Real Spike: Learning Real-valued Spikes for Spiking Neural Networks

Yufei Guo[⋆], Liwen Zhang[⋆], Yuanpei Chen, Xinyi Tong, Xiaode Liu, YingLei Wang, Xuhui Huang[✉], and Zhe Ma[✉]

Intelligent Science & Technology Academy of CASIC, Beijing 100854, China
`yfguo@pku.edu.cn, lwzhang9161@126.com, starhxh@126.com, mazhe_thu@163.com`

Abstract. Brain-inspired spiking neural networks (SNNs) have recently drawn more and more attention due to their event-driven and energy-efficient characteristics. The integration of storage and computation paradigm on neuromorphic hardwares makes SNNs much different from Deep Neural Networks (DNNs). In this paper, we argue that SNNs may not benefit from the weight-sharing mechanism, which can effectively reduce parameters and improve inference efficiency in DNNs, in some hardwares, and assume that an SNN with unshared convolution kernels could perform better. Motivated by this assumption, a training-inference decoupling method for SNNs named as **Real Spike** is proposed, which not only enjoys both unshared convolution kernels and binary spikes in inference-time but also maintains both shared convolution kernels and **Real**-valued **Spike**s during training. This decoupling mechanism of SNN is realized by a re-parameterization technique. Furthermore, based on the training-inference-decoupled idea, a series of different forms for implementing **Real Spike** on different levels are presented, which also enjoy shared convolutions in the inference and are friendly to both neuromorphic and non-neuromorphic hardware platforms. A theoretical proof is given to clarify that the Real Spike-based SNN network is superior to its vanilla counterpart. Experimental results show that all different **Real Spike** versions can consistently improve the SNN performance. Moreover, the proposed method outperforms the state-of-the-art models on both non-spiking static and neuromorphic datasets.

Keywords: Spiking neural network; Real spike; Binary spike; Training-inference-decoupled; Re-parameterization.

1 Introduction

Spiking Neural Networks (SNNs) have received increasing attention as a novel brain-inspired computing model that adopts binary spike signals to communicate between units. Different from the Deep Neural Networks (DNNs), SNNs transmit information by spike events, and the computation dominated by the addition operation occurs only when the unit receives spike events. Benefitting

[⋆] Equal contribution.

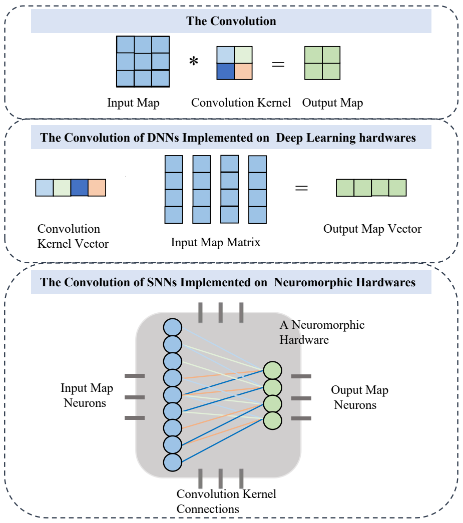
The Convolution
Input Map
Convolution Kernel
Output Map
The Convolution of DNNs Implemented on Deep Learning hardwares
Convolution Kernel Vector
Input Map Matrix
Output Map Vector
The Convolution of SNNs Implemented on Neuromorphic Hardwares
A Neuromorphic Hardware
Input Map Neurons
Ouput Map Neurons
Convolution Kernel Connections

Fig. 1. The difference of convolution computing process between DNNs and SNNs. For DNNs, the calculation is conducted in a highly-paralleled way on conventional hardwares. However, for SNNs, each connection between neurons will be mapped into a synapse on some neuromorphic hardwares, which cannot benefit from the advantages of the weight-shared convolution kernel, *e.g.*, inference acceleration and parameter reduction.

from this characteristic, SNNs can greatly save energy and run efficiently when implementing on neuromorphic hardwares, *e.g.*, SpiNNaker [17], TrueNorth [1], Darwin [28], Tianjic [32], and Loihi [5].

The success of DNNs inspires the SNNs in many ways. Nonetheless, the rich spatio-temporal dynamics, event-driven paradigm, and friendly to neuromorphic hardwares make SNNs much different from DNNs, and directly applying the successful experience of DNNs to SNNs may limit the performance of SNNs. As one of the most widely used techniques in DNNs, the weight-shared convolution kernel shows great advantages. It can reduce the parameters of the network and accelerate the inference. However, SNNs show great advantages in the condition of being implemented on neuromorphic hardwares which is very different from DNNs being implemented on deep learning hardwares. As shown in Fig. 1, in DNNs, the calculation is carried out in a highly-paralleled way on deep learning hardwares, thus sharing convolution kernel can improve the computing efficiency by reducing the data transferring between separated memory and processing units. However, for an ideal situation, to take full advantage of the storage-computation-integrated paradigm of neuromorphic hardwares, each unit and connection of the SNNs in the inference phase should be mapped into a neuron and synapse in neuromorphic hardware, respectively. Though these hardwares could be multiplexed, it also increases the complexity of deploymention and extra cost of data transfer. As far as we know, at least Darwin [28], Tianjic [32], and other memristor-enabled neuromorphic computing systems [41] adopt this one-to-one mapping form at present. Hence, all the components of an SNN will be deployed as a fixed configuration on these hardwares, no matter they share the same convolution kernel or not. Unlike the DNNs, the shared convolution kernels will not bring SNNs the advantages of parameter reduction and inference acceleration in this situation. Hence we argue that it would be better to learn unshared convolution kernels for each output feature map in SNNs.

Unfortunately, whether in theory or technology, it is not feasible to directly train an unshared convolution kernels-based SNN. First, there is no obvious proof that learning different convolution kernels directly will surely benefit the network performance. Second, due to the lack of mature development platforms for SNNs, many efforts are focusing on training SNNs with DNN-oriented programming frameworks, which usually do not support the learning of unshared convolution kernels for each feature map directly. Considering these limitations, we focus on training SNNs with unshared convolution kernels based on the modern DNN-oriented frameworks indirectly.

Driven by the above reasons, a training-time and inference-time decoupled SNN is proposed, where a neuron can emit *real-valued spikes* during training but binary spikes during inference, dubbed **Real Spike**. The training-time real-valued spikes can be converted to inference-time binary spikes via convolution kernel re-parameterization and a shared convolution kernel, which can be derived into multiples then (see details in Sec. 3.3). In this way, an SNN with different convolution kernels for every output feature map can be obtained as we expected. Specifically, in the training phase, the SNN will learn real-valued spikes and a shared convolution kernel for every output feature map. While in the inference phase, every real-valued spike will be transformed into a binary spike by folding a part of the value to its corresponding kernel weight. Due to the diversity of the real-valued spikes, by absorbing part of the value from each real spike, the original convolution kernel shared by each output map can be converted into multiple forms. Thus different convolution kernels for each feature map of SNNs can be obtained indirectly. It can be guaranteed theoretically that the **Real Spike** method can improve the performance due to the richer representation capability of real-valued spikes than binary spikes (see details in Sec. 3.4). Besides, **Real Spike** is well compatible with present DNN-oriented programming frameworks, and it still retains the advantages of DNN-oriented frameworks in terms of the convolution kernel sharing mechanism in the training. Furthermore, we extract the essential idea of training-inference-decoupled and extend **Real Spike** to a more generalized form, which is friendly to both neuromorphic and non-neuromorphic hardwares (see details in Sec. 3.5). The overall workflow of the proposed method is illustrated in Fig. 2.

Our main contributions are summarized as follows:

– We propose the **Real Spike**, a simple yet effective method to obtain SNNs with unshared convolution kernels. The Real Spike-SNN can be trained in DNN-oriented frameworks directly. It can effectively enhance the information representation capability of the SNN without introducing training difficulty.

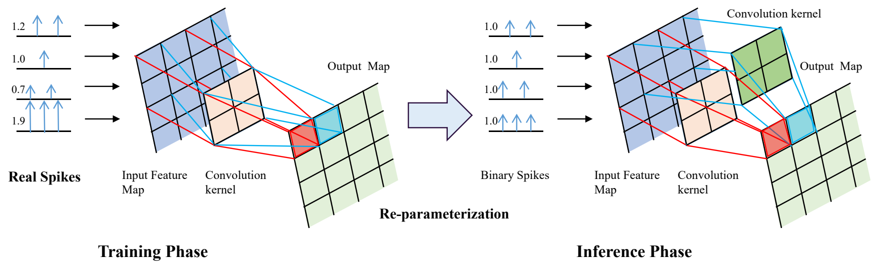

Fig. 2. The overall workflow of **Real Spike**. In the training phase, the SNN learns real-valued spikes and the shared convolution kernel for each output feature map. In the inference phase, the real spikes can be converted to binary spikes via convolution kernel re-parameterization. Then an SNN with different convolution kernels for every feature map is obtained.

- The convolution kernel re-parameterization is introduced to decouple a training-time SNN with real-valued spikes and shared convolution kernels, and an inference-time SNN with binary spikes and unshared convolution kernels.
- We extend the **Real Spike** to other different granularities (layer-wise and channel-wise). These extensions can keep shared convolution kernels in the inference and show advantages independent of specific hardwares.
- The effectiveness of **Real Spike** is verified on both static and neuromorphic datasets. Extensive experimental results show that our method performs remarkably.

2 Related Work

This work starts from training more accurate SNNs with unshared convolution kernels for each feature map. Considering the lack of specialized suitable platforms that can support the training of deep SNNs, powerful DNN-oriented programming frameworks are adopted. To this end, we briefly overview recent works of SNNs in three aspects: (i) learning algorithms of SNNs; (ii) SNN programming frameworks; (iii) convolutions.

2.1 Learning Algorithms of SNNs

The learning algorithms of SNNs can be divided into three categories: converting ANN to SNN (ANN2SNN) [2,13,36,25], unsupervised learning [6,14], and supervised learning [16,29,26,37,12]. ANN2SNN converts a pre-trained ANN to an SNN by transforming the real-valued output of the activation function to binary spikes. Due to the success of ANNs, ANN2SNN can generate an SNN in a short time with competitive performance. However, the converted ANN inherits the limitation of ignoring rich temporal dynamic behaviors from DNNs, it cannot handle neuromorphic datasets well. On the other hand, ANN2SNN usually requires hundreds of timesteps to approach the accuracy of pre-trained DNNs. Unsupervised learning is considered a more biologically plausible method. Unfortunately, the lack of sufficient understanding of the biological mechanism prevents the network from going deep, thus it is usually limited to small datasets and non-ideal performance. Supervised learning trains SNNs with error backpropagation. Supervised learning-based methods can not only achieve high accuracy within very few timesteps but also handle neuromorphic datasets. Our work falls under this category.

2.2 SNN Programming Frameworks

There exist several specialized programming frameworks for SNN modeling. However, most of them cannot support the direct training of deep SNNs. NEURON [3], a simulation environment for modeling computational models of neurons and networks of neurons, mainly focuses on simulating neuron issues with complex anatomical and physiological characteristics, which is more suitable for neuroscience research. BRIAN2 [11] and NEST [10], simulators for SNNs, aim at making the writing of simulation code as quick and easy as possible for the user, but they are not designed for the supervised learning for deep SNNs. SpikingJelly [7], an open-source deep learning framework for SNNs based on PyTorch, provides a solution to establish SNNs by mature DNN-oriented programming frameworks. However, so far most functions of the SpikingJelly are still under development. Instead of developing a new framework, we attempt to investigate an ingenious solution to develop the SNNs that have unshared convolutional kernels for each output feature map in the DNN-oriented frameworks, which have demonstrated an easy-to-use interface.

2.3 Convolutions

Convolutional Neural Networks have recently harvested a huge success in large-scale computer vision tasks [22,20,38,39,15,35]. Due to the abilities of input scale adaptation, translation invariance, and parameter reduction, the stacked convolution layers help train a deep and well-performed neural network with fewer parameters compared to dense-connected fully connected layers. For a convolution layer, feature maps of the previous layer are convolved with some learnable kernels and presented through the activation function to form the output feature maps as the current layer. Each learnable kernel is corresponding to an output map. In general, we have that

$$\mathbf{X}_{i,j} = f(\mathbf{I}_j * \mathbf{K}_i), \tag{1}$$

where $\mathbf{I}_j \in \mathbb{R}^{k\times k\times C}$ denotes j-th block of the input maps with total C channels, $\mathbf{K}_i \in \mathbb{R}^{k\times k\times C}$ is the i-th convolution kernel with $i = 1, \ \ldots, \ C$, then $\mathbf{X}_{i,j} \in \mathbb{R}$ is the j-th element in i-th output feature map. It can be seen that all the outputs in each channel share a same convolution kernel. However, when implementing a convolution-based SNN on above mentioned neuromorphic hardwares, all the kernels will be mapped as synapses no matter they are shared or not. Hence, keeping shared convolution kernel cannot show the same advantages for SNNs as DNNs. We argue that learning different convolution kernels for each output map may improve the performence of the SNN further. In this case, a convolution layer for SNNs can be written as

$$\mathbf{X}_{i,j} = f(\mathbf{I}_j * \mathbf{K}_{i,j}), \tag{2}$$

where $\mathbf{K}_{i,j} \in \mathbb{R}^{k\times k\times C}$ is the j-th convolution kernel for i-th output map. Unfortunately, it is not easy to directly implement the learning of unshared convolution kernel for SNNs in the DNN-oriented frameworks. And dealing with this issue is one of the important works in this paper.

3 Materials and Methodology

Aiming at training more accurate SNNs, we adopt the explicitly iterative leaky Integrate-and-Fire (LIF) model, which can be implemented on mature DNN-oriented frameworks easily to simulate the fundamental computing unit of SNNs. Considering that it is not easy to realize unshared convolution kernels for each output feature map with existing DNN-oriented frameworks, a modified LIF model that can emit the real-valued spike is proposed first. By using this modified LIF model, our SNNs will learn real spikes along with the shared convolution kernel for every output channel as DNNs during training. While for the inference phase, real spikes will be transformed into binary spikes and each convolution kernel will be re-parameterized as multiple different convolution kernels, so the advantages of SNNs can be recovered and unshared convolution kernel for each output map can be obtained. Then, to make the proposed design more general, we also propose layer-wise and channel-wise **Real Spike**, which can keep shared convolution kernels in both training phase and inference phase and will introduce no more parameters than its vanilla counterpart. In this section, we will introduce explicitly iterative LIF model, **Real Spike**, re-parameterization, and extensions of **Real Spike** successively.

3.1 Explicitly Iterative Leaky Integrate-and-Fire Model

The spiking neuron is the fundamental computing unit of SNNs. The LIF neuron is most commonly used in supervised learning-based methods. Mathematically, a LIF neuron can be described as

$$\tau\frac{\partial u}{\partial t} = -u + I, \quad u < V_{th} \tag{3}$$

$$u = u_{rest} \quad \& \quad fire\ a\ spike, \quad u \geq V_{th} \tag{4}$$

where u, u_{rest}, τ, I, and V_{th} represent the membrane potential, membrane resting potential, membrane time constant that controls the decaying rate of u, pre-synaptic input, and the given firing threshold, respectively. When u is below

V_{th}, it acts as a leaky integrator of I. On the contrary, when u exceeds V_{th}, it will fire a spike and propagate the spike to the next layer, then it will be reset to the resting potential, u_{rest}, which is usually set as 0.

The LIF model has a complex dynamics structure that is incompatible with nowadays DNN-oriented framework. By discretizing and transforming the LIF model to an explicitly iterative LIF model, SNNs can be implemented in these mature frameworks. The hardware friendly iterative LIF model can be described as

$$u(t) = \tau u(t-1) + I(t), \quad u(t) < V_{th} \tag{5}$$

$$o(t) = \begin{cases} 1, & if\ u(t) \geq V_{th}, \\ 0, & otherwise. \end{cases} \tag{6}$$

where V_{th} is set to 0.5 in this work. Up to now, there is still an obstacle for training SNNs in a direct way, *i.e.*, Eq. (6) is non-differentiable. As in other work [33,40,4], we appoint a rectangular function as the particular pseudo derivative of spike firing as follows,

$$\frac{do}{du} = \begin{cases} 1, & if\ 0 \leq u \leq 1, \\ 0, & otherwise. \end{cases} \tag{7}$$

With all these settings, now we can train an SNN on DNN-oriented frameworks.

3.2 Real Spike

As aforementioned, driven by the suppressed advantages of the shared convolution kernel and the expectation of enhancing the capacity of information representation for SNNs, we turn the problem of learning unshared convolution kernels to learning real spikes. To be more specific, the output of our modified LIF model in Eq. (6) is further rewritten as

$$\tilde{o}(t) = a \cdot o(t) \tag{8}$$

where a is a learnable coefficient. With this modification, our LIF model can emit a *real-valued spike*, dubbed **Real Spike**. Then we train SNNs with this modified LIF model along with the shared convolution kernel for each output map, which can be easily implemented in DNN-oriented frameworks. Obviously, unlike the binary spike, the real-valued spike will lose the advantage of computation efficiency of SNNs, since the corresponding multiplication cannot be substituted to addition. And another problem is that the learned convolution kernels for each output map are still shared at this time. Therefore, to jointly deal with these problems, we propose a training-inference decoupled framework, which can transform real spikes into binary spikes and convert the shared convolution kernel as different kernels by using re-parameterization, which will be introduced in the next subsection.

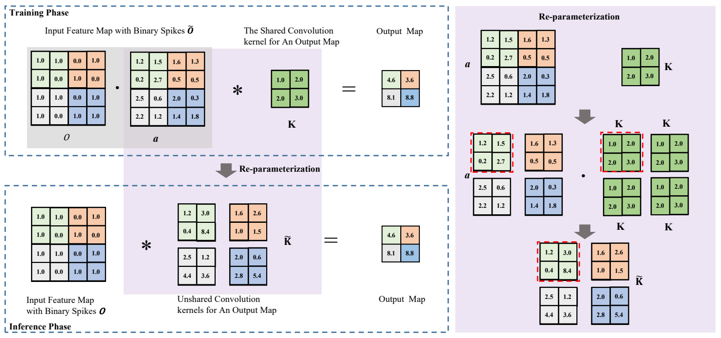
Training Phase
Input Feature Map with Binary Spikes $\tilde{O}$
The Shared Convolution kernel for An Output Map
Output Map
Re-parameterization
Inference Phase
Input Feature Map with Binary Spikes O
Unshared Convolution kernels for An Output Map
K
$\tilde{K}$
a
O

Fig. 3. The diagram of re-parameterization by a simple example.

3.3 Re-parameterization

Consider a convolution layer, which takes $\mathbf{F} \in \mathbb{R}^{D_F \times D_F \times M}$ as input feature maps, and generates the output feature maps, $\mathbf{G} \in \mathbb{R}^{D_G \times D_G \times N}$, where D_F and M denote the size of the square feature maps and the number of channels (depths) for the input, respectively; D_G and N denote the size of the square feature maps and the number of channels (depths) for the output, respectively. The convolution layer is actually parameterized by a group of convolution kernels, which can be denoted as a tensor, $\mathbf{K} \in \mathbb{R}^{D_K \times D_K \times M \times N}$ with a spatial dimension of D_K. Then, each element in $\mathbf{G}$ is computed as

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \tag{9}$$

For standard SNNs, the elements of input maps are binary spikes, while in this work, the SNN is trained with real-valued spikes for the purpose of enhancing the network representation capacity. In this case, we can further denote the input feature map, $\mathbf{F}$, according to Eq. (8) as follows

$$\mathbf{F} = \mathbf{a} \odot \mathbf{B} \tag{10}$$

where $\mathbf{B}$ and $\mathbf{a}$ denote a binary tensor and a learnable coefficient tensor, respectively. With this element-wise multiplication in Eq. (10), we can extract a part of the value from each element in $\mathbf{F}$, and fold it into the shared convolution kernel one-by-one according to the corresponding position during inference. Then the single shared convolution kernel can be turned into multiples without changing the values of the output maps. Through this decoupling process, a new SNN that can emit binary spikes and enjoy different convolution kernels will be obtained. This process can be illustrated from Eq. (11) to Eq. (13) as follows:

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot (\mathbf{a}_{k+i-1,l+j-1,m} \cdot \mathbf{B}_{k+i-1,l+j-1,m}) \tag{11}$$

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} (\mathbf{a}_{k+i-1,l+j-1,m} \cdot \mathbf{K}_{i,j,m,n}) \cdot \mathbf{B}_{k+i-1,l+j-1,m} \tag{12}$$

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \tilde{\mathbf{K}}_{k,l,i,j,m,n} \cdot \mathbf{B}_{k+i-1,l+j-1,m} \tag{13}$$

where $\tilde{\mathbf{K}}$ is the unshared convolution kernel tensor.

The whole process described above is called re-parameterization, which allows us to convert a real-valued-spike-based SNN into an output-invariant binary-spike-based SNN with unshared convolution kernels for each output map. That is, re-parameterization provides a solution to obtain an SNN with unshared convolution kernels under DNN-oriented frameworks by decoupling the training-time SNN and inference-time SNN. Figure 3 illustrates the details of re-parameterization by a simple example.

3.4 Analysis and Discussions

In this work, we assume that firing real-valued spikes during training can help increase the representation capacity of the SNNs. To verify our assumption, a series of analyses and discussions are conducted by using the information entropy concept. Given a scalar, vector, matrix, or tensor, $\mathbf{X}$, its representation capability is denoted as $\mathcal{R}(\mathbf{X})$, which can be measured by the information entropy of $\mathbf{X}$, as follows

$$\mathcal{R}(\mathbf{X}) = \max \mathcal{H}(\mathbf{X}) = \max(-\sum_{x\in\mathbf{X}} p_{\mathbf{X}}(x) log p_{\mathbf{X}}(x)) \tag{14}$$

where $p_{\mathbf{X}}(x)$ is the probability of a sample from $\mathbf{X}$. When $p_{\mathbf{X}}(x_1) = \cdots = p_{\mathbf{X}}(x_n)$, $\mathcal{H}(\mathbf{X})$ reaches its maximum (see Appendix A.1 for detailed proofs). For a binary spike o, it can be expressed with 1 bit, and the number of samples from o is 2. While the real-valued spike $\tilde{o}$ needs 32 bits, which consists of 2^{32} samples. Hence, $\mathcal{R}(o) = 1$ and $\mathcal{R}\ (\tilde{o}) = 32$ according to Eq. (14). Obviously, the representation capability of real spikes far exceeds that of binary spikes. This indicates that real spikes will enhance the information expressiveness of SNNs, which accordingly benefit the performance improvement. To further show the difference between real spikes and binary spikes intuitively, the visualizations of some channels expressed by real spikes and binary spikes are given respectively in the appendix.

Another intuitive conjecture to explain why the SNN with real-valued spikes performs better than its counterpart with binary spikes is that, for the former one, the information loss can be restrained to some extent by changing the fixed spike value to an appropriate value with a scalable coefficient, a; while for the later one, the firing function would inevitably induce the quantization error.

It can be concluded from the above analysis that, learning real-valued spikes instead of the binary spikes in the training phase, enables us to train a more accurate SNN. And by performing re-parameterization in the inference phase, real spikes can be converted to binary spikes and unshared convolution kernels can be obtained. In another word, learning **Real Spike** is actually used to generate a better information encoder, while the re-parameterization will transfer the

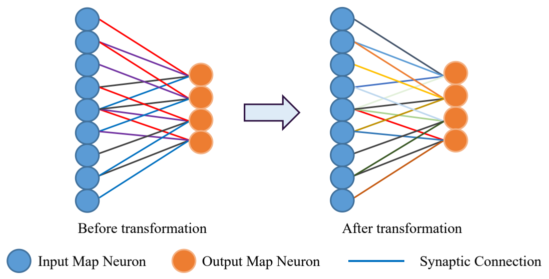

Fig. 4. The difference of adjacent layers in SNNs implemented on neuromorphic hardwares before re-parameterization and after re-parameterization. The convolution kernel re-parameterization will only change the values of the connections between neurons without introducing any additional computation and storage resource since the entire architecture topology of an SNN must be completely mapped into the hardwares.

rich information encoding capacity from **Real Spike** into information decoding. Moreover, the transformed model after re-parameterization can also retain the advantage coming from the binary spike information processing paradigm in standard SNNs. As shown in Fig. 4, when deploying an SNN into some neuromorphic hardwares, all the units and their connections of the network will be mapped as neurons and synaptic connections one by one. Hence, the SNN with multiple different convolution kernels will not introduce any computation and storage resource.

3.5 Extensions of Real Spike

The key observation made in the **Real Spike** is that re-scaling the binary spike of the LIF neuron with a real-valued coefficient, a, can increase the representation capability and accurancy of SNNs. As shown in Eq. (10), the default Real Spike is performed in the element-wise way. In a similar fashion, we argue that introducing scaling coefficent by layer-wise or channel-wise manners will also retain the benefit to some extent. Then, we further propose to re-formulate Eq. (8) for one layer as follows

$$\tilde{\mathbf{o}}(t) = \mathbf{a} \cdot \mathbf{o}(t) \tag{15}$$

With this new formulation we can explore various ways of introducing $\mathbf{a}$ during training. Specifically, we propose to introduce $\mathbf{a}$ for each layer in the following 3 ways:

Layer-wise:

$$\mathbf{a} \in \mathbb{R}^{1\times1\times1} \tag{16}$$

Channel-wise:

$$\mathbf{a} \in \mathbb{R}^{C\times1\times1} \tag{17}$$

Element-wise:

$$\mathbf{a} \in \mathbb{R}^{C\times H\times H} \tag{18}$$

Table 1. Ablation study for **Real Spike**.

Dataset	Architecture	Timestep	Accuracy
CIFAR10	ResNet20 w/ BS	2	88.91%
		4	91.73%
		6	92.98%
	ResNet20 w/ RS	2	90.47%
		4	92.53%
		6	93.44%
CIFAR100	ResNet20 w/ BS	2	62.59%
		4	63.27%
		6	67.12%
	ResNet20 w/ RS	2	63.40%
		4	64.87%
		6	68.60%

RS represents real spikes and BS represents binary spikes.

where C is the number of channels and H is the spatial dimension of a square feature map. Obviously, for layer-wise and channel-wise Real Spike, re-parameterization will only re-scale the shared convolution kernels without transferring them to different ones. In these two forms, our **Real Spike** act in the same way as conventional SNNs on any hardwares. That is to say, starting from the motivation of how to take full advantage of the integration of memory and computation, we propose element-wise **Real Spike**. Then we extract the essential idea of training-inference-decoupled and extend **Real Spike** to a more generalized form, which is friendly to both neuromorphic and non-neuromorphic hardware platforms.

4 Experiment

The performance of the **Real Spike**-based SNNs were evaluated on several traditional static datasets (CIFAR-10 [19], CIFAR-100 [19], ImageNet [20]) and one neuromorphic dataset (CIFAR10-DVS [24]). And multiple widely-used spiking archetectures including ResNet20 [33,36], VGG16 [33], ResNet18 [8], and ResNet34 [8] were used to verify the effectiveness of our **Real Spike**. Detailed introduction of the datasets and experimental settings are provided in the appendix. Extensive ablation studies were conducted first to compare the SNNs with real-valued spikes and their binary spike counterpart. Then, we comprehensively compared our SNNs with the existing state-of-the-art SNN methods.

4.1 Ablation Study for Real Spike

The ablation study of **Real Spike** was conducted on CIFAR-10/100 datasets based on ResNet20. The SNNs with real-valued spikes and binary spikes were trained with the same configuration, with timestep varying from 2 to 6. Results in Tab. 1 show that the test accuracy of the SNNs with real-valued spikes is always

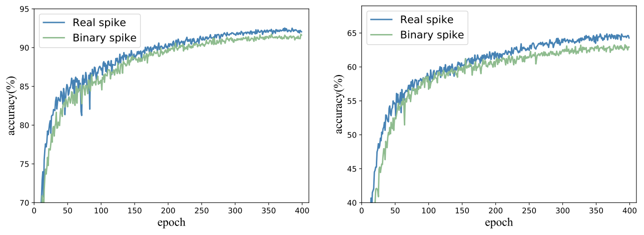

Fig. 5. The accuracy curves of ResNet20 with real-valued spikes and binary spikes with a timestep = 4 on CIFAR-10(left) and CIFAR-100(right). The real-valued spikes-based SNNs obviously enjoy higher accuracy and easier convergence.

Table 2. Performance comparison for different **Real Spike** versions.

Dataset	Architecture	version	Accuracy
CIFAR10	ResNet20	Vanilla	91.73%
		Layer-wise	92.12%
		Channel-wise	92.25%
		Element-wise	92.53%
CIFAR100	ResNet20	Vanilla	63.27%
		Layer-wise	64.28%
		Channel-wise	64.71%
		Element-wise	64.87%

higher than that with binary spikes. This is due to the richer representation capacity from Real Spike, which benefits the performance improvement of SNNs. Figure 5 illustrates the test accuracy curves of ResNet20 with real-valued spikes and its counterpart with binary spikes on CIFAR-10/100 during training. It can be seen that the SNNs with real-valued spikes obviously perform better on the accuracy and convergence speed.

4.2 Ablation Study for Extensions of Real Spike

The performance of SNNs with different extensions of Real Spike was evaluated on CIFAR-10(100). Results in Tab. 2 show that element-wise **Real Spike** always outperforms the other versions. But all the Real Spike-based SNNs conformably outperform the vanilla (standard) SNN. Another observation is that the accuracy difference between layer-wise **Real Spike** and the binary spike is greater than that between layer-wise **Real Spike** and element-wise **Real Spike**. This shows that our method touches on the essence of improving the SNN accuracy and is very effective.

4.3 Comparison with the State-of-the-art

In this section, we compared the **Real Spike**-based SNNs with other state-of-the-art SNNs on several static datasets and a neuromorphic dataset. For each

Table 3. Performance comparison with SOTA methods.

Dataset	Method	Type	Architecture	Timestep	Accuracy
CIFAR-10	SpikeNorm [36]	ANN2SNN	VGG16	2500	91.55%
	Hybrid-Train [34]	Hybrid	VGG16	200	92.02%
	SBBP [23]	SNN training	ResNet11	100	90.95%
	STBP [40]	SNN training	CIFARNet	12	90.53%
	TSSL-BP [42]	SNN training	CIFARNet	5	91.41%
	PLIF [9]	SNN training	PLIFNet	8	93.50%
	Diet-SNN [33]	SNN training	VGG16	5	92.70%
				10	93.44%
			ResNet20	5	91.78%
				10	92.54%
	STBP-tdBN [43]	SNN training	ResNet19	2	92.34%
				4	92.92%
				6	93.16%
	Real Spike	SNN training	ResNet19	2	**94.01%**±0.10
				4	**95.60%**±0.08
				6	**95.71%**±0.07
			ResNet20	5	**93.01%**±0.07
				10	**93.65%**±0.05
			VGG16	5	**92.90%**±0.09
				10	**93.58%**±0.06
CIFAR-100	BinarySNN [27]	ANN2SNN	VGG15	62	63.20%
	Hybrid-Train [34]	Hybrid	VGG11	125	67.90%
	T2FSNN [31]	ANN2SNN	VGG16	680	68.80%
	Burst-coding [30]	ANN2SNN	VGG16	3100	68.77%
	Phase-coding [18]	ANN2SNN	VGG16	8950	68.60%
	Diet-SNN [33]	SNN training	ResNet20	5	64.07%
			VGG16	5	69.67%
	Real Spike	SNN training	ResNet20	5	**66.60%**±0.11
			VGG16	5	**70.62%**±0.08
			VGG16	10	**71.17%**±0.07
ImageNet	Hybrid-Train [34]	Hybrid	ResNet34	250	61.48%
	SpikeNorm [36]	ANN2SNN	ResNet34	2500	69.96%
	STBP-tdBN [43]	SNN training	ResNet34	6	63.72%
	SEW ResNet [8]	SNN training	ResNet18	4	63.18%
			ResNet34	4	67.04%
	Real Spike	SNN training	ResNet18	4	**63.68%**±0.08
			ResNet34	4	**67.69%**±0.07
CIFAR10-DVS	Rollout [21]	Streaming	DenseNet	10	66.80%
	STBP-tdBN [43]	SNN training	ResNet19	10	67.80%
	Real Spike	SNN training	ResNet19	10	**72.85%**±0.12
			ResNet20	10	**78.00%**±0.10

run, we report the mean accuracy as well as the standard deviation with 3 trials. Experimental results are shown in Tab. 3.

CIFAR-10(100). On CIFAR-10, our SNNs achieve higher accuracy than the other state-of-the-art methods, and the best result of 95.71% top-1 accuracy is achieved by ResNet19 with 6 timesteps. And even trained with much fewer timesteps, *i.e.*, 2, our ResNet19 still outperforms the STBP-tdBN under 6 timesteps with 0.85% higher accuracy. This comparison shows that our method also enjoys the advantage of latency reduction for SNNs. On CIFAR-100, Real Spike also performs better and achieves a 2.53% increment on ResNet20 and a 0.95% increment on VGG16.

ImageNet. The ResNet18 and ResNet34 were selected as the backbones. Considering that the image size of the samples is much larger, the channel-wise Real Spike was used. For a fair comparison, we made our architectures consistent with SEW-ResNets, which are not typical SNNs, where the IF model and modified residual structure are adopted. Results show that, even with 120 fewer epochs of training (200 for ours, 320 for SEW-ResNets), the channel-wise Real Spike-based SNNs can still outperform SEW-ResNets. In particular, our method achieves a 0.5% increment on ResNet18 and a 0.65% increment on ResNet34. Moreover, Real Spike-based ResNet34 with 4 timesteps still outperforms STBP-tdBN-based RersNet34 with 6 timesteps by 3.97% higher accuracy.

CIFAR10-DVS. To further verify the generalization of the Real Spike, we also conducted experiments on the neuromorphic dataset CIFAR10-DVS. Using ResNet20 as the backbone, Real Spike achieves the best performance with 78.00% accuracy in 10 timesteps. For ResNet19, Real Spike obtains 5.05% improvement compared with STBP-tdBN. It's worth noting that, as a more complex model, ResNet19 performs worse than ResNet20. This is because that neuromorphic datasets usually suffer much more noise than static ones, thus more complex models are easier to overfit on these noisier datasets.

5 Conclusions

In this work, we focused on the difference between SNNs and DNNs and speculated that the unshared convolution kernel-based SNNs would enjoy more advantages than those with shared convolution kernels. Motivated by this idea, we proposed **Real Spike**, which aims at enhancing the representation capacity for an SNN by learning real-valued spikes during training and transferring the rich representation capacity into inference-time SNN by re-parameterizing the shared convolution kernel to different ones. Furthermore, a series of **Real Spike**s in different granularities were explored, which are enjoy shared convolution kernels in both training and inference phases and friendly to both neuromorphic and non-neuromorphic hardware platforms. Proof of why **Real Spike** has a better performance than vanilla SNNs was provided. Extensive experiments verified that our proposed method consistently achieves better performance than the existed state-of-the-art SNNs.

References

1. Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.J., et al.: 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 **34**(10), 1537–1557 (2015) 2
2. Cao, Y., Chen, Y., Khosla, D.: Spiking deep convolutional neural networks for energy-efficient object recognition. International Journal of Computer Vision **113**(1), 54–66 (2015) 4
3. Carnevale, N.T., Hines, M.L.: The NEURON book. Cambridge University Press (2006) 5
4. Cheng, X., Hao, Y., Xu, J., Xu, B.: Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition. In: IJCAI. pp. 1519–1525 (2020) 7
5. Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al.: Loihi: A neuromorphic manycore processor with on-chip learning. Ieee Micro **38**(1), 82–99 (2018) 2
6. Diehl, P.U., Cook, M.: Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Frontiers in computational neuroscience **9**, 99 (2015) 4
7. Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., Tian, Y., other contributors: Spikingjelly. https://github.com/fangwei123456/spikingjelly (2020) 5
8. Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., Tian, Y.: Deep residual learning in spiking neural networks. Advances in Neural Information Processing Systems **34**, 21056–21069 (2021) 11, 13
9. Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., Tian, Y.: Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 2661–2671 (2021) 13
10. Gewaltig, M.O., Diesmann, M.: Nest (neural simulation tool). Scholarpedia **2**(4), 1430 (2007) 5
11. Goodman, D.F., Brette, R.: The brian simulator. Frontiers in neuroscience p. 26 (2009) 5
12. Guo, Y., Tong, X., Chen, Y., Zhang, L., Liu, X., Ma, Z., Huang, X.: Recdis-snn: Rectifying membrane potential distribution for directly training spiking neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 326–335 (June 2022) 4
13. Han, B., Roy, K.: Deep spiking neural network: Energy efficiency through time based coding. In: European Conference on Computer Vision. pp. 388–404. Springer (2020) 4
14. Hao, Y., Huang, X., Dong, M., Xu, B.: A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. Neural Networks **121**, 387–395 (2020) 4
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 5
16. Huh, D., Sejnowski, T.J.: Gradient descent for spiking neural networks. Advances in neural information processing systems **31** (2018) 4

17. Khan, M.M., Lester, D.R., Plana, L.A., Rast, A., Jin, X., Painkras, E., Furber, S.B.: Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor. In: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). pp. 2849–2856. Ieee (2008) 2
18. Kim, J., Kim, H., Huh, S., Lee, J., Choi, K.: Deep neural networks with weighted spikes. Neurocomputing **311**, 373–386 (2018) 13
19. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html **5**(4), 1 (2010) 11
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012) 5, 11
21. Kugele, A., Pfeil, T., Pfeiffer, M., Chicca, E.: Efficient processing of spatio-temporal data streams with spiking neural networks. Frontiers in Neuroscience **14**, 439 (2020) 13
22. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998) 5
23. Lee, C., Sarwar, S.S., Panda, P., Srinivasan, G., Roy, K.: Enabling spike-based backpropagation for training deep neural network architectures. Frontiers in neuroscience p. 119 (2020) 13
24. Li, H., Liu, H., Ji, X., Li, G., Shi, L.: Cifar10-dvs: an event-stream dataset for object classification. Frontiers in neuroscience **11**, 309 (2017) 11
25. Li, Y., Deng, S., Dong, X., Gong, R., Gu, S.: A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In: International Conference on Machine Learning. pp. 6316–6325. PMLR (2021) 4
26. Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., Gu, S.: Differentiable spike: Rethinking gradient-descent for training spiking neural networks. Advances in Neural Information Processing Systems **34**, 23426–23439 (2021) 4
27. Lu, S., Sengupta, A.: Exploring the connection between binary and spiking neural networks. Frontiers in Neuroscience **14**, 535 (2020) 13
28. Ma, D., Shen, J., Gu, Z., Zhang, M., Zhu, X., Xu, X., Xu, Q., Shen, Y., Pan, G.: Darwin: A neuromorphic hardware co-processor based on spiking neural networks. Journal of Systems Architecture **77**, 43–51 (2017) 2
29. Neftci, E.O., Mostafa, H., Zenke, F.: Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. IEEE Signal Processing Magazine **36**(6), 51–63 (2019) 4
30. Park, S., Kim, S., Choe, H., Yoon, S.: Fast and efficient information transmission with burst spikes in deep spiking neural networks. In: 2019 56th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2019) 13
31. Park, S., Kim, S., Na, B., Yoon, S.: T2fsnn: Deep spiking neural networks with time-to-first-spike coding. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2020) 13
32. Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., Wang, G., Zou, Z., Wu, Z., He, W., et al.: Towards artificial general intelligence with hybrid tianjic chip architecture. Nature **572**(7767), 106–111 (2019) 2, 3
33. Rathi, N., Roy, K.: Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. arXiv preprint arXiv:2008.03658 (2020) 7, 11, 13
34. Rathi, N., Srinivasan, G., Panda, P., Roy, K.: Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. arXiv preprint arXiv:2005.01807 (2020) 13

35. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems **28** (2015) 5
36. Sengupta, A., Ye, Y., Wang, R., Liu, C., Roy, K.: Going deeper in spiking neural networks: Vgg and residual architectures. Frontiers in neuroscience **13**, 95 (2019) 4, 11, 13
37. Shrestha, S.B., Orchard, G.: SLAYER: Spike layer error reassignment in time. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31. pp. 1419–1428. Curran Associates, Inc. (2018), `http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf` 4
38. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014) 5
39. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015) 5
40. Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., Shi, L.: Direct training for spiking neural networks: Faster, larger, better. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 1311–1318 (2019) 7, 13
41. Yao, P., Wu, H., Gao, B., Tang, J., Zhang, Q., Zhang, W., Yang, J.J., Qian, H.: Fully hardware-implemented memristor convolutional neural network. Nature **577**(7792), 641–646 (2020) 3
42. Zhang, W., Li, P.: Temporal spike sequence learning via backpropagation for deep spiking neural networks. Advances in Neural Information Processing Systems **33**, 12022–12033 (2020) 13
43. Zheng, H., Wu, Y., Deng, L., Hu, Y., Li, G.: Going deeper with directly-trained larger spiking neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 11062–11070 (2021) 13