Weight Discretization for Quotient Model Abstraction in Spiking Neural Network Verification

Research Note — January 2026
(Corrected & Revised)

1. Introduction

This note extends the filtration-based quotient model abstraction [1] to preserve *synaptic weight information* during formal verification. While the original quotient model abstracts membrane potentials into equivalence classes, it treats all synapses uniformly, losing essential structural information.

We address this by introducing a *weight discretization scheme* that:
1. Maps continuous weights to a finite discrete range
2. Preserves the relative contribution of synapses to membrane potential
3. Ensures threshold feasibility is maintained
4. Retains weight visibility in the generated PRISM model

2. Weight Discretization

2.1. Formal Definition

> **Definition.** Given a weight range $[w_{\min}, w_{\max}] = [-100, 100]$ and a discretization parameter $W \in \mathbb{N}^+$, the *weight discretization function* $\delta_W : \mathbb{Z} \to \mathbb{Z}$ is defined as:
>
> $$\delta_{W(w)} = \text{round}\left(w \cdot \frac{W}{w_{\max}}\right)$$
>
> The discretized weight range is $[-W, W] \subset \mathbb{Z}$.

Example. For $W = 3$ (7 levels total):
- $\delta_3(100) = 3$ (strong excitatory)
- $\delta_3(67) = 2$ (medium excitatory)
- $\delta_3(33) = 1$ (weak excitatory)
- $\delta_3(0) = 0$ (negligible)
- $\delta_3(-50) = -2$ (medium inhibitory)
- $\delta_3(-100) = -3$ (strong inhibitory)

2.2. Threshold Calibration

The key challenge is ensuring that threshold reachability is preserved after discretization. We must calibrate the *discretized threshold* T_d to be consistent with the original threshold T.

> **Definition.** The *discretized threshold* for a neuron with original threshold T and weight discretization parameter W is:

$$T_d = \text{ceil}\left(T \cdot \frac{W}{w_{\max}}\right)$$

Theorem (Threshold Preservation)**.** Let $\mathcal{N}$ be a neuron with incoming weights $\{w_1, ..., w_m\}$ and threshold T. Let $\mathcal{N}'$ be the discretized version with weights $\{\delta_W(w_1), ..., \delta_W(w_m)\}$ and threshold T_d.

If $\mathcal{N}$ can fire in a single step (i.e., $\exists$ input pattern $\boldsymbol{y} \in \{0,1\}^m$ such that $\sum_{i=1}^m w_i \cdot y_i \geq T$), then $\mathcal{N}'$ can also fire in a single step.

Proof. Let $\boldsymbol{y}^*$ be an input pattern that causes $\mathcal{N}$ to fire:

$$S = \sum_{i=1}^{m} w_i \cdot y_i^* \geq T$$

The discretized contribution is:

$$S_d = \sum_{i=1}^{m} \delta_W(w_i) \cdot y_i^*$$

By the rounding property of δ_W:

$$\delta_W(w_i) \geq \frac{w_i \cdot W}{w_{\max}} - \frac{1}{2}$$

Therefore:

$$S_d \geq \left(\sum_{i=1}^{m} w_i \cdot y_i^*\right) \cdot \frac{W}{w_{\max}} - \frac{m}{2} = S \cdot \frac{W}{w_{\max}} - \frac{m}{2}$$

Since $S \geq T$:

$$S_d \geq T \cdot \frac{W}{w_{\max}} - \frac{m}{2}$$

For the discretized model to fire, we need $S_d \geq T_d$. By choosing:

$$T_d = \text{ceil}\left(T \cdot \frac{W}{w_{\max}}\right) \leq T \cdot \frac{W}{w_{\max}} + 1$$

A sufficient condition is:

$$S \cdot \frac{W}{w_{\max}} - \frac{m}{2} \geq T \cdot \frac{W}{w_{\max}} + 1$$

Which simplifies to:

$$(S - T) \cdot \frac{W}{w_{\max}} \geq \frac{m}{2} + 1$$

This holds when $S - T$ is sufficiently large relative to m. For the boundary case $S = T$, we need $W \geq w_{\max} \cdot (\frac{m}{2} + 1)/T$.

In practice, with $W = 3$, $w_{\max} = 100$, and typical networks ($m \leq 10$, $T = 100$):

$$W \geq 100 \cdot \frac{6}{100} = 6$$

Thus $W = 7$ (range $[-3, 3]$) is sufficient for most practical networks. □

Remark. Critical Constraint: The proof shows that weight discretization introduces a cumulative error of $-\frac{m}{2}$. If the fan-in m is large relative to W (specifically if $m > 2W$), the rounding noise may exceed the signal of the smallest synaptic weight.

For high-fanin neurons, we strictly recommend:
1. Using a finer discretization ($W \geq \frac{m}{2}$)
2. Applying a threshold correction factor: $T_{d'} = T_d - \text{floor}\left(\frac{m}{2W}\right)$ (Note: This may increase false positive firings).

3. Class Transition with Weighted Contributions

3.1. Contribution-Based Class Evolution

In the original quotient model, class evolution used a binary rule which loses weight information. We replace it with *weighted contribution-based* class evolution.

Definition. The *weighted contribution* for neuron n with incoming discretized weights $\{w_1^d, ..., w_m^d\}$ is:

$$C_n = \sum_{i=1}^{m} w_i^d \cdot y_i$$

where $y_i \in \{0, 1\}$ is the spike output of presynaptic neuron i.

Definition. The *class delta function* $\Delta : \mathbb{Z} \to \mathbb{Z}$ maps contribution to class change:

$$\Delta(C) = \text{clamp}\left(\text{round}\left(\frac{C}{\gamma}\right), -k, k\right)$$

where γ is the *class width* (typically $\gamma = \frac{T_d}{k}$) and k is the number of threshold levels.

3.2. Integration with Leak Rate

The quotient model must also account for membrane potential decay (leak). We propose a corrected *discretized leak* formula.

Definition. The *discretized leak factor* λ_d is:

$$\lambda_d = -\text{ round}((1 - r) \cdot k)$$

where $r \in [0, 1]$ is the original leak rate (retention) and k is the number of classes.

The class evolution rule becomes:

$$c'_n = \text{clamp}(c_n + \Delta(C_n) + \lambda_d, 0, k)$$

Example. For $r = 0.9$ (90% retention, i.e., 10% decay) and $k = 4$:

$$\lambda_d = -\text{ round}((1 - 0.9) \cdot 4) = -\text{ round}(0.4) = 0$$

Result: With no input, the class stays stable (decay is too small to drop a full class).

For $r = 0.5$ (50% decay):

$$\lambda_d = -\text{ round}((1 - 0.5) \cdot 4) = -\text{ round}(2.0) = -2$$

Result: With no input, class decreases by 2 per step.

Remark. We choose to apply leak *only when no excitatory input fires* to maintain consistency with the precise model's behavior, where leak is a multiplicative factor on the existing potential.

4. Threshold Feasibility Analysis

4.1. Definition

Definition. A neuron configuration is *threshold-feasible* if there exists at least one input pattern that can cause the neuron to reach the firing threshold within a finite number of steps.

Theorem (Feasibility Criterion)**.** A neuron with discretized weights $\{w_1^d, ..., w_m^d\}$ and threshold T_d is threshold-feasible if and only if:

$$\sum_{w_i^d > 0} w_i^d > |\lambda_d|$$

And specifically, for reliable firing:

$$\sum_{w_i^d > 0} w_i^d \geq \frac{T_d}{1 + |\lambda_d|}$$

Proof. Let $E = \sum_{w_i^d > 0} w_i^d$ be the maximum excitatory contribution per step.

Sufficiency: If $E \geq T_d$, the neuron can fire in a single step. Otherwise, the neuron accumulates potential. With leak factor $\lambda_d \leq 0$, each step adds $E + \lambda_d$ net contribution.

For accumulation to be possible at all, we must have $E + \lambda_d > 0$, or $E > |\lambda_d|$.

If this holds, the minimum steps required to reach class k is $n = \text{ceil}\left(\frac{k}{E+\lambda_d}\right)$. Since $T_d = k \cdot \gamma$, the condition $E \geq \frac{T_d}{1+|\lambda_d|}$ ensures firing is reachable.

Necessity: If $E \leq |\lambda_d|$, the potential cannot grow over time; the leak cancels or overpowers the excitation. □

4.2. Implementation

The feasibility check should be performed at PRISM generation time:

```rust
fn check_feasibility(
    weights: &[i32],  // discretized weights
    threshold: i32,
    leak_factor: i32, // expected to be <= 0 (e.g. -1, -2)
) -> Feasibility {
    let max_excitation: i32 = weights.iter()
        .filter(|&&w| w > 0)
        .sum();

    // Ensure we are working with the magnitude of the leak
    let leak_magnitude = leak_factor.abs();

    // Basic sanity check: Input must overcome leak
    if max_excitation <= leak_magnitude {
        return Feasibility::Impossible;
    }

    let min_required = threshold / (1 + leak_magnitude);

    if max_excitation >= threshold {
        Feasibility::SingleStep
    } else if max_excitation >= min_required {
        // Steps = ceil(Threshold / Net_Gain)
        let net_gain = max_excitation - leak_magnitude;
        let steps = (threshold + net_gain - 1) / net_gain;
        Feasibility::MultiStep { min_steps: steps }
    } else {
        Feasibility::Impossible
    }
}
```

5. PRISM Model Structure

The weighted quotient model generates PRISM code with explicit weight constants and contribution formulas. Note that weights are static constants, minimizing state explosion.

```
// Weight constants (discretized)
const int W = 3;  // Discretization parameter
const int W_in0_2 = 3;   // δ_3(100) = 3
const int W_n1_2 = -2;   // δ_3(-67) = -2
const int W_n3_2 = 1;    // δ_3(33) = 1

// Discretized threshold
const int T_d = 3;

// Contribution formula (evaluated at runtime)
formula contrib_2 = W_in0_2 * x0 + W_n1_2 * z1_2 + W_n3_2 * z3_2;

// Class delta (clamped)
formula delta_2 = max(-4, min(4, contrib_2));

// Class evolution with weighted contribution
// Note: Leak is applied via addition of negative constant lambda_d
[tick] y2=0 & pClass2=0 -> (pClass2' = max(0, min(4, pClass2 + delta_2)));
```

6. Conclusion

We have established a formal framework for weight discretization in quotient model abstraction. The key corrections in this version include:

1. **Corrected Leak Formulation:** Using ensures leak acts as a decay, not a driver.
2. **Fan-in Awareness:** Highlighting that must scale with fan-in to prevent rounding errors from dominating dynamics.
3. **Feasibility Logic:** Ensuring excitation strictly exceeds leak magnitude () for multi-step firing.

Bibliography

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.