POLITECNICO DI TORINO

Facoltà di Ingegneria Elettronica
Corso di Laurea in Ingegneria

Tesi di Laurea

Progetto di un convertitore Analogico-Informativo basato su Memristori

Relatori:
Prof. Gianluca Setti
Prof. Fabio Pareschi
Prof. Fernando Corinto

Candidato:
Fabrizio Congia

Marzo 2019

Ringraziamenti

Ringrazio i miei genitori Caterina e Giuseppe e mio fratello Valentino per essere la migliore famiglia che si possa desiderare. Il loro amore incondizionato, la loro pazienza e il loro supporto sono sempre stati la mia sicurezza. Ringrazio Stefania, la mia compagna, per la felicità che mi regala in ogni momento. Ringrazio i miei relatori, per la loro gentilezza e disponibilità durante questo lavoro. Un ultimo ringraziamento va ai miei amici, vecchi e nuovi, impossibile nominarli tutti.

Sommario

L'acquisizione di dati è un processo fondamentale nel mondo della tecnologia odierna. Il *Compressed Sensing* (CS) rappresenta un nuovo paradigma dell'acquisizione e processamento dei segnali che apre nuove opportunità e ha generato un notevole interesse in moltissimi campi di studio.
L'obiettivo di questa tesi è proporre un circuito che permetta di sfruttare il CS per acquisire segnali sparsi con un occhio all'efficienza energetica. Ne deriva un circuito di conversione analogico-informazione a basso consumo attraverso l'uso di memristori, che prende spunto da circuiti specifici per l'inferenza nelle reti neurali profonde.

Nel primo capitolo saranno presentati gli argomenti teorici a fondamento di quanto proposto. Dapprima si presenterà il CS, specificando quali sono le classi di segnali in cui è applicabile e quali sono i teoremi che ne garantiscono il funzionamento, mostrando infine alcune applicazioni pratiche attualmente in uso.
Nello stesso capitolo sarà presentato un breve riepilogo sulle reti neurali, per arrivare alle motivazioni che hanno spinto all'accostamento di queste e il CS. Per chiudere la parte teorica, si vedrà il Memristore, dispositivo nuovo e fondamentale nel funzionamento del circuito proposto, senza dimenticare lo stato dell'arte di tale dispositivo.

Il secondo capitolo esporrà il lavoro pratico svolto in questa tesi nei dettagli, dallo studio del Memristore alla simulazione finale del circuito proposto in un contesto tipico del CS, con relativa ricostruzione del segnale acquisito ed analisi dei risultati.

Indice

Capitolo 1

Teoria

Da qualche anno a questa parte il Compressed Sensing (CS) sta catturando l'attenzione di diversi campi della comunità scientifica come Signal processing, Statistica ed Elettronica. Sin dal suo sviluppo iniziale si stanno moltiplicando le pubblicazioni, così come le conferenze e i workshop sull'argomento.
Nel prossimo capitolo introdurremo le basi per la comprensione della teoria del CS, per poi proporre un'applicazione pratica nel capitolo successivo.

1.1 Compressed Sensing

Nell'epoca della rivoluzione digitale una grossa fetta dell'attenzione dell'ingegneria elettronica si è concentrata nella progettazione di sensori che possano permetterci di acquisire la realtà e ricostruirla con fedeltà e risoluzione sempre maggiori.
Con l'avvento di nuovi paradigmi come l'intelligenza artificiale o l'internet delle cose, le specifiche richieste ai sensori possono prendere strade diverse ma questi rimangono il punto di inizio di ogni sistema che interagisce con il mondo analogico.

Il lavoro di pionieri come Kotelnikov, Nyquist, Shannon e Whittaker ha gettato le basi teoriche di questa rivoluzione, dimostrando che un segnale può essere ricostruito esattamente a partire da un insieme di campioni acquisiti uniformemente nel tempo alla cosiddetta frequenza di Nyquist, ovvero il doppio della più alta frequenza del segnale che si vuole ricostruire.

La digitalizzazione che ne è conseguita ha permesso un'elaborazione dei dati in modo più semplice, veloce, economico e robusto rispetto alla controparte analogica. La quantità di dati che si è iniziata a generare è aumentata esponenzialmente mettendo a dura prova i progressi tecnologici nei campi associati al trattamento di tali dati (potenza di calcolo, velocità di acquisizione ed elaborazione, capacità di memorizzazione).

Per gestire tale mole di dati si sono dovuti cercare modi per rappresentare i segnali nel modo più conciso possibile, aprendo le porte allo sviluppo di tecniche di compressione dati. Una delle più popolari, nota come *transform coding*, si basa sul trovare una base nella quale il segnale in questione sia *sparso* o *comprimibile*. Il significato di questi termini verrà affrontato nella prossima sezione. Tale processo è chiamato *sparse approximation* ed è alla base di formati di compressione come il JPEG e l'mp3.

Sull'onda di questo concetto, nel caso in cui il segnale sia rappresentabile in forma sparsa e comprimibile, il *Compressed Sensing* cerca di sfruttare queste caratteristiche direttamente nel passo dell'acquisizione e nel design dei sensori. Piuttosto che campionare ad alta frequenza e poi comprimere il dato campionato, sarebbe meglio acquisire direttamente il segnale in una forma compressa.
Il lavoro di Candès, Romberg, Tao e Donoho [1] [2] ha dimostrato infatti come sia possibile ricostruire un segnale di dimensioni finite, con una rappresentazione sparsa e comprimibile, a partire da un piccolo insieme di misure lineari e non adattive.

Ci sono tre principali differenze tra il CS e la classica teoria del campionamento. La prima riguarda il tipo di segnale da acquisire: se la teoria classica tratta tipicamente segnali considerati di lunghezza infinita e continui nel tempo, il CS è un metodo matematico che si concentra su vettori di dimensioni finite in $\mathbb{R}^n$.
Come secondo punto, il CS solitamente acquisisce le misure come prodotto scalare tra il segnale e delle funzioni di test, piuttosto che campionare il segnale in un momento preciso.
Infine c'è una sostanziale differenza nel metodo di ricostruzione del segnale dalle misure compresse: la teoria classica ricompone il segnale attraverso una sinc interpolation che richiede una computazione limitata, il CS usa metodi altamente non lineari.

Il Compressed Sensing ha già dato al giorno d'oggi risultati apprezzabili in diversi campi di applicazione. Nel campo medico ha ad esempio permesso la velocizzazione della risonanza magnetica senza far perdere informazioni utili alla diagnosi.
Inoltre sono state avviate ricerche applicate a diversi campi pratici, come sistemi di campionamento sub-Nyquist, architetture di compressione immagini, o reti di sensori compresse.

1.1.1 Elementi di spazi vettoriali

Il *signal processing* si è da sempre focalizzato principalmente nei segnali prodotti da sistemi fisici. Molti di questi possono essere convenientemente modellati come lineari ed il naturale passo successivo è la rappresentazione tramite *vettori* esistenti nell'appropriato *spazio vettoriale*. Ciò permette di applicare ai segnali un insieme di regole proprie della geometria come lunghezza, distanze e angoli per lavorare

e descrivere i segnali d'interesse. Per questo motivo rivedremo brevemente alcuni concetti chiave necessari alla comprensione del CS.

Spazio vettoriale normato

Un segnale finito e discreto può essere visto come un vettore nello spazio Euclideo n-dimensionale ($\mathbb{R}^n$). Al suo interno possiamo definire il concetto di norma ℓ_p per $p \in [0, \infty]$ come:

$$\|x\|_p = \begin{cases} \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}, & p \in [1, \infty); \\ \max\limits_{i=1,2,\ldots,n} |x_i|, & p = \infty. \end{cases} \tag{1.1}$$

È utile ricordare il prodotto interno in $\mathbb{R}^n$ definito come:

$$\langle x, z \rangle = z^T x = \sum_{i=1}^{n} x_i z_i$$

che nel caso della norma ℓ_2 porta a: $\|x\|_2 = \sqrt{\langle x, x \rangle}$. Se $p < 1$, la norma nell'equazione 1.1 è detta quasi-norma in quanto non soddisfa la disuguaglianza triangolare. Inoltre, nonostante questa non sia nemmeno una quasi-norma, si userà la notazione $\|x\|_0 := |\text{supp}(x)|$ dove $\text{supp}(x) = \{i : x_i \neq 0\}$ indica il supporto di x e $|\text{supp}(x)|$ ne costituisce la cardinalità.
Le norme vengono utilizzate tipicamente per misurare la forza di un segnale o la dimensione di un errore. Al variare di p le loro proprietà variano notevolmente.

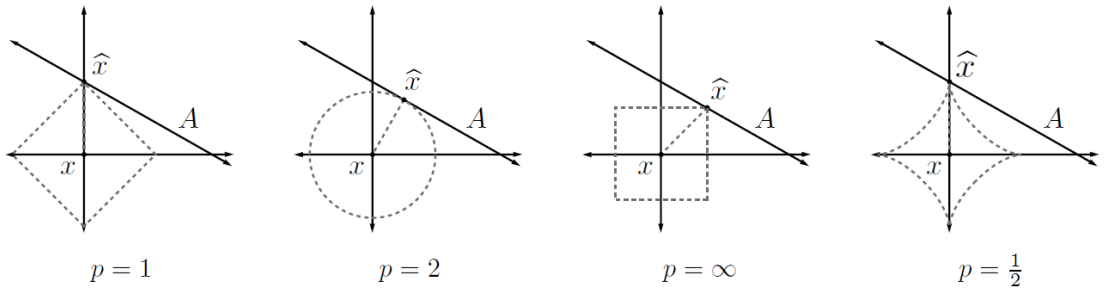


Figura 1.1: Approssimazione di un punto in $\mathbb{R}^2$ in uno sotto-spazio mono-dimensionale con l'uso di (quasi)norma ℓ_p per $p = 1, 2, \infty, \frac{1}{2}$[3]

Si prenda per esempio l'approssimazione di un punto in $\mathbb{R}^2$ su uno spazio affine mono-dimensionale A. Lo scopo è trovare $\hat{x} \in A$ tale che l'errore $\|x - \hat{x}\|_p$ sia minimo. Nella figura 1.1 si può notare come la scelta di p sia fondamentale nel risultante errore di approssimazione e che a un minore p corrisponda un errore distribuito più disomogeneamente e più sparso. Questo intuizione ha giocato un ruolo importante nello sviluppo della teoria del CS.

Basi e *frames*

Un insieme di vettori $\{\phi_i\}_{i=1}^n$ è definito una base di $\mathbb{R}^n$ se i vettori che lo compongono generano lo spazio $\mathbb{R}^n$ e sono linearmente indipendenti. Se tali vettori sono ortogonali fra loro e di modulo unitario, si parla di base ortonormale.
Si può generalizzare il concetto di base permettendo nell'insieme $\{\phi_i\}$ anche dei vettori non linearmente indipendenti. In questo caso si parla di *frame*, formalmente definito come insieme di vettori $\{\phi_i\}_{i=1}^d$ in $\mathbb{R}^d$, $d < n$ corrispondente ad una matrice $\Phi \in \mathbb{R}^{d \times n}$, tale che per ogni vettore $x \in \mathbb{R}^d$ si abbia

$$A\|x\|_2^2 \leq \|\Phi^T x\|_2^2 \leq B\|x\|_2^2$$

con $0 < A \leq B < \infty$. I *frame* possono fornire una più ricca rappresentazione di un segnale grazie alla loro ridondanza: data un segnale x esistono molteplici insiemi di coefficienti c che possono rappresentare lo stesso segnale. Nella letteratura del CS, le basi e i frame sono spesso chiamati *dictionary* o *overcomplete dictionary* i cui elementi sono detti *atoms*.

1.1.2 Rappresentazione sparsa

La possibilità di rappresentare un segnale attraverso modelli che sfruttino conoscenze a priori permette di distinguere caratteristiche importanti da altre superflue e migliora l'efficienza nell'acquisizione, il processing, la compressione e comunicazione di dati e informazioni.
Nonostante sia ragionevole usare i vettori come modello di rappresentazione, in molti casi non tutti i vettori vanno poi a rappresentare segnali validi. In risposta all'esplosione della dimensionalità dei dati, è aumentato l'interesse verso una rappresentazione in modelli di segnali a bassa dimensione. Si sfrutta la consapevolezza che il grado di libertà dei segnali ad alta dimensione è in realtà piuttosto bassa rispetto alla loro dimensionalità ambientale.
Un segnale può spesso essere approssimato con una combinazione lineare di pochi elementi del suo dizionario. Quando questa rappresentazione è esatta si dice che il segnale è *sparso*.
Formalmente, un segnale x è *k-sparse* quando ha al massimo k elementi diversi da 0. Si ha spesso a che fare con segnali che non sono sparsi ma ammettono una rappresentazione sparsa in una qualche base Φ. In questo caso ci si riferisce a x come *k-sparse* sottintendendo che si può esprimere x come $x = \Phi c$ dove $\|c\|_0 \leq k$.
Un esempio tradizionale dell'utilizzo di queste conoscenze può essere osservato nella compressione di immagini in figura 1.2.
Le immagini naturali sono spesso caratterizzate da ampie textures regolari e relativamente pochi margini netti. Questo tipo di struttura è conosciuta per essere molto vicina alla sparsità quando rappresentata con trasformata wavelet. Quest'ultima

(a) (b)

Figura 1.2: Approssimazione sparsa di un immagine naturale. (a) Immagine originale. (b) Approssimazione ottenuta tenendo esclusivamente i 10% maggiori coefficienti delle wavelet[4].

consiste nella divisione ricorsiva in componenti a bassa e alta frequenza. Le frequenze più basse danno una rappresentazione generale dell'immagine mentre le alte frequenze contengono i dettagli. Molti coefficienti delle wavelet sono vicini allo zero e impostandoli a 0 si ottiene una buona approssimazione dell'immagine nonostante la compressione applicata.
Si tratta di un'approssimazione non lineare in quanto i coefficienti da tenere sono determinati dal segnale stesso. La conseguenza di ciò è che la combinazione lineare di due segnali k-sparsi non sarà più k-sparsa dato che il supporto potrebbe non coincidere. Si può scrivere che, dati $x, z \in \Sigma_k$, non si avrà necessariamente $x + z \in \Sigma_k$ ma sarà sempre vero che $x + z \in \Sigma_{2k}$.
È importante sottolineare come pochi segnali nel mondo reale siano in verità sparsi, ma possono essere ben approssimati da segnali sparsi, chiamandosi in questo caso *comprimibili*, o *approssimativamente sparsi*. Calcolare l'errore dell'approssimazione a $\hat{x} \in \Sigma_k$ del segnale x permette di quantificarne la comprimibilità:

$$\sigma_k(x)_p = \min_{\hat{x} \in \Sigma_k} \|x - \hat{x}\|_p \tag{1.2}$$

1.1.3 Matrice di Sensing

Si restringa l'attenzione al modello CS a dimensioni finite. Dato un segnale $x \in \mathbb{R}^n$, consideriamo un sistema di acquisizione che effettui m misure lineari. Si può rappresentare il processo come:

$$y = Ax,$$

dove A è una matrice $m \times n$ e $y \in \mathbb{R}^m$. La matrice A rappresenta una riduzione dimensionale: associa $\mathbb{R}^n$, in cui n è solitamente grande, a $\mathbb{R}^m$, in cui m è tipicamente

più piccolo di n.
Nel modello CS standard le misure non sono adattive, non dipendono cioè dalle misure precedenti. Esiste però la possibilità di utilizzare un sistema adattivo che in certi scenari porta notevoli vantaggi. Inoltre in questo modello x è un vettore di lunghezza finita con dei valori discreti, ma si può estendere attraverso una rappresentazione discreta intermedia di segnali in realtà continui.
Due domande sorgono spontanee: come si dovrebbe progettare la matrice di sensing A per assicurarsi che preservi le informazioni contenute nel segnale x? Come si può ricostruire il segnale originale x partendo dalle misure y?
Nel caso il segnale x sia sparso o comprimibile, si vedrà che è possibile progettare una matrice A con $m \ll n$ che permetta di ricostruire accuratamente il segnale originale usando una varietà di algoritmi.

Condizioni di spazio nullo

Consideriamo lo spazio nullo della matrice A:

$$\mathcal{N}(A) = \{z : Az = 0\}.$$

Al fine di ricostruire tutti i segnali sparsi x dalle misurazioni Ax è necessario che, presi due vettori distinti x e $x' \in \Sigma_k$, si debba ottenere $Ax \neq Ax'$. Diversamente sarebbe impossibile distinguere x da x' basandosi solamente sulle misure. Formalmente si può dire che, se $Ax = Ax'$ allora $A(x - x') = 0$ con $x - x' \in \Sigma_{2k}$, A rappresenta unicamente tutte le $x \in \Sigma_k$ se e solo se $\mathcal{N}(A)$ non ha vettori in Σ_{2k}.
Questa proprietà può essere caratterizzata introducendo lo *spark*:

Definizione 1 *Si definisce* spark *di una matrice A il più piccolo numero di colonne di A linearmente dipendenti.*

Teorema 1 *Per ogni vettore $y \in \mathbb{R}^m$ esiste al massimo un segnale $x \in \Sigma_k$ tale che $y = Ax$ se e solo se spark$(A) > 2k$.*

Il teorema 1 fornisce classificazione completa di quando il segnale sparso è ricostruibile, nel caso si stia trattando con vettori *esattamente* sparsi.
Se si lavora con segnali approssimativamente sparsi sono necessarie condizioni più stringenti. Per dare una definizione formale, si parta dalla notazione $\Lambda \subset \{1, 2, ..., n\}$ rappresentante un sottoinsieme di indici e sia $\Lambda^c = \{1, 2, ..., n\} \setminus \Lambda$. Con x_Λ si intenda il vettore di lunghezze n ottenuto impostando a 0 gli elementi di x indicizzati da Λ^c. Allo stesso modo, si intenda con A_Λ la matrice $m \times n$ ottenuta impostando a 0 le colonne di A indicizzate da Λ^c.

Definizione 2 *Una matrice A soddisfa la* null space property *(NSP) di ordine k se esiste una costante C>0 tale che*

$$\|h_\Lambda\|_2 \leq C\frac{\|h_{\Lambda^c}\|_1}{\sqrt{k}}$$

sia valida per qualunque $h \in \mathcal{N}(A)$ *e per tutte le* Λ *tali che* $|\Lambda| \leq k$.

Dunque se un vettore h è esattamente k-sparso esiste una Λ tale che $\|h_{\Lambda^c}\|_1 = 0$ e dunque $h_\Lambda = 0$. Di conseguenza se una matrice A soddisfa la NSP, l'unico vettore k-sparso in $\mathcal{N}(A)$ è $h = 0$. Per spiegare le conseguenze della NSP, sia $\Delta : \mathbb{R}^m \to \mathbb{R}^n$ che rappresenta un metodo di ricostruzione specifico. Se

$$\|\Delta(Ax) - x\|_2 \leq C\frac{\sigma_k(x)_1}{\sqrt{k}} \tag{1.3}$$

per qualsiasi x, dove $\sigma_k(x)_1$ è definita nell'equazione 1.2. Questo garantisce l'esatta ricostruzione di tutti i segnali k-sparsi, ma assicura anche un grado di robustezza per segnali non sparsi che dipende da quanto questi siano ben approssimati da vettori k-sparsi. Si noti che la scelta della norma ℓ_p è arbitraria nell'equazione 1.3.

Teorema 2 *Sia* $A : \mathbb{R}^n \to \mathbb{R}^m$ *la matrice di sensing e* $\Delta : \mathbb{R}^m \to \mathbb{R}^n$ *l'algoritmo di ricostruzione arbitrario. Se la coppia* (A, Δ) *soddisfa la 1.3 allora A soddisfa la NSP di ordine 2k.*

Proprietà di isometria ristretta

Sebbene la NSP sia necessaria e sufficiente per garantire i risultati nella forma della 1.3, tali garanzie non tengono conto del rumore. Sono necessarie condizioni più stringenti nel caso in cui la numerazione sia in qualche modo corrotta da errori come la quantizzazione. Candès e Tao hanno introdotto la seguente condizione di isometria:

Definizione 3 *Una matrice A soddisfa la proprietà di isometria ristretta (RIP) di ordine* k *se esiste una* $\sigma_k \in (0, 1)$ *tale che*

$$(1 - \sigma_k)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1 + \sigma_k)\|x\|_2^2 \tag{1.4}$$

sia valida per qualunque $x \in \Sigma_k$.

Se una matrice soddisfa la RIP di ordine $2k$, si può interpretare la 1.4 dicendo che A preserva approssimativamente la distanza di ogni paio di vettori k-sparsi. Questo ha implicazioni fondamentali nella robustezza al rumore e permette di applicare una

vasta gamma di algoritmi per la ricostruzione del segnale.
Diversi studi sulla correlazione tra le dimensioni del problema (n, m e k) hanno portato a fissare dei limiti sul numero di misure necessarie affinché la matrice A rispetti la RIP di ordine k[4][5][6].
Si può inoltre dimostrare la relazione tra RIP e NSP:

Teorema 3 *Si supponga che A soddisfi la RIP di ordine 2k con* $\sigma_{2k} < \sqrt{2} - 1$. *Allora A soddisfa la NSP di ordine 2k con la costante*

$$C = \frac{2}{1 - (1 + \sqrt{2})\sigma_{2k}}.$$

Coerenza

Le proprietà *spark*, NSP e RIP forniscono garanzie sulla possibilità di ricostruzione di segnali k-sparsi, ma verificare che una matrice le soddisfi richiede una ricerca su tutte le $\binom{n}{k}$ sotto-matrici. In molti casi si preferisce ricorrere ad una proprietà facilmente calcolabile chiamata coerenza.

Definizione 4 *La coerenza di una matrice A,* $\mu(A)$, *è il maggior prodotto scalare fra due colonne* a_i,a_j *di A:*

$$\mu(A) = \max_{1 \leq i < j \leq n} \frac{|\langle a_i, a_j \rangle|}{\|a_i\|_2 \|a_j\|_2}.$$

I limiti della coerenza di una matrice rientrano sempre nell'intervallo $\mu(A) \in [\sqrt{\frac{n-m}{m(n-1)}}, 1]$. Per $n \gg m$ il limite inferiore si può semplificare a $\mu(A) \geq \frac{1}{\sqrt{m}}$.
La coerenza si può talvolta mettere in relazione a *spark*, NSP e RIP. Per esempio è dimostrato che:

$$spark(A) \geq 1 + \frac{1}{\mu(A)}.$$

Costruzione della matrice di sensing

Si sono definite le proprietà di una matrice riguardanti il CS ed esistono diversi modi di costruire delle matrici che le rispettino. Tuttavia molti metodi portano problemi numericamente instabili[7] o richiedono un numero di misure m che può diventare spropositato[8][9].
Queste limitazioni possono però essere superate randomizzando la costruzione della matrice. Per esempio, matrici random A di dimensione $m \times n$ i cui elementi sono indipendenti e identicamente distribuiti con distribuzioni continue hanno uno

$spark(A) = m + 1$ con probabilità certa. Ancora più importante è il fatto che una matrice random soddisferà la RIP con alta probabilità se gli elementi sono scelti con distribuzioni gaussiane, di Bernoulli o più genericamente sub-gaussiane.

Teorema 4 (Sub-gaussian restricted isometries) *Sia A una matrice $m \times n$ di distribuzione random sub-gaussiana con righe e colonne indipendenti e*

$$m = O\left(k \log \frac{\binom{n}{k}}{\sigma_{2k}^2} \right).$$

Allora A soddisferà la RIP di ordine $2k$ con probabilità di almeno

$$1 - 2\exp(-c_1 \sigma_{2k}^2 m).$$

Dal teorema 3 segue che queste matrici costruite con distribuzioni randomiche rispettano anche la NSP. Inoltre può essere dimostrato che se la distribuzione usata ha media nulla e varianza finita, la coerenza converge a $\mu(A) = \sqrt{(2\log n)/m}$ al crescere di m e n.
L'utilizzo di matrici random porta altri benefici, i cui principali sono la *democraticità* e l'*universalità*. Il primo indica la possibilità di ricostruire il segnale anche da sottoinsiemi di m abbastanza grandi, aumentando la robustezza alla perdita o la corruzione di un numero limitato di misure. Il secondo intende la proprietà di una matrice random di rispettare la RIP con alta probabilità se m è abbastanza ampio. Ciò vale sia per distribuzioni gaussiane che per sub-gaussiane.
È inoltre importante sottolineare come si riesca ad ottenere matrici che soddisfino la RIP e abbiano una bassa coerenza nel caso la costruzione non sia completamente random ma abbia delle restrizioni aggiuntive. Questo è infatti il caso dell'implementazione delle matrici in hardware come avviene in questa tesi.

1.1.4 Ricostruzione del segnale

Esistono diversi approcci alla ricostruzione del segnale x. Molti di questi sono precedenti al CS e sono utilizzati anche in altre applicazioni come statistica e geofisica. Si presentano brevemente alcuni di questi.

Algoritmo di minimizzazione ℓ_1

Il primo e naturale approccio al problema della ricostruzione di un segnale sparso è l'ottimizzazione di un problema nella forma:

$$\hat{x} = \arg\min_z \|z\|_0 \quad \text{soggetto a} \quad z \in \mathcal{B}(y), \tag{1.5}$$

in cui $\mathcal{B}(y)$ assicura che $\hat{x}$ sia coerente con le misure y. Nel caso le misure siano esatte e prive di rumore si può porre $\mathcal{B}(y) = \{z : Az = y\}$.
Nel caso ci sia del rumore si può scrivere invece $\mathcal{B}(y) = \{z : \|Az - y\|_2 \leq \epsilon\}$ e in entrambi i casi la 1.5 le x più sparse in accordo con le misure y.
In entrambi i casi però viene intrinsecamente supposto che x sia sparso. Nel caso più generico in cui $x = \Sigma c$, si può scrivere:

$$\hat{c} = \arg\min_z \|z\|_0 \quad \text{soggetto a} \quad z \in \mathcal{B}(y), \tag{1.6}$$

in cui $\mathcal{B}(y) = \{z : A\Phi z = y\}$ o $\mathcal{B}(y) = \{z : \|A\Phi z - y\|_2 \leq \epsilon\}$. Adottando la notazione $\tilde{A} = A\Phi$ la 1.5 e 1.6 sono essenzialmente identiche e l'introduzione di Φ non complica significativamente la costruzione della matrice A. Tuttavia la funzione obbiettivo $\|\cdot\|_0$ è non convessa e diventa difficile da calcolare. Per risolvere questo problema si sostituisce la norma ℓ_0 con la ℓ_1. Nello specifico si considera il problema:

$$\hat{x} = \arg\min_z \|z\|_1 \quad \text{soggetto a} \quad z \in \mathcal{B}(y), \tag{1.7}$$

la quale, se $\mathcal{B}(y)$ è convessa, diventa facilmente calcolabile. Quando $\mathcal{B}(y) = \{z : Az = y\}$, la 1.7 può essere posta come programma lineare, mentre con $\mathcal{B}(y) = \{z : \|Az - y\|_2 \leq \epsilon\}$ diventa un programma convesso con limite conico.
Nel tempo è stata creata una grande varietà di risolutori sia generici che specifici del CS per questo tipo di problema.

Algoritmo *greedy*

L'algoritmo *greedy* applica un approssimazione iterativa dei coefficienti del segnale e del supporto fino a che non viene riscontrato un criterio di convergenza. È stato dimostrato che alcuni metodi *greedy* hanno le stesse performance sui risultati garantiti del metodo precedente esposto della minimizzazione ℓ_1.
Due dei più vecchi e semplici algoritmi di questa categoria sono l'*Orthogonal Matching Pursuit* (OMP) e l'*Iterative Hard Thresholding*(IHT). L'OMP[10] inizia trovando la colonna di A più correlata alle misure. Il passo è poi ripetuto correlando le colonne con il residuo del segnale, ottenuto sottraendo la stima parziale al vettore di misure originale. Il criterio di interruzione può essere il numero di iterazioni o la convergenza di y a valori prossimi a $A\hat{x}$.
L'IHT[11] inizia da una stima $\hat{x} = 0$, per poi iterare un gradino a gradiente discendente seguito da un *hard thresholding* fino a che non viene riscontrato il criterio di interruzione.

Algorithm 1.1 Orthogonal Matching Pursuit

Inputs: CS matrix/dictionary A, measurement vector y
Initialize: $\widehat{x}_0 = 0$, $r_0 = y$, $\Lambda_0 = \emptyset$.
for $i = 1$; $i := i + 1$ until stopping criterion is met **do**
  $g_i \leftarrow A^T r_{i-1}$ {form signal estimate from residual}
  $\Lambda_i \leftarrow \Lambda_{i-1} \cup \text{supp}(H_1(g_i))$ {add largest residual entry to support}
  $\widehat{x}_i|_{\Lambda_i} \leftarrow A^{\dagger}_{\Lambda_i} y$, $\widehat{x}_i|_{\Lambda_i^c} \leftarrow 0$ {update signal estimate}
  $r_i \leftarrow y - A\widehat{x}_i$ {update measurement residual}
end for
Output: Sparse representation $\widehat{x}$

Algorithm 1.2 Iterative Hard Thresholding

Inputs: CS matrix/dictionary A, measurement vector y, sparsity level k
Initialize: $\widehat{x}_0 = 0$.
for $i = 1$; $i := i + 1$ until stopping criterion is met **do**
  $\widehat{x}_i = H_k\left(\widehat{x}_{i-1} + A^T(y - A\widehat{x}_{i-1})\right)$
end for
Output: Sparse representation $\widehat{x}$

Algoritmo combinatorio

In aggiunta alle due classi presentate, ne esiste una terza di algoritmi combinatori. Si tratta di algoritmi inizialmente sviluppati per un altro tipo di problemi (principalmente *theoretical computer science*) ma divenuta rilevante per il CS. Si tratta della ricerca di una soluzione ottimale da un insieme incompleto di misure. In questo tipo di problemi una ricerca esaustiva non è possibile. Pur ponendo condizioni più stringenti sulla struttura della matrice di sampling A, questo tipo di algoritmi può rivelarsi più veloce quando applicabile.

Implementazioni di ricostruttori

Sono disponibili all'uso diversi ricostruttori di carattere commerciale e non che sfruttano gli algoritmi presentati. Tra questi si può citare SPGL1[12] (utilizzato in questa tesi) e NESTA[13].
Tra i ricostruttori che sfruttano la minimizzazione ℓ_1 ci sono CVX[14] e unlocbox[15], mentre tra i ricostruttori euristici si trovano i già citati OMP e IHT.

Valutazione dei risultati

La valutazione della qualità della ricostruzione richiede l'introduzione di figure di merito la cui prima e più naturale è:

$$\mathrm{RSNR[dB]} = 20\log_{10}\left(\frac{\|x\|_2}{\|\hat{x}-x\|_2}\right),$$

definita rapporto segnale-rumore di ricostruzione. Il suo significato è intuitivo, ad un maggiore RSNR corrisponde una ricostruzione migliore.
Ma non ci si deve dimenticare che si sta operando con matrici random, e dunque l'entità dell'errore può variare ad ogni iterazione. Per ovviare a questo problema si usa spesso il metodo Montecarlo per poter valutare un valore medio del RSNR.

$$\mathrm{ARSNR[dB]} = \mathbf{E}\left[20\log_{10}\left(\frac{\|x\|_2}{\|\hat{x}-x\|_2}\right)\right] \approx \frac{1}{W}\sum_{j=0}^{W-1} 20\log_{10}\left(\frac{\|x^{(j)}\|_2}{\|\hat{x}^{(j)}-x^{(j)}\|_2}\right)$$

In alternativa si può ritenere che la ricostruzione è corretta se la RSNR[dB] supera una $\mathrm{RSNR[dB]}_{\min}$ minima e si definisce Probabilità di ricostruzione corretta (PCR) come:

$$\mathrm{PCR} = \Pr\{\mathrm{RSNR[dB]} \geq \mathrm{RSNR[dB]}_{\min}\} \approx \frac{\left|\left\{\frac{\|x^{(j)}\|_2}{\|\hat{x}^{(j)}-x^{(j)}\|_2} \geq 10^{\frac{\mathrm{RSNR[dB]}_{\min}}{20}}\right\}\right|}{W}$$

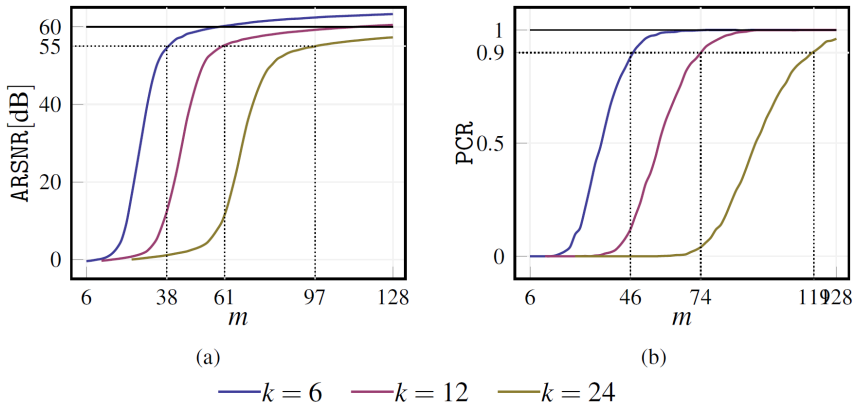


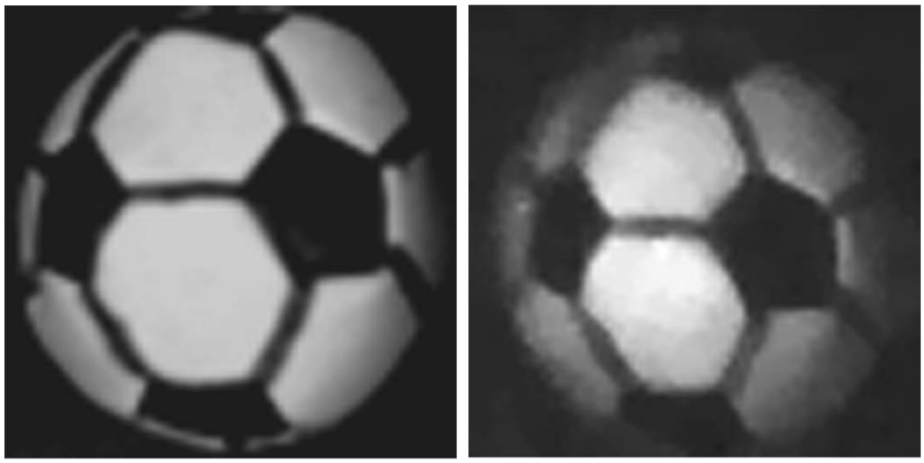
Figura 1.3: Risultati da un'analisi Montecarlo di un sistema CS classico: all'aumentare di misure sia ARSNR(a) che PCR(b) aumentano, con una dipendenza dalla sparsità k[16]

1.1.5 Stato dell'Arte

L'utilizzo delle tecniche di Compressed Sensing si sta diffondendo in diversi campi del *signal processing* e matematica computazionale. Un elenco parziale delle sue applicazioni include fotografia[17], riconoscimento facciale[18], acquisizione di segnali biomedici[19], caratterizzazione di antenne[20].

Fotografia

L'esempio più classico del CS nella fotografia è la *single-pixel camera*. Come si può capire dal nome, questa camera sfrutta un singolo fotodiodo per acquisire le informazioni sulla luminosità che normalmente vengono recepite da una matrice di milioni di sensori (o pixel).
La SPC funziona facendo riflettere l'immagine da un *digital micromirror device* (DMD) attraverso una lente fino al fotodiodo. Il DMD è una matrice di specchi microscopici la cui angolazione può essere modificata fino a $\pm 15^{\circ}$. Ad ogni acquisizione gli specchi sono mossi in modo tale che solo alcuni focalizzino parte dell'immagine sul fotodiodo in un dato istante. Applicare questo comportamento m volte è l'equivalente di costruire la matrice di Sampling di un sistema CS. È possibile ricostruire l'immagine intera attraverso gli algoritmi discussi nella sezione 1.1.4.
Questo metodo richiede l'acquisizione dell'immagine in sequenza rendendo l'operazione lenta, ma propone un nuovo modo di acquisizione dove la velocità non è fondamentale e sono già stati avviati studi per affrontare questo problema[17].



(a) Fotografia convenzionale 64x64 (b) Fotografia da single-pixel camera

Figura 1.4: Una fotografia scattata da una *single-pixel camera* costruita da Richard Baraniuk e Kevin Kelly della Rice University. L'immagine è derivata matematicamente da 1600 misure random[21]

Riconoscimento facciale

Trovare soluzioni di un sistema sotto-determinato è la caratteristica principale del CS. Il riconoscimento di una faccia nonostante il cambiamento di espressione, luminosità fino a camuffamento o coperture parziali del volto appare subito come un altro modo di porre lo stesso quesito. In figura 1.5 è riportato un esempio di metodi utilizzati in questo campo[18]. L'immagine test a sinistra, potenzialmente occlusa (a) o corrotta (b), è rappresentata come una combinazione lineare sparsa di tutte le immagini di training (al centro) più errori sparsi (destra). I coefficienti in rosso corrispondono all'immagine dell'individuo corretto. L'algoritmo determina l'identità (indicata da un riquadro rosso) da un set di 700 immagini di training di 100 individui diversi.

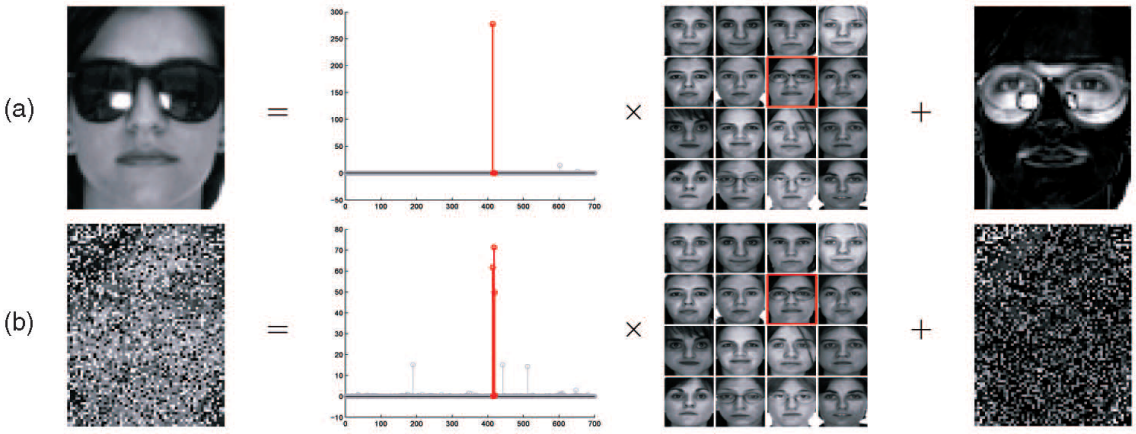


Figura 1.5: Approccio al problema del riconoscimento facciale

Il sistema riceve un set di immagini rappresentato da una matrice in cui ogni riga è derivata dalla foto di un soggetto (per esempio concatenando le righe della sua foto). Questa matrice viene associata ad un vettore di indici che contiene l'informazione su quale soggetto è rappresentato in una determinata riga. Diversi studi indicano che le righe afferenti allo stesso soggetto risiederanno in un sottospazio a bassa dimensione.
Questo approccio è stato in seguito evoluto per tener conto di diversi allineamenti (volto non frontale) e tipi di illuminazione (naturale o artificiale, soleggiato o nuvoloso), e migliorare le performance in presenza dei problemi già considerati[22].

Risonanza magnetica e altri segnali medici

La velocità con cui si possono campionare dati nella risonanza magnetica è ridotto da limiti fisici e fisiologici. Spesso il criterio di Nyquist è violato e bisogna cercare un diverso modo di ricostruire il segnale senza far degradare troppo la qualità

dell'immagine. Per fortuna le immagini MR presentano un implicita sparsità che permette di applicare le tecniche del CS. In realtà molti segnali medici presentano sparsità, se non nel segnale in se, almeno in una sua qualche trasformata. Nell'immagine 1.6 troviamo un esempio di tomografia assiale di un cervello e un angiografia di una gamba che mostrano diversi gradi di sparsità a seconda della trasformata usata. Scegliendo la più adeguata all'occasione, la ricostruzione può avvenire con le tecniche già discusse nonostante il sotto-campionamento.

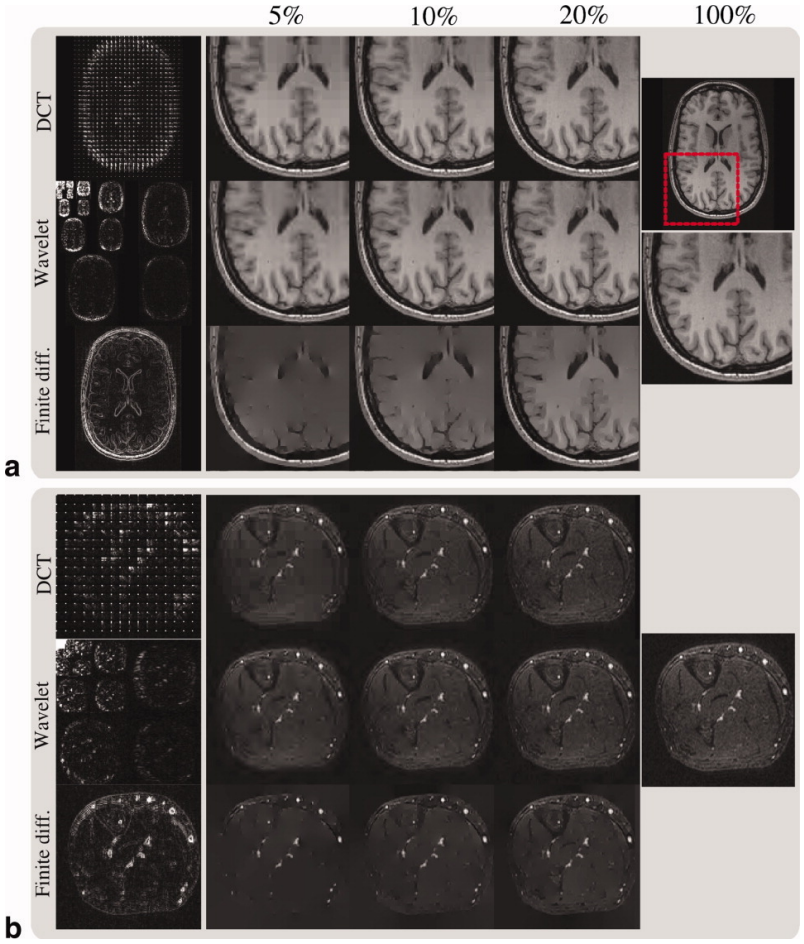


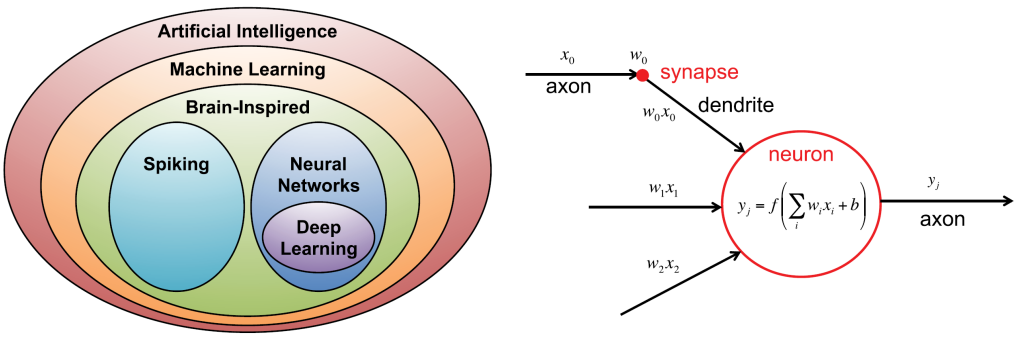
Figura 1.6: Ricostruzione di una TAC di un cervello (a) e Angiografia periferica ad alto contrasto di una sezione di gamba (b) utilizzando solo $n\%$ coefficienti maggiori per 3 diverse trasformate[19].

1.2 Reti neurali

In un era in cui la raccolta dati è enorme e pervasiva, l'intelligenza artificiale (in particolare le reti neurali) ha acquisito un'importanza fondamentale per l'estrazione di informazioni significative. Questa ci permette di analizzare e comprendere i dati per identificare comportamenti specifici (per esempio elettronica indossabile) o per prendere decisioni e agire (veicoli autonomi).
Il motivo che ci spinge ad analizzare questo campo di studi è l'ampio utilizzo di operazioni in comune con il Compressed Sensing e la quantità di studi che vengono ogni giorno pubblicati su come gestire al meglio questo carico computazionale.
La crescente complessità e lo sviluppo sia delle reti neurali che dei compiti che devono eseguire richiede sempre di più un hardware specifico dedicato da cui si potrebbe prendere spunto per l'utilizzo nel CS.

1.2.1 Intelligenza Artificiale e DNNs

Per Intelligenza artificiale (AI), termine coniato nel 1950 da John McCarthy, si intende il campo della scienza e l'ingegneria del creare macchine in grado di raggiungere determinati scopi come succede negli esseri umani[23].
All'interno dell'AI esiste una grossa branca chiamata Machine Learning, definita nel 1959 da Arthur Samuel come lo studio di dare ai computers la capacità di imparare senza essere esplicitamente programmati. Il vantaggio è chiaro: invece di creare un programma che preveda ogni evenienza in un problema, la macchina può imparare attraverso un processo chiamato *training* come gestire ogni problema.



(a) Deep Neural Networks nel panorama dell'Intelligenza Artificiale

(b) Connessioni di un neurone[24]

Figura 1.7: I concetti base di rete neurale

Dato che il cervello è al momento l'esempio migliore di "macchina" atta a imparare

e risolvere i problemi, una parte del Machine Learning (chiamata Brain-Inspired) tenta di emulare alcuni aspetti del modo in cui si pensa che il cervello funzioni.
L'elemento computazionale principale del cervello è chiamato *neurone*, che riceve segnali attraverso i cosiddetti *dendriti*, elabora e genera un segnale su un elemento uscente chiamato *assone*. I segnali in ingresso e uscita sono detti attivazioni. A loro volta gli assoni sono collegati a dendriti di altri neuroni attraverso delle *sinapsi*. Queste ultime si occupano di scalare il segnale x_i di un fattore chiamato peso w_i. Si presuppone che il cervello impari modificando i pesi associati alle sinapsi pur mantenendo la propria struttura. In figura 1.7b vi è lo schema di un neurone in cui x_i sono le attivazioni, w_i sono i pesi, f$(\cdot)$ è la funzione non lineare e b è il bias.
A questo punto possiamo distinguere fra Spiking Computing e Reti Neurali in cui i primi, a differenza delle seconde, ritengono che la computazione avvenga non solo in funzione dei segnali di attivazione ma anche del momento in cui essi arrivano.
Le Reti Neurali a loro volta portano il concetto di layer. I risultati non sono solo somme pesate, ma funzioni non lineari di tali somme. Mettendo in serie diversi strati neurali si è in grado di affrontare problemi più complessi con risultati più precisi. Per questo motivo se all'interno di una rete neurale troviamo più di un layer nascosto (nè di ingresso nè di uscita) si parla di Deep Neural Networks (DNNs).

I concetti di Inferenza e Training

In una rete neurale *imparare* implica il trovare il valore corretto di pesi e bias attraverso un processo chiamato *training*. Una volta terminato, la rete può eseguire il compito assegnato utilizzando i pesi ottenuti nel training. Utilizzare una rete con questi pesi è detto inferenza (figura 1.8).
Il concetto dietro il training presuppone trovare dei pesi che minimizzino la differenza media tra il risultato ideale e il risultato ottenuto dalla DNN. Può trattarsi di *supervised training* in cui alla rete neurale viene fornita insieme al set di training anche il risultato atteso, o *unsupervised training* nel quale la rete si limita a riconoscere strutture nell'insieme di dati. Esistono altri tipi di training che ricadono tra queste categorie e dipendono dal risultato che si vuole ottenere.
Il training richiede tipicamente un'enorme mole di dati e una notevole potenza computazionale e viene spesso eseguita sul cloud con tempi d'esecuzione che arrivano a diversi giorni.
In molte applicazioni invece è preferibile eseguire l'inferenza vicino al sensore. Nell'elaborazione di un immagine appena scattata da un sensore fotografico si possono ridurre i costi di trasmissione e memorizzazione; nella guida di un veicolo autonomo si possono ridurre i tempi di reazione. Molte piattaforme embedded inoltre hanno parametri molto stringenti per memoria, consumo energetico e potenza computazionale.

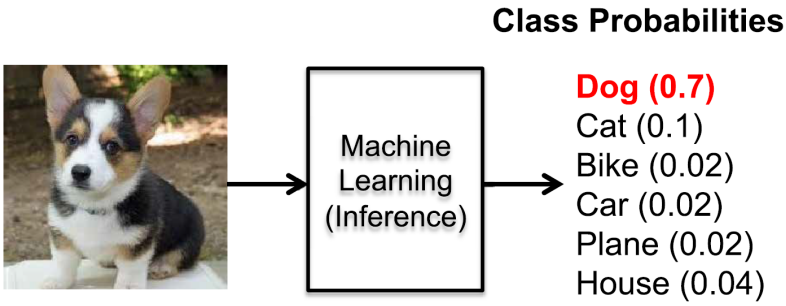
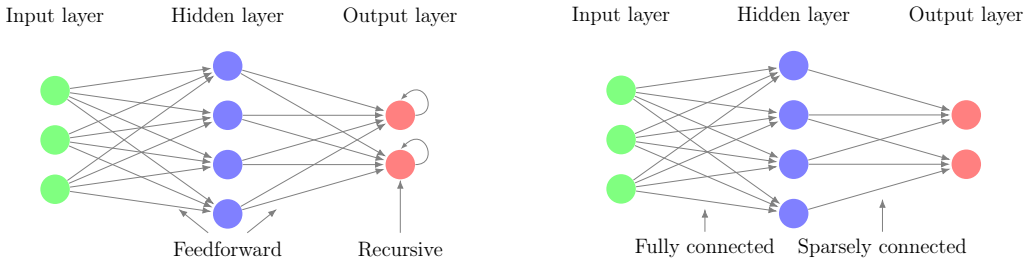


Figura 1.8: Esempio di classificazione immagini[23]

1.2.2 Panoramica delle DNNs

Esistono DNN di diverse dimensioni e forme, e le più popolari cambiano rapidamente per migliorare accuratezza ed efficacia. Si possono distinguere due forme principali: *feedforward* e *recurrent*.
Nel caso di rete feedforward la computazione avviene in una serie di operazioni in sequenza e ad un determinato ingresso corrisponderà sempre lo stesso risultato. Al contrario le *recurrent neural networks* (RNN) hanno una memoria interna che permette una dipendenza dei risultati dai dati precedenti.
Le DNN posso essere composte da *fully connected* layers (FC). In uno strato FC ogni attivazione d'uscita è composta dalla somma pesata di tutte le attivazioni d'ingresso. In molte applicazioni questo non è necessario e alcuni pesi sono ridotti a zero, rendendo il layer *sparsely connected*.



(a) Feedforward e recurrent networks

(b) Fully connected e sparse networks

Figura 1.9: Diversi tipi di reti neurali

Il calcolo può essere reso più efficiente limitando il numero di pesi che contribuiscono all'output. Inoltre si può ridurre la memoria necessaria a salvare i pesi, riutilizzando lo stesso set (*weight sharing*).
Ha così acquisito popolarità un layer strutturato come una convoluzione, in cui la somma pesata per un output è ottenuta sfruttando solo un piccolo insieme di input

vicini chiamato *receptive field*. In più viene utilizzato lo stesso set di pesi per tutti gli output. Questo layer è chiamato *convolutional* layer (CONV).
Le reti neurali che contano principalmente sui CONV sono dette convolutional neural networks e le DNNs più popolari utilizzano sempre di più questo tipo di approccio. Un CONV layer è basato su un forte utilizzo di moltiplicazioni e addizioni.
In queste reti gli input di attivazioni sono organizzati in insiemi di *input features map* (ifmaps) 2D chiamati canali. Ogni canale viene convoluto per un filtro 2D preso da uno stack. Ci si riferisce allo stack di filtri 2D come singolo filtro 3D. Il risultato della convoluzione è sommato fra tutti i canali. In più un bias 1D può essere addizionato ai risultati del filtraggio. Il risultato derivato da un canale è chiamato *output feature map* (ofmap).
Oltre a CONV ed FC esistono anche layers opzionali come di nonlinearità, *pooling* e normalizzazione. I primi sono usati dopo ogni CONV o FC per introdurre nonlinearità nel sistema con funzioni tipiche come sigmoid, ReLu (e sue derivate). Il pooling costituisce una sorta di pretrattamento degli input con funzioni come media o max. La normalizzazione modifica la distribuzione degli input per ottenere una media nulla e una deviazione standard unitaria.
Nella tabella sottostante è riportato un riepilogo delle DNN più popolari dedicate alla ImageNet Challenge, con un indicazione del numero di *multiply-and-accumulate* che le costituiscono.

Metrics	**LeNet 5**	**AlexNet**	**Overfeat fast**	**VGG 16**	**GoogLeNet v1**	**ResNet 50**
Top-5 error[†]	n/a	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)[†]	n/a	19.8	17.0	8.8	10.7	7.0
Input Size	28×28	227×227	231×231	224×224	224×224	224×224
# of CONV Layers	2	5	5	13	57	53
Depth in # of CONV Layers	2	5	5	13	21	49
Filter Sizes	5	3,5,11	3,5,11	3	1,3,5,7	1,3,7
# of Channels	1, 20	3-256	3-1024	3-512	3-832	3-2048
# of Filters	20, 50	96-384	96-1024	64-512	16-384	64-2048
Stride	1	1,4	1,4	1	1,2	1,2
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G
# of FC Layers	2	3	3	3	1	1
Filter Sizes	1,4	1,6	1,6,12	1,7	1	1
# of Channels	50, 500	256-4096	1024-4096	512-4096	1024	2048
# of Filters	10, 500	1000-4096	1000-4096	1000-4096	1000	1000
Weights	58k	58.6M	130M	124M	1M	2M
MACs	58k	58.6M	130M	124M	1M	2M
Total Weights	60k	61M	146M	138M	7M	25.5M
Total MACs	341k	724M	2.8G	15.5G	1.43G	3.9G

Tabella 1.1: Riepilogo DNN più popolari [23]

1.2.3 Stato dell'Arte

Nonostante le prime proposte di reti neurali siano datate agli anni 1940, la prima implementazione pratica con neuroni multipli non è arrivata prima del 1989, con la rete LeNet per il riconoscimento di scrittura a mano[25].
Tuttavia il periodo d'oro delle reti neurali è arriva negli anni 2010 con il sistema di riconoscimento vocale di Microsoft nel 2011[26] e AlexNet per il riconoscimento di immagini nel 2012[27]. Il successo delle reti neurali può essere attribuito a 3 cause: l'avvento dei big data (Facebook riceve ogni giorno un miliardo di immagini), la crescita nel campo dei semiconduttori e architetture dei PC, e lo sviluppo di nuovi algoritmi che ha permesso l'applicazione di DNN a campi finora esclusi.
Un esempio eclatante dello sviluppo delle DNNs può essere visto in figura 1.10, una sfida di riconoscimento immagini con una dataset per il training di 1.2 milioni di immagini e 1000 categorie di oggetti possibili. Nel 2015 la rete ResNet ha superato l'accuratezza media umana in un test che considera le prime 5 categorie fornite come risposta per ciascuna immagine[28].

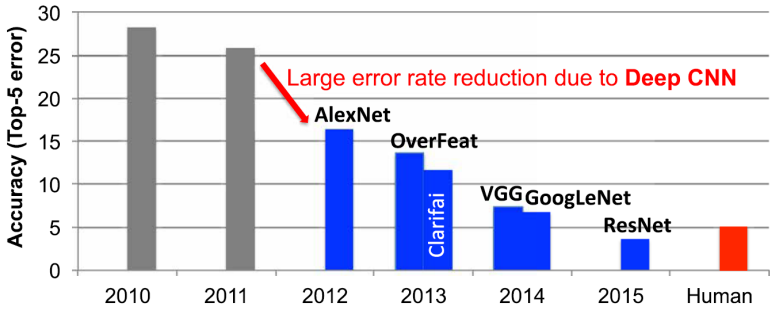


Figura 1.10: Risultati della sfida ImageNet[28]

Al giorno d'oggi le reti neurali sono applicate in diversi aree, dal multimedia alla medicina. A titolo d'esempio, è bene citare:

- Immagini e video: rappresentano la più grande mole di dati ad oggi disponibile, le DNN sono utilizzate per l'estrazione di informazioni utili. Esempi tipici possono essere il riconoscimento immagini, localizzazione di oggetti, riconoscimento di azioni e segmentazione di forme.

- Parola e linguaggio: le DNNs hanno migliorato l'efficacia del riconoscimento vocale, processing di linguaggio e generazione audio.

- Medicina: si è dimostrato il ruolo importante nella genomica, migliorando la comprensione di malattie genetiche come autismo, cancro e atrofia muscolare.

Le DNNs sono state anche utilizzate nella classificazione di immagini relative a casi medici per il rilevamento di tumori della pelle, del cervello e della mammella.

- Videogiochi: hanno dimostrato la loro efficacia in contesti in cui non è possibile prevedere tutte le conseguenze di una scelta. Hanno superato le capacità umane in giochi come Atari e Go e stanno rivaleggiando in videogiochi più complessi come Dota 2.
- Robotica: stanno avendo successo nel controllo di movimenti senza supervisione come la stabilizzazione di quadricotteri, strategie di guida di veicoli autonomi, navigazione a vista e controllo della presa di braccia meccaniche.

La varietà di applicazione delle reti neurali ha come conseguenza anche i diversi requisiti, in relazione alla complessità.

1.3 Il Memristore

Il terzo argomento di questa tesi è il memristore. Teorizzato nel 1971 da Leon Chua[29], è spesso indicato come il quarto bipolo passivo fondamentale oltre a resistore, induttore e condensatore. Una definizione assiomatica, data dallo stesso Chua, è la seguente:

> "Ogni dispositivo a due terminali che mostra un ciclo d'isteresi pizzicato con intersezione nell'origine nel grafico corrente-tensione quando pilotato da un qualsiasi sorgente di tensione o di corrente periodica a media nulla è chiamato memristore. Se l'ingresso è una sorgente di corrente, si tratta di un memristore controllato in corrente. Se l'ingresso è una sorgente di tensione, è chiamato memristore controllato in tensione."

Questa definizione *black-box* indica che la composizione interna del dispositivo è irrilevante. In effetti il comportamento tipico del memristor può essere ottenuto con diversi materiali ed è stato osservato in natura in molteplici animali e piante[30].

1.3.1 Classificazione

Per ragioni didattiche i memristori sono solitamente classificati in 4 categorie in ordine di complessità decrescente[31]. La più semplice e ultima nella tabella 1.2, è chiamata memristore ideale e coincide con la prima definizione del 1971.
La relazione costitutiva, data una condizione iniziale $\varphi(0)$, è data da:

$$\varphi \triangleq \varphi(0) + \int_0^q R(q)dq \triangleq \hat{\varphi}(q) \tag{1.8}$$

Differenziando entrambi i lati rispetto al tempo si ottiene:

$$\frac{d\varphi}{dt} = R(q)\frac{dq}{dt} \qquad o \qquad v = R(q)i \tag{1.9}$$

una volta definiti $\frac{d\varphi}{dt} = v$ e $\frac{dq}{dt} = i$.
L'equazione 1.8 o la sua equivalente 1.9 è chiamata relazione costitutiva di un memristore controllato in carica.
La relazione costitutiva duale di un memristore controllato in flusso è:

$$q = \hat{q}(\varphi) \tag{1.10}$$

equivalente ad un memristore controllato in tensione:

$$i = G(\varphi)v \tag{1.11}$$

dopo aver sostituito $\frac{\varphi}{dt} = v$.

	Controllo in corrente	Controllo in tensione
Memristore esteso	$v = R(\mathbf{x}, i)i$ $R(\mathbf{x}, i) \neq \infty$ $\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, i)$	$v = G(\mathbf{x}, v)v$ $G(\mathbf{x}, i) \neq \infty$ $\frac{d\mathbf{x}}{dt} = \mathbf{g}(\mathbf{x}, v)$
Memristore generico	$v = R(\mathbf{x})i$ $\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, i)$	$v = G(\mathbf{x})v$ $\frac{d\mathbf{x}}{dt} = \mathbf{g}(\mathbf{x}, v)$
Memristore ideale generico	$v = R(x)i$ $\frac{dx}{dt} = \hat{f}(x)i$	$v = G(x)v$ $\frac{dx}{dt} = \hat{g}(x)v$
Memristore ideale	$v = R(q)i$ $\frac{dq}{dt} = i$	$v = G(\varphi)v$ $\frac{d\varphi}{dt} = v$

Tabella 1.2: Classificazione dei memristori

Memristore ideale

D'ora in poi si focalizzerà l'attenzione sul memristore controllato in tensione, da cui si può facilmente intuire il duale.
Il memristore ideale è definito da una funzione scalare $G(\varphi)$ chiamata mem-conduttanza. Dato che l'input è la tensione, la variabile indipendente è il flusso φ definito come

$$\varphi(t) \triangleq \int_{-\infty}^{t} v(\tau)d\tau = \varphi(0) + \int_{0}^{t} v(\tau)d\tau. \tag{1.12}$$

Una volta nota l'onda in ingresso $v = v(t)$ e la condizione iniziale $\varphi(0)$, si può ricavare la forma d'onda $\varphi(t)$ utilizzando l'equazione 1.12. Una volta sostituita nell'equazione $G(\varphi)$ si ottiene la funzione $G(\varphi(t))$ che descrive la mem-conduttanza

variante nel tempo e la corrispondente $i(t)$ utilizzando la legge di Ohm. Se si riporta in un piano i vs. v i valori di $v(t)$ e la corrispondente $i(t)$ si ottiene un curva chiusa e passante per l'origine. Tale grafico è il già citato ciclo d'isteresi pizzicato ed è mostrato in figura 1.11.

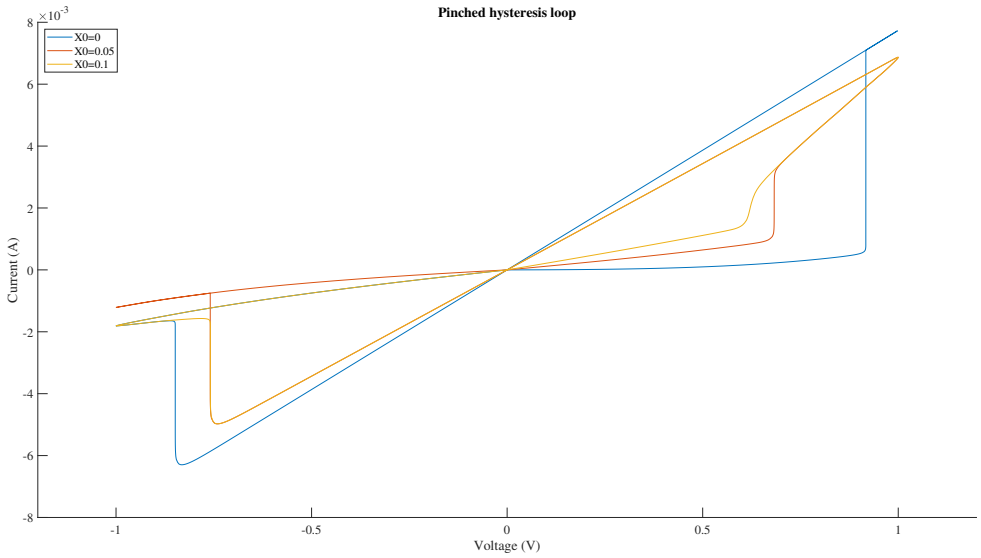


Figura 1.11: Pinched hysteresis loop per il memristore TaO data una sinusoide in ingresso

Si noti come i cicli d'isteresi di un memristore con stati iniziali differenti e la stessa sorgente periodica possano essere diversi fra loro.
Data l'importanza del ciclo d'isteresi pizzicato in qualità di firma tipica dei memristori, le sue caratteristiche sono state studiate approfonditamente. Nel caso di memristori ideali tale funzione è dispari ed è stato stabilito che, se pilotato da una funzione dispari, si incrocia all'origine con due diverse pendenze[32].

Memristore ideale generico

Si può ricavare un insieme infinito di memristori ideali generici derivati partendo da un memristore ideale.
Quattro passaggi sono necessari per ottenerne un memristore ideale generico una volta nota la funzione tra carica q e flusso φ di un memristore ideale controllato in tensione, insieme alla sua funzione mem-conduttanza G(φ) dipendente dal flusso.

Il primo passo è la scelta di una funzione $x = \hat{x}(\varphi)$ da cui si ricavi anche la funzione inversa $\varphi = \hat{x}^{-1}(x)$.

Il secondo passo prevede di derivare la funzione $\mathrm{G}(\varphi)=\frac{d\hat{q}(\varphi)}{d\varphi}$ nella quale si sostituisce $\varphi = \hat{x}^{-1}(x)$ per ottenere una mem-conduttanza $\mathrm{G}(x)$ dipendente da uno stato x.

Nel terzo passo si deve derivare $\frac{d\hat{x}(\varphi)}{d\varphi}$ e sostituirci nuovamente $\varphi = \hat{x}^{-1}(x)$ per ottenere la funzione $\hat{g}(x)$.

Nell'ultimo passo si definiscono i memristori derivati con le funzioni ottenute $i = G(x)v$ e $\frac{dx}{dt} = \hat{g}(x)v$.

È stato stabilito che ognuno degli infiniti memristori ideale generici così ottenuti ha lo stesso ciclo d'isteresi pizzicato del memristore ideale da cui sono stati generati. Per questo motivo è spesso consigliato ricavare il memristore ideale padre per studiare più facilmente il memristore ideale generico.

Memristore generico

Un memristore è chiamato generico se la sua legge di Ohm dipendente dallo stato e la sua equazione di stato assumono la forma riportata nella seconda riga della tabella 1.2. Diversamente dal Memristore ideale generico, la corrente i (o tensione v rispettivamente) appare all'interno della funzione $\hat{f}(x)$ (o $\hat{g}(x)$).
Un esempio di memristore generico non ideale può essere un *Termistore a coefficiente di temperatura negativo* (NTC)[33]. Il suo modello matematico è il seguente:

Legge di Ohm in funzione dello stato	$i = W(x)v$
Equazione di stato	$\frac{dx}{dt} = \frac{\delta_N}{H_{CN}}(T_{0N} - x) + \frac{W(x)}{H_{CN}}v^2$

La mem-conduttanza $M(x)$ è definita da:

$$W(x) = [R_{ON}e^{\beta_N(\frac{1}{x}-\frac{1}{T_{0N}})}]^{-1}$$

Memristore esteso

Un memristore è chiamato esteso se la sua mem-resistenza $R(\mathbf{x}, i)$, o mem-conduttanza $G(\mathbf{x}, v)$ è funzione non solo delle variabili di stato $x = (x_1, x_2, ..., X_n)$ ma anche della corrente (o tensione) di input e la mem-resistenza $R(\mathbf{x}, 0)$ o mem-conduttanza $G(\mathbf{x}, 0)$ è un numero finito.

Il memristore a base di Ossido di Tantalio che verrà utilizzato in questa tesi è un esempio di memristore esteso e le relazioni costitutive sono riportate nella sezione 2.2

1.3.2 La fisica del memristore

Data la sua definizione, non esiste nè una composizione, nè un comportamento fisico tipico del memristore. Si può però esaminare un esempio e cercare di capirne i meccanismi di switching, cioè del passaggio da uno stato memristivo ad un altro.
I memristori ad ossido di Tantalio (TaO_x) stanno trovando successo grazie alle buone capacità di switching multi-livello e sono i più promettenti candidati ad essere applicati in diversi contesti come memorie analogiche e neuromorfiche. Il dispositivo usato in questa tesi fa parte di questo tipo di memristori e un suo esempio in serie con un transistore (1T1M) può essere visualizzato in figura 1.12.
Il dispositivo in questione è integrato sopra un CMOS ed è composto da 3 strati:

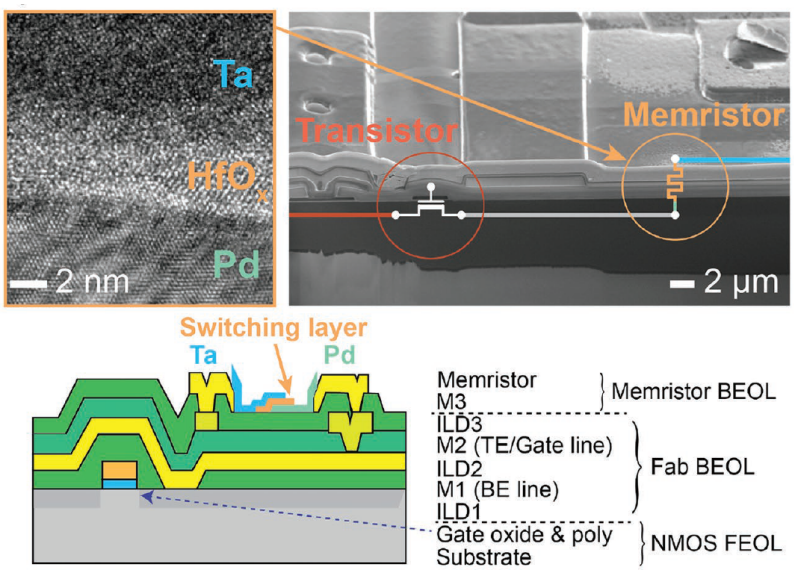


Figura 1.12: Memristore TaO[34]

Tantalio, ossido di Afnio (che ne costituisce lo strato di switch) e Palladio.
Tuttavia ci sono state difficoltà nella comprensione e visualizzazione dei filamenti di conduzione (CF) durante il passaggio da uno stato ON ad uno stato OFF attraverso diversi esperimenti come il *transmission electron microscopy* (TEM) e lo *scanning electron microscopy* (SEM). Per adesso solo lo stato ON statico è stato visualizzato attraverso il *pressure-modulated conductance microscopy* (PMCM), studi con l'*X-ray spectromicroscopic* e il TEM.

Secondo un recente studio[35] è stata riconosciuta la coesistenza di 3 maggiori movimenti di ioni a caratterizzare lo switch tra stati di questo tipo di memristori: un drift verticale dovuto al campo elettromagnetico, una diffusione laterale che porta gli ioni di ossigeno nel CF e la termoforesi che spinge gli ioni di ossigeno ad allontanarsi dalle regioni del CF. A seconda delle condizioni uno o due di questi fenomeni sono prevalenti e portano alla chiusura o apertura del canale di conduzione.

1.3.3 Stato dell'Arte

Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine[34]

Il memristore TaO è stato utilizzato per costruire un Dot Product Engine (DPE) applicato ad una rete neurale per la classificazione del database MNIST.
L'implementazione 1T1M (figura 1.12) permette la regolazione precisa della memresistenza su circa 180 livelli differenti, consentendo di rappresentare alcune matrici in analogico in modo molto preciso e di calcolare prodotti vettore matrice (VMM) convertite dal digitale fintanto che la matrice scelta sia rappresentabile. Il segnale digitale viene quindi convertito e calcolato in analogico, venendo poi riconvertito in digitale.
Il gruppo di ricerca ha costruito una matrice di 128 × 64 memristori che lavora alla velocità di 100 MHz. Dato che i memristori non sono volatili, la programmazione può avvenire una sola volta e poi ignorata fino a che non è necessaria una nuova matrice.

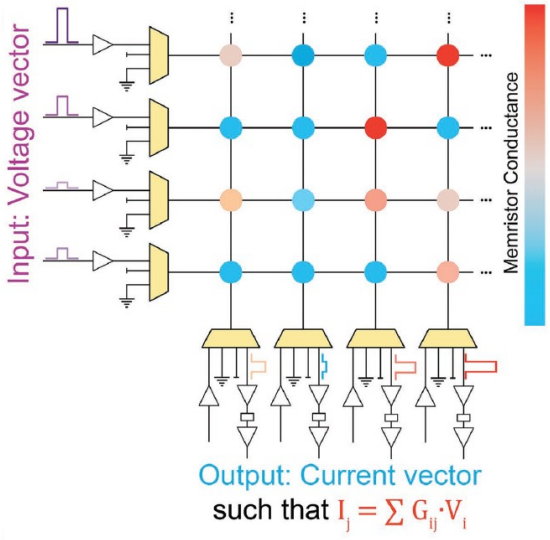


Figura 1.13: Schematico di DPE con matrice di memristori[34]

Il sistema di selezione di righe e colonne permette di programmare singolarmente ogni cella. Una tensione positiva equivale ad un'operazione di RESET e una tensione negativa porta un SET. Con degli impulsi crescenti di SET e RESET, inframezzati da dei segnali di lettura della mem-conduttanza, si è riuscito ad ottenere il *fine tuning* mostrato in figura 1.14.

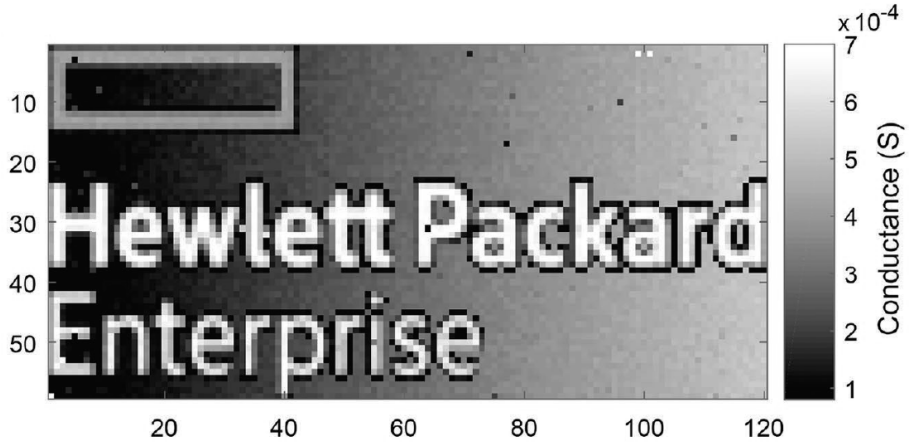


Figura 1.14: Dimostrazione dell'accuratezza sui livelli di mem-conduttanza[34]

Inoltre l'integrazione CMOS-memristor può essere applicata in fonderia con un processo BEOL (back-end-of-the-line), cioè viene depositato sul CMOS già prodotto un layer di transizione, e il collegamento viene fatto con una tecnologia a 2 μm.
Il memristore è lo stesso già visto precedentemente che consiste in uno stack di Ta/HfO_2/Pd sviluppato per ottimizzare una conduttività multilivello.
Tutto il sistema è stato disegnato per utilizzare in input degli impulsi di tensione regolabile tra -10V e 10V, e le colonne sono in grado di misurare la corrente tramite amplificatori a transimpedenza.
Il risultato sono 16000 moltiplicazioni e addizioni per ciclo di clock che portano ad una performance prevista di 115 trilioni di operazioni al secondo per Watt.
La VMM risultante ha una precisione di 6 bit e può essere programmata per diverse applicazioni. Il suo utilizzo nel riconoscimento di lettere scritte a mano del database MNIST ha portato ad un accuratezza dei risultati dell'89.9%.

Capitolo 2

Progetto

In questo capitolo sarà presentato un circuito di conversione analogico-informativo a basso consumo. Il lavoro prende spunto dalla letteratura presentata nel capitolo precedente e verrà esposto nell'ordine logico/cronologico in cui è stato sviluppato.

2.1 Dalle reti neurali al Compressed Sensing

Gli algoritmi del Compressed Sensing prevedono un ampio utilizzo di circuiti di moltiplicazione e addizione. Come esposto nella sezione 1.1 il segnale analogico viene campionato e moltiplicato per delle funzioni di test. Nella sezione 1.2 sono state introdotte le reti neurali modo tale da sfruttare una somiglianza cruciale con il CS: l'uso esteso di somme pesate.
La bibliografia sulle DNNs è in enorme crescita, data l'espansione che tale campo ha al giorno d'oggi e la quantità di energia e hardware richiesto da queste tecnologie. L'ottimizzazione dell'esecuzione di questo tipo di calcolo è una questione sia economica che ecologica, e punto decisivo nello sviluppo di Intelligenze Artificiali.
Dopo aver studiato diverse soluzioni tipiche applicate alle reti neurali, averne valutato efficienza e peculiarità, una soluzione in particolare ha suscitato interesse ai fini di questa tesi. Tra le varie proposte per rendere più efficiente la somma pesata è stata infatti presentato e valutato un Dot Product Engine (DPE) (presentato nella sezione 1.3.3) costituito da una matrice di Memristenze che utilizzi le leggi di Kirchhoff per eseguire il prodotto scalare direttamente in un circuito mixed-signals. Nello specifico, si ottiene una moltiplicazione usando la conduttanza di un resistore come peso, la tensione come ingresso e la corrente come uscita, come mostrato in figura 2.1.
Alcune pubblicazioni mostrano [34] come questo tipo di approccio sia promettente e adottando lo stesso con tecnologie adatte [36] si riesca ad ottenere, secondo le proiezioni, un'efficienza computazionale pari a 115 TOPS W^{-1} (Tera Operazioni al

secondo per Watt).

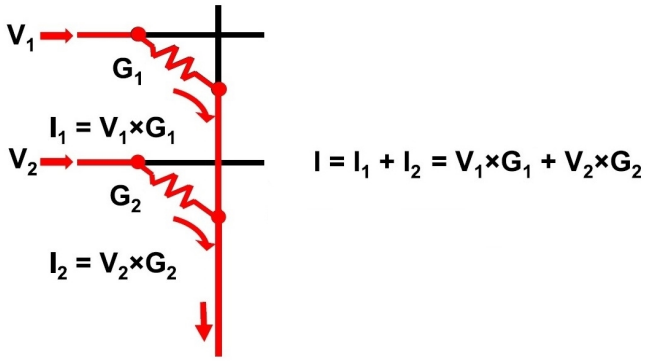


Figura 2.1: Computazione analogica con Memristors. G è la conduttanza[23].

Per avere un metro di misura, per eseguire lo stesso tipo di operazioni con tecnologia digitale CMOS a 40 nm è stimata un'efficienza pari a 7 TOPS W $^{-1}$.
Inoltre, spostare la conversione in digitale al termine del calcolo allevierebbe ulteriormente i consumi e la richiesta di area.

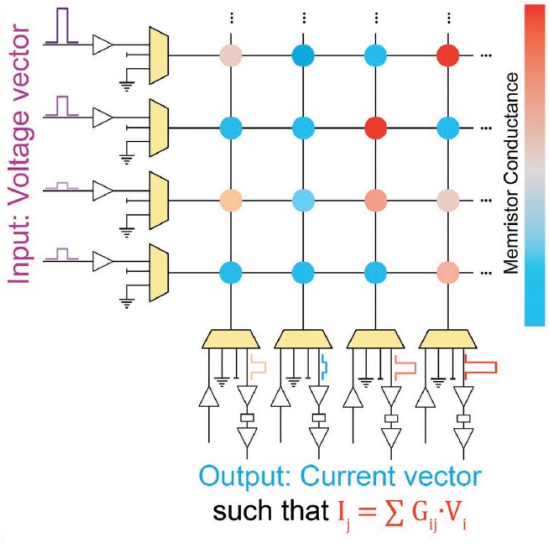


Figura 2.2: Schematico di DPE con matrice di memristori

Si stima che un DPE (figura 2.2) applicato in un contesto più pratico come una Convolutional Neural Network (CNN) per l'inferenza di immagini, porterebbe un miglioramento pari a 14.8x, 5.5x e 7.5x in throughput (inferenze per secondo), energia e densità computazionale (inferenze per secondo per area del chip) rispettivamente in confronto all'attuale implementazione ASIC digitale [37] di riferimento per

lo stesso compito.
Sebbene i risultati siano sensazionali, l'uso di mixed circuit in larga misura nel campo delle DNN sembra ancora lontano. La scelta delle resistenze variabili non volatili adatte è da fare caso per caso. Tra i possibili candidati troviamo phase change memory (PCM), resistive RAM (RRAM or ReRAM), conductive bridge RAM (CBRAM), e spin transfer torque magnetic RAM (STT-MRAM)[38]. Ognuna di queste possibilità presenta pro e contro in termini di durata, tempo di ritenzione del dato, correnti di scrittura, densità, variazione e velocità.
Tra i vari svantaggi di tali memorie vi sono inoltre la ridotta precisione e consumo aggiunto dai convertitori ADC e DAC, la limitazione delle dimensioni di matrici dovuta alla resistenza delle connessioni, l'energia richiesta per scrivere il dato e la variazione di valore tra un ciclo e un altro[39].
Per risolvere molti di questi problemi sono state presentate delle implementazioni in cui eDRAM (ISAAC [37]) e DRAM (PRIME[40]) sono stati sostituiti con i Memristori (Sezione 1.3). In particolare ISAAC esegue un prodotto scalare sfruttando 8 memristori con memoria di 2 bits, eseguendo una moltiplicazione 1b×2b e richiedendo 16 cicli per completare il calcolo. Inoltre riorganizza milioni di Memristori in ordine gerarchico per evitare problemi con matrici troppo grandi. PRIME invece usa matrici da 256×256 memristori che possono essere configurati a 4 bit (per il calcolo) o a singolo bit (come memoria).
È doveroso sottolineare come questi risultati siano stati ottenuti attraverso simulazioni dato che la fabbricazione di grosse matrici di memristori è ancora in evoluzione: esistono in ogni caso esempi di matrici di memristori fabbricate.
Nel caso del CS inoltre trattiamo un segnale analogico ancora prima della conversione in digitale. Per cui alcuni svantaggi elencati non sono considerati un problema. C'è una differenza cruciale che non permette di applicare lo stesso identico concetto: nel caso del CS gli elementi del vettore in ingresso sono campioni presi in momenti diversi fra di loro e quindi l'operazione deve essere sequenziale e non parallela.
Passiamo quindi da:

$$I_j = \sum G_{ij} \cdot V_i \qquad \text{a:} \qquad I_j[t] = \sum G_j \cdot V[t]$$

2.2 Il modello di Memristore

È stato scelto il Memristore ad Ossido di Tantalio (TaO), sviluppato e fabbricato nei laboratori della Hewlett Packard. Un approfondimento sulla teoria e lo stato dell'arte dei Memristori può essere trovato nel capitolo 1.3.
Il memristore in questione è oggetto di diversi studi e HP ha pubblicato un modello matematico fedele ai dati ricavati in laboratorio [41]. Tale modello è rappresentato dal seguente sistema di equazioni differenziali algebriche:

$$\begin{aligned}\frac{dx}{dt} = \quad f(x, v_m) = \quad & A \sinh\Big(\frac{v_m}{\sigma_{off}}\Big) \exp\Big(- \frac{1}{1 + \beta_m i_m v_m}\Big) \text{ step}(-v_m) + \\ & B \sinh\Big(\frac{v_m}{\sigma_{on}}\Big) \exp\Big(- \frac{x^2}{x_{on}^2}\Big) \exp\Big(\frac{i_m v_m}{\sigma_p}\Big) \text{ step}(v_m),\end{aligned}$$

$$i_m = \quad W(x, v_m) v_m$$

Si ha quindi un memristore esteso del primo ordine, secondo la classificazione vista nel capitolo 1.3.
L'ingresso e l'uscita sono date rispettivamente da $u = v_m$ e $y = i_m$ mentre $H(x, u) = W(x, v_m)$ è la funzione di mem-conduttanza dipendente dall'input e stato data da:

$$W(x, v_m) = G_m x + a \exp(b\sqrt{|v_m|})(1 - x)$$

Lo stato x del memristore rappresenta la frazione di volume del canale di conduzione. Di conseguenza può variare tra 0 e 1 che corrispondono rispettivamente allo stato della nanostruttura completamente isolante/conduttivo.
La funzione step$(\cdot)$ è la funzione gradino, definita come $step(v_m) = \frac{1+sign(v_m)}{2}$, nella quale sign$(\cdot)$ rappresenta la funzione segno.
I valori dei parametri del modello sono riportati nella tabella:

A/s^{-1}	σ_{off}/V	x_{off}	$\beta/A^{-1}V^{-1}$	$B/s * -1$	σ_{on}
10^{-10}	$1.3{\cdot}10^{-2}$	$4{\cdot}10^{-1}$	500	$1{\cdot}10^{-4}$	$4.5{\cdot}10^{-1}$
x_{on}	$\sigma_p/A^{-1}V^{-1}$	G_m/S	a/S	$b/V^{-\frac{1}{2}}$	
$6 \cdot 10^{-2}$	$4 \cdot 10^{-5}$	$2.5 \cdot 10^{-2}$	$7.2 \cdot 10^{-6}$	4.7	

Data la natura del sistema, si è reso necessario un adattamento del modello al fine di semplificare l'integrazione numerica da parte di risolutori numerici di sistemi DAE. Sono state quindi eliminate le funzioni discontinue e a tratti, sostituite da approssimazioni continue e differenziabili [42]. Nelle prossime righe sarà presentato un breve riassunto del lavoro pubblicato da Ascoli e utilizzato in questa tesi.
La funzione modulo $|v_m|$ è stata sostituita dalla sua approssimazione g_ρ data da:

$$g_\rho = \left(\frac{1}{1+exp(-\rho v_m)}\frac{1}{1+exp(\rho v_m)}\right)$$

dove ρ è il parametro che definisce la concavità della curva. La figura 2.3 mostra l'approssimazione per ρ pari a 10^1, 10^2 e 10^3:

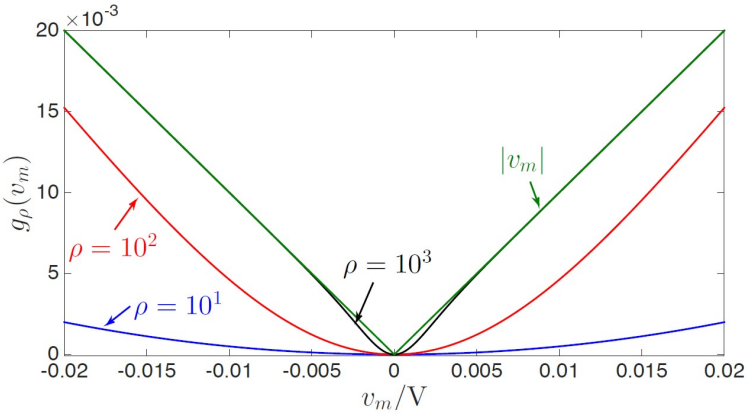


Figura 2.3: La funzione modulo $|v_m|$ e la sua approssimazione g_ρ

I risultati dell'approssimazione numerica sono migliori nel caso di $\rho = 10^3$.
Il grafico 2.4 mostra la i_m al variare di v_m compreso tra [-0.3, 0]V dati con stato interno iniziale $x_0 \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$. Il grafico 2.5 rappresenta più nello specifico la stessa funzione per il valore scelto $\rho = 10^3$ al variare di x_0 in tutto il dominio di esistenza a passi di 0.1. Nel confronto visivo con il modello originale si può notare come l'approssimazione sia ottimale.

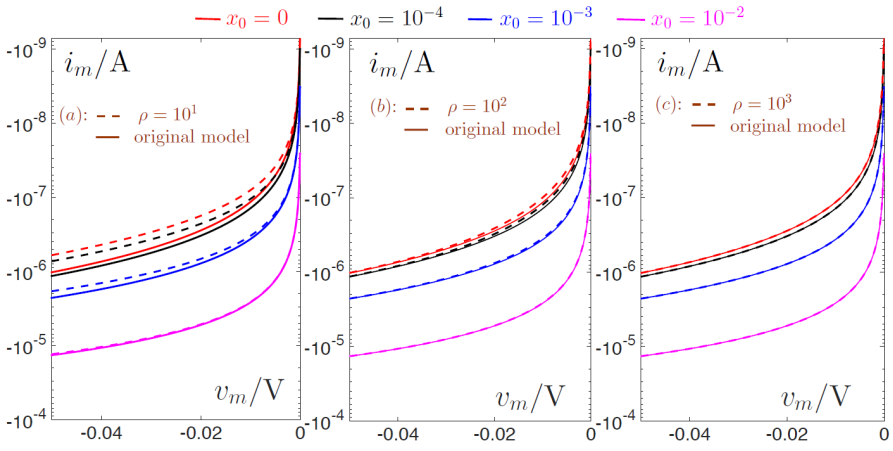


Figura 2.4: i_m al variare di v_m per $\rho \in \{10^1, 10^2, 10^3\}$

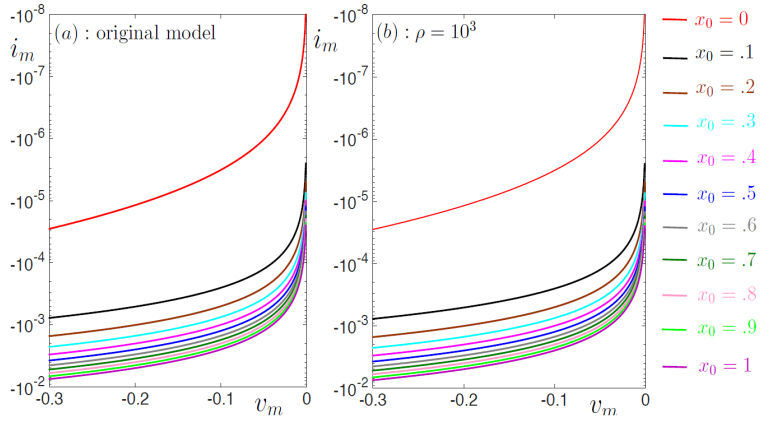


Figura 2.5: i_m al variare di v_m per $\rho = 10^3$ a diversi stati iniziali x_0

La seconda funzione presa in considerazione è step$(\cdot)$, sostituita da $f_k(v_m)$ data da:

$$f_k(v_m) = \left(\frac{1}{1 + exp(-kv_m)}\right)$$

nella quale k controlla la pendenza vicino a $v_m = 0$. La figura 2.6 mostra il grafico di esempio con $k \in \{30, 40, 50\}$ insieme con la funzione gradino di riferimento.

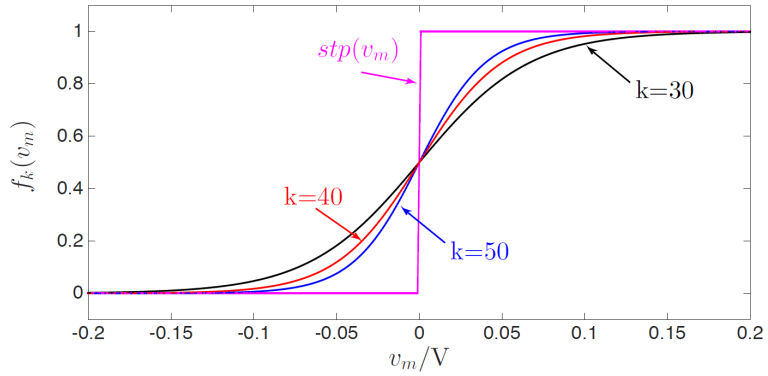


Figura 2.6: Approssimazione della funzione gradino

Per verificare la bontà dell'approssimazione sono state applicate diverse forme d'onda a varie frequenze per confrontare il comportamento del modello originale fornito dalla HP e il modello approssimato. Il circuito di test, in seguito riprodotto in questa tesi come base della matrice di memristori è rappresentato in figura 2.7.

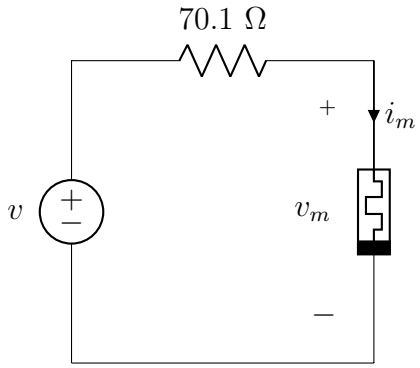


Figura 2.7: Circuito di test

Nella figura 2.8 possiamo notare il ciclo d'isteresi passante per l'origine tipico del memristore, e il valore $k = 50$ ci fornisce un'approssimazione indistinguibile dall'originale.

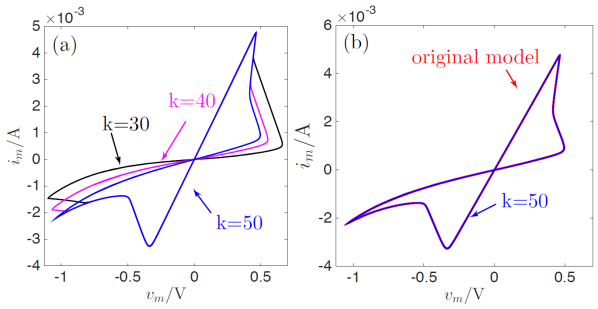
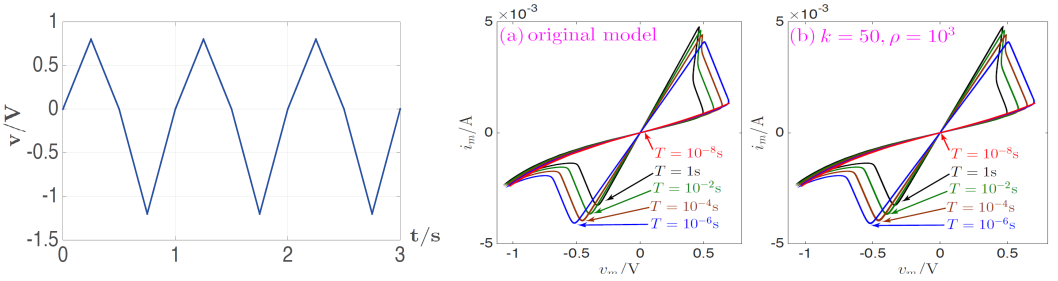


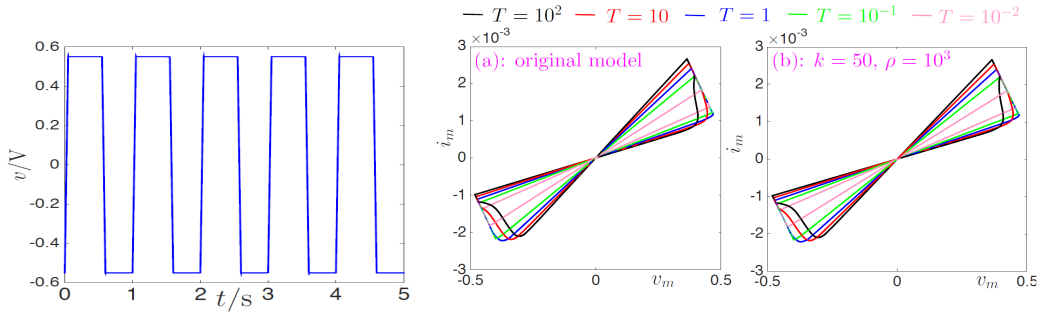
Figura 2.8: Ciclo d'isteresi approssimato al variare con $k \in \{30, 40, 50\}$

Il valore ottenuto con un'onda d'ingresso triangolare è poi sottoposto a verifica utilizzando come ingresso un'onda triangolare (figura 2.9a), un'onda quadra (figura 2.9b) e un'onda sinusoidale (figura 2.9c) al variare della frequenza.

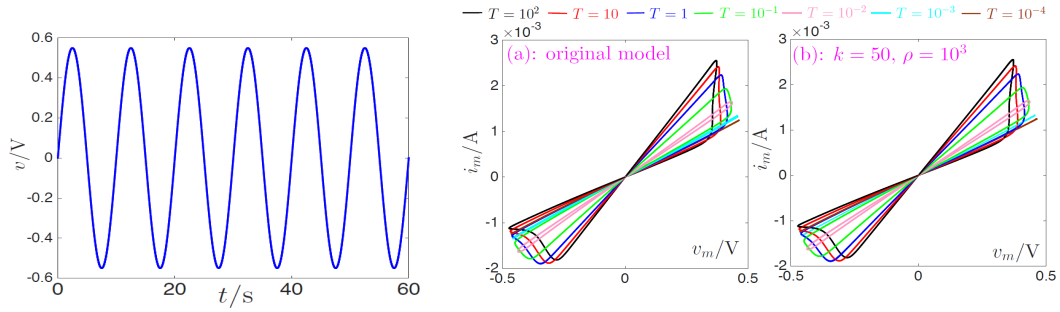
Una volta convalidate le scelte è stato elaborato un'implementazione del modello del TaO Memristor per il software LTSpice. Il circuito è rappresentato da un sottocircuito con 2 terminali d'accesso (plus e minus). Nel sottocircuito troviamo un generatore di corrente controllato in tensione G_{res} che definisce la corrente passante fra i 2 terminali, rappresentando così i_m. La capacità C_{int} modella lo stato x_0 del memristore, la cui velocità di carica dipende un altro generatore di corrente controllato in tensione G_Y, specificando l'evoluzione dello stato del memristor secondo la DAE del modello. La resistenza R_{aux} è inserita per prevenire problemi di convergenza nel simulatore.
La netlist per LTSpice risultante è riportata nell'appendice A.1.1.



(a) Ciclo d'isteresi per onda triangolare



(b) Ciclo d'isteresi per onda quadra



(c) Ciclo d'isteresi per onda sinusoidale

Figura 2.9: Confronto ciclo d'isteresi per $k = 50$ e $\rho = 10^3$ al variare della frequenza

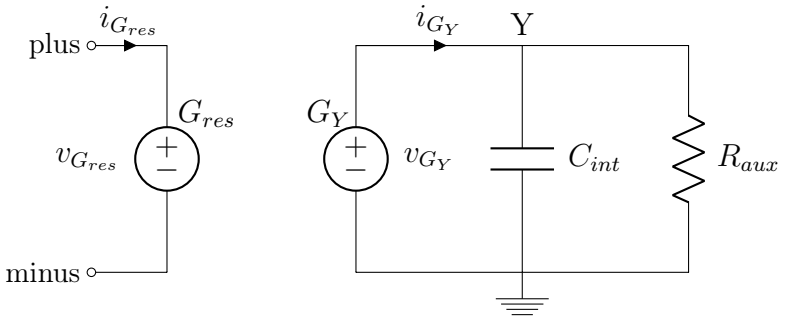


Figura 2.10: Implementazione circuitale del memristore TaO

2.2.1 Test sul Memristore TaO

Il primo passo nello studio del modello scelto è stata la replicazione dei risultati ottenuti nella pubblicazione. Il modello è stato quindi importato in LTSpice e si è riprodotto il circuito di test in figura 2.7.
Il segnale applicato in ingresso è composto da una serie di 4 impulsi: i primi due positivi, gli ultimi due negativi. Il primo impulso di ogni coppia rappresenta l'impulso di scrittura, in grado di cambiare lo stato interno del memristore. Il secondo impulso delle coppie è in grado di non modificare lo stato del memristore permettendo quindi la lettura, quale che sia il valore di interesse che si vuole ottenere. In figura 2.11 possiamo notare l'analisi parametrica per diversi valori di scrittura.

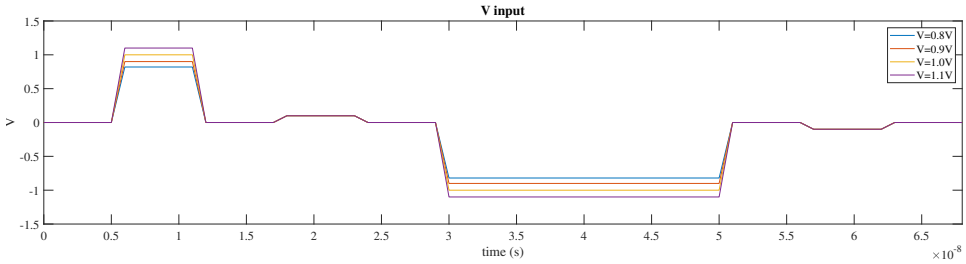
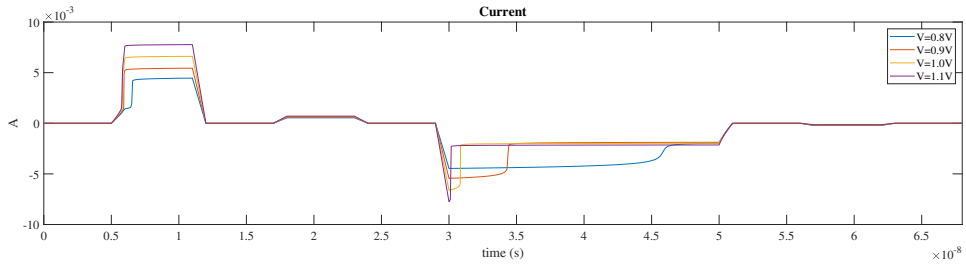
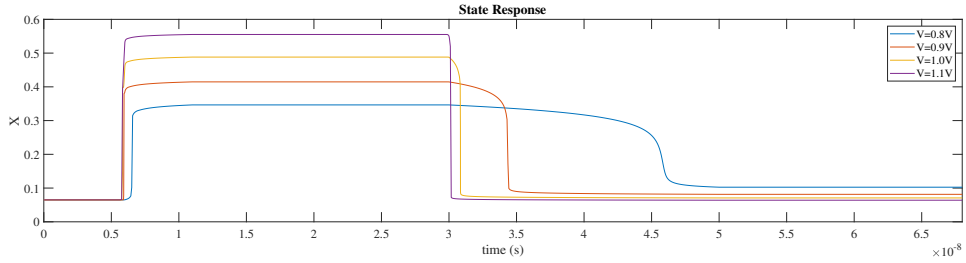


Figura 2.11: Segnale di ingresso per l'analisi iniziale del modello LTSpice



(a) Evoluzione della corrente i_m



(b) Evoluzione dello stato x

Figura 2.12: Risposta per Vin rappresentata in figura 2.11

È stato impostato uno stato iniziale $x_0 = 0.065$ secondo le scelte fatte dagli autori del modello. Gli effetti dei segnali di scrittura $V_w \in \{0.8, 0.9, 1.0, 1.1\}$ sono mostrati in figura 2.12a per quanto riguarda la corrente i_m passante nel memristore e in figura 2.12b per quanto riguarda lo stato interno x:
I risultati ottenuti corrispondono a quelli ottenuti nella pubblicazione. Il passo successivo è quindi stato l'esame del comportamento in diverse condizioni per conoscere i limiti del modello per quanto riguarda programmabilità, lettura e precisione.
Tre informazioni sono necessarie al corretto utilizzo del dispositivo:

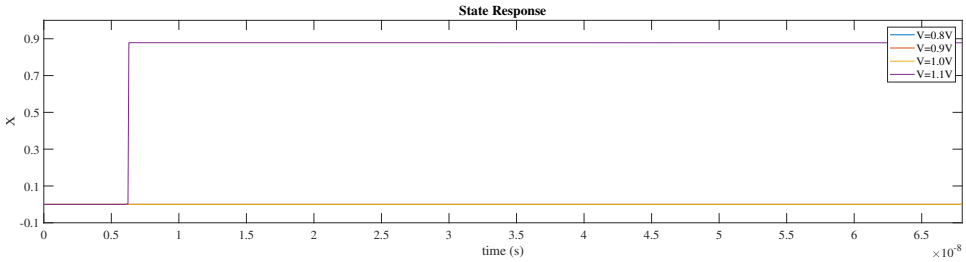
1. Quali sono i segnali adatti alla programmazione?
2. Quali sono le ampiezze dei segnali adatti alla lettura?
3. Che legame esiste tra stato interno x e mem-resistenza?

Programmazione

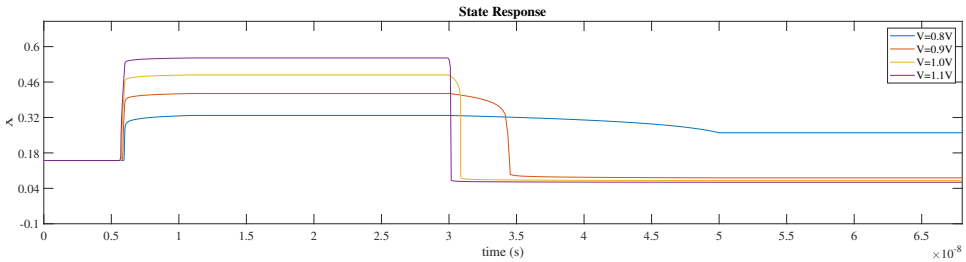
Si è replicata la prova appena eseguita partendo da condizioni iniziali differenti. I grafici in figura 2.13 sono ottenuti per $x_0 \in \{0, 0.15, 5, 1\}$:
Diverse considerazioni possono essere tratte dai risultati ottenuti:

- Quando ci si trova in condizioni vicine agli estremi del dominio di x i risultati non sembrano affidabili per un utilizzo in un circuito. Potrebbe essere determinato da un limite del modello o un comportamento fisico quando il canale è completamente aperto/chiuso.
- Se la condizione iniziale si discosta dai limiti [0, 1] la programmazione è coerente, i risultati sono prevedibili e replicabili. Abbiamo trovato in 0.065 i limiti minimi per iniziare la programmazione con impulsi positivi.
- La programmazione con tensioni positive è più semplice rispetto alle negative: alcune simulazioni con tensioni negative non convergono a risultati possibili e le ampiezze richieste sono spesso più alte di quelle positive. La risposta ad un impulso negativo può mostrare una certa incoerenza come mostrato a t=34ns nei grafici 2.13b e 2.13c.
- Utilizzando la forma d'onda scelta, partendo da x_0 diverse è applicando una tensione di programmazione positiva si arriva allo stesso risultato (differenze minime) purché x_0 sia minore dello stato che si vuole raggiungere. Nei grafici 2.13b e 2.13c si può notare che l'impulso di scrittura a 1,1V ha permesso di ottenere entrambe le volte lo stesso stato.

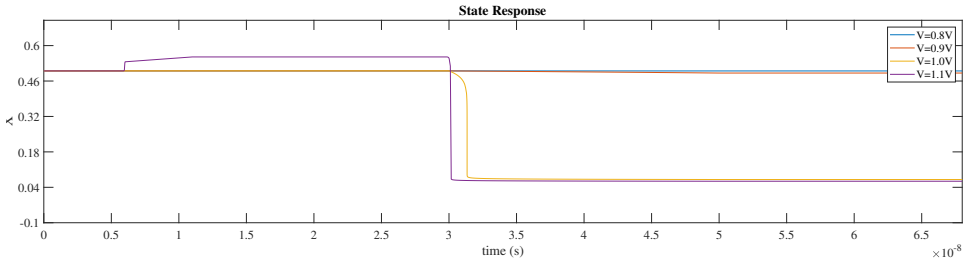
Dato che l'obbiettivo della tesi non si ferma allo studio del memristore, si è scelto di sfruttare le informazioni ottenute per programmare il memristore nel modo più



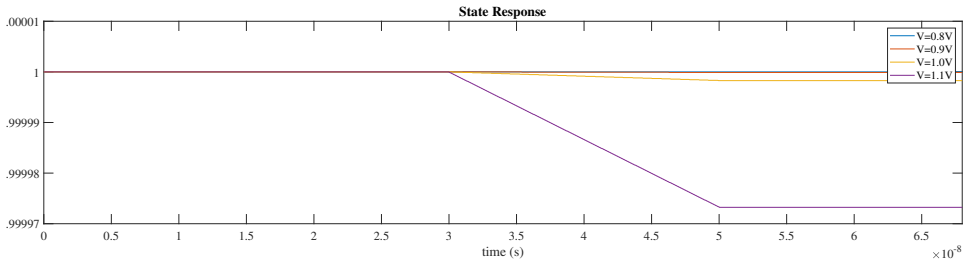
(a) Evoluzione dello stato x per $x_0 = 0$



(b) Evoluzione dello stato x per $x_0 = 0.15$



(c) Evoluzione dello stato x per $x_0 = 0.5$

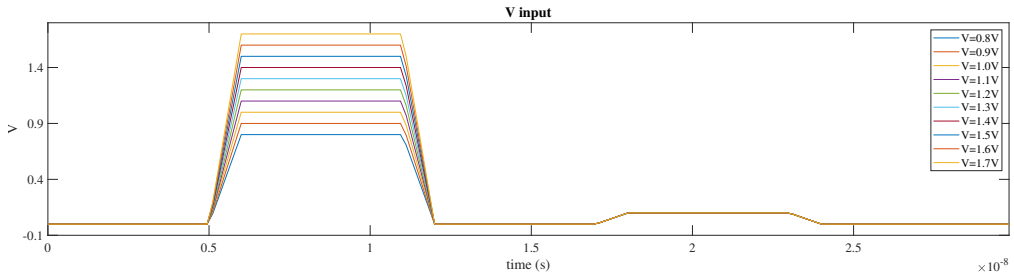


(d) Evoluzione dello stato x per $x_0 = 1$

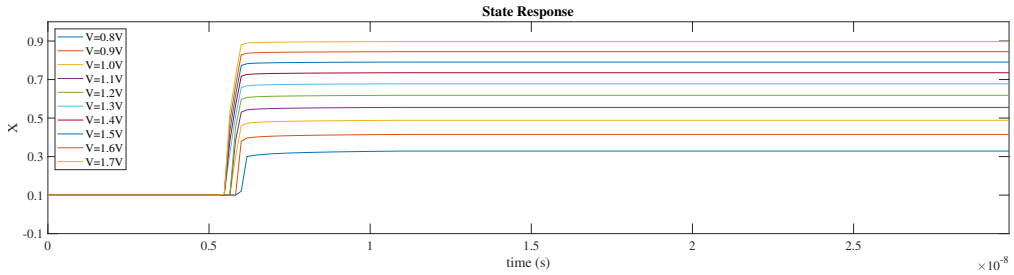
Figura 2.13: Risposta per Vin rappresentata in figura 2.11 al variare di x_0

semplice, efficace e affidabile possibile, in linea con l'utilizzo che se ne deve fare. Precisazioni, programmazioni più complesse o che sfruttino le intere possibilità del memristore possono essere lasciate a futuri sviluppi.

Il test successivo è stato studiare la risposta ad un impulso di scrittura rappresentato in figura 2.14a per un numero maggiore di ampiezze.



(a) Impulsi crescenti di scrittura

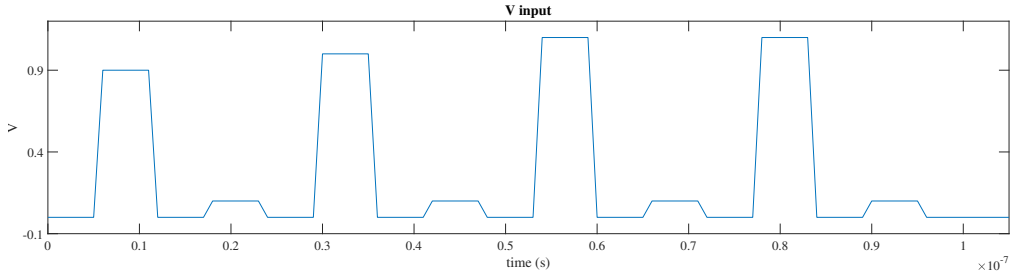


(b) Evoluzione dello stato x

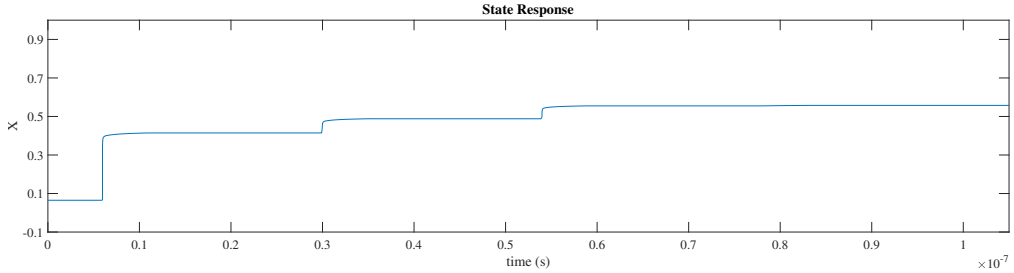
Figura 2.14: Variazione di x per impulsi positivi

Lo stesso test non è stato soddisfacente per tensioni negative, LTSpice non ha risolto il circuito nei limiti di valori possibili.
Si è passato quindi ad analizzare la risposta a vari impulsi, facendo diverse prove di cui la figura 2.15 è un esempio. Ci sono diverse conclusioni importanti:

- La scrittura avviene principalmente negli momenti iniziali dell'impulso, si può riconoscere il caricamento della capacità del sotto-circuito.
- Un secondo impulso identico o minore del primo non modifica lo stato scritto dal primo.
- Un secondo impulso maggiore del primo modifica lo stato come se il primo impulso non sia mai avvenuto.
- Un impulso lungo ha lo stesso effetto di un impulso breve.
- L'impossibilità di affidarsi ad ampiezze negative limita le opzioni di programmazione.



(a) Treno di impulsi in ingresso



(b) Evoluzione dello stato x

Figura 2.15: Esempio di variazione di x per impulsi successivi crescenti o doppi

Il segnale più adatto ed efficace alla programmazione si è rivelato dunque essere un impulso di lunghezza pari a 5ns e di ampiezza variabile. Con un segnale in ingresso pari a quello in figura 2.14a abbiamo trovato il legame tra ampiezza del segnale Vw di scrittura e stato interno x.
Nel grafico 2.16 è mostrato il risultato dello studio sulla programmazione dello stato interno del memristore. Si è ottenuto un metodo univoco ed efficace di programmazione.

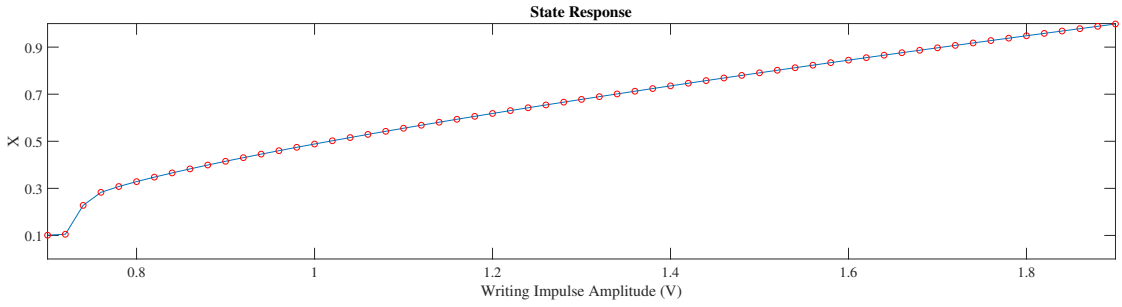


Figura 2.16: Evoluzione dello stato x al variare della tensione di scrittura

Tre appunti sono necessari:

- Il valore dello stato iniziale x_0 deve essere basso ma non zero (un valore ottimale sarebbe tra 0.065 e 0.25), ottenibile tramite un reset ad inizio programmazione. Tale segnale consiste in un impulso negativo abbastanza elevato da rientrare nel range di x_0 accettati.

- Valori di programmazione minori di 0.8V sono sconsigliabili in quanto il risultato è troppo dipendente da x_0.

- Valori di programmazione superiori a 1,8V sono sconsigliabili in quanto chiudere il canale conduttivo vicino all'apertura totale richiede tensioni molto alte.

I valori di stato intermedi rappresentano quindi il dominio di utilizzo del memristore in questa tesi.
Usando un tool di curve fitting si può ottenere una funzione matematica che, partendo dalle condizioni scelte in precedenza, dato un valore di stato interno richiesto ci fornisce la tensione dell'impulso necessaria ad ottenerlo. Con una funzione di tipo polinomiale di grado 3 otteniamo:

$$V(x) = p1 \cdot x^3 + p2 \cdot x^2 + p3 \cdot x + p4$$

con i valori dei coefficienti rappresentati in tabella:

$p1$	$p2$	$p3$	$p4$
-0.383	1.405	0.3333	0.5473

La funzione si dimostra precisa per i nostri scopi, come si può notare dalla figura 2.17, e presenta un errore quadratico medio di $4 \cdot 10^{-4}$:

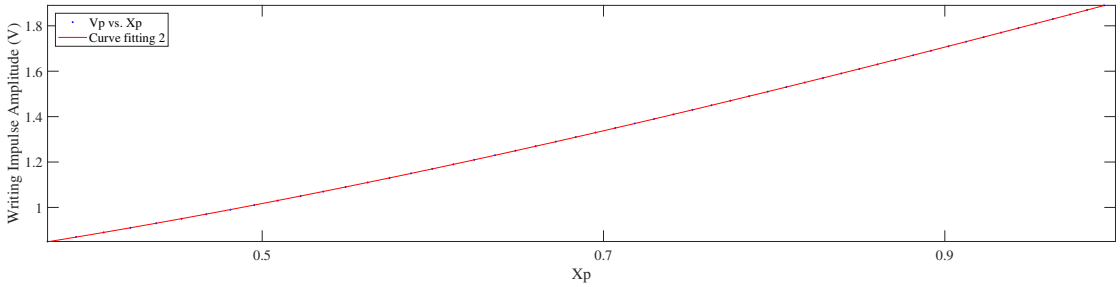
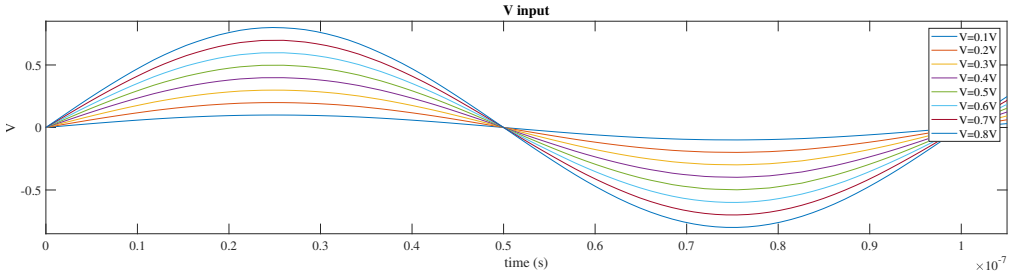


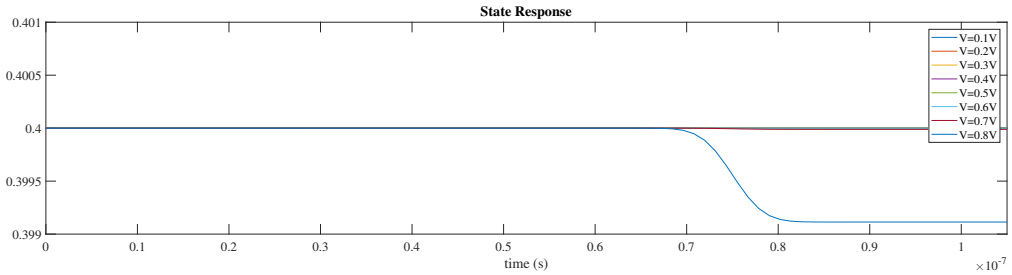
Figura 2.17: Ampiezza della tensione richiesta per la scrittura di un dato x

Lettura

Una volta programmato il memristore è necessario conoscere il range di ampiezza delle tensioni che possono scorrere senza modificarne lo stato. Si ricorda che tutti i test sono eseguiti usando il circuito in figura 2.7 e la tensione a cui ci si riferisce è quella in uscita dal generatore.
Il primo test di lettura è uno sweep di ampiezze differenti e frequenza fissa. In figura 2.18a si può notare un ingresso sinusoidale di ampiezza compresa tra 0.1 e 0.8 V e frequenza 10MHz, il memristore è stato configurato ad uno stato interno iniziale x_0 pari a 0.4.



(a) Sinusoidi di ampiezza $\in \{0.1, 0.8\}$ e frequenza 10MHz

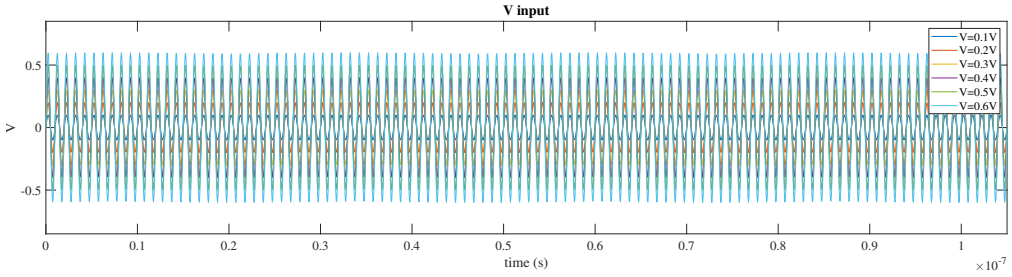


(b) Evoluzione dello stato x

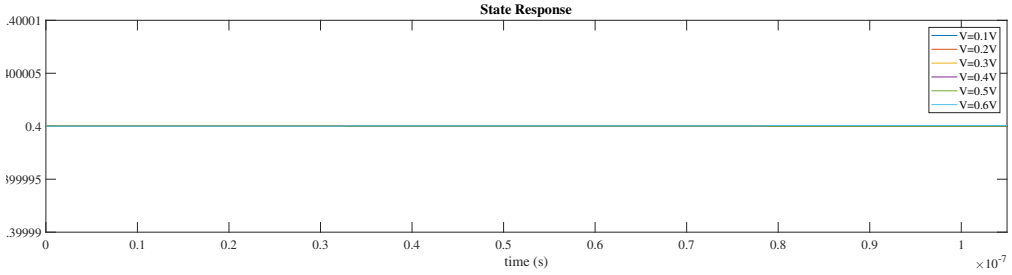
Figura 2.18: Variazione di x per potenziali segnali di lettura

Nell'analisi delle ampiezze permesse lo stato inizia a degradare per $V_{in} \geq 0.7$V (figura 2.18b) il che porta ad un intervallo di valori di lettura tra $\{-0.6, 0.6\}$.
Il secondo test mantiene le condizioni appena viste ma ne cambia la frequenza. Nella figura 2.19a vi è un esempio per frequenza pari a 1GHz. Non sembra esserci una correlazione tra il mantenimento dello stato interno del memristore x e la frequenza del segnale.
A questo punto si sono stabiliti i limiti dei segnali accettati in ingresso che, secondo il modello approssimato in uso, possono essere acquisiti senza che sia necessario ripetere la programmazione del memristore. Anche prolungando la durata del test i risultati non si discostano da quanto mostrato.

È il caso di ricordare che le ampiezze accettate dipendono anche dallo stato interno iniziale x_0, e se questo fosse ridotto si dovrebbe evitare una $V_{in} \geq 0.5$V. Tuttavia in questa tesi verrà utilizzato un range di x per il quale sono valide le scelte fatte.



(a) Sinusoidi di ampiezza $\in \{0.1, 0.6\}$ e frequenza 1GHz



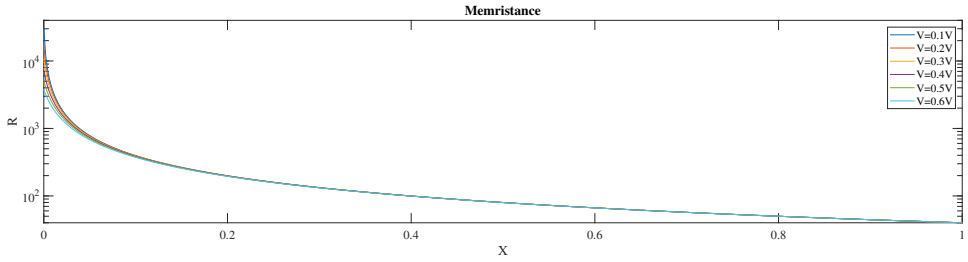
(b) Evoluzione dello stato x

Figura 2.19: Variazione di x per potenziali segnali di lettura

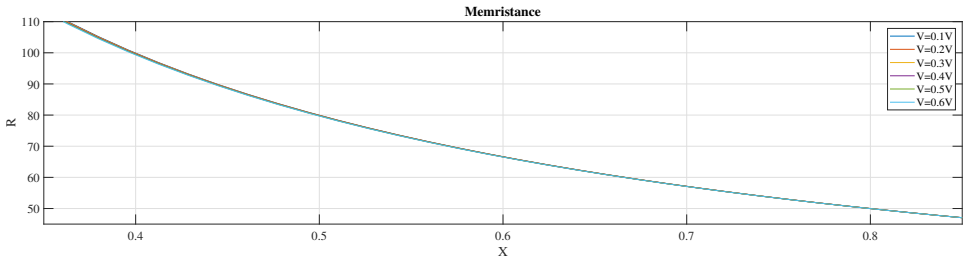
Legame Stato Interno-mem-resistenza

L'ultimo passo per l'utilizzo consapevole del memristore è trovare la relazione tra stato x e resistenza equivalente R, o mem-resistenza. Eseguendo un'analisi parametrica per $0 \leq x \leq 1$ con un tensione di lettura tra $0.1 \leq V_{in} \leq 0.6$ si ottiene il grafico semilogaritmico rappresentato in figura 2.20a.
La mem-resistenza mostra dipendenza dalla tensione in ingresso. Tuttavia, se ci si sofferma sull'intervallo $0.4 \leq x \leq 0.8$, ovvero l'intervallo selezionato in fase di programmazione, questa dipendenza è limitata. Si ha anche il vantaggio di poter selezionare più precisamente la mem-resistenza scelta, in quanto ad una piccola variazione di x non corrisponde ad un enorme variazione di R come accade nella parte sinistra del grafico. Si ricava dunque la funzione che collega la mem-resistenza e lo stato del memristore:

$$R = \frac{40}{x}$$



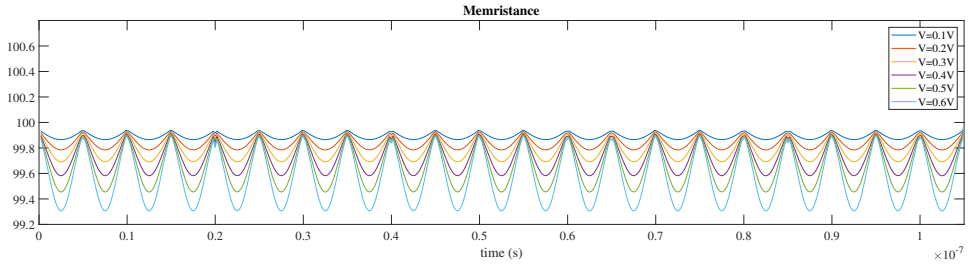
(a) Variazione di R per $0 \leq x \leq 1$



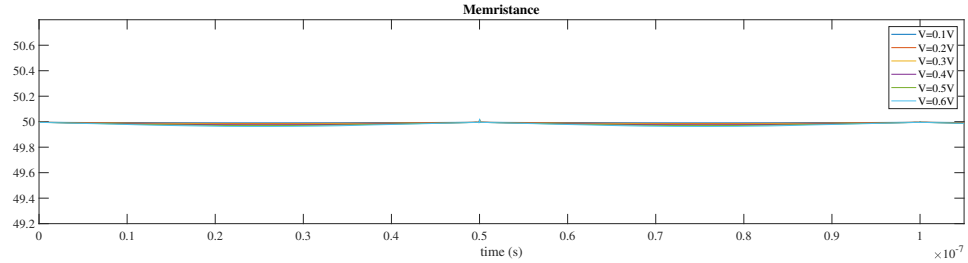
(b) Variazione di R per $0.4 \leq x \leq 0.8$

Figura 2.20: Legame tra mem-resistenza e stato interno del Memristore

Ulteriori test sulla risposta del memristore a segnali periodici con diverse ampiezze e frequenze la già citata dipendenza dall'ampiezza del segnale (crescente al diminuire di x) in figura 2.21a e 2.21b e nessuna dipendenza dalla frequenza.
Si ricorda che in qualità di memristore esteso (vedasi la classificazione nella sezione 1.3.1) questo tipo di relazione con l'ampiezza del segnale non è inaspettato, mentre l'indipendenza dalla frequenza fa pensare ad una semplificazione del modello rispetto al dispositivo reale.
Rimane infine da fare l'ultimo collegamento tra tensione di scrittura e mem-resistenza scelta: un'analisi parametrica ha permesso di tracciare il profilo di programmazione rappresentato in figura 2.22.
Unendo le informazioni delle ultime due prove si nota come le tensioni di programmazione per ottenere mem-resistenze comprese tra 100 e 50 Ω variano tra 0.85 e 1.5V.



(a) Variazione mem-resistenza per $x = 0.4$



(b) Variazione mem-resistenza per $x = 0.8$

Figura 2.21: Esempi di variazione di R dato il segnale in figura 2.18a

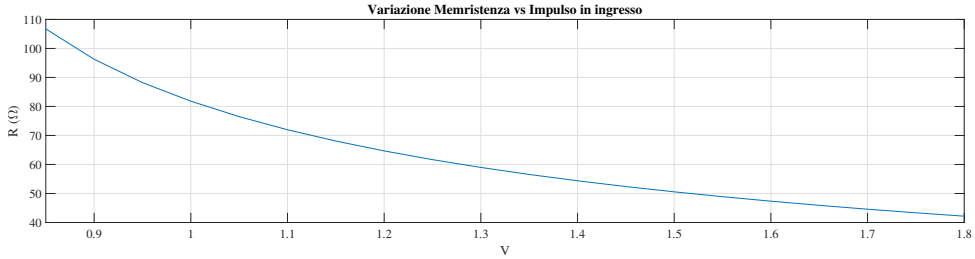


Figura 2.22: Variazione di R per 0.8V$\leq V_w \leq$ 1.8V

Durante questo studio del memristore TaO si sono determinate le modalità di scrittura e di lettura del memristore, ottenendo un range di resistenze e relativi stati usabili nel proseguimento di questa tesi.
La tabella riassume i risultati ricavati nelle pagine precedenti:

	x	V_w (V)	V_r (V)	R (Ω)	Durata V_w (ns)	x_0
min	0.4	0.85	0	50	5	0.065
max	0.8	1.8	0.6	100		0.25

dove x rappresenta lo stato del memristore, x_0 lo stato iniziale pre-programmazione, V_w è la tensione di scrittura, V_r è la tensione di lettura e R è la mem-resistenza.

2.3 Sviluppo del circuito

2.3.1 Prima bozza

Viene così alla luce la prima bozza di circuito, presentata in figura 2.23.

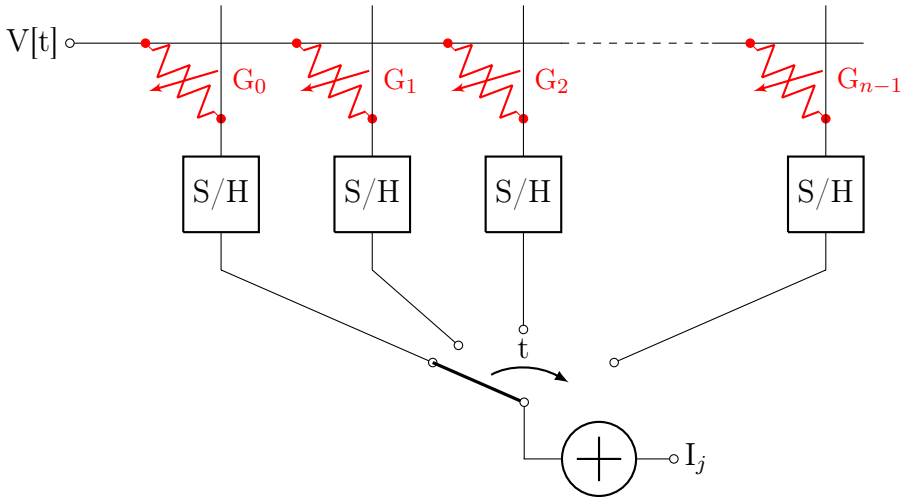


Figura 2.23: Primo schematico

Nel circuito possiamo notare un addizionatore che lavora in modalità *time interleaved*: il suo ingresso è infatti dato da un solo percorso alla volta. All'istante t_i il campione $V[t_i]$ sarà moltiplicato dal peso G_i e sommato all'uscita dell'istante t_{i-1}; nell'istante successivo t_{i+1} l'addizionatore aggiunge al risultato così ottenuto il segnale $V[t_{i+1}]$ campionato e moltiplicato per il peso G_{i+1} e così via. L'algoritmo è rappresentato nel diagramma 2.24.
Supponendo di partire in condizioni iniziali tali che l'uscita I[0] sia nulla, avremo:

$$\begin{aligned} I[t] &= G_0 \times V[t] \\ I[2t] &= G_0 \times V[t] + G_1 \times V[2t] = I[t] + G_1 \times V[2t] \\ I[xt] &= [...] \qquad\qquad\qquad\quad I[(x\text{-}1)t] + G_1 \times V[xt] \end{aligned}$$

Sono stati valutati anche schematici alternativi ma la scelta di minimizzare complessità e area del circuito ha portato questo schematico ad essere la base per il circuito finale. I Sample and Hold in corrente sono stati in seguito eliminati.
Uno degli obbiettivi primari di questa tesi è realizzare un convertitore a basso consumo, spingendo verso una riduzione ai minimi dell'hardware necessario ad effettuare la conversione. Molte delle scelte sui dettagli del circuito sono rimandate ad un secondo momento, ma è già chiaro il prodotto scalare tipico del Compressed Sensing.

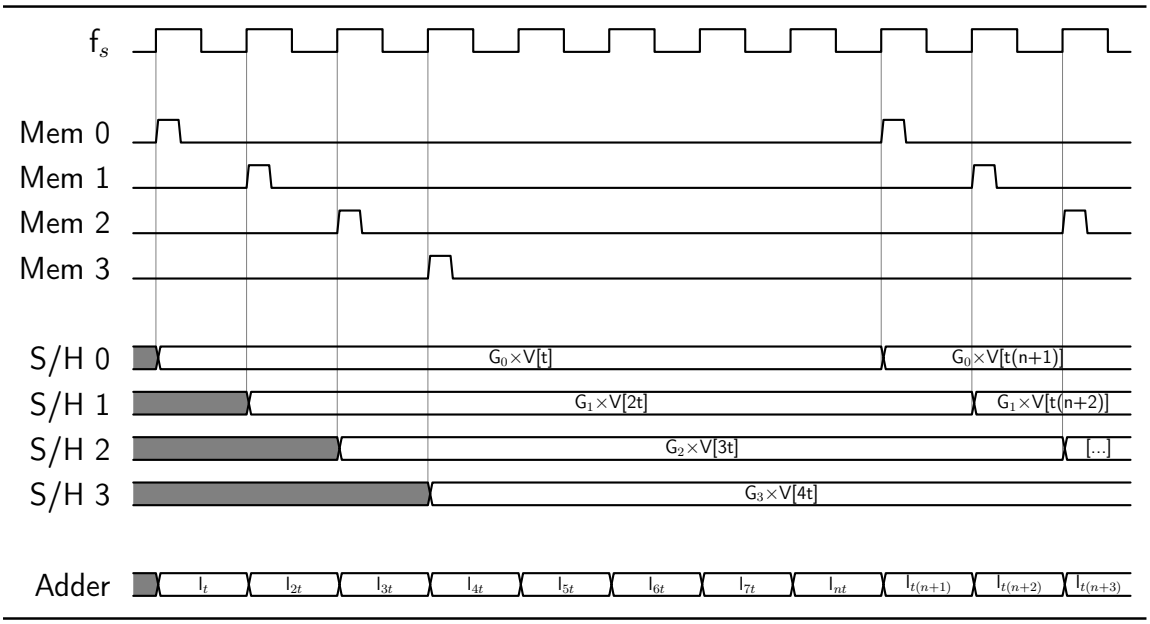


Figura 2.24: Algoritmo convertitore Analogico-Informativo

2.3.2 Da bozza a schematico

Da una bozza concettuale si passa ad uno schematico più preciso, rappresentato in figura 2.25. Dato un vettore (segnale in ingresso), il circuito deve calcolare un prodotto scalare in analogico.
Analizziamo il circuito partendo dall'ingresso fino all'uscita: il segnale V_{in} è caratterizzato dai limiti esposti nella sezione precedente. Può essere il segnale di scrittura o da acquisire, la selezione potrebbe avvenire con un multiplexer (ommesso).
Andando avanti si può notare come sia stato riprodotto il circuito di test utilizzato precedentemente per analizzare il memristore. È presente la resistenza R=70,1 Ω. Questo consente di mantenere la validità dei risultati ottenuti fino ad ora, sia in fase di scrittura che di lettura.
Successivamente è posta una riga di 8 memristori, e il capo negativo è connesso a massa virtuale o flottante quando non in uso. Questi dispositivi creano, con la loro mem-conduttanza, i pesi del prodotto scalare e ai fini della simulazione del comportamento sono già considerati programmati.

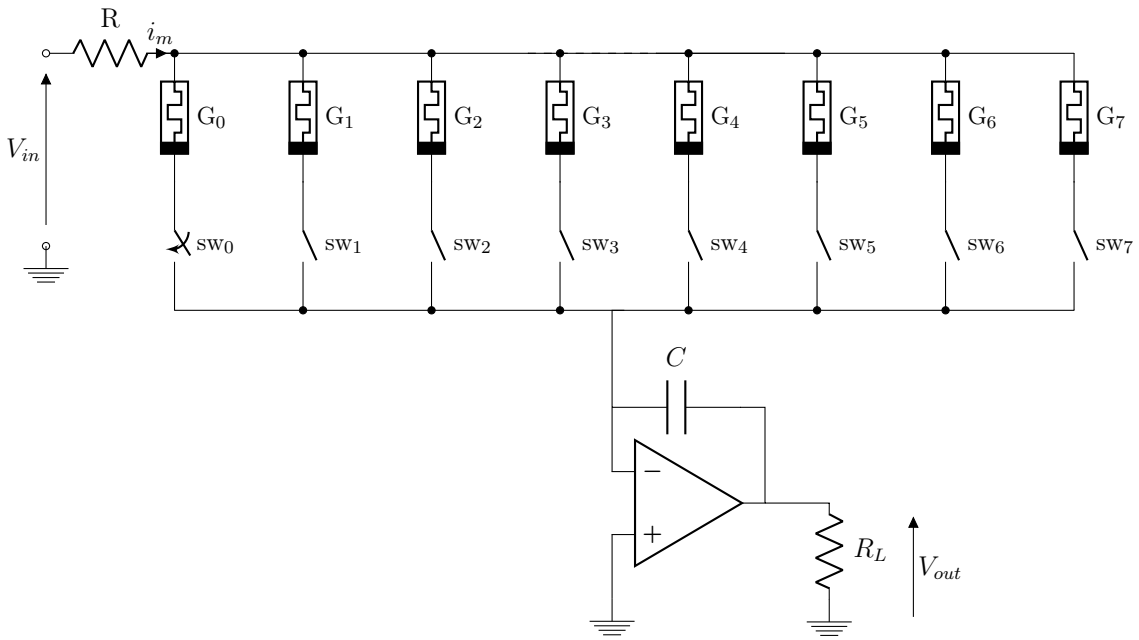


Figura 2.25: Prima implementazione circuitale

Ogni peso sarà dato da

$$p_n = \frac{1}{R + M_n}$$

nella quale M è la mem-resistenza nel percorso selezionato.
Grazie agli interruttori il circuito lavora in modalità time interleaved: solo un percorso è chiuso in un determinato istante, obbligando la corrente a scorrere attraverso la mem-conduttanza selezionata. Questi interruttori si occupano del campionamento del segnale e la loro frequenza totale costituisce quella di campionamento. Nella simulazione si è deciso di campionare a f_s =20 KHz, dunque ogni interruttore si chiude alla frequenza di f_s/n =2.5 KHz.
Il passo successivo è l'accumulazione, ad opera di un integratore ideale. Ai fini della simulazione è stato scelto un amplificatore operazionale con una banda e un guadagno che si possono considerare infiniti rispetto al resto del circuito. Non sono presenti tensioni o correnti di offset.
A completare l'integratore, la capacità C accumula una tensione proporzionale alla corrente in ingresso. Per questo motivo il suo valore è stabilito in base ai tempi di chiusura degli interruttori e alla dinamica di uscita desiderata. Inoltre è necessario considerare le condizioni iniziali, per semplicità la simulazione partirà sempre da una condensatore scarico.

2.3.3 Prima simulazione

Il test sul circuito è stato eseguito in contemporanea su Matlab e su LTSpice. Il primo ha fornito i risultati teorici attesi, mentre il secondo ha eseguito la stessa simulazione sul circuito in figura 2.25. Importare i risultati di LTSpice su Matlab ha reso possibile fare un confronto per valutare la bontà del lavoro svolto.
Il segnale in ingresso è una sinusoide di frequenza $\leq$ 10 KHz, campionata a frequenza pari a 20 KHz. In figura 2.26 è mostrato un esempio a 9.3 KHz.

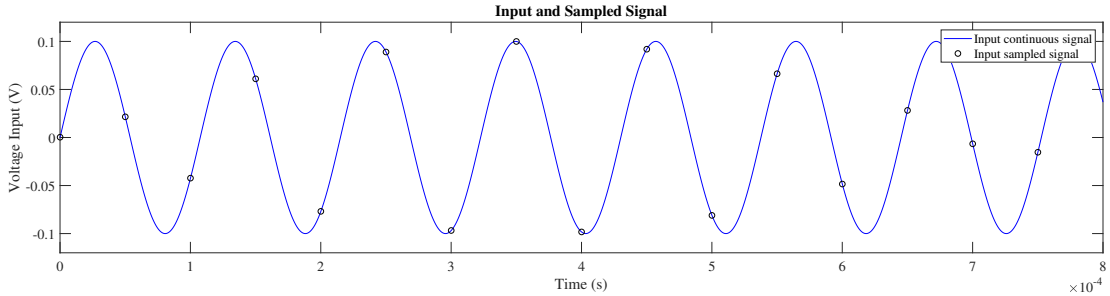
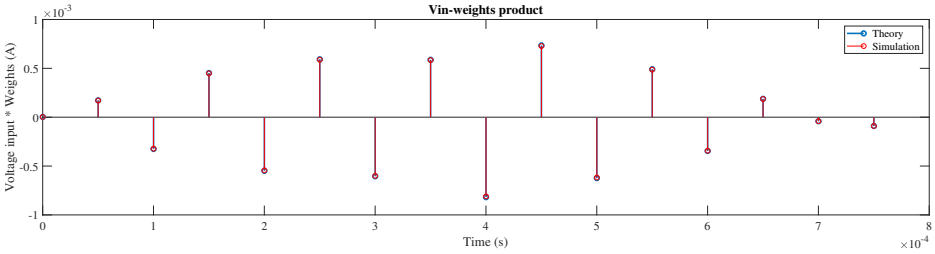


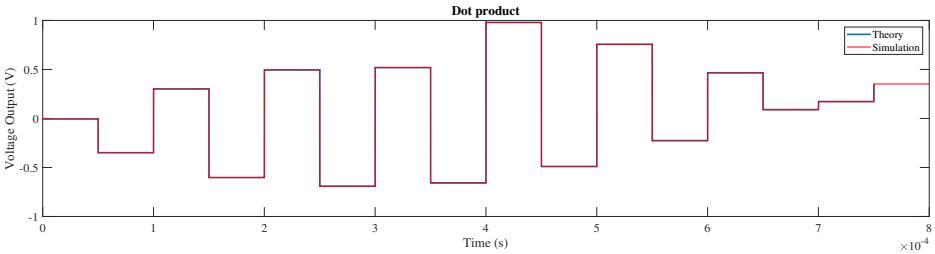
Figura 2.26: Esempio di segnale di test a 9.3 KHz

Gli interruttori si chiudono seguendo una funzione porta di durata pari a 100 ns con tempi di salita e discesa pari a 1 ns. La vicinanza di questo segnale alla funzione porta ideale è fondamentale ai fini della correttezza dei risultati. Tempi di salita e discesa lunghi allontanerebbero il segnale integrato dai risultati teorici.
Dato che nella realtà il primo campione non può venire all'istante t=0, se ne è tenuto conto anche su Matlab inserendo un ritardo nell'istante di campionamento nello script che esegue il calcolo teorico.
I valori dei pesi p è stato deciso in modo arbitrario, al solo fine di testare il corretto funzionamento del circuito. La resistenza è compresa tra 50 e 100 Ω, secondo i limiti decisi precedentemente.
Questo ha permesso di esaminare tra le altre cose la precisione della funzione di trasformazione tra resistenza e stato del memristore, in quanto nel circuito possiamo selezionare lo stato ma non la mem-resistenza equivalente.
Otteniamo così dei pesi $p : \{5.9 \leq p \leq 8.3\} \cdot 10^{-3}$.
Nel grafico 2.27a è riportata la corrente passante per C che rappresenta dunque l'elemento singolo del prodotto scalare $V(t_n) \times p_n$ utile ai fini di visualizzazione e comprensione del processo.
La figura successiva 2.27b mostra l'evoluzione della tensione di uscita, accumulata nella capacità C. I risultati teorici sono stati scalati di un fattore $-\frac{1}{C}$ per ottenere una sovrapposizione con la simulazione come conseguenza del principio di funzionamento dell'integratore utilizzato.
Si può vedere come il risultato sia preciso e per completezza d'informazione nel

grafico 2.28 sono riportati i residui sia della moltiplicazione che dell'integrazione, entrambi 2 o 3 ordini di grandezza più piccoli del valore nominale.

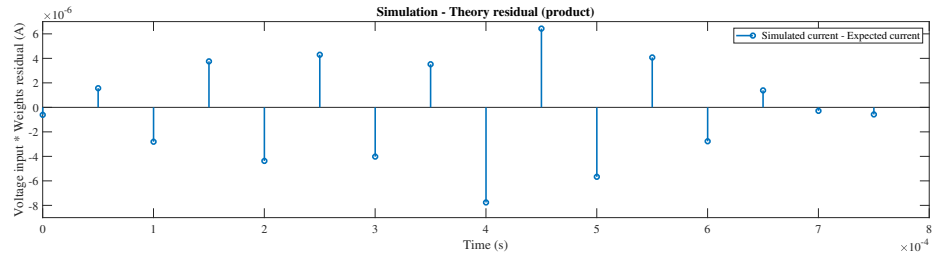


(a) Correnti passanti per la Capacità C

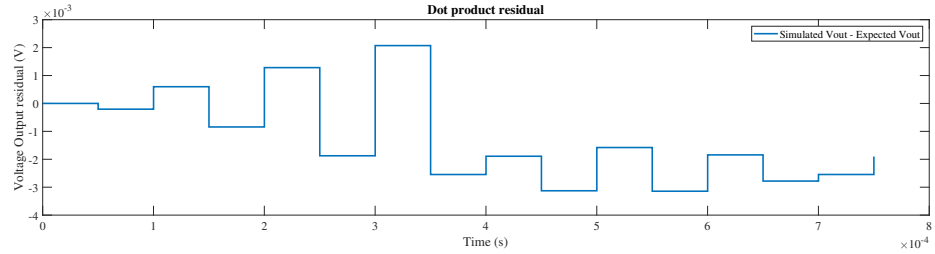


(b) Evoluzione della tensione di uscita

Figura 2.27: Risultati per il segnale rappresentato in figura 2.26



(a) Residuo della moltiplicazione



(b) Residuo del prodotto scalare

Figura 2.28: Residui dei risultati in figura 2.27

2.3.4 Correzione dell'offset

Normalmente avere un insieme di pesi di media nulla può essere una comodità, ma quando si utilizza un integratore diventa una necessità. A meno che non si utilizzi un segnale a media nulla, in un integratore non ideale la tensione di uscita è destinata a raggiungere presto la saturazione. Per questo motivo è stata aggiunta una correzione dell'offset al circuito iniziale.

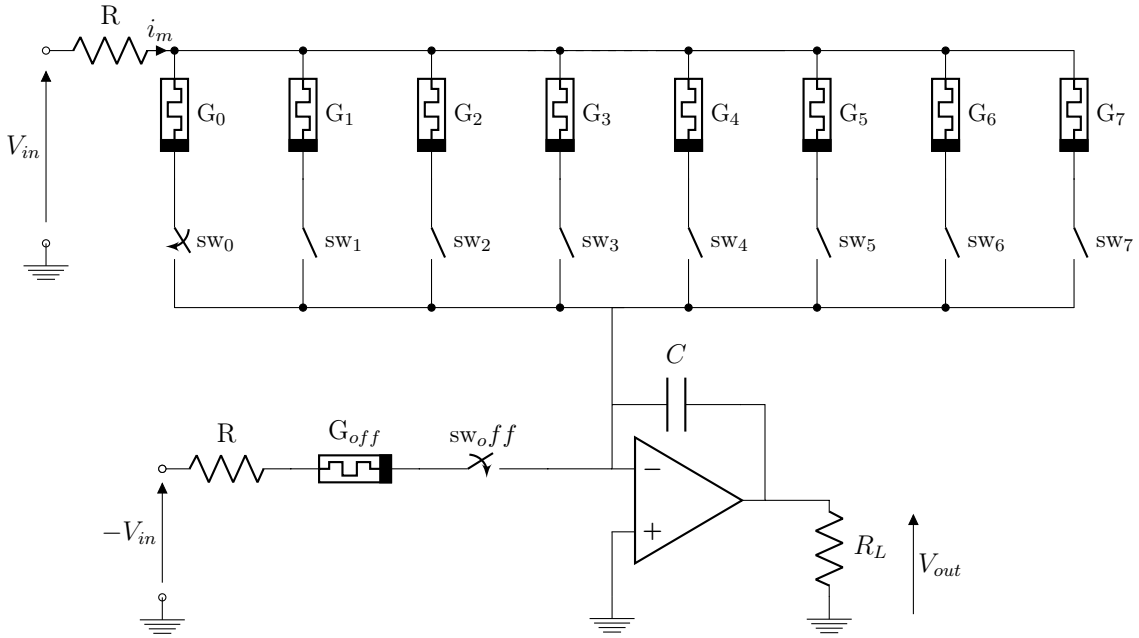
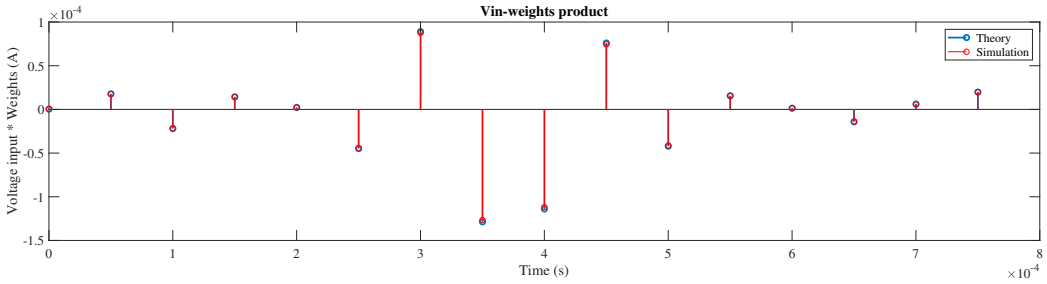


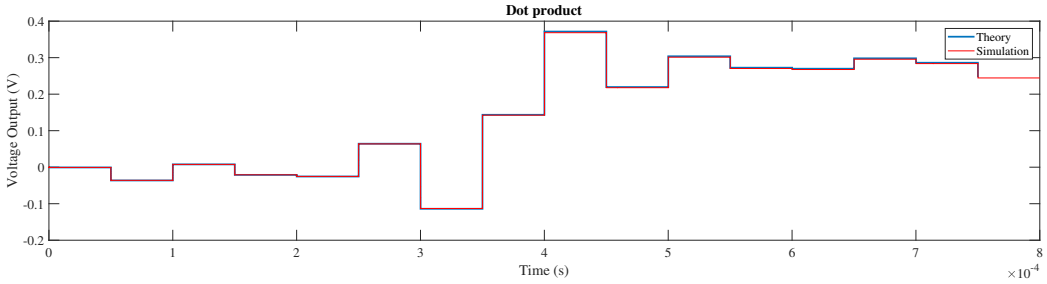
Figura 2.29: Seconda implementazione circuitale

Nello schematico 2.29 si può vedere una resistenza, un memristore e uno switch addizionali che ricevono in ingresso il segnale negato. Le resistenze R sono equivalenti a 70.1 Ω e lo switch di offset lavora ad una frequenza n volte più alta degli altri switch. In un periodo di campionamento il prodotto z sarà dato da $z = V(in) \times \bar{p} - V(in) \times p_{off} = V(in) \times (\bar{p} - p_{off})$. È evidente che per avere un peso nullo è necessario che $p_{off} = \bar{p}$ e di conseguenza $G_{off} = \bar{G}$.
In questo modo, programmando la G di offset al valore medio delle mem-conduttanza degli altri memristori, si può traslare il range dei pesi da $p : \{5.9 \leq p \leq 8.3\} \cdot 10^{-3}$ a $p : \{-1.2 \leq p \leq 1.2\} \cdot 10^{-3}$.

Utilizzando lo stesso ingresso visto nella simulazione precedente (figura 2.26) si può notare la riduzione dei risultati del prodotto a causa dei nuovi valori dei pesi, ma il risultato rimane accurato (figura 2.30).
Con questo accorgimento, anche nel caso avessimo un segnale a media non nulla (per esempio totalmente positivo o negativo), l'uscita dell'integratore rimarrebbe statisticamente intorno allo zero grazie ai pesi a media nulla.



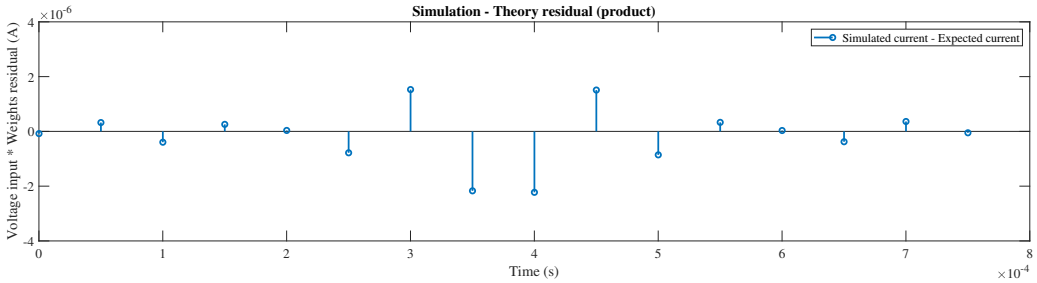
(a) Correnti passanti per la Capacità C



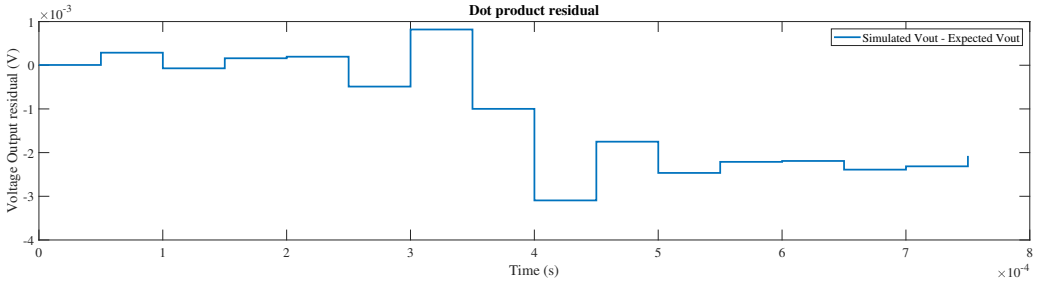
(b) Evoluzione della tensione di uscita

Figura 2.30: Risultati per il segnale rappresentato in figura 2.26

Dall'analisi dei residui si può notare anche che l'entità dell'errore relativo non ha subito cambiamenti (figura 2.31) confermando la validità della correzione apportata. Un esempio di codice Matlab utilizzato per effettuare il confronto può essere trovato nell'appendice A.2.8.



(a) Residuo della moltiplicazione



(b) Residuo del prodotto scalare

Figura 2.31: Residui dei risultati in figura 2.30

2.3.5 Applicazione con un segnale sparso

In questa sezione si mostrerà l'applicabilità del circuito elaborato in un contesto di CS. A questo fine è stato dato un segnale sparso in ingresso al circuito, il quale provvede all'acquisizione e conversione per la ricostruzione finale tramite il ricostruttore SPGL1 già citato nella sezione 1.1.4.
Il circuito è stato ampliato fino ad arrivare a 16 memristori, modificando inoltre la sorgente di V_{in} per fornire il segnale sparso in una base scelta arbitrariamente. Il resto è rimasto immutato.

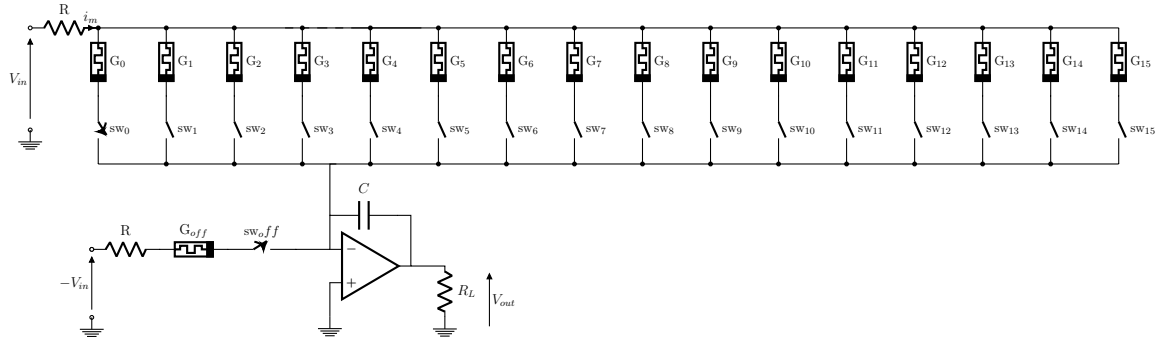


Figura 2.32: Implementazione circuitale finale

I passaggi matematici applicati sono i seguenti: si abbia una base di sparsità D e una rappresentazione sparsa del segnale xi. Il segnale in ingresso è dato da $x = \mathrm{D}xi$. Il CS richiede l'acquisizione con una matrice di sensing A con caratteristiche viste nella sezione 1.1.3 per ottenere delle misure y tramite $y = \mathrm{A}x$. Questa è l'operazione svolta dal circuito, ogni elemento-peso di una riga di A è replicato dalla mem-conduttanza G di un memristore.
Se il numero m delle misure y è adeguato, un ricostruttore può ricavare da y una stima $\hat{xi}$ dei coefficienti xi. Più sono le misure, più alta sarà la probabilità di ricostruzione corretta e l'acuratezza del risultato.
Riutilizzando la matrice di sparsità D si può ricavare la ricostruzione $\hat{x} = \mathrm{D}\hat{xi}$ del segnale originario x.

Integrazione tra LTSpice e Matlab

Con l'aumento graduale della complessità è incrementata l'integrazione tra il simulatore LTSpice e Matlab per dare maggiori responsabilità a quest'ultimo.
Si indicherà con N il numero di campioni in un periodo T e con m il numero di misure.
Su Matlab viene definita la matrice di sparsità D con dimensione N × N, è stata scelta una base di Fourier (DFT). Si applica l'operazione $x = Dxi$ per ottenere il segnale in ingresso. Il vettore di coefficienti xi, rappresentazione sparsa del segnale, e stato scelto arbitrariamente nei limiti stabiliti dall'analisi del memristore. In un segnale sparso la maggior parte di questi coefficienti sarà 0. In figura 2.33 si può visualizzare il processo appena descritto.
Anche il numero di misure m da effettuare è scelto arbitrariamente, in modo tale che $m<N$ ma sufficientemente grande da permettere la ricostruzione del segnale (come presentato nel teorema 4).
Matlab ha anche il compito di generare la matrice di sensing A con dimensione m × N, contenente i pesi (o funzioni di test) da applicare al segnale campionato. Questa matrice è generata con la funzione `Rand` di Matlab e vi sono applicati dei vincoli per fare rientrare i pesi nei parametri realizzabili dai memristori (calcolati nella sezione 2.3.4). Ogni riga di questa matrice contiene i pesi relativi ad una sola misura.
A questo punto Matlab esegue un ciclo `for` per ogni riga della matrice di sensing: il vettore di pesi p della misurazione in atto viene tradotto in un vettore di stati interni x dei memristori che acquisiranno una mem-induttanza G_i equivalente al peso p_i. In questa fase viene anche considerato e calcolato il memristore di offset (funzione `Rnd2C.m` in appendice A.2.3).
I coefficienti del segnale in ingresso (ampiezza e frequenza delle sinuoidi) e gli stati interni dei memristori, compreso quello di offset, sono passati a LTSpice attraverso

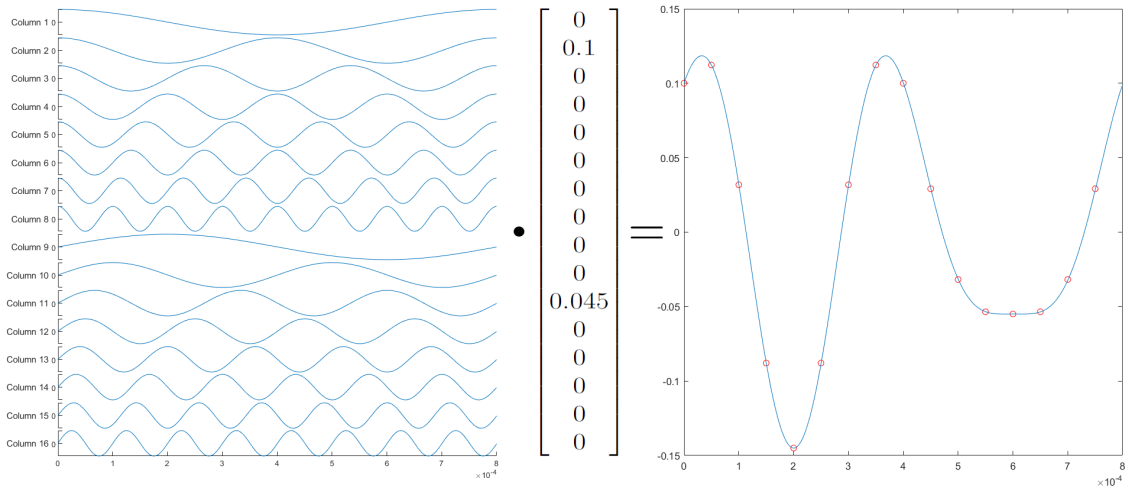


Figura 2.33: Rappresentazione della costruzione del segnale di input $x = Dxi$. A sinistra la Matrice di Sparsità D (base di Fourier). Ogni colonna rappresenta un seno o un coseno di ampiezza unitaria e frequenza crescente. Al centro il vettore di rappresentazione sparsa. A destra il segnale di ingresso risultante.

un file di testo. Ad occuparsi di questo lavoro, che deve essere fatto per ogni misurazione, è la funzione `Matlab2LTspice` riportata in appendice A.2.5.
Lo schematico su LTSpice è stato preparato a ricevere le informazioni contenute nel file di testo. Sono stati parametrizzati i valori relativi agli stati interni dei memristori e alla sorgente in ingresso. Al singolo generatore di tensione è stata sostituita una serie di 16 generatori di sinusoidi, uno per ogni colonna della matrice di sparsità, con frequenza e ampiezza parametrizzati e stabiliti dal file di testo prodotto nel passo precedente.
Ogni iterazione del ciclo `for` produce una misura y. Questo lavoro non è per forza sequenziale, ma può essere ottenuto in parallelo utilizzando una matrice di m righe, ognuna equivalente al circuito proposto.
L'ultimo passo spetta ancora a Matlab che acquisisce il vettore di misure y, e lo fornisce insieme alla matrice di sparsità D e la matrice di sensing A al ricostruttore SPGL1[12].
Il vettore $\hat{xi}$, ricostruzione del vettore xi, è infine moltiplicato per la matrice di sparsità D per ottenere il segnale ricostruito $\hat{x}$. Una funzione di `plot` permette di paragonare visivamente i segnali x e $\hat{x}$.
Infine è stato scritto del codice Matlab opzionale per visualizzare l'andamento del circuito durante il calcolo di una singola misura. Questo codice non è necessario per il funzionamento della simulazione, ma rimane utile in caso di problemi.

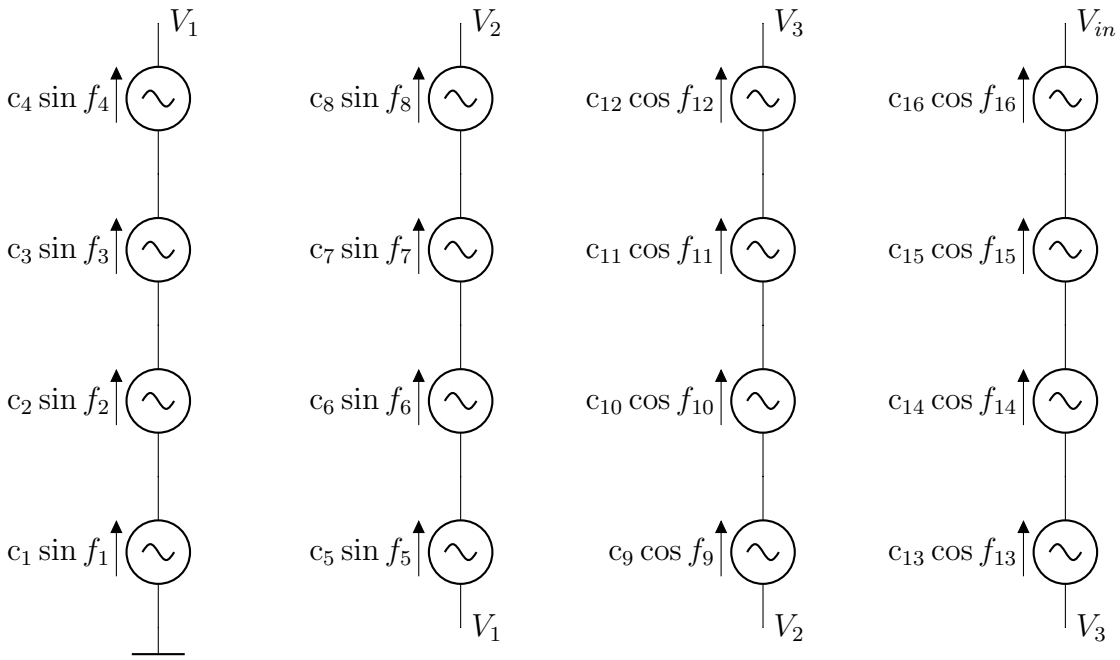


Figura 2.34: Implementazione automatica sorgente con base di Fourier

2.4 Risultati

Ai fini della valutazioni del circuito, come accaduto in precedenza, si è effettuato un confronto tra risultati ottenuti in un ambiente privo di errori come Matlab e i risultati ottenuti dalla simulazione.
Nel codice Matlab ci si riferisce alle variabili teoriche con le lettere presentate nel capitolo appena visto, mentre i dati ricavati dalla simulazione si distinguono per avere lo stesso nome con l'aggiunta di `Sim` alla fine (ad esempio `y` e `ySim`).

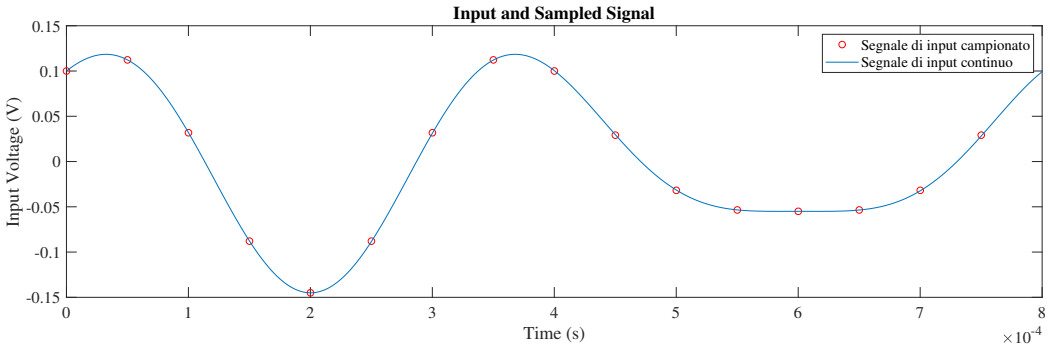


Figura 2.35: Esempio di segnale in ingresso continuo e campionato

Da una parte si ha quindi un segnale esattamente 2-sparso, campionato senza

errori, a cui su Matlab è applicata la procedura matematica appena descritta. Dall'altra parte si ha lo stesso segnale esattamente sparso, ma elaborato dal circuito presentato in questa tesi. Il circuito è semi-ideale, gli errori sono introdotti solo in alcuni passaggi: fra questi ricordiamo errori di campionamento (non più impulsivo), errori nella traduzione dei pesi (in programmazione dei memristori e a causa della Mem-conduttanza dipendente dalla tensione) e infine errori dovuti ai tempi di carica e scarica del condensatore dell'integratore. Nonostante questo, il circuito si rivela particolarmente preciso e in grado di garantire prestazioni molto vicine al calcolo ideale.

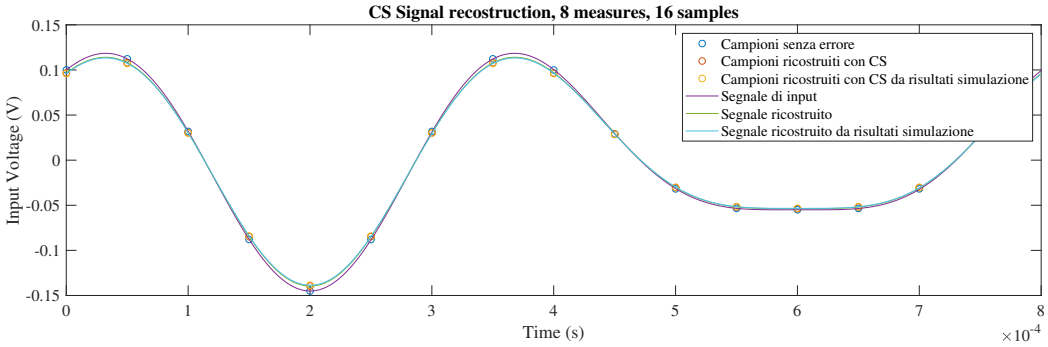


Figura 2.36: Confronto tra segnale originale x, segnale ricostruito dai calcoli teorico $\hat{x}$ e segnale ricostruito dai risultati della simulazione del circuito $\hat{x}_{Sim}$

È doveroso precisare che la valutazione sulla bontà del circuito va fatto confrontando fra loro i due segnali ricostruiti. Il confronto tra la ricostruzione e il segnale originale mostra invece le performance del Compressed Sensing e in quanto tale, non sono imputabili al circuito.

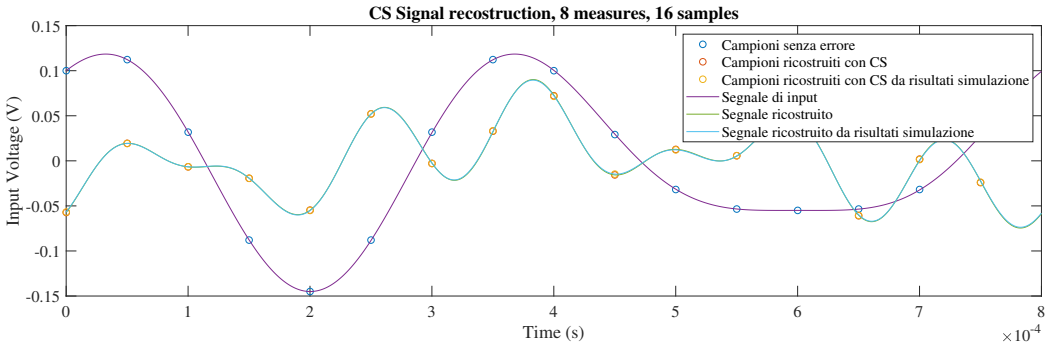


Figura 2.37: Esempio di ricostruzione fallita, sia con risultati teorici che simulati

Può capitare talvolta che la funzione SPGL1 fallisca la ricostruzione (figura 2.37).

Ciò accade perchè la matrice di Sensing A non è ottenuta verificando le proprietà necessarie all'ottenimento garantito di una ricostruzione valida, operazione lunga e pesante dal punto di vista computazionale. Come spiegato nella sezione 1.1.3 la scelta di una matrice random fornisce solo una certo livello di probabilità che le proprietà (in questo caso basta lo *spark*) vengano rispettate, probabilità che cresce all'aumentare del numero di misure m.
Tuttavia non è mai capitato che la ricostruzione sia fallita con i risultati simulati e riuscita con quelli teorici. In ogni simulazione svolta nel corso di questa tesi, le ricostruzioni sono fallite o riuscite insieme, segno che nessuna di queste è stata imputabile ad errori del circuito. Questo dimostra una certa robustezza nell'implementazione proposta.
Dunque per eliminare questa causa di errore è sufficiente agire sul numero di misure m, il cui incremento non solo riduce la possibilità di fallimento ma aumenta anche la fedeltà di ricostruzione, ottenendo un segnale ricostruito progressivamente più vicino all'originale come mostrato in figura 2.38.

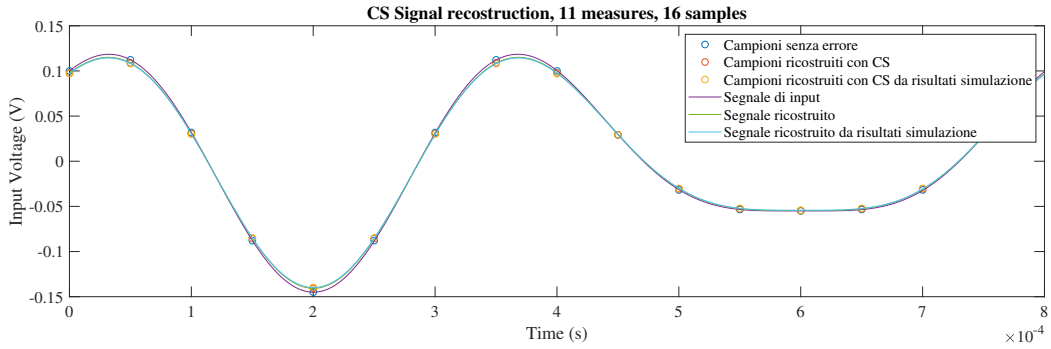


Figura 2.38: Ricostruzione con 11 misure

Sono state infine svolte due indagini statistiche sulle prestazioni del circuito. Nella prima si vuole mostrare quindi quanto la misura $\hat{y}$ ricavata dal circuito proposto sia vicina alla misura ideale y. Nella seconda si rende evidente la correlazione tra numero di misure m e accuratezza della ricostruzione, sia in termini di rapporto segnale rumore che in termini di percentuale di ricostruzioni riuscite.

2.4.1 Indagine sull'errore di misura

Si è verificata la risposta a 5 segnali sia sul circuito provvisto di correzione di offset che su quello sprovvisto. Il campione statistico è composto da 100 simulazioni, ognuna con un diverso set di pesi random. La durata delle simulazioni è stata 80ms, tale da permettere 100 misure sequenziali senza reset e verificare l'andamento

dell'errore in tempi prolungati. Nella tabella 2.1 è riportato l'errore quadratico medio normalizzato per le misure numero 1, 10 e 100.
I segnali testati, espressi in volt, sono i seguenti:

1. V=0.1
2. V= $0.1\sin(2\pi \cdot 5000)$
3. V= $0.1\sin(2\pi \cdot 5000) + 0.1$
4. V= $0.1\sin(2\pi \cdot 2500) + \cos(2\pi \cdot 3750)$
5. V= $0.1\sin(2\pi \cdot 2500) + 0.045\cos(2\pi \cdot 3750) + 0.1$

L'ingresso numero 4 rappresenta il segnale sparso di riferimento utilizzato precedentemente e successivamente. L'equazione utilizzata è la seguente:

$$NRMSE = \frac{\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(\hat{y} - y)^2}}{\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y)^2}} \tag{2.1}$$

in cui n rappresenta il numero di simulazioni effettuate.

NRMSE	1^a misura		10^a misura		100^a misure	
	no offset	offset	no offset	offset	no offset	offset
Segnale 1	0.0027	0.0069	0.0029	0.0067	0.0029	0.0068
Segnale 2	0.0140	0.0079	0.0074	0.0074	0.0070	0.0075
Segnale 3	0.0040	0.0073	0.0041	0.0072	0.0041	0.0073
Segnale 4	0.0084	0.0063	0.0061	0.0062	0.0062	0.0063
Segnale 5	0.0027	0.0076	0.0030	0.0072	0.0031	0.0073

Tabella 2.1: Errore quadratico medio normalizzato, 100 campioni, 5 segnali. No offset rappresenta il circuito privo di correzione di offset

Ci sono alcune considerazioni sui risultati: gli errori dei segnali con un bias (segnali 1, 3 e 5) si rivelano minori rispetto agli altri nel circuito privo di correzione di offset. Questo avviene perchè, data la natura dell'integratore, il segnale tende a crescere o decrescere più di un segnale a media nulla. Il fatto che la potenza del segnale si trovi al denominatore della formula 2.1 rende l'errore generalmente più piccolo. In realtà si sta lavorando con un integratore ideale con una dinamica d'uscita infinita ma in un contesto reale questa divergenza porterebbe velocemente l'uscita verso

saturazione e quindi il circuito senza correzione di offset non è comunque da preferire. Si può inoltre notare che il circuito con correzione di offset ottiene risultati piuttosto omogenei per tutti i tipi di segnale applicati anche se il memristore aggiuntivo di offset porta un'altra piccola causa d'errore.

2.4.2 Indagine sull'accuratezza della ricostruzione

Preso il segnale di test 2-*sparso* già indicato precedentemente, sono state eseguite un totale di 600 simulazioni, 100 per ogni numero di misure m compreso tra 8 e 13. Da questo campione sono stati calcolati l'ARSNR presentato nella sezione 1.1.4 e la percentuale di ricostruzioni riuscite. Si è scelto un RSNR pari a 17 dB come limite oltre il quale la ricostruzione può dirsi fedele all'originale. In tabella 2.2 sono riportati i valori ottenuti da valori ideali e i valori ottenuti dalla simulazione.

Misure m		8	9	10	11	12	13
ARSNR (dB)	Ideale	25.95	26.90	26.96	27.22	28.51	28.73
	Simul.	25.14	26.0313	25.84	26.07	27.33	27.33
Successo (%)	Ideale	75	88	93	98	100	99
	Simul.	75	87	93	98	99	100

Tabella 2.2: Risultati di ricostruzione con misure ideali e simulate

I risultati sono in linea con le aspettative, la differenza fra le prestazioni del circuito proposto e i risultati ideali rimane nell'ordine di 1dB.
La differenza di percentuale di successo nelle misure 9 e 11 sono dovute al limite rigido di 17dB che si è scelto per differenziare il segnale ricostruito correttamente da quello ricostrito non correttamente. Confrontando le singole coppie di ricostruzioni, ci si rende conto che la differenza è minima e può addirittura causare un risultato visto nella colonna 11, nella quale solo il rapporto segnale rumore del circuito simulato riesce a passare la soglia imposta. Questa piccola deviazione si potrebbe correggere con un campione di risultati più ampio, facilmente ottenibile con una potenza di calcolo maggiore.

2.5 Conclusioni

In questa tesi si è proposto con successo un convertitore analogico-informazione, attraverso un'implementazione hardware per il calcolo del prodotto scalare tramite l'uso di memristori.
Si sono descritte le considerazioni che hanno portato a prendere ispirazione dai circuiti specifici per le DNNs e a scegliere il memristore come dispositivo cruciale del circuito proposto.
Sono stati eseguiti test estensivi su un modello di memristore TaO, per stabilirne i modi d'uso adatti al nostro caso e i limiti da prendere in considerazione. Il circuito proposto è stato disegnato attorno a queste informazioni, e ampliato per tenere in considerazioni eventuali problematiche emerse in fase di test.
I risultati sono stati confrontati durante ogni fase dello sviluppo con i risultati attesi attraverso i programmi Matlab e LTSpice.
Sono stati infine simulati segnali sparsi per valutarne l'effettivo funzionamento in un contesto inerente al Compressed Sensing. I risultati sono stati in linea con le attese, permettendo di ricostruire il segnale in modo consistente alle aspettative.
Sono possibili degli sviluppi futuri, prima fra tutti l'estensione a matrice per il calcolo in parallelo (implicita in questa tesi) e la progressiva sostituzione di componenti ideali con le controparti reali, con relativa valutazione di efficienza nei consumi per l'applicabilità in contesti a basso o bassissimo consumo.

Appendice A

Codice di riferimento

A.1 Modello di TaO Memristor

Listato di modello di Memristor adatto alla simulazione con LTSpice, fornito dalla pubblicazione *Robust simulation of a TaO memristor model* [42].

A.1.1 Memristor.sp

```
*LTspice IV-coded netlist of the circuit realization
*of the continuous and differentiable variant
*of the original TaO memristor model
************************************************
* plus, minus: memristor terminals
* V(Y): voltage at node Y, modeling the state
.SUBCKT TaO_memristor plus minus
* memristor model parameters
.PARAMS B=1e-13 sigma_on=0.45 y_on=0.06 sigma_p=4e-5 A=1e-19 sigma_off=0.013
    ↪ y_off=0.4 beta=500 Gm=0.025 aa=7.2e-6 bb=4.7 rho=1e-3 k={1/50} Cini=1
* voltage-dependent current source
* modelling the memristor current
Gres plus minus value={V(plus ,minus)*(Gm*V(Y)+(1-V(Y))*aa*exp(bb*sqrt(abss(V(
    ↪ plus ,minus),rho ))))}
* voltage-dependent current source
* modelling the right hand side of the ODE
Gy 0 Y value={A*sinh(V(plus ,minus)/sigma_off)*exp(-pow(y_off/V(Y),2))*exp (1/(1+
    ↪  beta*I(Gres)*V(plus ,minus)))*stps(-V(plus ,minus),k)+B*sinh(V(plus ,
    ↪ minus)/sigma_on)*exp(-pow(V(Y)/y_on ,2))*exp(I(Gres)*V(plus ,minus)/
    ↪ sigma_p)*stps(V(plus ,minus),k)}
* capacitor, whose voltage models the state
Cint Y 0 {1n} IC={Cini}
* large resistance preventing convergence issues
Raux Y 0 100T
* step approximation
```

```
.func stps(x,p)={1/(1+ exp(-x/p))}
* modulus approximation
.func abss(x,p)={x*(stps(x,p)-stps(-x,p))}
.ends TaO_memristor
```

A.2 Matlab

Listati di test in Matlab. È compreso il calcolo dei valori attesi, l'acquisizione delle simulazioni in LTSpice e il confronto dei dati e diverse funzioni necessarie al funzionamento.

A.2.1 Res2C.m

Funzione di trasformazione da Memristenza desiderata a stato interno del memristore.

```
function y = Res2C(res)
    a=40;
    b=-1;
if ismatrix(res)
    y = a*res.^b;
else
    y = a*res^b;
end
```

A.2.2 C2Res.m

Funzione di trasformazione da stato interno del memristore a memristenza equivalente.

```
function y = C2Res(Cini)
    a=40;
    b=-1;
    y=a*Cini^b;
end
```

A.2.3 Rnd2C.m

Conversione della matrice di Sensing Random A in matrice di stato interno dei memristori tali che la mem-conduttanza sia pari al corrispondente peso in A.

```
function y = Rnd2C(A)
if ~ismatrix(A)
    error('Input must be a Matrix')
end
    B=A+1/140.8; %empirico
```

```
    y=Res2C(1./B-70.1);
end
```

A.2.4 ContSign.m

Costruzione di una base di Sparsità DFT alternativa con un numero maggiore di elementi per la rappresentazione del segnale continuo

```
function [y,t]= ContSign(a, b, c)
%a=N (dimensione di D), b=tempo di simulazione, c=campioni segnale continuo
switch nargin
    case 3
        N=a;
        M=c;
        tMax=b; %Simulation time
    case 2
        N=a;
        tMax=b;
        M=1000000;
    case 1
        N=a;
        tMax=0.0008;
        M=1000000;
end
    t=linspace(0,tMax,M);
    G=zeros(N, M);
    %Costruzione Base di Fourier con c punti
for k=1:N/2
    for l=0:M-1
        G(k,l+1)=cos((2*pi*k*l)/M);
        G(k+N/2,l+1)=sin((2*pi*k*l)/M);
    end
end
y=G';
end
```

A.2.5 Matlab2LTspice.m

Costruzione del file di testo `init.txt` con i parametri necessari alla simulazione in LTSpice

```
function Matlab2LTspice(C, xi, tMax, Coffset, N)
fid = fopen( 'init.txt', 'wt' );
%parameters for memristor internal state, vector C
for i = 1:N
    fprintf( fid, '.param cini%d %f\n', i, C(i));
end
%parameters for Sinusoid amplitude, vector xi
```

```
for j = 1:N
    fprintf( fid, '.param x%d %f\n', j, xi(j));
end
%parameters for Sine and Cosine freq, vector fq
for k = 1:N/2
    fq=k/(tMax); %correggere
    fprintf( fid, '.param f%d %f\n', k, fq);
    fprintf( fid, '.param f%d %f\n', k+N/2, fq);
end
%Offset Memristor state
fprintf( fid, '.param cini21 %f\n', Coffset);
fclose(fid);
end
```

A.2.6 LTspice2Matlab.m

Acquisizione del file di simulazione LTSpice su Matlab. Funzione creata da Paul Wagner e disponibile online[43]

A.2.7 runme.m

Codice centrale che riunisce tutte le funzioni introdotte in appendice. Il suo funzionamento è descritto nella sezione 2.3.5.

```
clc
close all
clear

addpath('spgl1-1.9'); %path del risolutore
opts = spgSetParms('verbosity',0);

% parametri
N = 16; %numero memristori
M = 8; %numero misure
tMax=0.0008; %tempo per una misura
 ton=102*10^-9; %durata porta di campionamento
tSamp=0:tMax/N:tMax; %periodo di campionamento
tSampC = tSamp + (ton/2); %traslazione campionamento a meta porta di
    ↪ campionamento
tSampC(end)=[]; %l'ultimo elemento e eliminato - N elementi rimanenti

% base di sparsita: D e la base di fourier (sin + cos)
D=zeros(N);
for k=1:N/2
    for l=0:N-1
        D(k,l+1)=cos((2*pi*k*l)/N);
        D(k+N/2,l+1)=sin((2*pi*k*l)/N);
    end
```

```
end
D=D';

%Creazione base di sparsita e tempi associati con piu campioni per
    ↪ visualizzazione come segnale continuo
[G, t]=ContSign(N, tMax, 1088);

% generazione segnale - xi e la sua rappresentazione sparsa
xi = zeros(N,1);
xi(2) = 0.1;
xi(11) = 0.045;
x = D*xi;
xCont=G*xi; %segnale continuo

%plot segnale e sample
figure
plot(tSampC, x, 'or', t, xCont);
legend('Segnale di input campionato', 'Segnale di input continuo')
title('Input and Sampled Signal');
ylabel('Input Voltage (V)');
xlabel('Time (s)');
set(gca,'Units','normalized','FontUnits','points','FontWeight','normal','FontSize
    ↪ ',15,'FontName','Times')

Coffset=Res2C(140.8-70.1); %Calcolo stato del memristor di offset data la
    ↪ resistenza scelta

%calcolo range matrice di sampling
Range=1/120.1-1/170.1;
d=Range/2;

% Matrice di sampling NxM casuale nei valori replicabili con i memristori
A = rand(M,N)*Range-d;

ySim=zeros(M,1); %inizializzo vettore di misure risultanti dalla simulazione

%calcolo vettore di misure date dalla simulazione
for i=1:M
    C=Rnd2C(A(i,:)); %crea vettore di stati dei memristori per replicare i pesi
        ↪ dati dalla riga della matrice A
    Matlab2LTspice(C, xi, tMax, Coffset, N); %scrive parametri su init.txt per
        ↪ LTSpice

    % now run LTspice
    !"C:\Program Files\LTC\LTspiceXVII\XVIIx64.exe" -Run -b Circuito_semiideale.
        ↪ asc
        pause(3)%time for simulation

    %Importa simulazione
```

```
    warning('off','MATLAB:iofun:UnsupportedEncoding');
    SimResult=LTspice2Matlab('Circuito_semiideale.raw');
    warning('on','MATLAB:iofun:UnsupportedEncoding');

    %Trova la variabile corretta
    Vvout= find(ismember(SimResult.variable_name_list, 'V(vout)'));

    %Importa and riscala vettori
    VoutSim=SimResult.variable_mat(Vvout,end); %Segnale di output della
        ↪ simulazione, ultimo elemento
    VoutSim=-VoutSim/2000; %eliminazione scaling dell'integratore (dipende dalla
        ↪ capacita)
    ySim(i)=VoutSim; %salva risultato in un vettore di misure (M componenti)

end

% vettore di misure
y = A*x;

% utilizzo del ricostruttore
xihat = spg_bpdn(A*D/Range,y/Range,0.01,opts);
xihatSim = spg_bpdn(A*D/Range,ySim/Range,0.01,opts); %ricostruzione da
    ↪ simulazione

% moltiplicazione per la matrice di sparsita per ottenere il segnale
% ricostruito
xhat = D*xihat;
xhatSim = D*xihatSim; %risultato da simulazione
xhat_resampled= G*xihat; %rappresentazione continua
xhatSim_resampled = G*xihatSim; %risultato da simulazione(continuo)

% plot risultati
figure
plot(tSampC, [x,xhat,xhatSim], 'o', t,[xCont, xhat_resampled, xhatSim_resampled])
title('CS Signal recostruction, 8 measures, 16 samples');
ylabel('Input Voltage (V)');
xlabel('Time (s)');
xlim([0 tMax]);
set(gca,'Units','normalized','FontUnits','points','FontWeight','normal','FontSize
    ↪ ',15,'FontName','Times')
legend('Campioni senza errore', 'Campioni ricostruiti con CS', 'Campioni
    ↪ ricostruiti con CS da risultati simulazione','Segnale di input','Segnale
    ↪ ricostruito','Segnale ricostruito da risultati simulazione')
```

A.2.8 test.m

Esempio di codice aggiuntivo per la verifica del funzionamento del circuito durante la misurazione. Permette la visualizzazione delle grandezze quali corrente e tensione,

confrontate con i valori attesi.

```
clc
clear
close all

N=16; %number of memristors
Iout=zeros(1,N); %preallocate Iout
prod=zeros(1,N); %vector of product, needed for debug
Goffset=(1./140.8); %offset
Coffset=Res2C(140.8-70.1);
tMax=0.0008;
ton=102*10^-9; %sampling pulse time
tSamp=0:tMax/N:tMax;
tSampC = tSamp + (ton/2); %sampling in the mid of pulse
tSampC(end)=[]; %drop last element

%Creazione base di sparsita con piu campioni(segnale continuo)
[G, t]=ContSign(N, tMax, 100000);

% base di sparsita: D e la base di fourier (sin + cos)
D=zeros(N);
for k=1:N/2
    for l=0:N-1
        D(k,l+1)=cos((2*pi*k*l)/N);
        D(k+N/2,l+1)=sin((2*pi*k*l)/N);
    end
end
D=D';

% generazione segnale - xi e la sua rappresentazione sparsa
xi = zeros(N,1);
xi(2) = 0.07;
xi(11) = 0.05;
x = D*xi;
xCont=G*xi; %segnale continuo

%plot segnale e sample
figure
plot(tSampC, x, 'or', t, xCont);
grid on;
legend('Segnale di input campionato', 'Segnale di input continuo')
title('Input and Sampled Signal');
ylabel('Input Voltage (V)');
xlabel('Time (s)');
set(gca,'Units','normalized','FontUnits','points','FontWeight','normal','FontSize
    ↪ ',15,'FontName','Times')


%calcolo range matrice di sampling
```

```
Range=1/120.1-1/170.1;
d=Range/2;

% Creazione singola riga della matrice random per test funzionamento
A = rand(1,N)*Range-d;
Pesi=A(1,:);
C=Rnd2C(Pesi);

%Main program
for i=1:N
    prod(i)=x(i).*Pesi(i);

    if i>1
        sum=Iout(i-1)+prod(i);
    else
        sum=prod(i);
    end
    Iout(i)=sum;
end
m=sprintf('%.4f ', Pesi);
h=sprintf('%.4f ', Goffset);
o=sprintf('%.4f ', Coffset);
fprintf('Il segnale passa per un set di %d memristori con pesi pari a %s\n', N, m
    ↪ )
fprintf('Offset implementato con un memristore di memduttanza pari a %s siemens e
    ↪  uno stato pari a %s \n', h,o)

%Create parameter for simulation on LTSpice
Matlab2LTspice(C, xi, tMax, Coffset, N);

% now run LTspice
!"C:\Program Files\LTC\LTspiceXVII\XVIIx64.exe" -Run -b Circuito_semiideale.asc
 pause(2)%time for simulation

%Import simulation
warning('off','MATLAB:iofun:UnsupportedEncoding');
SimResult=LTspice2Matlab('Circuito_semiideale.raw');
warning('on','MATLAB:iofun:UnsupportedEncoding');

%Find correct variables
IC1= find(ismember(SimResult.variable_name_list, 'I(C1)'));
Vvin= find(ismember(SimResult.variable_name_list, 'V(vin)'));
Vvout= find(ismember(SimResult.variable_name_list, 'V(vout)'));

%Import and rescale vectors
VinSim=SimResult.variable_mat(Vvin,:); %Simulation Input Signal
VoutSim=SimResult.variable_mat(Vvout,:); %Simulation Output Signal
timeSim=SimResult.time_vect; %Simulation time vector
```

```
ISim=SimResult.variable_mat(IC1,:); %Simulation Current through C (Integrator
    ↪ Capacitance)

%Interpolation to have vector of the same lenght
ISimInter=interp1(timeSim,ISim,tSampC,'nearest'); %Sample reduction (square in
    ↪ simulation vs impulse in theory)
deltaI=ISimInter+prod; %Difference between product in simulation and theory, for
    ↪ debug purpose
Tint=tSampC+50*10^-9; %shift the time vector to interpolate after the change of
    ↪ value
VoutInter=interp1(timeSim,VoutSim,Tint,'next'); %Sample reduction of Vout
Vout=Iout.*2000; %Dividing I by C to obtain Voltage
deltaV=VoutInter+Vout; %Difference between Output voltage (Simulation and theory)

%plot segnali ed errori
figure;
stairs(tSampC,-Vout,'linewidth',2);
hold on
stairs(timeSim,VoutSim,'r');
title('Dot product');
ylabel('Voltage Output (V)');
xlabel('Time (s)');
xlim([0 tMax]);
grid on;
legend('Theory','Simulation');

figure;
stem(tSampC,prod,'linewidth',2);
hold on;
xlim([0 tMax]);
stem(tSampC, -ISimInter,'r');
title('Vin-Memductance product')
ylabel('Voltage input * Memductance (A)');
xlabel('Time (s)')
grid on;
legend('Theory','Simulation');

figure;
stem(tSampC,deltaI,'linewidth',2);
title('Simulation - Theory error (product)')
ylabel('Voltage input * Memductance (A)');
xlabel('Time (s)');
xlim([0 tMax]);
legend('Difference between current though Memristors in simulation and theory');

figure;
stairs(tSampC,deltaV,'linewidth',2);
title('Dot product error');
ylabel('Voltage Output [V]');
```

```
xlabel('Time [s]');
xlim([0 tMax]);
grid on;
legend('Difference between output voltage in simulation and theory');
```

Bibliografia

[1] Candes, E., Romberg, J., Tao, T.: Stable Signal Recovery from Incomplete and Inaccurate Measurements. Technical report (2005)

[2] Donoho, D.: Compressed sensing. IEEE Transactions on Information Theory **52**(4) (apr 2006) 1289–1306

[3] Eldar, Y.C., Kutyniok, G.: Compressed Sensing: Theory and Applications, Cambridge University Press. (2012)

[4] Davenport, M.A., DeVore, R.A., Koss, W.E.: Random Observations on Random Observations: Sparse Signal Acquisition and Processing. Technical report (2010)

[5] Jayram, T.S., Woodruff, D.: Optimal Bounds for Johnson-Lindenstrauss Transforms and Streaming Problems with Sub-Constant Error. Technical report (2011)

[6] Krahmer, F., Ward, R.: New and Improved Johnson–Lindenstrauss Embeddings via the Restricted Isometry Property. SIAM Journal on Mathematical Analysis **43**(3) (jan 2011) 1269–1281

[7] Cohen, A., Dahmen, W., DeVore, R.A.: Compressed sensing and best k-term approximation. Technical report (2006)

[8] DeVore, R.A.: Deterministic constructions of compressed sensing matrices. Journal of Complexity **23**(4-6) (aug 2007) 918–925

[9] Indyk, P.: Explicit Constructions for Compressed Sensing of Sparse Signals. In: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics Philadelphia (2007) 30–33

[10] Mallat, S., Zhifeng Zhang: Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing **41**(12) (1993) 3397–3415

[11] Blumensath, T., Davies, M.E.: Iterative hard thresholding for compressed sensing. Applied and Computational Harmonic Analysis **27**(3) (nov 2009) 265–274

[12] Van, E., Friedlander, M.P.: PROBING THE PARETO FRONTIER FOR BASIS PURSUIT SOLUTIONS. Technical report (2008)

[13] Becker, S., Bobin, J., Candes, E.: NESTA: A Fast and Accurate First-order Method for Sparse Recovery. (apr 2009)

[14] Grant, M.C., Boyd, S.P.: Graph Implementations for Nonsmooth Convex Programs. Technical report

[15] Combettes, P.L., Pesquet, J.C.: A Proximal Decomposition Method for Solving Convex Variational Inverse Problems. (jul 2008)

[16] Mangia, M., Pareschi, F., Cambareri, V., Rovatti, R., Setti, G.: Adapted Compressed Sensing for Effective Hardware Implementations. 1 edn. Springer International Publishing, Bologna (2018)

[17] Satat, G., Tancik, M., Raskar, R.: Lensless Imaging with Compressive Ultrafast Sensing. Technical report (2016)

[18] Wright, J., Yang, A., Ganesh, A., Sastry, S., Yi Ma: Robust Face Recognition via Sparse Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence **31**(2) (feb 2009) 210–227

[19] Lustig, M., Donoho, D., Pauly, J.M.: Sparse MRI: The application of compressed sensing for rapid MR imaging. Magnetic Resonance in Medicine **58**(6) (dec 2007) 1182–1195

[20] Massa, A., Rocca, P., Oliveri, G.: Compressive Sensing in Electromagnetics - A Review. IEEE Antennas and Propagation Magazine **57**(1) (feb 2015) 224–238

[21] Baraniuk, R.: Compressive Sensing [Lecture Notes]. IEEE Signal Processing Magazine **24**(4) (jul 2007) 118–121

[22] Ganesh, A., Wagner, A., Zhou, Z., Yang, A.Y., Ma, Y., Wright, J.: Face Recognition by Sparse Representation. Technical report

[23] Sze, V.: Designing Hardware for Machine Learning: The Important Role Played by Circuit Designers. IEEE Solid-State Circuits Magazine **9**(4) (2017) 46–54

[24] Li, F.F., Johnson, J., Yeung, S.: Stanford University CS231n: Convolutional Neural Networks for Visual Recognition

[25] Le Cun, Y., Jackel, L., Boser, B., Denker, J., Graf, H., Guyon, I., Henderson, D., Howard, R., Hubbard, W.: Handwritten digit recognition: applications of neural network chips and automatic learning. IEEE Communications Magazine **27**(11) (nov 1989) 41–46

[26] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, Yifan Gong, Acero, A.: Recent Advances in Deep Learning for Speech Recognition at Microsoft. (2013) 0–4

[27] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems. Curran Associates Inc. (2012) 1097—-1105

[28] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. (sep 2014)

[29] Chua, L.: Memristor-The missing circuit element. IEEE Transactions on Circuit Theory **18**(5) (1971) 507–519

[30] Chua, L.: If it's pinched it's a memristor. Memristors and Memristive Systems **9781461490** (2014) 17–90

[31] Chua, L.: Everything you wish to know about memristors but are afraid to ask. Radioengineering **24**(2) (2015) 319–368

[32] Biolkova, V., Biolek, D., Biolek, Z.: Pinched hysteretic loops of ideal memristors, memcapacitors and meminductors must be 'self-crossing'. Electronics Letters **47**(25) (dec 2011) 1385–1387

[33] Chua, L., Kang, S.M.: Memristive devices and systems. Proceedings of the IEEE **64**(2) (1976) 209–223

[34] Hu, M., Graves, C.E., Li, C., Li, Y., Ge, N., Montgomery, E., Davila, N., Jiang, H., Williams, R.S., Yang, J.J., Xia, Q., Strachan, J.P.: Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine. Advanced Materials **30**(9) (2018) 1–10

[35] Lian, X., Wang, M., Rao, M., Yan, P., Yang, J.J., Miao, F.: Characteristics and transport mechanisms of triple switching regimes of TaO _x memristor. Applied Physics Letters **110**(17) (apr 2017) 173504

[36] Van Der Plas, G., Verbruggen, B.: A 150MS/s 133uW 7b ADC in 90nm digital CMOS Using a Comparator-Based Asynchronous Binary-Search sub-ADC. In: 2008 IEEE International Solid-State Circuits Conference. Digest of Technical Papers - IEEE International Solid-State Circuits Conference, IEEE (2008) 242–243

[37] Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S., Srikumar, V.: ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In: Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016, United States, Institute of Electrical and Electronics Engineers Inc. (2016) 14–26

[38] Sze, V., Chen, Y.H., Tien-Ju, Y.: Efficient Processing of Deep Neural Networks: A Tutorial and Survey. **105**(12) (2017) 2295–2329

[39] Eryilmaz, S.B., Joshi, S., Neftci, E., Wan, W., Cauwenberghs, G., Wong, H.S.: Neuromorphic architectures with electronic synapses. Proceedings - International Symposium on Quality Electronic Design, ISQED **2016-May**(Cd) (2016) 118–123

[40] Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., Xie, Y.: PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In: 2016 ACM/IEEE 43rd Annual

International Symposium on Computer Architecture (ISCA), IEEE (jun 2016) 27–39

[41] Strachan, J.P., Torrezan, A.C., Miao, F., Pickett, M.D., Joshua Yang, J., Yi, W., Medeiros-Ribeiro, G., Stanley Williams, R.: State dynamics and modeling of tantalum oxide memristors. IEEE Transactions on Electron Devices **60**(7) (2013) 2194–2202

[42] Ascoli, A., Tetzlaff, R., Chua, L.: Robust simulation of a TaO memristor model. Radioengineering **24**(2) (2015) 384–392

[43] Wagner, P.: Fast Import of Compressed Binary .RAW Files Created with LTspice Circuit Simulator