UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Introduzione alle reti Neuromorphic Computing e alle Spike Neural Networks

Relatore
Prof. Schenato Luca

Laureando
Bissiato Martino

ANNO ACCADEMICO 2024-2025

Data di laurea 22/09/2025

Indice

1 Sommario **1**

2 Reti Neurali Artificiali **3**

2.1 Introduzione alle Reti Neurali 3

2.2 Il Neurone 4

2.3 Struttura delle Reti Neurali Artificiali 5

2.3.1 Input 5

2.3.2 Sinapsi (Pesi) 6

2.3.3 Bias 6

2.3.4 Strati Nascosti 6

2.3.5 FUNZIONI DI ATTIVAZIONE 6

2.4 Modelli di base delle reti neurali artificiali 10

2.4.1 Il Percettrone 10

2.4.2 Multilayer perceptron 11

2.5 Limiti delle Reti Neurali Artificiali 12

3 Principi del Neuromorphic Computing **13**

3.1 Introduzione al Neuromorphic Computing 13

3.2 Caratteristiche principali 14

3.2.1 Elaborazione event-driven 14

3.2.2 Elaborazione parallela e distribuita 14

3.2.3 Plasticità sinaptica implementata in hardware 14

3.2.4 Architetture hardware dedicate 15

3.3 Superamento delle architetture Von Neumann 15

4 Spiking Neural Networks **17**

4.1 Introduzione alle Spiking Neural Networks 17

4.1.1 Cos'è uno spike? 17

4.1.2 Codifica temporale dell'informazione 18

4.1.3 Vantaggi potenziali delle SNN 18
4.2 Modelli di neuroni spiking 19
4.2.1 Introduzione ai modelli di neuroni 19
4.2.2 Modello di McCulloch-Pitts 19
4.2.3 Modello Leaky Integrate-and-Fire (LIF) 19
4.2.4 Modello di Hodgkin-Huxley 21
4.3 Fondamenti teorici dell'Apprendimento Sinaptico 22
4.3.1 La Regola di Hebb 22
4.3.2 Spike Timing Dependent Plasticity (STDP) 22
4.3.3 Regola di aggiornamento sinaptico STDP 23

5 Hardware Neuromorfico: Loihi 1 e Loihi 2 25
5.1 Loihi 1: Architettura Neuromorfica Digitale 25
5.1.1 Caratteristiche di apprendimento 25
5.1.2 Primitive computazionali 26
5.1.3 Componenti hardware 26
5.1.4 Struttura neuronale e fan-out 26
5.1.5 Efficienza e paradigma event-driven 27
5.2 Loihi 2 27
5.2.1 Architettura di base 27
5.2.2 Nuove funzionalità 28
5.2.3 Scalabilità e I/O 28
5.3 Lava: il framework software 28
5.3.1 Introduzione a Lava 28
5.3.2 Architettura e Magma 28
5.3.3 Funzionalità principali 29

6 Costruzione di una SNN tramite Brian2 31
6.1 Introduzione a Brian2 31
6.2 Installazione e setup 32
6.2.1 Creazione dell'ambiente virtuale 32
6.2.2 Installazione di Brian2 e librerie ausiliarie 32
6.2.3 Verifica dell'installazione 33
6.2.4 Definizione del modello neuronale 33
6.2.5 Implementazione di Spike-Timing-Dependent Plasticity 35
6.2.6 Simulazione e rappresentazione della rete 36
6.2.7 Analisi dei risultati 42

6.2.8 Possibili estensioni e sviluppi futuri 43

7 Conclusioni **45**

7.1 Limitazioni del Neuromorphic Computing 45

7.2 Sviluppi futuri 46

Bibliografia **47**

Capitolo 1

Sommario

Lo scopo principale di questo lavoro è fornire al lettore una chiara introduzione al campo del Neuromorphic Computing (NC) e alle Reti Neurali Spiking (SNN). La tesi propone innanzitutto le ragioni per cui il Neuromorphic Computing rappresenta una valida alternativa alle reti neurali tradizionali. Per contestualizzare l'argomento si parte con l'analisi del neurone dal punto di vista biologico, passando poi ad un'analisi delle reti neurali artificiali, evidenziando strutture principali e limiti. Successivamente, viene approfondito l'ambito del NC, spiegandone i concetti chiave e i miglioramenti che può portare rispetto alle architetture tradizionali. Per concretizzare i concetti del NC, vengono analizzate le SNN, in particolare i modelli di neuroni spiking e i fondamenti teorici dell'apprendimento sinaptico, e implementazioni su architetture hardware, in particolare Loihi e Loihi 2. Viene poi descritto un esempio pratico di SNN attraverso Brian2, un simulatore open-source di SNN. Viene, infatti, implementata una SNN utilizzando le caratteristiche descritte nei primi capitoli attraverso una guida che permette di capire i concetti e le possibili applicazioni. Infine, vengono tratte le conclusioni sul lavoro svolto. Vengono evidenziati i limiti al giorno d'oggi del NC e i possibili cambiamenti che può portare in futuro.

Capitolo 2

Reti Neurali Artificiali

2.1 Introduzione alle Reti Neurali

Il termine ”Pattern Recognition” comprende un’ampia gamma di problemi di elaborazione di informazioni di particolare importanza, come quesiti di classificazione di caratteri, riconoscimento vocale, il riconoscimento di guasti nei macchinari e le diagnosi mediche. Questi sono problemi che per la mente umana possono sembrare banali. Tuttavia, trovare una soluzione attraverso un computer si è rivelata estremamente difficile.[1] Per trovare una soluzione efficace a questi problemi, si adotta un approccio di tipo statistico, che permette di riconoscere la natura probabilistica sia delle informazioni che vogliamo cercare di elaborare sia del modo in cui vorremmo esprimere i risultati desiderati. Un campo che adotta questo approccio statistico è quello delle reti neurali, che negli ultimi anni sta prendendo particolare risvolto per via dei grandi risultati che permettono di ottenere. Il termine ”rete neurale” deriva dalla biologia e prende ispirazione dalla struttura e dal funzionamento del cervello umano, utilizzando processi che imitano il modo in cui i neuroni biologici lavorano insieme per identificare fenomeni, pesare le opzioni e arrivare alle conclusioni. Più specificamente sono chiamate ’reti neurali artificiali’ (”Artificial Neural Networks”, ANN) poiché si basano sul concetto di neuroni artificiali interconnessi. A causa del crescente interesse nella scoperta del funzionamento del cervello umano, le ricerche in questo ambito stanno avendo una crescita esponenziale, soprattutto negli ultimi anni. Grazie a questi studi, infatti, nelle prossime sezioni verranno approfondite le reti neurali e come queste tentano di emulare il comportamento dei neuroni biologici.

2.2 Il Neurone

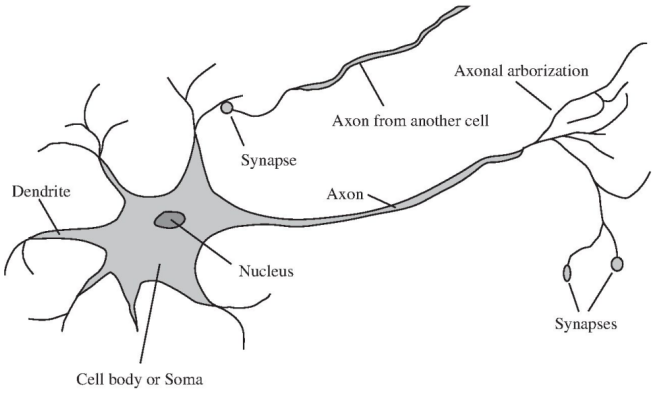


Figura 2.1: Struttura del Neurone

Il Neurone è una cellula complessa composta principalmente da: corpo cellulare, dendriti e assone.[2] Il corpo cellulare è detto anche ”Soma” e svolge il lavoro di ”Computing Powering Unit” del cervello, essendo il luogo dove avviene l’elaborazione non lineare delle informazioni. I Dendriti sono ramificazioni sottili, la cui funzione principale è quella di ricevere segnali in entrata, portando l’informazione da altri neuroni. L’Assone, invece, è una sorta di prolungamento che trasporta l’output del soma verso altri neuroni. Ogni neurone possiede un alto numero di connessioni, andando così a formare una vera e propria rete di neuroni. Un Neurone è formato, così come la maggior parte delle cellule del corpo umano, da una membrana, che è permeabile a certe molecole. In particolare, questa membrana è capace di regolare il flusso, in entrambe le direzioni, di alcuni ioni (come ad esempio calcio e potassio) grazie alla presenza di canali ionici, che trasportano ioni fuori e dentro la membrana a causa di un certo stimolo. Inoltre, questi ioni possono essere scambiati con altri neuroni, quando l’informazione, sotto forma di segnale elettrico, è diffusa attraverso la rete di neuroni. La membrana è attraversata da molti canali ionici, che controllano la permeabilità rispetto agli ioni. La differenza di potenziale attraverso la membrana è data dalla differenza di concentrazione degli ioni. Infatti, secondo l’equazione di Nerst:

$$\Delta u(x) = \left(\frac{kT}{q}\right) \ln\left(\frac{n_1}{n_2}\right) \tag{2.1}$$

in cui n1 e n2 sono le differenze di concentrazione dentro e fuori dalla membrana, rispettivamente; k è la costante di Boltzmann, T è la temperatura e q è la carica elementare. Quando il potenziale della membrana supera una certa soglia, il neurone emette un impulso, anche chiamato ”Action Potential”. Quest’impulso viene poi utilizzato da altri neuroni come input e eventualmente genera altri impulsi. L’informazione nel cervello è costituita da impulsi che i neuroni

emettono in un certo tempo. Inoltre, il cervello è costituito da altri tipi di sistemi biologici: i più importanti sono le Sinapsi, che fungono da interfaccia tra un Dendrite e un Assone di neuroni differenti. Le sinapsi sono complessi processi biologici che colmano il divario tra l'output di un certo Neurone Presinaptico(assone) e l'input di uno Postsinaptico (dendrite). In effetti, le caratteristiche biologiche delle sinapsi si adattano durante il funzionamento del cervello, diventando più forti o più deboli durante il processo di trasmissione di informazioni da un neurone all'altro. Il progresso nella comprensione della fisiologia del neurone e nello sviluppo di metodi per riprodurre sperimentalmente i suoi comportamenti ha portato alla definizione di modelli sempre più accurati e complessi. Tuttavia, l'elevato livello di dettaglio richiesto da tali modelli comporta costi computazionali molto alti, spesso incompatibili con le attuali capacità tecnologiche. Questo ha reso necessario un compromesso tra fedeltà biologica e praticità computazionale. In questo contesto nascono le reti neurali artificiali, modelli ispirati al funzionamento del cervello umano ma progettati per essere implementabili in modo efficiente su sistemi informatici. A partire da una semplificazione della struttura del neurone biologico, queste reti hanno dimostrato un enorme potenziale nel campo dell'apprendimento automatico, offrendo soluzioni efficaci a problemi complessi.

2.3 Struttura delle Reti Neurali Artificiali

2.3.1 Input

Lo strato di input è il primo livello della rete e ha il compito di acquisire e presentare i dati grezzi nel formato richiesto dai livelli successivi. Ogni neurone di questo strato corrisponde a una caratteristica del vettore di input $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. In fase di inferenza, il layer di input non applica trasformazioni pesate: si limita a inoltrare ciascun valore x_i allo strato successivo, preservando l'informazione originale.

Prima di essere introdotti nella rete, i dati in ingresso vengono spesso sottoposti a operazioni di pre-processing come:

- normalizzazione o standardizzazione per rendere comparabili scale diverse;
- codifica di variabili categoriche ("one-hot", "embeddings");
- "data augmentation" per immagini o segnali audio, al fine di migliorare la generalizzazione.

In questo modo, lo strato di input garantisce che ogni neurone di livello superiore riceva un dato omogeneo, riducendo le disomogeneità che potrebbero ostacolare l'addestramento.

2.3.2 Sinapsi (Pesi)

Ogni connessione tra due neuroni è associata a un peso scalare w_{ij} che modula l'intensità del segnale trasmesso dal neurone i a j. Durante l'addestramento, questi pesi vengono affinati tramite algoritmi di ottimizzazione (per esempio discesa del gradiente) per minimizzare una funzione di costo e guidare la rete verso la soluzione desiderata.

2.3.3 Bias

Il bias b_j è un termine aggiuntivo introdotto in ciascun neurone che consente di spostare l'uscita della somma pesata prima dell'applicazione della funzione di attivazione. Grazie al bias, la funzione di attivazione può assumere valori non nulli anche quando tutti i pesi e gli input tendono a zero, aumentando la capacità di modellare dati con offset costanti.

2.3.4 Strati Nascosti

Gli strati nascosti svolgono il compito cruciale di trasformare progressivamente la rappresentazione dell'input in "feature" di ordine crescente di astrazione.

1. Ogni neurone riceve come input la combinazione lineare pesata dei valori dei neuroni del layer precedente:

$$z_j = \sum_i w_{ij}\, x_i + b_j.$$

2. Al valore z_j si applica una funzione di attivazione non lineare $\phi(\cdot)$ (ad esempio ReLU, Sigmoid o Tanh), generando l'output $a_j = \phi(z_j)$.

3. L'output a_j diventa parte dell'input per i neuroni del livello successivo.

In reti profonde, questa catena di trasformazioni permette di scoprire rappresentazioni gerarchiche: dalle feature di basso livello (contorni, bordi, toni nei dati visivi) fino a pattern complessi e concetti ad alto livello, facilitando la capacità di generalizzazione e la robustezza del modello.

2.3.5 FUNZIONI DI ATTIVAZIONE

Le funzioni di attivazione introducono non linearità nel modello, permettendo di risolvere problemi complessi. [3] Gli input sono le unità più importanti nella struttura di una rete neurale, in quanto vengono elaborati e trasformati in un risultato di output, attraverso l'applicazione di una funzione chiamata 'funzione di attivazione', funzione soglia o funzione di trasferimento.

Per limitare l'ampiezza dell'output di un neurone e mantenerlo entro un intervallo definito, si utilizzano le cosiddette "squashing functions" ("funzioni di compressione"). Una funzione di compressione riduce l'ampiezza del segnale di output a un valore finito. Le più comuni sono:

- Funzione di attivazione lineare: la funzione di attivazione lineare è direttamente proporzionale all'input. Può essere definita come: $F(x) = ax$, in cui il valore della variabile a può essere qualsiasi valore costante.

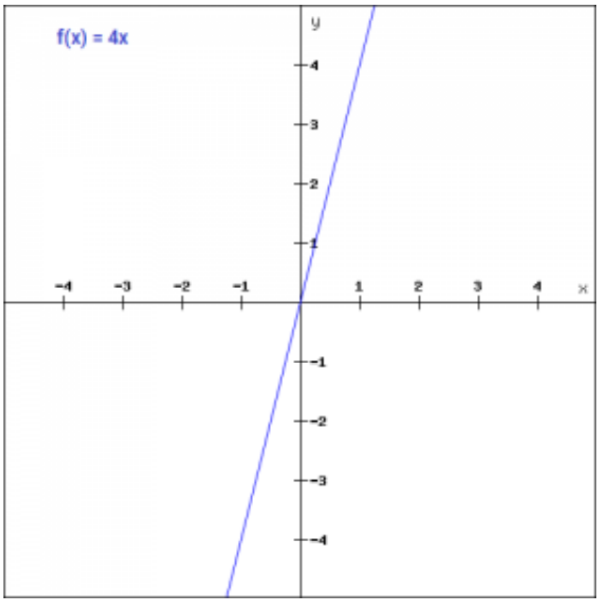


Figura 2.2: Grafico della funzione di attivazione Lineare

- Sigmoide: la funzione di attivazione non lineare più utilizzata in reti semplici, con output compreso tra 0 e 1. Può essere definita come $F(x) = 1/e^{-}x$. La funzione sigmoide è differenziabile in modo continuo. La derivata della funzione è: $f'(x) = 1 - sigmoide(x)$. Inoltre, la funzione sigmoide non è simmetrica rispetto allo zero, ciò significa che i segni di ogni valore di output sarà lo stesso. Questo problema può essere risolto scalando la funzione.

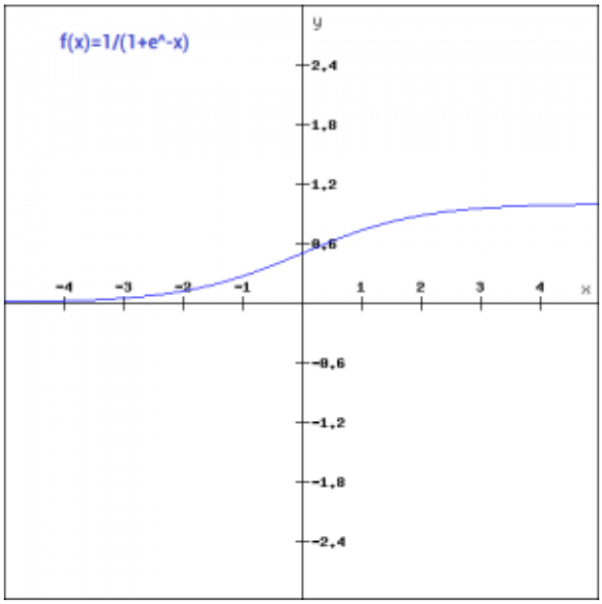


Figura 2.3: Grafico della funzione Sigmoide

- Tanh: è una funzione tangente iperbolica, simile alla sigmoide ma simmetrica attorno all'origine. Ciò comporta output con segni differenti dagli strati precedenti che saranno forniti come input agli strati successivi. Può essere definita come: $f(x) = 2sigmoid(2x) - 1$ La funzione Tanh è continua e differenziabile e i valori sono compresi tra -1 e 1. In relazione alla funzione sigmoide, il gradiente è più ripido. La funzione Tanh è preferita alla sigmoide poiché ha gradienti che non possono variare in una certa direzione e per il fatto che sia centrata sullo zero.

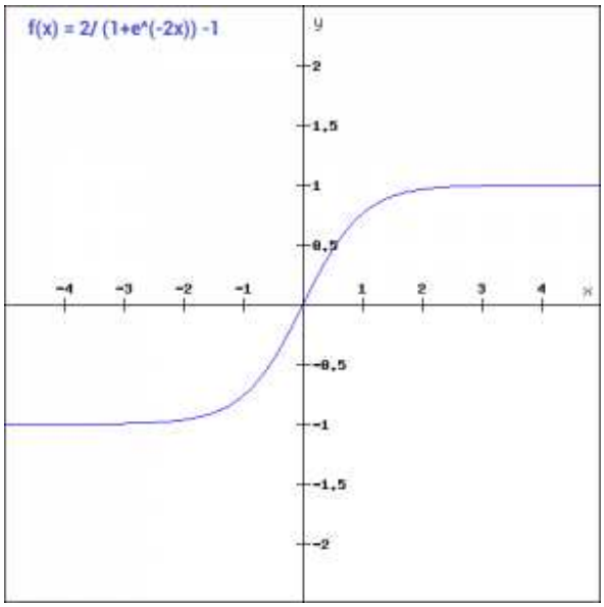


Figura 2.4: Grafico della funzione Tanh

- ReLU (Rectified Linear Unit) : ReLU è una funzione di attivazione non lineare che è ampiamente utilizzata nelle reti profonde(”Deep Networks”) poichè migliora la convergenza. Il principale vantaggio dell’utilizzo di questa funzione è il fatto che i neuroni nella rete non vengono attivati allo stesso momento. Questo implica che un neurone sarà non attivo solo quando l’output della trasformazione lineare sarà zero. Può essere matematicamente definita come: $f(x) = max(0, x)$.

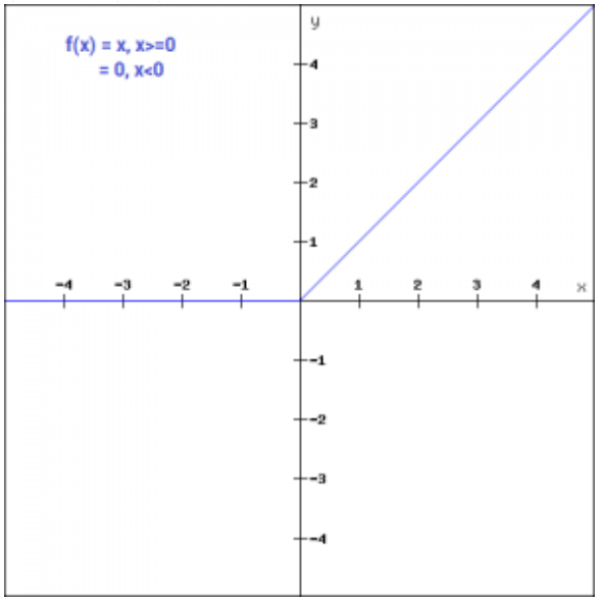


Figura 2.5: Grafico della funzione ReLU

La funzione ReLU è più efficiente di altre funzioni di attivazione poiché non tutti i neuroni sono attivati contemporaneamente, ma solo alcuni per volta. In alcuni casi, il valore del gradiente risulta uguale a zero, per cui i pesi e i valori di bias non sono aggiornati nella fase di backpropagation nell’addestramento della rete neurale.

- Softmax(Softmax Activation Function): La funzione Softmax è una combinazione di più funzioni sigmoide. Come sappiamo, una funzione sigmoide restituisce valori compresi tra 0 e 1, questi possono essere trattati come probabilità dei punti dati di una particolare classe. La funzione Softmax, a differenza delle funzioni sigmoide che vengono utilizzate per la classificazione binaria, può essere utilizzata per problemi di classificazione multiclasse. La funzione, per ogni punto dati di tutte le singole classi, restituisce la probabilità. Può essere espressa come:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K$$

OUTPUT: Lo strato di output è l'ultimo livello della rete e fornisce il risultato finale del modello. Il numero di neuroni in questo strato dipende dal tipo di problema da risolvere:

- Classificazione binaria: assegnare l'input ad una delle due categorie possibili. Per esempio, in ambito medico stabilire se un paziente è "sano" oppure "malato";
- Classificazione multi-classe: attribuire l'input a una sola tra tre o più classi mutuamente esclusive. Per esempio, riconoscere quale animale è ritratto in una fotografia;
- Regressione: stimare un valore numerico continuo. Per esempio prevedere il prezzo di una casa o la temperatura atmosferica.

2.4 Modelli di base delle reti neurali artificiali

2.4.1 Il Percettrone

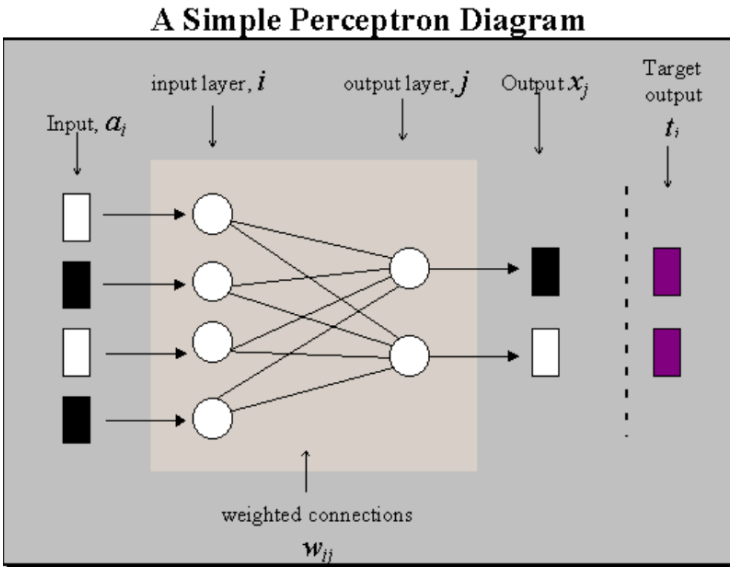


Figura 2.6: Struttura del percettrone

Nel 1958 Frank Rosenblatt, psicologo e pioniere dell'informatica, propose il Percettrone, uno dei primi algoritmi di apprendimento supervisionato,realizzandone implementazioni sia software che hardware[4]. Ideato per i problemi di classificazione e di riconoscimento di pattern (ad esempio lettere o semplici immagini), il modello era composto da:

- Unità di elaborazione: un insieme fisso di elementi di elaborazione riceveva in ingresso sottoinsiemi scelti casualmente dei pixel dell'immagine; ciascun nodo applicava una funzione di soglia.
- Strato di pesi adattativi: ad ogni uscita dei nodi di elaborazione erano associati pesi sinaptici w_i, oltre a un termine di bias b.

- Funzione di attivazione: la somma pesata degli ingressi veniva infine processata da una funzione di soglia $f(\cdot)$, tipicamente funzione a gradino (o, in versioni successive, sigmoidale):

$$y = f\Big(\sum_{i=1}^{n} w_i x_i + b\Big).$$

Per misurare la bontà del modello, Rosenblatt definì il "Criterio del Percettrone", ossia una funzione di errore proporzionale al numero di campioni misclassificati:

$$E(\mathbf{w}) = -\sum_{i \in \mathcal{M}} y^{(i)} \big(\mathbf{w} \cdot \mathbf{x}^{(i)}\big),$$

dove $\mathcal{M}$ è l'insieme degli esempi erroneamente classificati e $y^{(i)} \in \{-1,1\}$ la loro etichetta corretta. L'apprendimento avviene mediante un aggiornamento iterativo dei pesi, applicando di volta in volta la regola del gradiente discendente su singolo esempio:

$$w_j^{(\tau+1)} = w_j^{(\tau)} + \eta\, \delta^{(n)}\, x_j^{(n)},$$

con η tasso di apprendimento e $\delta^{(n)}$ errore sul pattern n. Il teorema di convergenza del Percettrone garantisce che, se il dataset è linearmente separabile, questa procedura trova in un numero finito di passi un vettore di pesi che separa perfettamente le classi, poiché a ogni aggiornamento il margine tra le classi aumenta fino a raggiungere la separazione ottimale.

2.4.2 Multilayer perceptron

Il Multilayer Perceptron (MLP) è un'estensione del percettrone introdotto da Rosenblatt e costituisce ancora oggi una delle architetture fondamentali alla base di modelli più complessi, come i Transformers [5]. A differenza del percettrone originale, il MLP presenta uno o più strati intermedi tra lo strato di input e quello di output. Tali strati, non direttamente connessi agli input provenienti dall'ambiente di osservazione, vengono definiti nascosti ("hidden layers") [6].

L'introduzione di più strati è stata motivata dalla necessità di aumentare la complessità delle regioni decisionali. Infatti, un percettrone singolo con un solo input genera regioni decisionali rappresentabili come semipiani. Aggiungendo uno strato nascosto, ciascun neurone agisce come un percettrone classico sugli output dello strato precedente, consentendo alla rete di approssimare regioni decisionali convesse, ottenute dall'intersezione dei semipiani generati dai singoli neuroni.

Per quanto riguarda le funzioni di attivazione, si è dimostrato che le reti multi-strato non aumentano la potenza espressiva rispetto alle reti a singolo strato se le funzioni di attivazione

sono lineari, poiché la composizione di funzioni lineari rimane lineare. La vera potenza del MLP deriva quindi dall'adozione di funzioni di attivazione non lineari.

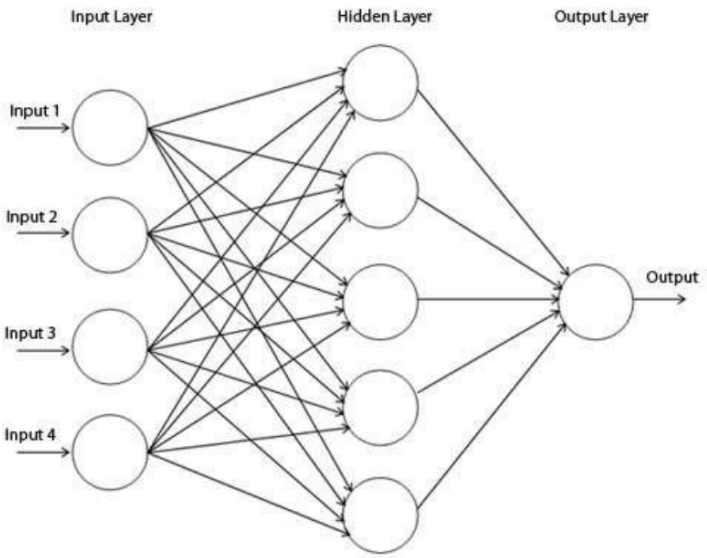


Figura 2.7: Struttura del Multylayer Perceptron

2.5 Limiti delle Reti Neurali Artificiali

Le Artificial Neural Networks (ANNs) hanno rivoluzionato numerosi ambiti dell'intelligenza artificiale grazie alla loro capacità di apprendere rappresentazioni complesse da grandi quantità di dati. Tuttavia, presentano una serie di limiti strutturali e funzionali.

In primo luogo, il loro funzionamento richiede un elevato consumo energetico, soprattutto nella fase di addestramento ("training"), in quanto si basano su processi sincroni e su hardware non specializzato. Inoltre, le ANNs convenzionali operano principalmente in domini statici e supervisionati, rendendo difficile la generalizzazione in ambienti dinamici o con pochi dati etichettati.

Dal punto di vista biologico, esse risultano poco aderenti ai meccanismi del cervello umano, in quanto non includono la temporalità dell'attività neuronale, i potenziali d'azione (spikes) o la plasticità sinaptica dipendente dal tempo. Infine, la loro interpretabilità è spesso limitata, trattandosi di modelli complessi e poco trasparenti, con conseguenti difficoltà nell'analisi e nella validazione delle decisioni prodotte.

Questi limiti motivano la ricerca di architetture alternative, come quelle neuromorfiche, ispirate alla neurobiologia.

Capitolo 3

Principi del Neuromorphic Computing

Prima di immergerci nel mondo del Calcolo Neuromorfo (”Neuromorphic Computing”, NC), è utile analizzarne la storia e le motivazioni che hanno portato allo sviluppo di tale disciplina. L’NC nasce dalla congiunzione tra l’esigenza di hardware più efficiente per applicazioni di Machine Learning e il desiderio di implementare su hardware le fondamenta delle Neuroscienze, per una migliore comprensione del funzionamento del cervello.

3.1 Introduzione al Neuromorphic Computing

Il termine Neuromorphic Computing fu coniato nel 1990 da Carver Mead[7], che originariamente lo intese per indicare tutte le forme di elettronica VLSI (Very Large Scale Integration) ispirate al funzionamento del cervello. Nel corso della storia, l’NC ha assunto nuove caratteristiche e obiettivi, influenzato e ispirato da altre discipline. Le basi principali per lo sviluppo dell’NC sono state le Neuroscienze, il Machine Learning e l’Elettronica (per quanto riguarda l’implementazione hardware). A causa del repentino progresso che la nostra società sta vivendo nell’ultimo periodo, le tecnologie informatiche e i dispositivi elettronici stanno producendo una quantità crescente di dati ogni anno e le persone hanno accesso a quantità sempre maggiore di informazioni. Per far fronte alle richieste che stanno emergendo con questa rivoluzione tecnologica, soprattutto negli ultimi anni stanno nascendo nuove tecniche di apprendimento automatico e algoritmi di intelligenza artificiale(”Artificial Intelligence”, AI). Tali algoritmi utilizzano in genere reti neurali artificiali(”Artificial Neural Networks”, ANN) e tecniche di apprendimento profondo(”Deep Learning”, DL) per risolvere attività di riconoscimento di modelli. Tuttavia, i metodi che riguardano la loro implementazione richiedono una esponenziale mole di dati e risorse informatiche, che a loro volta richiedono una quantità insostenibile di energia. D’altra parte, i cervelli biologici superano chiaramente i sistemi di AI in termini del numero di campioni di addestramento, efficienza energetica e la loro costante capacità di adattarsi a situazioni

impreviste. Per questo motivo, un approccio promettente verso lo sviluppo di nuovi paradigmi computazionali è quello che riguarda il campo del Neuromorphic Computing(NC), che propone di progettare sistemi hardware e software ispirati al funzionamento del cervello umano, imitando il comportamento dei neuroni biologici e delle sinapsi.

3.2 Caratteristiche principali

Le architetture neuromorfiche riproducono in hardware i principi di funzionamento delle reti neuronali biologiche, superando le limitazioni delle architetture di von Neumann in termini di scalabilità ed efficienza energetica. In questa sezione ne vengono illustrate le caratteristiche principali.

3.2.1 Elaborazione event-driven

Nei sistemi neuromorfici l'elaborazione è guidata da eventi discreti ("spikes"). Un neurone artificiale viene attivato solo al ricevimento di uno spike in ingresso e calcola:

$$\text{se } V_i(t) \geq \theta \quad \Rightarrow \quad \text{emit_spike}(i),$$

dove V_i è il potenziale di membrana e θ la soglia di firing. Ciò riduce il consumo energetico nei periodi di inattività, in cui il neurone non riceve stimoli esterni.

3.2.2 Elaborazione parallela e distribuita

L'architettura neuromorfica è intrinsecamente parallela: ogni neurone hardware agisce in modo indipendente e parallelo, con memoria locale per pesi e stati sinaptici. Questo aspetto consente di sfruttare al massimo la capacità di calcolo, consentendo scalabilità ed evitando sprechi di potenza.

3.2.3 Plasticità sinaptica implementata in hardware

Nel nostro cervello, la plasticità sinaptica è un meccanismo con cui i collegamenti sinaptici si rafforzano o si indeboliscono a seconda di quanto vengono sollecitati, cioè in relazione all'attività neurale a cui sono sottoposti. Nei chip neuromorfici, questo meccanismo viene implementato fisicamente, attraverso una memorizzazione locale dei pesi sinaptici. Permettendo al sistema di adattarsi dinamicamente agli stimoli ricevuti.

3.2.4 Architetture hardware dedicate

Al giorno d'oggi esistono diverse tipologie di architetture neuromorfiche, ognuna delle quali è dedicata a compiti specifici. Le più conosciute e citate nei progetti di ricerca sono:

- Intel Loihi 1 e Loihi 2;
- IBM TrueNorth;
- SpiNNaker;
- BrainScaleS e BrainScaleS-2;
- Akida.

3.3 Superamento delle architetture Von Neumann

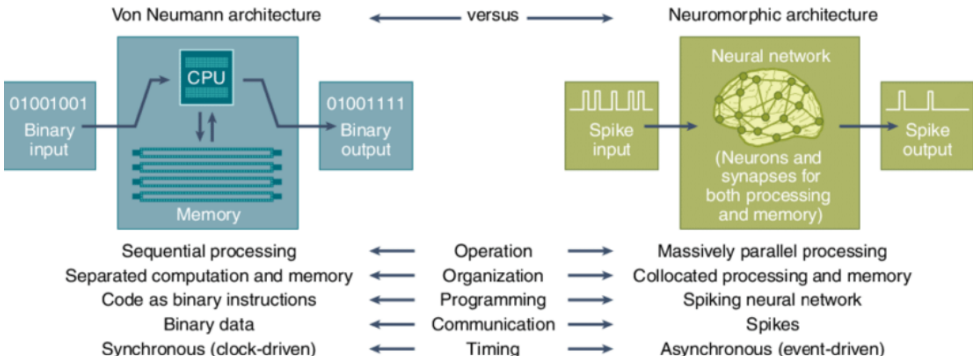


Figura 3.1: Architetture a confronto

Le architetture computazionali tradizionali sono, per la maggior parte dei casi, ispirate all'architettura di Von Neumann. Questi sistemi sono costituiti da diverse componenti fisiche separate, come la memoria (MU), l'unità di controllo dei processi (CPU) e l'unità aritmetica/logica (ALU).

Questa separazione tra le diverse unità fisiche produce un problema dal punto di vista energetico, poiché l'informazione deve essere continuamente trasferita da un'unità all'altra e ciò pone un limite ben definito a dove si possono spingere i sistemi computazionali odierni.

Sebbene ci siano stati dei miglioramenti in termini di prestazioni conseguiti dall'evoluzione tecnologica, le componenti di base dei sistemi di elaborazione sono rimaste sostanzialmente invariate, non risolvendo i limiti già predisposti.

Al contrario, il cervello umano presenta notevoli vantaggi: un'efficienza energetica superiore, una flessibilità che gli consente di adattarsi a situazioni impreviste ed la capacità di risolvere problemi complessi. Sebbene si debba ancora scoprire un sistema in grado di elaborare informazioni in modo equivalente al cervello umano, la scoperta e lo studio di principi biologici negli

ultimi anni hanno permesso significativi progressi verso architetture altamente più efficienti e adattive.

Per superare i limiti delle architetture Von Neumann è necessaria una tecnologia rivoluzionaria, basata su sviluppi scientifici innovativi. L'architettura neuromorfica risponde a questa esigenza integrando strettamente unità di elaborazione e memoria in un unico sistema dinamico, fondato su meccanismi di apprendimento mediante i quali ciascun elemento si adatta e si riconfigura in base agli stimoli ricevuti.

In sintesi, l'architettura neuromorfica tenta di superare i vincoli dell'approccio classico Von Neumann, proponendo un modello di calcolo rivoluzionario, in cui l'elaborazione è distribuita.

Capitolo 4

Spiking Neural Networks

4.1 Introduzione alle Spiking Neural Networks

Le Spiking Neural Networks (SNN) rappresentano uno dei concetti fondamentali alla base del Neuromorphic Computing. A differenza delle reti neurali tradizionali (”rate-based”), che elaborano informazioni in modo continuo, le SNN trasmettono dati attraverso eventi discreti chiamati spikes o potenziali d’azione. Questa caratteristica conferisce loro una natura event-driven, in cui l’elaborazione di dati e la comunicazione avvengono soltanto quando si verificano eventi significativi, potenzialmente riducendo il consumo energetico e incrementando l’efficienza computazionale.

4.1.1 Cos’è uno spike?

Uno ”spike” è un impulso elettrico di brevissima durata, tipicamente rappresentato come un’onda unitaria o un segnale binario, che codifica l’informazione in forma temporale. In biologia, gli spikes sono generati dai neuroni quando il potenziale di membrana supera una certa soglia. Nel contesto computazionale, lo spike è l’unità elementare di informazione.

Un segnale si può differenziare in due principali tipologie:

- Segnali continui: utilizzati nei modelli classici di reti neurali artificiali. Le uscite dei neuroni sono valori reali (ad es. probabilità o intensità), aggiornati in modo sincrono a ogni passo temporale di calcolo.
- Spike discreti: nelle SNN, i neuroni comunicano tramite eventi binari (presenza o assenza di spikes) che avvengono in istanti specifici. L’informazione è codificata principalmente nel tempo di emissione e/o nella frequenza degli spike.

Il passaggio da un segnale di input continuo a uno basato su eventi discreti permette di adottare forme di codifica temporale (”temporal coding”) e di sfruttare meccanismi biologicamente ispirati di apprendimento, come la Spike-Timing-Dependent Plasticity (STDP).

4.1.2 Codifica temporale dell’informazione

Una delle differenze chiave tra reti neurali convenzionali e SNN risiede nel ”temporal coding”. Mentre le reti tradizionali spesso codificano l’informazione in valori statici o medie di attivazioni, le SNN possono sfruttare:

- Time-to-first-spike: l’informazione è trasmessa nel tempo che intercorre tra lo stimolo e il primo spike del neurone. In pratica, più forte è lo stimolo e più velocemente il neurone raggiunge la soglia e genera il primo spike. [8]
- Spike rate coding: l’informazione è rappresentata dalla frequenza di spike in un dato intervallo. Più intenso è lo stimolo e più spikes vengono generati in quel periodo di tempo. [9]
- Precise spike timing: l’informazione dipende dal pattern temporale esatto degli spikes, cioè ogni impulso generato ha un tempo preciso che contribuisce alla codifica del segnale. [10]

Questi metodi permettono alle SNN di rappresentare e interpretare segnali che cambiano nel tempo. Infatti, sono particolarmente efficaci per analizzare dati sensoriali, come immagini o video, in cui il tempo rappresenta un ruolo fondamentale.

4.1.3 Vantaggi potenziali delle SNN

Le SNN offrono numerosi vantaggi teorici e pratici rispetto alle architetture convenzionali:

1. Efficienza energetica: essendo event-driven, il calcolo avviene solo in risposta a quando avviene un impulso, riducendo così il numero di operazioni rispetto a modelli di rete neurale tradizionali. Questa caratteristica è particolarmente rilevante per implementazioni su hardware neuromorfico.

2. Scalabilità: la comunicazione asincrona e locale tra neuroni riduce la necessità di sincronizzazione globale, consentendo la costruzione di reti su larga scala con latenza ridotta.

3. Capacità di codifica temporale: l’utilizzo del dominio temporale come ulteriore dimensione di codifica aumenta la capacità informativa a parità di numero di neuroni.

4. Maggiore plausibilità biologica: rispetto ai modelli ”rate-based”, le SNN riproducono in maniera più fedele le dinamiche temporali osservate nei sistemi nervosi biologici.

4.2 Modelli di neuroni spiking

4.2.1 Introduzione ai modelli di neuroni

Nel campo delle reti neurali artificiali e del Neuromorphic Computing, la scelta del modello di neurone costituisce un aspetto fondamentale per definire il comportamento complessivo del sistema. I modelli variano notevolmente in termini di complessità e fedeltà biologica: da rappresentazioni fortemente semplificate, ideali per studi teorici e implementazioni ad alta efficienza, a modelli biofisici dettagliati, sviluppati per simulazioni neuroscientifiche. In questa sezione vengono descritti tre modelli rappresentativi: il neurone di McCulloch-Pitts, considerato la base storica per lo sviluppo delle reti neurali artificiali; il Leaky Integrate-and-Fire (LIF), uno dei modelli più diffusi nell'ambito del calcolo neuromorfico; infine vengono messi in luce gli aspetti chiave del modello Hodgkin-Huxley, utilizzato per rappresentazioni neuronali più complesse.

4.2.2 Modello di McCulloch-Pitts

Il modello di McCulloch-Pitts rappresenta il più semplice modello di neurone. La sua estrema semplicità comporta la trascuratezza delle complesse dinamiche e caratteristiche del neurone biologico, motivo per cui non è impiegato in contesti dove la plausibilità biologica è fondamentale. Nonostante ciò, questo modello costituisce la base da cui ha avuto origine lo sviluppo delle reti neurali artificiali. [11]

In questo modello, gli impulsi ("spikes") ricevuti in ingresso sono modulati da un peso sinaptico w_{ij}, tipicamente un numero reale compreso nell'intervallo $[-1,1]$ e spesso limitato a $[0,1]$. Il neurone riceve N input in ingresso, ciascuno derivante dall'output di altri neuroni, ponderato dal relativo peso sinaptico. Viene quindi calcolata la somma lineare di tali valori: se la somma supera una soglia prefissata, il neurone emette un impulso.

$$u_i = \sum_{j=1}^{N} w_{ij} \cdot I_j \tag{4.1}$$

dove:

- u_i è il potenziale di membrana dell'i-esimo neurone,
- I_j è l'input proveniente dal j-esimo neurone,
- w_{ij} è il peso della sinapsi tra il neurone i e il neurone j.

4.2.3 Modello Leaky Integrate-and-Fire (LIF)

Il Leaky Integrate-and-Fire (LIF) è una variante dell'Integrate-and-Fire model proposta nei primi del '900 da Louis Lapicque[12]. Il termine "leaky" ("perdita") si riferisce alla presenza di un

termine di perdita nell'equazione che regola il potenziale di membrana. LIF è uno dei modelli più utilizzati nel campo del Neuromorphic Computing a causa della sua semplicità e della sua capacità di riprodurre in maniera realistica alcuni aspetti del comportamento neuronale.

Il modello LIF può essere descritto come un circuito equivalente costituito da un resistore R e un condensatore C in parallelo:

- il resistore rappresenta la perdita di carica della membrana,
- il condensatore rappresenta la capacità della membrana di immagazzinare carica.

La dinamica del potenziale di membrana è descritta dalla seguente equazione differenziale del primo ordine:

$$\tau_m \frac{dV(t)}{dt} = -\big(V(t) - V_{\text{rest}}\big) + R\,I(t) \tag{4.2}$$

dove:

- $V(t)$ è il potenziale di membrana al tempo t,
- V_{rest} è il potenziale di riposo,
- $\tau_m = R\,C$ è la costante di tempo di membrana,
- $I(t)$ è la corrente di input al tempo t.

Soglia e reset

Quando il potenziale $V(t)$ raggiunge la soglia V_{th}, il neurone emette uno spike (o potenziale d'azione) e il potenziale viene immediatamente riportato a un valore di ripristino V_{reset}. Può inoltre essere previsto un periodo refrattario t_{ref} durante il quale il neurone non può emettere ulteriori spike.

$$\text{Se } V(t) \geq V_{\text{th}} \Rightarrow \begin{cases} \text{Emetti spike} \\ V(t) \leftarrow V_{\text{reset}} \\ \text{Inizia periodo refrattario} \end{cases} \tag{4.3}$$

Pur non catturando le complessità biofisiche della generazione del potenziale d'azione, il modello LIF è estremamente utile nelle simulazioni che richiedono velocità e semplicità implementativa . Tuttavia, per applicazioni che richiedono un'elevata fedeltà biologica, vengono preferiti modelli più sofisticati, come il modello Hodgkin–Huxley, che verrà discusso nel paragrafo seguente.

4.2.4 Modello di Hodgkin-Huxley

Il modello Hodgkin e Huxley è un modello matematico che descrive il processo di depolarizzazione della membrana cellulare, dedotto da numerose osservazioni sperimentali utilizzando gli assoni giganti dei calamari.[13][14] Il modello descrive il contributo delle correnti ioniche variabili in funzione della tensione alla dinamica del potenziale di membrana $V(t)$.

Equazione del potenziale di membrana

$$C_m \frac{dV}{dt} = -\,\bar{g}_{\text{Na}}\, m^3 h\,(V - E_{\text{Na}}) -\, \bar{g}_{\text{K}}\, n^4\,(V - E_{\text{K}}) -\, g_{\text{L}}\,(V - E_{\text{L}}) +\, I_{\text{ext}} \tag{4.4}$$

dove

- C_m è la capacità di membrana per unità di area;
- $\bar{g}_{\text{Na}}$, $\bar{g}_{\text{K}}$ sono le conduttanze massime di sodio, potassio;
- g_{L} è la conduttanza di leak;
- E_{Na}, E_{K}, E_{L} sono i potenziali di inversione;
- I_{ext} è la corrente esterna applicata.

Variabili di gating Le variabili m, h e n definiscono l'apertura o la chiusura dei canali ionici. In particolare:

- m è la probabilità di apertura dei gate di attivazione dei canali Na^+;
- h la probabilità di apertura dei gate di inattivazione dei canali Na^+;
- n la probabilità di apertura dei gate di attivazione dei canali K^+.

Queste variabili soddisfano la seguente equazione del primo ordine:

$$\frac{dx}{dt} = \alpha_x(V)\,(1 - x) \;-\; \beta_x(V)\,x \quad \text{con } x \in \{m, h, n\} \tag{4.5}$$

Dove:

- $\alpha_x(V)$ è il tasso di transizione del gate x dallo stato chiuso a quello aperto in funzione del potenziale di membrana V.
- $\beta_x(V)$ è il tasso di transizione del gate x dallo stato aperto a quello chiuso in funzione del potenziale di membrana V.

Caratteristiche del modello

- Riproduce con elevata fedeltà le fasi di depolarizzazione e ripolarizzazione del potenziale d'azione.
- Richiede la soluzione numerica di quattro equazioni differenziali accoppiate.
- Fondamentale per studi biofisici, ha ispirato versioni ridotte e modelli semplificati

4.3 Fondamenti teorici dell'Apprendimento Sinaptico

4.3.1 La Regola di Hebb

Una delle più importanti ipotesi postulate sui meccanismi dell'apprendimento sinaptico è la regola di Hebb, formulata per la prima volta dallo psicologo canadese Donald Hebb nel 1949. Questa regola afferma che se un neurone presinaptico contribuisce ripetutamente all'attivazione di un neurone postsinaptico, la connessione sinaptica tra i due verrà rafforzata. Questo concetto è spesso riassunto nella frase: "neuroni che attivano insieme, si connettono insieme" ("cells that fire together, wire together")[15]. Da questa formulazione si può comprendere che esiste una potenziale relazione causale tra l'attivazione dei due neuroni. Questa causalità tra neuroni richiede, quindi, che il neurone presinaptico si attivi leggermente prima di quello postsinaptico per rafforzare il legame sinaptico che c'è tra i due. La regola di Hebb può essere espressa come:

$$\Delta w_{ij} = \eta \cdot x_i \cdot y_j \tag{4.6}$$

dove:

- Δw_{ij} è la variazione del peso sinaptico tra il neurone presinaptico i e quello postsinaptico j;
- x_i è l'attività del neurone presinaptico
- y_j è l'attività del neurone postsinaptico;
- η è il tasso di apprendimento ("learning rate").

Quindi il peso sinaptico si aggiorna con:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \tag{4.7}$$

Questa regola ha gettato le fondamenta teoriche per comprendere come la plasticità sinaptica possa scaturire l'apprendimento e la memoria nel cervello biologico, ovvero che derivano dal rafforzamento delle connessioni tra neuroni che si attivano in modo contiguo nel tempo. Inoltre, la regola di Hebb ha gettato le basi per modelli di apprendimento non supervisionato ('Unsupervised Learning') basati sulla correlazione tra attività pre- e post-sinaptica.

4.3.2 Spike Timing Dependent Plasticity (STDP)

La Spike Timing Dependent Plasticity (STDP) è una forma temporalmente asimmetrica della regola di apprendimento di Hebb citata in precedenza, dovuta alle strette correlazioni tra gli spike del neurone pre–sinaptico e quelli del neurone post–sinaptico. Diversamente dal postulato originale di Hebb, che prevedeva solo potenziamento sinaptico, la STDP introduce anche

un meccanismo di indebolimento: in particolare, l'arrivo ripetuto di spike pre–sinaptici pochi millisecondi prima del potenziale d'azione post–sinaptico genera un Long-Term Potentiation (LTP), mentre uno spike pre–sinaptico che arriva subito dopo l'attivazione post–sinaptica induce un Long-Term Depression (LTD). Questo doppio meccanismo era in realtà già anticipato nel 1973 da Stent, il quale affermò che un input presinaptico non costantemente co-attivo con la cellula postsinaptica dovesse subire un indebolimento.[16] [17] Il cambiamento sinaptico in funzione del ritardo $\Delta t = t_{\text{post}} - t_{\text{pre}}$ è descritto dalla funzione STDP:

$$\Delta w(\Delta t) = \begin{cases} A_+ \, e^{-\Delta t/\tau_+}, & \Delta t > 0 \quad \text{(LTP)}, \\ -\, A_- \, e^{\,\Delta t/\tau_-}, & \Delta t < 0 \quad \text{(LTD)}, \end{cases}$$

dove $A_\pm$ e $\tau_\pm$ sono parametri che variano a seconda del tipo di sinapsi. L'alta sensibilità di Δw al timing relativo degli spike suggerisce la possibilità di codificare informazioni su scale temporali di pochi millisecondi.

La STDP è quindi una regola di apprendimento locale, in cui ogni sinapsi modifica il proprio peso basandosi esclusivamente sugli spike che riceve e sugli spike generati dal neurone a cui è connessa. Questa caratteristica la rende particolarmente adatta a implementazioni neuromorfiche, dove memoria ed elaborazione sono co-localizzate e le operazioni avvengono in maniera distribuita ed event-driven.

4.3.3 Regola di aggiornamento sinaptico STDP

Come discusso in precedenza, nella Spike-Timing-Dependent Plasticity(STDP) la variazione del peso sinaptico Δw_j associato alla sinapsi proveniente dal neurone pre-sinaptico j dipende dalla differenza temporale tra gli spike pre- e post-sinaptici. Indichiamo con:

- t_j^f l'istante di arrivo del f-esimo spike pre-sinaptico sulla sinapsi j;
- t_i^n l'istante di emissione del n-esimo spike post-sinaptico del neurone i.

Il contributo al cambiamento di peso dovuto alla singola coppia di spike (f, n) è espresso da:

$$\Delta w_{ij}^{(f,n)} = W\big(t_i^n - t_j^f\big), \tag{4.8}$$

dove $W(\Delta t)$ è la funzione di apprendimento temporale. Il cambiamento complessivo del peso w_j si ottiene sommando su tutte le coppie di spike pre- e post-sinaptici:

$$\Delta w_j = \sum_f \sum_n W\big(t_i^n - t_j^f\big). \tag{4.9}$$

Una forma comunemente adottata per $W(\Delta t)$ è:

$$W(\Delta t) = \begin{cases} A_+ \exp\left(-\dfrac{\Delta t}{\tau_+}\right), & \Delta t > 0, \\ -A_- \exp\left(\dfrac{\Delta t}{\tau_-}\right), & \Delta t < 0, \\ 0, & \Delta t = 0 \end{cases} \tag{4.10}$$

dove A_+ e A_- sono ampiezze positive (eventualmente dipendenti dal peso corrente w_j) e τ_+, τ_- sono costanti di tempo tipicamente dell'ordine di 10 ms.

In questa formulazione, gli impulsi pre-sinaptici che precedono di poco quelli post-sinaptici ($\Delta t > 0$) inducono un rafforzamento della sinapsi (potenziamento a lungo termine), mentre quelli pre-sinaptici successivi a quelli post-sinaptici ($\Delta t < 0$) ne provocano l'indebolimento (depressione a lungo termine). Questa regola rappresenta uno dei meccanismi fondamentali dell'apprendimento nelle SNN.

Capitolo 5

Hardware Neuromorfico: Loihi 1 e Loihi 2

5.1 Loihi 1: Architettura Neuromorfica Digitale

Presentato da Intel nel 2018, Loihi è uno dei primi chip neuromorfici digitali programmabili. Questo chip è stato concepito per simulare Spiking Neural Networks (SNN) in maniera efficiente e scalabile. Esso presenta un'architettura completamente digitale che approssima il funzionamento dei neuroni a tempo continuo tramite un modello discreto.

5.1.1 Caratteristiche di apprendimento

Loihi integra un motore di apprendimento ("learning engine") programmabile che supporta plasticità sinaptica avanzata direttamente in hardware. Le principali funzionalità includono:

- Tracce di impulsi ("spike traces"): segnali filtrati di spike pre- e post-sinaptici, con costanti di tempo configurabili.
- Tracce multiple: possibilità di applicare filtri multipli alla stessa sequenza di spike, generando segnali temporali diversificati.
- Variabili sinaptiche aggiuntive: ogni sinapsi dispone di due variabili di stato extra, oltre al peso, per regole di plasticità complesse.
- Tracce di ricompensa: spike speciali che trasportano valori di reward o punishment, utili per algoritmi di reinforcement learning.

Queste capacità rendono Loihi il primo chip SNN digitale a supportare in hardware modelli con plasticità estesa e modulata.

5.1.2 Primitive computazionali

Il chip fornisce meccanismi innovativi per la ricerca algoritmica:

- Generazione di rumore stocastico;
- Ritardi configurabili (assonali, sinaptici, refrattari);
- Elaborazione dinamica di compartimenti dendritici;
- Adattamento della soglia di attivazione per omeostasi dell'eccitabilità;
- Ridimensionamento e saturazione dei pesi per preservare la "permanence".

5.1.3 Componenti hardware

L'architettura è organizzata come una mesh multicore composta da:

- 128 neural cores per il calcolo neuromorfico;
- 3 core x86 embedded per controllo e gestione;
- Interfacce off-chip per estendere la rete a più chip.

La Network-on-Chip (NoC) asincrona gestisce:

- Comandi di lettura/scrittura;
- Comunicazione tra core x86;
- Spike messages per le SNN;
- Barriere di sincronizzazione.

Il protocollo supporta fino a 4096 core on-chip e, teoricamente, fino a 16384 chip tramite indirizzamento gerarchico.

5.1.4 Struttura neuronale e fan-out

Ogni "neural core" implementa fino a 1.024 compartimenti neurali, organizzati in alberi che formano neuroni multi-compartimentali. La connettività è flessibile:

- reti fully connected;
- reti sparse o binarie;
- pesi ad alta precisione per l'apprendimento.

Per ridurre la complessità $O(n^2)$ delle reti dense, Loihi adotta:

- Compressione di reti sparse;
- Multicast core-to-core;
- Formati sinaptici variabili (1–9 bit, con/ senza segno);
- Connettività gerarchica a livello di popolazione.

5.1.5 Efficienza e paradigma event-driven

L'implementazione digitale asincrona (bundled data) sfrutta la natura sparsa delle SNN, attivando il calcolo solo in presenza di eventi. Questo approccio riduce drasticamente il consumo energetico (meno di 1.5 W in molte applicazioni) e consente latenze dell'ordine dei millisecondi.

5.2 Loihi 2

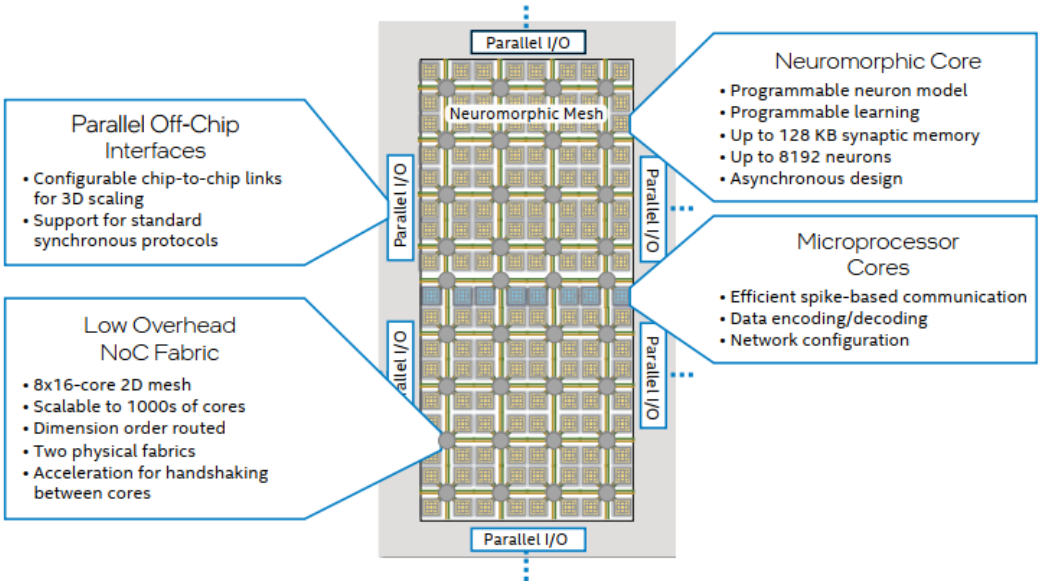


Figura 5.1: Struttura del chip Loihi 2

5.2.1 Architettura di base

Presentato da Intel nel 2021, Loihi 2 è la seconda generazione del processore neuromorfico sperimentale, progettato per migliorare velocità, programmabilità e densità rispetto al predecessore.[18] Il chip mantiene un'architettura multicore asincrona, con:

- core neuromorfici programmabili, ciascuno dotato di memoria locale riconfigurabile;
- rete on-chip (Network-on-Chip, NoC) ridisegnata per aumentare la banda e ridurre la latenza;
- supporto a modelli neurali personalizzati tramite microcodice.

Ogni core neuromorfico integra unità di calcolo specializzate per l'elaborazione di eventi (event-driven) e gestione diretta di sinapsi e neuroni virtuali.

5.2.2 Nuove funzionalità

Rispetto a Loihi 1, Loihi 2 introduce miglioramenti significativi:

- Spike con payload fino a 32 bit, permettendo di codificare informazione sia in ampiezza che in tempo;
- Modelli neurali completamente programmabili via microcodice, con possibilità di definire dinamiche personalizzate di neuroni e sinapsi;
- Plasticità a tre fattori in hardware, combinando STDP con segnali di modulazione esterni;
- Maggiore densità rispetto alla prima generazione, con capacità fino a circa 1 milione di neuroni e 120 milioni di sinapsi per chip (a seconda della configurazione);
- Compressione avanzata della connettività e gestione flessibile della memoria per ottimizzare l'uso delle risorse.

5.2.3 Scalabilità e I/O

Loihi 2 è progettato per scalare in sistemi multi-chip e integrarsi in piattaforme eterogenee:

- comunicazione locale di spike per ridurre il traffico inter-chip;
- supporto a interconnessioni ad alta densità per configurazioni multi□chip ("3D stacking");
- interfacce standard a livello di sistema (ad esempio PCIe o Ethernet) per l'integrazione in server, sistemi embedded o cloud.

5.3 Lava: il framework software

5.3.1 Introduzione a Lava

Lava è un framework open source sviluppato da Intel per sfruttare al meglio le potenzialità di Loihi 2 e altre piattaforme neuromorfiche. È modulare, componibile e platform-agnostic, consentendo prototipazione su CPU/GPU e deployment su hardware neuromorfico.

5.3.2 Architettura e Magma

Lava è organizzato in processi asincroni che comunicano tramite passaggio di messaggi ("Communicating Sequential Processes"). Magma, il runtime di Lava, gestisce la compilazione e l'esecuzione dei processi Lava su diversi backend (CPU, GPU, Loihi).

5.3.3 Funzionalità principali

- Supporto a modelli neurali e topologie personalizzate;
- Addestramento offline e quantizzazione;
- Supporto all'importazione di modelli addestrati in PyTorch o TensorFlow tramite toolchain esterni;
- Profilazione di prestazioni e consumi;
- Rilasciato con licenza BSD-3 per promuovere la partecipazione attiva di e uno sviluppo collaborativo. [1]

Grazie a Lava, la programmazione neuromorfica diventa più accessibile, accelerando la transizione dalla ricerca teorica alle applicazioni reali.

[1] `https://github.com/lava-nc/lava/blob/main/LICENSE`

Capitolo 6

Costruzione di una SNN tramite Brian2

6.1 Introduzione a Brian2

Brian2 è un simulatore di SNN interamente sviluppato in Python. A differenza di molti altri simulatori, che costringono a scegliere tra linguaggi di alto livello, espressivi ma meno efficienti, e linguaggi di basso livello, performanti ma poco adatti a logiche sperimentali complesse, Brian2 adotta un approccio basato sulla generazione di codice a runtime. I modelli, definiti in Python, vengono tradotti automaticamente in codice compilato ottimizzato (C++ per impostazione predefinita, oppure CUDA per l'esecuzione su GPU tramite `brian2cuda`).

Brian2 è la seconda versione di Brian1, introdotto nel 2008 da Dan Goodman e Romain Brette come alternativa più accessibile rispetto a simulatori complessi quali NEURON e NEST. Brian1, pur basato su Python e NumPy, soffriva di limiti significativi: esecuzione interamente interpretata, assenza di generazione di codice compilato e difficoltà nell'estendere le funzionalità senza intervenire sul motore interno. Questi vincoli lo rendevano adatto solo a modelli di piccola o media scala.

Brian2 supera tali limitazioni grazie a un'architettura modulare e componibile, in cui equazioni neurali, sinapsi, regole di plasticità e strumenti di monitoraggio sono moduli autonomi e combinabili. Le dinamiche possono essere definite in modo dichiarativo e tradotte nel backend più adatto (C++, CUDA o Python), mentre il sistema di unità fisiche integrato previene errori dimensionali e garantisce coerenza.

Questo approccio incrementa la velocità di simulazione, rendendo gestibili reti con milioni di sinapsi, e consente di modellare fenomeni biologici complessi (STDP, bursting, adattamento, plasticità non lineare) con regole di apprendimento personalizzate. Grazie a queste caratteristiche, Brian2 si conferma uno strumento ideale per simulazioni di reti neurali biologicamente plausibili e scalabili, pur non essendo ancora ottimizzato per simulazioni in tempo reale o per reti di dimensioni estreme tipiche di ambienti High-Performance Computing.

6.2 Installazione e setup

Per eseguire le simulazioni con Brian2 è consigliabile predisporre un ambiente Python isolato, così da gestire in modo controllato le dipendenze e garantire la riproducibilità degli esperimenti.

6.2.1 Creazione dell'ambiente virtuale

È possibile creare un ambiente virtuale sia con `venv`. Da terminale quindi scriviamo:

Con `venv`:

```
Su Linux/Mac:
python3 -m venv brian_env
source brian_env/bin/activate

Su Windows:
python -m venv brian_env
brian_env\Scripts\activate
```

6.2.2 Installazione di Brian2 e librerie ausiliarie

Ora che ci troviamo dentro l'ambiente virtuale, bisogna installare Brian2 e le librerie più comuni utilizzate per il preprocessing e la visualizzazione dei dati:

```
pip install brian2 numpy matplotlib scikit-learn
```

Per maggiori informazioni riguardanti l'installazione di Brian2 e le librerie si consiglia la visione della documentazione ufficiale:

- `https://brian2.readthedocs.io/en/stable/introduction/install.html`
- `https://brian2.readthedocs.io/`

6.2.3 Verifica dell'installazione

Per verificare che Brian2 sia correttamente configurato, possiamo far eseguire il seguente codice:

```
from brian2 import *

start_scope()

tau = 10*ms
eqs="""
dv/dt = (1-v)/tau : 1
"""
G = NeuronGroup(1, eqs, method='exact')

run(1*ms)

print("Brian2 installato correttamente, versione:", brian2.
    __version__)
```

Se lo script stampa la versione di Brian2 senza produrre errori, allora possiamo procedere con la definizione di modelli LIF, la configurazione di sinapsi STDP e l'esecuzione di semplici modelli di Spiking Neural Networks.

6.2.4 Definizione del modello neuronale

Come descritto nella sezione 4.2.3, il modello di neurone Leaky Integrate-and-Fire (LIF) è stato scelto per la simulazione della rete spiking per via della sua semplicità e della capacità di riprodurre in modo plausibile la dinamica di base del potenziale di membrana. In questa sezione ci concentriamo sulla sua implementazione in `Brian2`.

Il modello è definito come un `NeuronGroup` in cui la variabile di stato `v` rappresenta il potenziale di membrana e `I` la corrente sinaptica in ingresso. La soglia di attivazione, il reset e il periodo refrattario sono impostati secondo i parametri scelti per la simulazione:

```
from brian2 import *

# Parametri
tau_m   = 20*ms
Vrest   = -70*mV
Vth     = -54*mV
Vreset  = -60*mV
tau_ref = 5*ms
N = 100 # numero di neuroni

# Equazioni del neurone LIF
eqs = '''
dv/dt = (Vrest - v + I)/tau_m : volt (unless refractory)
I      : volt
'''

# Creazione del gruppo neuronale (N neuroni)
G = NeuronGroup(N, model=eqs,
                     threshold='v > Vth',
                     reset='v = Vreset',
                     refractory=tau_ref,
                     method='euler')
hidden.v = Vrest
```

Lo script appena proposto, infatti, non restituisce ancora nulla, ma è utile osservare degli aspetti. In particolare, in questo script:

- `v` è il potenziale di membrana;
- `I` raccoglie le correnti sinaptiche in ingresso;
- il metodo `euler` è sufficiente per integrare l'equazione nel contesto di reti di piccola scala.

Questa struttura modulare consente di estendere facilmente il modello, ad esempio introducendo adattamento o dinamiche dendritiche, e costituisce la base per l'applicazione delle regole di plasticità STDP nelle sinapsi.

6.2.5 Implementazione di Spike-Timing-Dependent Plasticity

Spike-Timing-Dependent-Plasticity (STDP), che regola l'apprendimento sinaptico, che aggiorna i pesi w in base all'intervallo temporale Δt tra impulsi ("spikes") pre- e post-sinaptici. La legge di aggiornamento adottata è:

$$\Delta w = \begin{cases} A_+ \exp\left(-\dfrac{\Delta t}{\tau_+}\right), & \Delta t > 0 \quad \text{(LTP)}, \\ -A_- \exp\left(\dfrac{\Delta t}{\tau_-}\right), & \Delta t < 0 \quad \text{(LTD)} \end{cases}$$

dove $\Delta t = t_{\text{post}} - t_{\text{pre}}$, A_+ e A_- sono le ampiezze di potenziamento e depressione, mentre τ_+ e τ_- sono costanti di decadimento.

In `Brian2`, questa regola si implementa mantenendo due variabili (`Apre` e `Apost`) che si aggiornano in corrispondenza degli eventi pre- e post-sinaptici. Il peso viene modificato nelle variabili `on_pre` e `on_post` secondo la funzione $W(\Delta t)$. Inizialmente, con le informazioni acquisite nel paragrafo precedente, creiamo 2 gruppi di neuroni, chiamati `group1` e `group2`.

```
from brian2 import *
import numpy as np
import matplotlib.pyplot as plt

# Parametri
tau_m   = 20*ms
Vrest   = -70*mV
Vth     = -54*mV
Vreset  = -60*mV
tau_ref = 5*ms
N = 100 #numero di neuroni

# Parametri STDP
tau_pre  = 20*ms
tau_post = 20*ms
A_pre    = 0.01
A_post   = -A_pre * 1.05
w_max    = 1.0

# Modello LIF
eqs = '''
dv/dt = (Vrest - v + I)/tau_m : volt (unless refractory)
I      : volt
```

```
'''

# Definizione dei gruppi di neuroni di esempio
group1 = NeuronGroup(N, eqs, threshold='v>1', reset='v=0')
group2 = NeuronGroup(N, eqs, threshold='v>1', reset='v=0')

# Sinapsi con STDP
stdp = Synapses(group1, group2,
    model='''
        w : 1
        dApre/dt  = -Apre/tau_pre  : 1 (event-driven)
        dApost/dt = -Apost/tau_post: 1 (event-driven)
    ''',
    on_pre='''
        I_post += w * mV
        Apre   += A_pre
        w       = clip(w + Apost, 0, w_max)
    ''',
    on_post='''
        Apost  += A_post
        w       = clip(w + Apre, 0, w_max)
    '''
)
stdp.connect(p=1.0)
stdp.w = 'rand()'
```

In questo schema:

- `dApre/dt` e `dApost/dt` descrivono il decadimento esponenziale delle tracce temporali.
- In `on_pre`, il potenziale post-sinaptico viene incrementato di w e il peso aggiornato in base alla traccia post-sinaptica.
- In `on_post`, la traccia post-sinaptica viene incrementata e il peso modificato in base alla traccia pre-sinaptica.
- `clip` mantiene w entro l'intervallo $[0, w_{\text{max}}]$.

6.2.6 Simulazione e rappresentazione della rete

Una volta definiti i gruppi neuronali e le sinapsi STDP, possiamo procedere nel comporre una simulazione della SNN. In questa fase utilizziamo due tipi di monitor:

- SpikeMonitor, per registrare i tempi di spike di ciascun neurone.

- StateMonitor, per tracciare nel tempo variabili come il potenziale di membrana o i pesi sinaptici.

Di seguito un esempio di codice in Brian2 che configura i monitor e lancia una simulazione:

```
from brian2 import *
import numpy as np
import matplotlib.pyplot as plt


#INIZIO DELLA SIMULAZIONE

start_scope()                #inizializza l'ambiente di simulazione


#PARAMETRI GRUPPI NEURONALI

tau_m    = 20*ms        # costante di tempo di membrana
Vrest    = -70*mV       # potenziale di riposo
Vth      = -54*mV       # soglia di attivazione (spike)
Vreset   = -60*mV       # potenziale di reset dopo lo spike
tau_ref  = 5*ms         # periodo refrattario
N        = 100          # numero di neuroni in ciascun gruppo
sim_time = 500*ms       # durata totale della simulazione


#PARAMETRI STDP

tau_pre   = 20*ms       # decadimento del tracciato pre-sinaptico
tau_post  = 20*ms       # decadimento del tracciato post-sinaptico
A_pre     = 0.01        # incremento del tracciato pre-sinaptico
A_post    = -A_pre*1.05 # incremento del tracciato post-sinaptico
w_max     = 1.0         # peso sinaptico massimo


#EQUAZIONE LIF

eqs = '''
dv/dt = (Vrest - v + I)/tau_m : volt (unless refractory)  #
   variazione di v
I      : volt                                             # corrente
```

```
    sinaptica
'''


#GRUPPI DI NEURONI

group1 = NeuronGroup(
    N, eqs,
    threshold='v > Vth',          # condizione per emettere spike
    reset='v = Vreset',           # reset di v dopo lo spike
    refractory=tau_ref,           # refrattarietà
    method='euler',               # metodo di integrazione
    namespace={'Vth': Vth, 'Vreset': Vreset}  # variabili esterne
    usate nel modello
)

group2 = NeuronGroup(
    N, eqs,
    threshold='v > Vth',
    reset='v = Vreset',
    refractory=tau_ref,
    method='euler',
    namespace={'Vth': Vth, 'Vreset': Vreset}
)

#si inizializza lo stato di v e I in tutti i neuroni di entrambi i
    gruppi
for grp in (group1, group2):
    grp.v = Vrest    # imposta v 'allinizio al potenziale di riposo
    grp.I = 0*mV     # imposta la corrente sinaptica iniziale a zero


#INPUT POISSON → group1

poisson = PoissonGroup(N, rates=50*Hz)    # generatore Poisson di
    spike
inp1    = Synapses(poisson, group1, on_pre='I_post += 2.0*mV')  #
    sinapsi 'dingresso
inp1.connect(j='i')                       # connessione one-to-one
```

```
#SINAPSI STDP group1 → group2

stdp = Synapses(
    group1, group2,
    model='''
        w                : 1                                    # peso
   sinaptico
        dApre/dt  = -Apre/tau_pre   : 1 (event-driven)       #
   tracciato pre
        dApost/dt = -Apost/tau_post : 1 (event-driven)       #
   tracciato post
    ''',
    on_pre='''
        I_post += w * 1.0*mV     # aggiunge corrente postsinaptica
        Apre   += A_pre          # aggiorna il tracciato pre
        w      = clip(w + Apost, 0, w_max)  # modifica il peso
    ''',
    on_post='''
        Apost  += A_post         # aggiorna il tracciato post
        w      = clip(w + Apre, 0, w_max)   # modifica il peso
    '''
)
stdp.connect(p=1.0)   # collega tutti a tutti
stdp.w     = 'rand()' # inizializza i pesi casualmente [0,1]
stdp.Apre  = 0        # inizializza Apre a zero
stdp.Apost = 0        # inizializza Apost a zero


#MONITORAGGI DI group1 E group2

spikes1  = SpikeMonitor(group1)                # registra spike di
   group1
state1_v = StateMonitor(group1, 'v', record=0) # registra v del
   neurone 0 in group1

spikes2  = SpikeMonitor(group2)                # registra spike di
   group2
```

```
state2_v = StateMonitor(group2, 'v', record=0) # registra v del
    neurone 0 in group2


# SIMULAZIONE

run(sim_time)  # esegue la simulazione per sim_time


# PLOTTAGGI

plt.figure(figsize=(6, 4))
# raster di group1
plt.subplot(2,1,1)
plt.plot(spikes1.t/ms, spikes1.i, '.b')
plt.xlabel('t (ms)')
plt.ylabel('neurone')
plt.title('Raster - group1')
# potenziale membrana gruppo 1
plt.subplot(2,1,2)
plt.plot(state1_v.t/ms, state1_v.v[0]/mV, 'b')
plt.xlabel('t (ms)')
plt.ylabel('v (mV)')
plt.title('v - group1 neuron 0')
plt.tight_layout()
plt.show()

plt.figure(figsize=(6, 4))
# raster di group2
plt.subplot(2,1,1)
plt.plot(spikes2.t/ms, spikes2.i, '.r')
plt.xlabel('t (ms)')
plt.ylabel('neurone')
plt.title('Raster - group2')
# potenziale membrana gruppo 2
plt.subplot(2,1,2)
plt.plot(state2_v.t/ms, state2_v.v[0]/mV, 'r')
plt.xlabel('t (ms)')
plt.ylabel('v (mV)')
```

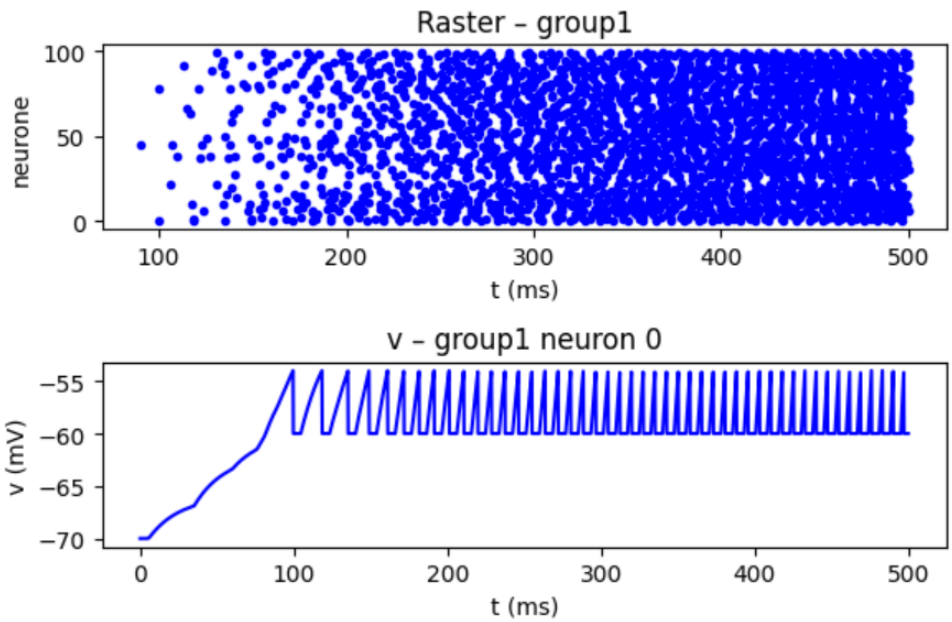
```
plt.title('v - group2 neuron 0')
plt.tight_layout()
plt.show()
```

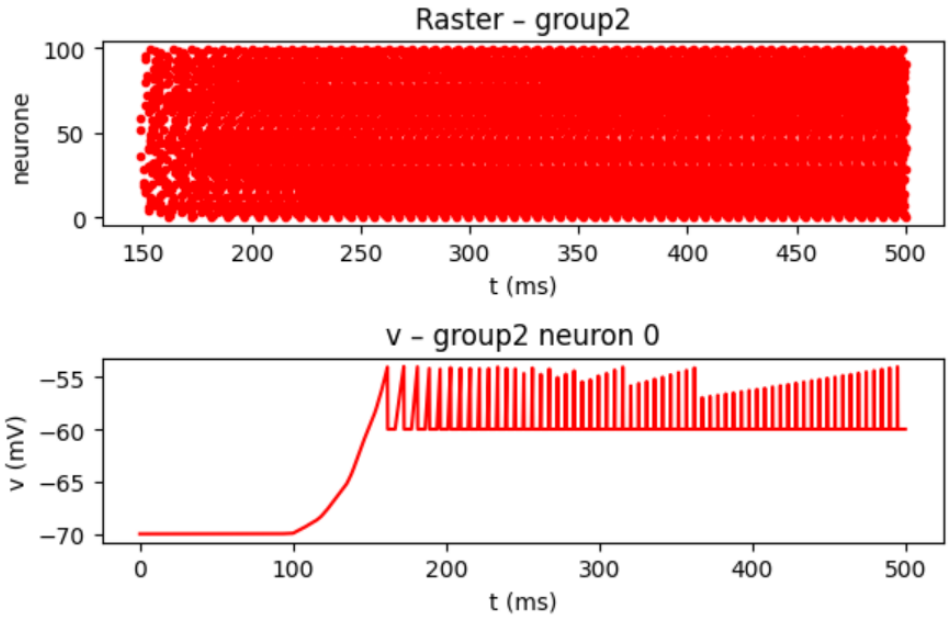
Al termine della simulazione, i dati raccolti dai monitor possono essere utilizzati per generare:

- Raster plot: distribuzione temporale degli spike per ogni gruppo neuronale.
- Tracce di potenziale: evoluzione del potenziale di membrana dei neuroni selezionati.
- Evoluzione dei pesi: andamento temporale di alcuni pesi sinaptici per evidenziare l'effetto della plasticità.

In particolare, nella simulazione appena proposta, i grafici mostrati rappresentano:

- raster plot di `group1`;
- raster plot di `group2`;
- potenziale di membrana di `group1` e `group2` attraverso il tracciamento di solo di un singolo neurone, mettendo in evidenza gli impulsi che emette prima e dopo il rafforzamento delle sinapsi.





6.2.7 Analisi dei risultati

Al termine della simulazione, i dati raccolti con i monitor sono elaborati per comprendere le dinamiche interne della rete. Le principali metriche considerate sono:

1. Spike Raster: Dal raster plot di group1 si osserva un'attività irregolare con firing rate medio vicino a 50 Hz (dato dal PoissonGroup). Il raster plot di group2 mostra che, inizialmente, solo poche sinapsi casuali causano spike; in seguito, STDP rafforza le connessioni causali e cresce la densità di eventi.

2. Dinamica del potenziale di membrana: in group1 (un neurone) lo StateMonitor evidenzia transienti rapidi da –70 mV fino alla soglia (–54 mV) e reset a –60 mV, con refrattarietà di 5 ms. In group2 (un neurone) i picchi di v sono meno frequenti all'inizio (pesi deboli), quindi crescono di ampiezza e frequenza man mano che STDP rafforza i collegamenti sinaptici .

3. Tasso di firing medio: il firing rate medio di ciascun layer viene calcolato attraverso le istruzioni:

```
        rate1 = len(spikes1.i)/(N*sim_time)  # firing rate
group1
        rate2 = len(spikes2.i)/(N*sim_time)  # firing rate
group2
```

dove tipicamente: rate1 = circa 50 Hz, rate2 < rate1 all'inizio.

4. Effetto della STDP sui pesi: sebbene sulla simulazione proposta non è tracciato direttamente stdp.w, l'aumento di attività in group2 riflette il potenziamento dei pesi. Per un'analisi quantitativa si consiglia di aggiungere:

```
w_mon = StateMonitor(stdp, 'w', record=range(10))
```

Questa analisi fornisce una panoramica semplice, ma esplicativa, del comportamento dinamico del modello, mettendo in evidenza l'efficacia della plasticità STDP.

6.2.8 Possibili estensioni e sviluppi futuri

Il prototipo di rete SNN presentato nei paragrafi precedenti rappresenta una base solida, ma può essere ulteriormente sviluppato in diverse direzioni:

- Utilizzo di dataset più complessi e codifica degli spikes: ampliare l'insieme di dati di ingresso oltre il semplice MNIST binario, sperimentando tecniche di codifica temporale degli spike come il "latency coding" (dove l'informazione è nel ritardo di emissione dell'impulso) o il "rank-order coding" (dove conta l'ordine di arrivo degli spike). Questo permetterebbe di affrontare dataset più ricchi, come CIFAR-10 per immagini a colori o insiemi di dati audio già convertiti in sequenze di impulsi. [19] [20]
- Adozione di modelli neuronali più sofisticati: sostituire il modello LIF con modelli che riproducono in modo più realistico il comportamento dei neuroni biologici, come il modello di Izhikevich.[21] Questi modelli consentono di simulare fenomeni complessi come il "bursting" (scariche multiple ravvicinate) o l'adattamento della frequenza di scarica.
- Plasticità combinata e apprendimento supervisionato: integrare la STDP con altre tecniche di apprendimento, ad esempio il "surrogate gradient descent" o metodi di addestramento supervisionato locale, per migliorare le prestazioni della rete in compiti di classificazione. [22]
- Implementazione su hardware neuromorfico: trasferire la rete su piattaforme fisiche dedicate, come Intel Loihi o SpiNNaker, per confrontare direttamente tempi di esecuzione, consumo energetico e accuratezza rispetto alla simulazione software, valutando così la fattibilità in contesti embedded o a basso consumo.
- Apprendimento continuo ("lifelong learning"): introdurre protocolli in cui i dati vengono presentati in sequenza, senza azzerare lo stato interno della rete, per studiare il fenomeno del "catastrophic forgetting" (perdita di conoscenze precedenti) e sperimentare strategie di stabilizzazione come meccanismi di homeostasi o il "replay" di esperienze passate. [23] [24]

- Ottimizzazione automatica dei parametri: applicare tecniche di ricerca automatizzata degli iperparametri, come la ”grid search” o la ”Bayesian optimization”, per individuare i valori ottimali di parametri chiave (ad esempio costanti di tempo τ, tasso di apprendimento η, densità di connessioni e pesi iniziali).[25][26]

Queste estensioni non solo arricchiscono la validità sperimentale dell'approccio, ma offrono spunti per lavori futuri sia in ambito accademico sia industriale, esplorando il potenziale delle Spiking Neural Networks in scenari di apprendimento real-time e a basso consumo energetico.

Capitolo 7

Conclusioni

In questa tesi si è cercato di spiegare i principi chiave del Neuromorphic Computing attraverso un'analisi approfondita, con l'obiettivo di far comprendere al lettore le potenzialità di tale campo. Il Neuromorphic Computing si ispira a riprodurre le caratteristiche tipiche del cervello umano, come l'elaborazione event-driven; il parallelismo massivo dei neuroni e delle sinapsi e la plasticità locale, ponendo le basi per sistemi più efficienti e con risposte a tempo reale.

7.1 Limitazioni del Neuromorphic Computing

Il Neuromorphic Computing nasce dall'ambizione di replicare i meccanismi del cervello biologico, soprattutto l'efficienza e la versatilità. Tuttavia, ad oggi si scontrano con una serie di sfide che ne limitano l'impiego su larga scala. In particolare, la natura "a gradino" degli impulsi generati dalle SNN non rende possibile l'utilizzo di metodi di ottimizzazione ("Gradient descent") tipici del Deep Learning tradizionale. Per ottimizzare le SNN, infatti, vengono utilizzate delle tecniche che possono aggiungere complessità computazionale e possono ridurre la precisione del modello. Inoltre, sebbene le SNN offrano potenziali risparmi energetici grazie all'elaborazione event-driven, esse faticano ancora a eguagliare l'accuratezza delle reti tradizionali che, supportate da GPU("Graphics Processing Unit") e TPU("Tensor Processing Unit"), forniscono risultati più veloci e affidabili. Infine, sul fronte hardware sono diverse le lacune da colmare per poter competere con le architetture tradizionali, soprattutto in termini di scalabilità e variabilità dei dispositivi. La piattaforma Loihi, spiegata precedentemente, dimostra dei consumi ridotti per singoli chip, ma quando si sale a decine o centinaia di core neuromorfici la complessità di routing delle connessioni e la variabilità di fabbricazione diventano degli ostacoli concreti.

7.2 Sviluppi futuri

In futuro, il Neuromorphic Computing è destinato a progredire principalmente su tre fronti in relazione tra loro:

- a livello algoritmico, per quanto riguarda lo sviluppo di nuovi algoritmi di training nativi per SNN che permetteranno di rendere differenziabili gli impulsi
- a livello hardware, con architetture in grado di ospitare migliaia di core sinaptici con bassa variabilità di fabbricazione (come Loihi 2, già citato in precedenza)
- a livello software, invece, sono necessari framework che traducono modelli da Pytorch (o TensorFlow) a configurazioni neuromorfiche ottimizzate, migliorando: la compatibilità delle SNN con l'hardware dedicato; il "mapping" dei neuroni e sinapsi sui core neuromorfici e la calibrazione dei parametri fisici del chip neuromofico, che compensa variabilità di processo e una maggior precisione.

Nel complesso, sebbene il Neuromorphic Computing riscontri delle limitazioni, lo sviluppo di nuove tecnologie in questo campo prelude a un futuro in cui i sistemi di intelligenza artificiale potranno operare in modo più sostenibile e adattivo, avvicinandosi così al comportamento complesso del cervello umano.

Bibliografia

[1] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[2] C. F. Stevens, «The neuron,» *Scientific American*, vol. 241, n. 3, pp. 54–65, 1979.

[3] S. Sharma, S. Sharma e A. Athaiya, «Activation functions in neural networks,» *Towards Data Sci*, vol. 6, n. 12, pp. 310–316, 2017.

[4] F. Rosenblatt, «The perceptron: a probabilistic model for information storage and organization in the brain.,» *Psychological review*, vol. 65, n. 6, p. 386, 1958.

[5] A. Vaswani, N. Shazeer, N. Parmar et al., «Attention is all you need,» *Advances in neural information processing systems*, vol. 30, 2017.

[6] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu e N. Mastorakis, «Multilayer perceptron and neural networks,» *WSEAS Transactions on Circuits and Systems*, vol. 8, n. 7, pp. 579–588, 2009.

[7] C. Mead, «Neuromorphic electronic systems,» *Proceedings of the IEEE*, vol. 78, n. 10, pp. 1629–1636, 2002.

[8] T. Gollisch e M. Meister, «Rapid neural coding in the retina with relative spike latencies,» *Science*, vol. 319, n. 5866, pp. 1108–1111, 2008.

[9] E. D. Adrian e Y. Zotterman, «The impulses produced by sensory nerve endings: Part I. Impulses set up by single end□organs,» *The Journal of Physiology*, vol. 61, n. 2, pp. 145–171, 1926.

[10] J. D. Victor e K. P. Purpura, «Nature and precision of temporal coding in visual cortex: a metric-space analysis,» *Journal of Neurophysiology*, vol. 76, n. 2, pp. 1310–1326, 1996.

[11] W. S. McCulloch e W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» *The bulletin of mathematical biophysics*, vol. 5, n. 4, pp. 115–133, 1943.

[12] S. Lu e F. Xu, «Linear leaky-integrate-and-fire neuron model based spiking neural networks and its mapping relationship to deep neural networks,» *Frontiers in neuroscience*, vol. 16, p. 857 513, 2022.

[13] Wikipedia contributors. «Modello di Hodgkin–Huxley — Wikipedia, l'enciclopedia libera.» (2025), indirizzo: https://it.wikipedia.org/wiki/Modello_di_Hodgkin-Huxley.

[14] A. L. Hodgkin e A. F. Huxley, «A quantitative description of membrane current and its application to conduction and excitation in nerve,» *The Journal of Physiology*, vol. 117, n. 4, pp. 500–544, 1952.

[15] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[16] H. Markram, W. Gerstner e P. J. Sjöström, «Spike-timing-dependent plasticity: a comprehensive overview,» *Frontiers in synaptic neuroscience*, vol. 4, p. 2, 2012.

[17] G. S. Stent, «A physiological mechanism for Hebb's postulate of learning,» *Proceedings of the National Academy of Sciences*, vol. 70, n. 4, pp. 997–1001, 1973.

[18] Intel Corporation. «Loihi 2 Technology Brief – Neuromorphic Computing.» Intel: technology brief; consultare la pagina per dettagli tecnici e riferimenti. (2024), indirizzo: https://www.intel.com/content/www/us/en/research/neuromorphic-computing-loihi-2-technology-brief.html.

[19] R. Guyonneau, R. Van Rullen e S. J. Thorpe, «Neurons' race: Very rapid encoding of natural scenes in simulation and in vivo,» *Vision Research*, vol. 46, n. 24, pp. 4021–4035, 2006.

[20] R. Van Rullen e S. J. Thorpe, «Temporal and rate coding in the visual cortex: What the arguments tell us,» *Trends in Neurosciences*, vol. 24, n. 1, pp. 40–45, 2001.

[21] E. M. Izhikevich, «Simple model of spiking neurons,» *IEEE Transactions on Neural Networks*, vol. 14, n. 6, 2003. indirizzo: https://www.izhikevich.org/publications/spikes.htm.

[22] E. O. Neftci, H. Mostafa e F. Zenke, «Surrogate Gradient Learning in Spiking Neural Networks,» *IEEE Signal Processing Magazine*, vol. 36, n. 6, pp. 61–63, 2019.

[23] J. M. Allred e Others, «Controlled Forgetting: Targeted Stimulation and Dopaminergic Plasticity Modulation for Unsupervised Lifelong Learning in Spiking Neural Networks,» *Frontiers in Neuroscience*, vol. 14, p. 7, 2020.

[24] H. Shin, J. K. Lee, J. Kim e R. Socher, «Continual Learning with Deep Generative Replay,» in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 2990–2999.

[25] J. Snoek, H. Larochelle e R. P. Adams, «Practical Bayesian Optimization of Machine Learning Algorithms,» in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 2951–2959.

[26] J. Bergstra e Y. Bengio, «Random Search for Hyper-Parameter Optimization,» *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[27] I. K. Schuller, R. Stevens, R. Pino e M. Pechan, «Neuromorphic computing–from materials research to systems architecture roundtable,» USDOE Office of Science (SC)(United States), rapp. tecn., 2015.

[28] Brian2 developers. «Brian2 documentation — Stable version.» Documentazione ufficiale del simulatore Brian2; consultare la pagina per esempi, API e guide. (2025), indirizzo: `https://brian2.readthedocs.io/en/stable/`.

[29] W. Maass, «Networks of spiking neurons: the third generation of neural network models,» *Neural networks*, vol. 10, n. 9, pp. 1659–1671, 1997.

[30] M. Pfeiffer e T. Pfeil, «Deep learning with spiking neurons: Opportunities and challenges,» *Frontiers in neuroscience*, vol. 12, p. 409 662, 2018.