



Politecnico
di Torino

PROGETTO DI PDS

SPIKING NEURAL NETWORKS

Alessandro Savino (Professore)
Alessio Carpegna

Angelo Iannielli

Artificial Intelligence

Francesco Recchia

Software Engineering

Nicolai Slav

Cloud Computing and
Network Infrastructures

Indice

- Scopo del progetto
- Perché usare le SNN
- Metodologia
- Componenti
- Ambito e limiti
- Risultati

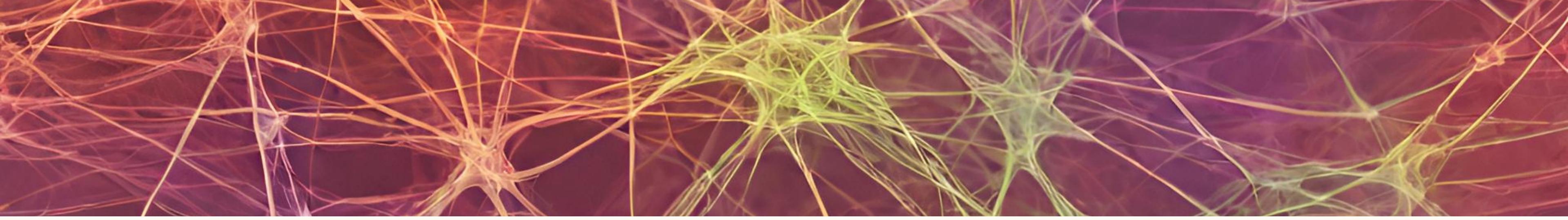
Scopo del progetto

**Studio della resilienza nelle
spiking neural networks**

Obiettivi

- Definire le strutture dati necessarie alla rappresentazione della SNN.
- Simulare una SNN.
- Adattare il multithreading alla SNN.
- Ridefinire le strutture dati per poter studiare la resilienza di una SNN.
- Definire i componenti in cui iniettare gli stuck
- Simulare una rete con gli stuck.
- Analizzare i dati raccolti

[Torna all'indice](#)



Perché usare le spiking NN?

[Torna all'indice](#)

Affinità coi modelli biologici reali

- Questo tipo di reti neurali replica più fedelmente il comportamento biologico dei neuroni. Le informazioni sono codificate in impulsi temporali (spike) in maniera simile a quanto avviene con impulsi elettrici.
- I processi sono rappresentati in modo più dettagliato ed è possibile comprendere meglio le dinamiche temporali che ne sono alla base.



Perché usare le spiking NN?

[Torna all'indice](#)

Elaborazione temporale dei dati efficiente

- Gli impulsi spiking permettono una elaborazione dei dati basata sullo scorrere del tempo.
- Le connessioni sinaptiche possono essere rafforzate o indebolite in base al momento in cui avvengono gli impulsi, permettendo alla rete di adattarsi dinamicamente seguendo un modello temporale complesso.

Modellazione del problema

```
pub trait Model {  
    type Neuron: 'static + Sized + Clone + Send + Sync + Debug;  
    type Config: Clone;  
  
    fn handle_spike(  
        neuron: &mut Self::Neuron,  
        weighted_input_val: f64,  
        ts: u128  
    ) -> f64;
```

Il tratto Model è definito per espandere la libreria e rappresentare anche modelli diversi da quello LIF

```
pub struct NN<M: Model + Clone + 'static> {  
    pub layers: Vec<Layer<M>>,  
}
```

La SNN è un vettore di layer, ciascuno dei quali a sua volta è un vettore di neuroni e contiene i pesi di input e i pesi interni

All'interno del Neurone

```
pub struct LifNeuron {  
    pub v_rest: f64,  
    pub v_reset: f64,  
    pub v_th: f64,  
    pub tau: f64,  
    pub v_mem: f64,  
    pub ts_old: u128,  
    pub heap_tree: Option<HeapCalculator<f64, u64>>,  
    pub injection_vmem: Option<InjectionStruct>,  
    pub comparator: Option<InjectionStruct>,  
}
```

Il neurone contiene i suoi parametri interni e le strutture dati necessarie a effettuare i test di resilienza.

```
pub struct HeapCalculator<T: Clone, U> {  
    heap_vec: Vec<Link<T, U>>,  
}
```

Per modellare la sommatoria viene utilizzato un vettore che concettualmente rappresenta l'albero di sommatori e può simulare il guasto di un qualsiasi link

Studio della Resilienza

```
pub struct Resilience {  
    pub components: Vec<String>,  
    pub stuck_type: Stuck,  
    pub times: u128,  
}
```

Per calcolare la Resilienza viene effettuata una prima simulazione della rete senza guasti. Poi, successivamente alla scelta di un componente e di uno stuck si simula la rete “danneggiata” per n volte e si confrontano i risultati con quello precedente

Componenti

[Torna all'indice](#)

Neuron

Quando si seleziona il neuron come componente su cui iniettare gli stuck, si stanno selezionando tutti i valori di default di un neurone: il potenziale di riposo, tau, il potenziale di reset e il potenziale di soglia. Ciascuno di questi valori è un f64 e studiando la struttura di un floating point a doppia precisione si ha un cambiamento significativo se si inverte il bit di segno (1) o si inverte uno dei bit dell'esponente (11). Perciò la probabilità di causare un errore è di circa 18,75% che va poi moltiplicata per la probabilità che il neurone riceva e spari lo spike.

Membrane voltage

Quando si seleziona il potenziale di membrana, lo stuck viene applicato al potenziale di membrana di un neurone scelto a caso.

Full adder

Per applicare lo stuck al sommatore, viene costruita una struttura che internamente è costituita da un vettore di strutture (Link) che contengono il valore attuale, il tipo di stuck e la maschera. Concettualmente questo vettore rappresenta l'albero di sommatori e può simulare il guasto di un qualsiasi link. Per motivi di prestazioni tutte queste strutture sono degli Option e ci si entra solo nel caso in cui il guasto è presente.

Comparator

Essendo l'output binario lo stuck ha una probabilità del 50% di causare l'errore

Metodologia

[Torna all'indice](#)

Input spikes

Abbiamo usato la strategia locale per gli spike di input, ovvero ciascun neurone riceve il proprio vettore di spike di input.

Stuck

Per indicare lo stuck viene utilizzato un Enum dove:

Zero indica che un bit viene forzato a 0

One indica che un bit viene forzato a 1

Transient indica che un bit viene forzato al suo valore opposto

Multithreading

Per ciascun layer viene creato un thread che elabora gli spike di ingresso e emette al layer successivo gli spike prodotti dal layer stesso.

La concorrenza è gestita mediante l'utilizzo di canali

Programmazione generica

Utilizzo di funzioni generiche per gestire l'injection di un bit di errore in valori di diversi tipi

Ambito e limiti

[Torna all'indice](#)

Il linguaggio di programmazione usato è Rust

- Rust è un linguaggio moderno che ci permette di avere prestazioni simili al C. Purtroppo però una rete neurale vera non può essere eseguita su un calcolatore general purpose, ma ha bisogno di hardware dedicato.

Dimensione delle SNN

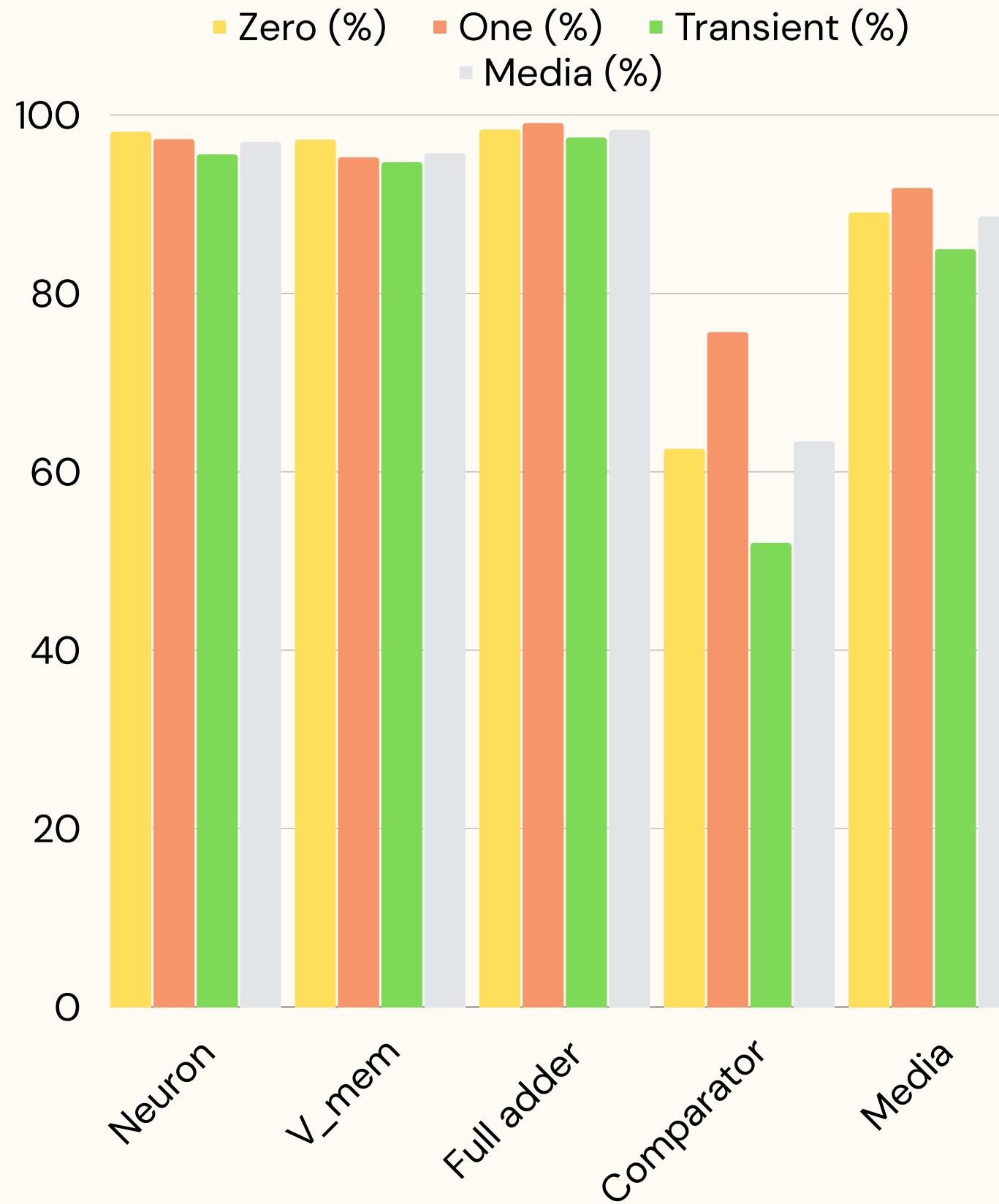
- Per avere tempi di risoluzione accettabili, considerando di lanciare il programma su un PC con 10 core e ripetendo il test di resilienza 1000 volte si consiglia di non usare reti con più di 10 layer ciascuno avente più di 10 neuroni.

Multithreading

- Per il nostro progetto abbiamo usato i thread nativi di rust per sfruttare al massimo le capacità del computer.

Risultati

[Torna all'indice](#)



I test sono stati eseguiti su una rete $10 \times 5 \times 10 \times 5 \times 5 \times 10 \times 4 \times 4 \times 8 \times 8$, con 96 spike in ingresso e 36 in uscita. Il numero di test è 10000 per ciascun componente e tipo di stuck. Tempo di esecuzione medio: 19s.

Abbiamo notato che il numero di output errati si verifica principalmente con gli stuck iniettati sui comparatori.

- La spiegazione è che il comparatore fornisce in output un valore binario, perciò la probabilità di emettere un output errato è del 50%.
- La soluzione potrebbe essere quella di costruire un sistema ridondante che lavori in parallelo e decidere l'output con un sistema di votazione maggioritario.