Title: Secure Spiking Neural Network on FPGA

Author: Rupesh Raj Karn[1]

1 Introduction to SNN

A Spiking Neural Network (SNN) is a computational model inspired by biology, designed to emulate the workings of the human brain with greater fidelity compared to traditional artificial neural networks. Unlike conventional neural networks, which utilize continuous activation values, SNNs operate based on discrete events known as spikes, mirroring the firing of neurons in the human brain. These spikes are generated in response to input stimuli and propagate through an interconnected network of neurons. SNNs are renowned for their exceptional capacity to effectively represent temporal information, rendering them particularly well-suited for tasks that involve time-dependent data, such as pattern recognition, event-driven processing, and even the implementation of neuromorphic hardware. By harnessing the principles of neural spiking and synaptic plasticity, Spiking Neural Networks provide a promising avenue for investigating complex cognitive processes and constructing models that closely mimic biological systems, spanning a wide array of applications, from robotics to cutting-edge neuroscience research. In recent years, these networks have garnered significant attention for their potential in real-time processing, event recognition, and the development of neuromorphic hardware, offering a distinctive perspective on neural computation and cognitive functions.

1.1 Dissecting SNN at High Level

We delve into the complex mechanisms driving SNNs, providing insights into the neurons, synapses, and the distinctive temporal dynamics that differentiate them from conventional artificial neural networks.

1. Spiking Neurons: At the heart of every Spiking Neural Network lie its neurons. In contrast to the continuous activation values found in typical neural networks, SNNs utilize spiking neurons, which function in a discrete, event-driven manner. Each neuron accumulates incoming signals and emits an output spike once a specific membrane potential threshold is met.
2. Membrane Potential: The membrane potential plays a crucial role in the behavior of spiking neurons. It serves as a vital indicator of the neuron's internal state and is subject to influences from incoming spikes originating in connected neurons. This potential undergoes dynamic changes over time, either increasing or decreasing, as it integrates incoming signals. Once the membrane potential exceeds a predefined threshold, the neuron triggers a spike, thus initiating communication with other neurons within the network.
3. Synaptic Weighting: Within SNNs, synapses assume a central role in shaping the strength of connections between neurons. Every synapse is linked to a distinct weight that regulates the impact of an incoming spike on the membrane potential of the recipient neuron. These synaptic weights are acquired through the training process and serve as a cornerstone of the network's capacity to acquire knowledge and adjust to diverse tasks.
4. Time-Based Encoding: A notable characteristic of SNNs lies in their priority on time-based encoding. Rather than focusing on the frequency or rate of spike occurrences, the exact timing of spikes takes precedence. This emphasis on temporal encoding empowers SNNs to excel in tasks that revolve around dynamic and time-critical data.

1.2 Mathematics of Training SNN

In the training of a Spiking Neural Network (SNN), several mathematical concepts come into play.

1.2.1 Objective Function

The training process begins with defining an objective function or loss function, often denoted as $L(\theta)$, where θ represents the set of parameters, including synaptic weights. For example, the mean squared error (MSE) loss is often used for regression tasks:

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - f(x_i, \theta)\right)^2$$

1.2.2 Gradient Descent

Gradient descent is commonly used to minimize the objective function and update the parameters iteratively. The gradient descent update rule is given by:

$$\theta \leftarrow \theta - \eta \cdot \nabla L(\theta)$$

[1]Postdoctoral Associate at Center for Cyber Security, Department of Electrical Engineering and Computer Science, New York University, Abu Dhabi, UAE. rupesh.k@nyu.edu, +971-552959384, rkarn, rupesh-raj-karn-51614417 Scholar

Where η is the learning rate.

1.2.3 Spiking Neuron Models

To compute gradients, a differentiable approximation of spiking neuron models, such as the leaky integrate-and-fire (LIF) model, is used. This approximation allows for the calculation of gradients based on the neuron's membrane potential dynamics.

The membrane potential of neuron i at time t is given by:

$$V_i(t) = \sum_j w_{ij} * S_j(t)$$

Where w_{ij} is the synaptic weight from neuron j to neuron i, and $S_j(t)$ is the spike train of neuron j.

1.2.4 Spike-Time-Dependent Plasticity (STDP)

Many SNNs use learning rules based on the relative timing of pre-synaptic and post-synaptic spikes, known as Spike-Time-Dependent Plasticity (STDP). The weight update rule for STDP can be represented as:

$$\Delta w_{ij} = A^+ \cdot \exp(-\Delta t/\tau^+) - A^- \cdot \exp(\Delta t/\tau^-)$$

Where Δt is the time difference between pre-synaptic and post-synaptic spikes, and A^+ and A^- are positive and negative learning rates, respectively.

1.3 Mathematics of Inference on SNN

Inference on a Spiking Neural Network (SNN) involves several mathematical concepts and principles.

1.3.1 Membrane Potential Dynamics

The dynamics of the membrane potential ($V_i(t)$) of neuron i at time t can be described using the leaky integrate-and-fire (LIF) model. It follows the equation:

$$\tau_m \frac{dV_i(t)}{dt} = R_m \left(I_i(t) - V_i(t)\right)$$

Where:

- τ_m is the membrane time constant.
- R_m is the membrane resistance.
- $I_i(t)$ is the input current to neuron i at time t.

The neuron fires a spike when the membrane potential reaches a threshold V_{th}:

$$V_i(t) = V_{\text{th}} \quad \text{if} \quad V_i(t^-) < V_{\text{th}}$$

1.3.2 Spike Generation

When a neuron fires a spike, it generates a delta function spike, often represented as $\delta(t - t_i)$, where t_i is the time of the spike. The spike is then transmitted to post-synaptic neurons.

1.3.3 Synaptic Transmission

The effect of a spike generated by neuron j on the membrane potential of neuron i can be represented as:

$$V_i(t) = V_i(t^-) + w_{ij} \cdot \delta(t - t_j)$$

Where:

- w_{ij} is the synaptic weight from neuron j to neuron i.
- $\delta(t - t_j)$ represents the effect of the spike from neuron j at time t_j on neuron i.

1.3.4 Spiking Activity

The spiking activity of an SNN can be analyzed through spike trains, which represent the occurrence of spikes over time. The spike train of neuron i can be denoted as $S_i(t)$ and is typically a binary sequence.

$$S_i(t) = \sum_j \delta(t - t_j^i)$$

where, t_j^i is the time of the jth spike of neuron i.

2 Defining Attack

An attack procedure aims to compromise the security and integrity of Spiking Neural Network (SNN) models and their computations when they are deployed on Field-Programmable Gate Arrays (FPGAs). In this scenario, attackers may employ various tactics, including side-channel attacks, to gain insights into the operation of the FPGA-based neural network without direct access to the model or data. They may scrutinize power consumption, electromagnetic emissions, or timing patterns to infer details about the network's architecture, calculations, or even the input data. Another potential attack avenue involves manipulating the FPGA hardware, which can be achieved through physical tampering with the device or malicious reprogramming. By exploiting vulnerabilities in the implementation or communication interfaces, attackers may attempt to pilfer sensitive *"model parameters"* or introduce malicious backdoors, potentially compromising the confidentiality and integrity of the neural network's operations. Implementing robust security measures is essential to shield FPGA-based neural networks from such attack models and to ensure the preservation of their functionality and data privacy.

2.1 Attack Model

For any security framework, it is crucial to explicitly define the assumptions related to which entities are considered trusted and untrusted, as well as the specific assets accessible to each entity. According to the existing literature, the threat model is described in the following manner:

The attacker possesses information about the encryption algorithm in use and is aware of the specific locations where the key is implemented to secure various components of the Spiking Neural Network (SNN). These components include *weights* $w_{ij} \in \boldsymbol{\theta}$*, spiking thresholds* V_{th}*, encoding* $\delta(t - t_i)$*, activation functions* $\phi(y)$*,* and others. The sole element remaining unknown to the attacker is the secret key's value, which is represented as a binary vector. Furthermore, the adversary may have physical access to the FPGA that runs the encrypted SNN. The adversary can extract the gate-level netlist from the FPGA through I/O query-based attacks like SAT, oracle-less attacks, such as machine learning-based methods, or via reverse engineering techniques.

As detailed in Section 1.1, three crucial elements, specifically the membrane potential, weights, and encoding, draw the focus of potential attackers. If any of these components remains unencrypted, it could potentially provide a vulnerability for attackers to exploit. This is because the knowledge of one component may aid the attacker in deducing the others. *Hence, it is clear that ensuring the equal security of all these components is of utmost importance.*

2.2 Challenges

As described in Section 2.1, it's evident that a Spiking Neural Network (SNN) implementation lacking the simultaneous encryption of the membrane potential V_{th}, weights $w_{ij} \in \boldsymbol{\theta}$, and encoding $\delta(t - t_i)$ cannot be considered a secure deployment. Therefore, one effective means to bolster security is the integration of homomorphic encryption.

Although the integration of homomorphic encryption with SNNs offers immense promise, it introduces several challenges. SNNs are inherently computationally demanding, and performing computations on encrypted data is notably slower than on unencrypted data. The computational complexity escalates with the number of operations, potentially causing significant delays in SNN training and inference processes.

Another hurdle lies in the enlarged data size when encrypted. Encrypted data typically exceeds the size of the original data, thereby increasing storage and transmission demands. Striking a balance between data security and practicality remains an ongoing challenge, demanding efficient algorithms and specialized hardware solutions.

Furthermore, FPGAs come with finite resources, encompassing logic cells, memory, and routing capabilities. Implementing a homomorphic encryption scheme while staying within these constraints can be difficult. Developers need to carefully design their circuits to make the most of the available resources, which may sometimes require making trade-offs between security and efficiency.

Additionally, ensuring the security of the homomorphic encryption implementation is of paramount importance. FPGA-based systems face unique vulnerabilities, such as susceptibility to side-channel attacks. The implementation

of robust key management strategies, data protection during processing, and the mitigation of potential threats are crucial aspects that require careful consideration.

3 Possible Solution

3.1 Stochastic Computing

Stochastic computing (SC) offers an alternative computing paradigm where information is represented and processed using probabilities and random variables.

SC represents numbers using random binary digits, where each digit is a probability of being '1' or '0'. This stochastic representation allows for operations such as addition, multiplication, and division to be performed using probabilistic logic gates. Stochastic binary arithmetic can be used in applications where precision can be traded for efficiency, such as in probabilistic algorithms.

1. Probabilistic Neural Networks: In traditional neural networks, computations are typically performed using fixed-precision arithmetic. Stochastic neural networks, on the other hand, can leverage stochastic computing to represent and process neural activations as probabilities. This approach can be valuable in scenarios where neural networks need to cope with uncertainty and imprecise data, such as in autonomous systems and robotics.

2. Randomized Algorithms: Randomized algorithms often rely on random numbers and probabilistic operations to solve complex problems efficiently. SC provides a natural framework for designing and implementing randomized algorithms, which are widely used in fields like cryptography, data analysis, and optimization.

3. Error-Tolerant Applications: Stochastic computing can be advantageous in error-tolerant applications, where small errors in computations can be acceptable. By embracing stochasticity, these applications can benefit from reduced hardware and power requirements while still achieving satisfactory results.

3.2 Securing Neural Networks with Logic Locking

Logic locking is a security technique that can be applied to protect neural networks from unauthorized access and tampering. By adding an extra layer of security through encryption and obfuscation, logic locking enhances the confidentiality and integrity of neural network models and their computations. Logic locking offers a potent defense against various security threats, including intellectual property theft and adversarial attacks on neural networks. It helps maintain the confidentiality and integrity of neural network models, ensuring that the networks perform as intended and preventing unauthorized access to critical information.

1. Model Encryption: In logic locking for neural networks, the model parameters, architecture, and other critical components can be encrypted to prevent unauthorized access. By encrypting the model, it becomes challenging for potential attackers to reverse-engineer and extract sensitive information, ensuring the confidentiality of the neural network. Representing neural network computation with SC allows to apply locking techniques directly on top of the SC gates.

2. Obfuscated Logic Gates: Logic locking involves adding obfuscated logic gates within the neural network design. These gates introduce complexity and make it challenging for attackers to reverse-engineer the network or modify its behavior. Obfuscation techniques can include adding redundant gates, introducing noise, or using custom gate structures.

3. Resilience to Reverse Engineering: By implementing logic locking, neural networks can become more resilient to reverse engineering and side-channel attacks. This is particularly important when the networks are deployed in untrusted environments or as part of proprietary applications.

****