A Logical Degeneration of Vector Symbolic Architectures for Explainable AI: Among Distributed Inference and Propositional Reasoning; a Wittgenstenian Approach.

Abstract

This paper introduces a theoretical framework for Explainable Artificial Intelligence (XAI) based on a logical degeneration of high-dimensional vector symbolic architectures (VSAs). While VSAs provide a powerful substrate for neuro-symbolic reasoning through distributed representations, their internal operations remain mostly opaque to human understanding. We argue that interpretability can be achieved by reducing the dimensional complexity of such systems collapsing functionally but preserving their symbolic function. We propose a method to recast distributed logic into a one-dimensional propositional form, enabling transparent inspection of vector operations such as binding, bundling, vectorial inversion and so on. The framework is grounded in formal logic and epistemology, offering a conceptual bridge between symbolic transparency and distributed expressiveness, radicated in Wittgenstein's *Tractatus*. This work contributes to the development of controllable, auditable reasoning systems, opening novel perspectives on the design of interpretable neuro-symbolic architectures.

Introduzione

Il crescente interesse verso l'Explainable Artificial Intelligence (XAI) testimonia una consapevolezza sempre più diffusa rispetto le implicazioni epistemologiche ed etiche derivanti dall'opacità dei sistemi di apprendimento automatico. Le architetture di deep learning di paradigma connessionista, sebbene straordinariamente efficaci in numerosi compiti computazionali, risultano spesso inaccessibili alla comprensione umana. È necessario infatti ricordare che il paradigma connessionsita procede in modo algebrico, destrutturando i concetti in strati di funzioni probabilistiche. Tale deficit interpretativo ha promosso lo sviluppo di approcci ibridi neuro-simbolici, in grado di coniugare la potenza di apprendimento delle reti neurali con la trasparenza del ragionamento simbolico.

Queste sono le cosidette reti Vector Symbolic Architectures (VSA), una famiglia di modelli che consente la rappresentazione distribuita di concetti attraverso vettori iperdimensionali e l'esecuzione di operazioni logiche su tali rappresentazioni, al fine di coniugare, sommare e manipolare concetti in maniera distribuita. Le VSA, tuttavia, nonostante la loro promettente capacità di modellare inferenze complesse, restano in larga parte non interpretabili: l'informazione è dispersa nello spazio vettoriale e le operazioni interne non sono immediatamente ricostruibili da un osservatore umano: il senso delle proposizioni si smarrisce nell'algebra degli spazi vettoriali.

Il presente contributo getta le basi teoriche per lo sviluppo di un framework - ispirato alla logica del *Tractatus Logico Philosophicus* wittgensteiniano - volto a risolvere tale criticità mediante una semplificazione logica strutturata: si tratta di ridurre ad una forma unidimensionale riconducibile alla logica proposizionale (comprendendo anche operazioni di verifica della validità del sistema, da cui l'ispiraizone wittgensteiniana) lo spazio vettoriale distribuito, affinché la logica binaria[1] risulti concettualmente una degenerazione, per riduzione vettoriale, dello spazio vettoriale a rappresentazione distribuita. L'ipotesi di fondo è che la trasparenza non risieda necessariamente nella semplicità del modello o nella semplificazione dei termini, ma nella possibilità di tracciare una corrispondenza sistematica tra inferenze computazionali distribuite e strutture linguistiche o logiche formalmente intelligibili, riaccompagnando il paradigma informatico al suo ruolo di rappresentazione, di riflesso, non di sostituto ontologico.

Seguendo alcune intuizioni wittgensteiniane[2], proponiamo per i nostri scopi una riduzione sia matematica che epistemologica dell'architettura vettoriale che preservi l'intenzionalità semantica originaria, pur semplificandone la struttura formale, proponendo un collasso funzionale che conservi l'identità rappresentativa del vettore → simbolo. In questo modo, le operazioni simboliche interne alla rete possono essere esplicitate e controllate come sequenze inferenziali proposizionali, rendendo il comportamento del sistema finalmente accessibile alla comprensione e alla verifica umana. Il framework mira non solo ad essere uno strumento di lettura, ma anche di verifica, laddove, spesso, il rumore prodotto dalle operazioni tra vettori iperdimensionali rischia di creare errori non trascurabili all'interno del sistema: errori che sono dalla macchina letti come soluzioni contingenti di insiemi discreti (che evidentemente sfuggono alle previsioni del programmatore),

1 Riferendoci alla logica proposizionale come "logia binaria" non stiamo commettendo un superficiale errore di semplificazione dei termini, ma stiamo inscrivendo la logica proposizionale nel campo dell'informatica. Nella computazione, infatti, è comodo distinguere la logica distribuita (nel nostro caso iperdimensionale) da una logica a due termini, 0 e 1.

2 Per una più approfondita disamina di quanto lega Wittgenstein allo spazio vettoriale rimando a *Wittgenstein e il neuro-simbolismo: come ChatGPT e il Tractatus risolvono un antico rompicapo logico.*

dall’ingegnere di software come bug di sistema, ma dall’utente come vere e proprie allucinazioni concettuali. È necessaria un framework che applichi sullo spazio vettoriale una teoria della degenerazione fedelmente rappresentativa e non distruttiva, al fine di ricostruire la coerenza senza smarrire, o almeno sospendendo, il significato epistemologico che le dimensioni vettoriali assumono nel contesto della rappresentazione.

Presentando il sistema, ci occuperemo di spiegare nel modo più chiaro possibile le tappe del nostro percorso e il funzionamento delle tecnologie da noi citate.

Contesto dei sistemi vettoriali di rappresentazione e sviluppi attuali

La necessità di un approccio neuro-simbolico all’intelligenza artificiale è stata efficacemente argomentata da Gary Marcus, ricercatore statunitense noto per i suoi studi sull’intersezione tra psicologia cognitiva, neuroscienze e IA, nonché professore emerito di psicologia e scienze neurali alla New York University. Già nel 2020, Marcus metteva in luce i limiti dell’approccio dominante all’IA, connessionista, fortemente orientato verso l’apprendimento automatico generalizzato, fondato sull’utilizzo di dataset sempre più ampi e su una crescente potenza computazionale. A suo avviso, questo paradigma non è sufficiente per raggiungere una vera intelligenza artificiale generale; aggiungiamo noi, questo paradigma rischia di diventare una formalizzazione puramente matematica e non rappresentativa della realtà. L’IA connessionista, infatti, struttura la conoscenza in una serie di funzioni di attivazione che, attraverso un’architettura a strati, concorrono assieme per produrre un risultato coerente con la domanda[3]. Come si potrà intuire, questo paradigma ha dei problemi piuttosto concreti: le strutture neuronali che si articolano nella rete smarriscono, mano a mano, il senso semantico delle informazioni trattate, riducendosi a vettori numerici la cui interpretabilità risulta opaca sia per l’utente umano sia per il progettista. Ne consegue che la spiegazione del processo decisionale si affida a tecniche algebriche, spesso incapaci di restituire la logica interna della rete in termini comprensibili e verificabili. Inoltre, per quanto siano negli anni state avanzate diverse ipotesi di connessionismo come modello plausibile per la ricostruzione della mente[4], il simbolismo non ha mancato di notare la perdita della logica e del senso all’interno della struttura connesionista[5].

Secondo Marcus, è necessario un cambiamento di rotta: occorre un approccio ibrido, che integri apprendimento statistico e conoscenza esplicita, capace di valorizzare il ragionamento simbolico e ispirato a modelli cognitivi umani[6]. Solo così si potrà costruire un’intelligenza artificiale più solida, interpretabile e generalizzabile rispetto ai sistemi attuali. La prospettiva è stata di successo[7] e, per quanto molte aziende ancora preferiscano il paradigma connessionista per via del suo low effort training, si fanno strada molti progetti che mettono la trasparenza e l’interpretabilità del codice davanti al big data training.

3 Cfr. Rumelhart, D. E., McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition; Bechtel, W., & Abrahamsen, A. (2002). Connectionism and the Mind: An Introduction to Parallel Processing in Networks;* mentre per un approccio neuroscientifico rimandiamo a Churchland, P. S., & Sejnowski, T. J. (1992). *The Computational Brain*

4 Clark, A. (1993). *Associative Engines: Connectionism, Concepts, and Representational Change*

5 Fodor, J. A., & Pylyshyn, Z. W. (1988). *Connectionism and Cognitive Architecture: A Critical Analysis*

6 Marcus, G. (2001). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*

7 Offre un’ottimo stato dell’arte sull’integrazione neuro-simbolica Besold, T. R., Garcez, A. d., & Bader, S. (2017). *Neural-Symbolic Learning and Reasoning: A Survey and Interpretation*

Il funzionamento della rete neuro-simbolista si fonda più nel dettagli sull'integrazione di due componenti principali: da un lato, una rete neurale che apprende dati[8] (immagini, testo, segnali) tramite l'estrazione automatica di pattern distribuiti; dall'altro, un sistema simbolico che manipola concetti, regole logiche e strutture gerarchiche in modo esplicito. La rappresentazione dei concetti, invece di essere data in pasto, a seguito della tokenizzazione, a delle strutture gerarchizzate di funzioni di attivazione, sono rappresentate all'interno di uno spazio vettoriale ad alta dimensione, dove simboli, relazioni e strutture logiche vengono codificati in vettori distribuiti. Le reti neurali possono così apprendere a generare, trasformare e ragionare su queste rappresentazioni, rendendo possibile l'interpretazione di dati grezzi (come immagini o testo) in termini di significati simbolici e viceversa. Il risultato è un'architettura ibrida capace non solo di riconoscere pattern, ma anche di comporre e manipolare concetti in modo astratto, spiegabile e generalizzabile. Questo spazio matematico iperdimensionale di vettori è detto Vector Symbolic Architectures (VSA).

Per quanto in stato avanzato, anche i sistemi ad architettura vettoriale hanno dei limiti non trascurabili, sia di tipo tecnico che filosofico.

Dal punto di vista della costruzione del sistema, dobbiamo tenere conto del difetto delle sue operazioni. All'interno dello spazio vettoriale, infatti, le operazioni tra vettori — come il binding, l'unbinding o la somma — introducono inevitabilmente rumore. Questo rumore è una inevitabile conseguenza della natura distribuita e non ortogonale dei vettori iperdimensionali: ogni combinazione perde in parte l'identità dei simboli originari, e la decodifica può restituire risultati approssimativi o ambigui. La struttura rimane intatta, ma reiterando le operazioni per un determinato numero di volte, è possibile riscontrare ambiguità. Conseguenza diretta di questo fatto è la possibilità che dei vettori, densi di valori derivanti dal rumore tra le operazioni, perdano la loro quasi ortogonalità nello spazio. L'ortogonalità è necessaria, invece, affinché si possa dire che i vettori sono tra loro indipendenti (o meglio non è possibile riscrivere i vettori di base come combinazione lineare l'uno dell'altro). In un'architettura sparsa, a basso contenuto di valori attivi, l'operazione distribuita – che si avvicina nel caso del bundling alla somma vettoriale – è associata all'operazione logica OR. Il medesimo fatto di reiterazione delle operazioni porta una sovrabbondanza di valori nei vettori generati ad architettura sparsa, cosicché per mantenere una similarità coseno minima tra loro è necessaria l'operazione ulteriore di thinning. Questa consiste nel tagliare determinate posizioni a valore uno per ricondurle a vettori a basso contenuto di valori attivi. Si può ben comprendere come, anche quando si tratti di architetture sparse, il problema della quasi somiglianza dei vettori ha comunque un'incidenza importante, considerando che, per quanto basse, le probabilità di errore, dovute ad allucinazioni conseguenti la somiglianza di vettori composti, sussistono.

I problemi interpretativi principali, invece, emergono nella fase di codifica semantica. L'encoder, incaricato di trasformare simboli logici e concetti in vettori iperdimensionali, effettua una mappatura che non è trasparente: le strutture logiche in input vengono effettivamente mantenute nella forma vettoriale, ma il significato interno delle operazioni nello spazio matematico resta opaco. Per il programmatore o l'analista non è possibile seguire direttamente il tracciato semantico all'interno dello spazio vettoriale se non attraverso strumenti esterni di decodifica. Questa opacità ha una radice strutturale: l'algebrizzazione della semantica nel campo dell'informatica — cioè la sua

8 Per una presentazione strutturata delle reti neurali proponiamo Haykin, S. (2009). *Neural Networks and Learning* Machines, ma rimandiamo anche all'introduzione del seguente testo per comprendere appieno il significato di apprendimento dei dati nel constesto del deep learning. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.*

traduzione in operazioni su vettori, nel nostro caso — è una condizione necessaria affinché una macchina possa effettuare inferenze o comunque sia operare sulla semantica stessa. Tuttavia, se questa algebrizzazione non è sempre costruita in modo fedele alla natura dell'oggetto rappresentato; ciò significa che il significato può rapidamente disperdersi nel corso delle operazioni.

A livello più profondo, questo implica un problema filosofico e strutturale: le singole dimensioni di un vettore non hanno un significato rappresentazionale fondato aderente a delle caratterstiche dell'oggetto rappresentato. Non rappresentano parti discrete dell'oggetto originario, ma sono elementi funzionali di una struttura distribuita. In questo senso, la rappresentazione non può uscire dal paradigma connessionista-funzionalista: in fondo, ciò che conta non è la singola dimensione, ma la configurazione globale del vettore, la sua struttura. Il vantaggio è evidente: possiamo trattare significati complessi come unità vettoriali coese, riducendo il peso dell'interpretazione strutturale. Tuttavia, questo comporta che non possiamo leggere direttamente un vettore come una somma di parti interpretabili, ma dobbiamo ricorrere a un decoder (spesso una rete neurale o una mappa simbolica pre-addestrata) per risalire al significato originario, riconoscendo l'oggetto rappresentato tramite l'intera configurazione delle dimensioni e dovendo, comunque, affidarci alla macchina in modo quasi "oracolare".

Teoria dell’architettura di rappresentazione vettoriale

Lo spazio vettoriale iperdimensionale è, in senso matematico, un’estensione dello spazio vettoriale classico, dove, semplicemente, il numero di dimensioniè estremamente elevato - nell’ordine delle migliaia - e ciascun vettore è un elemento di R^d o di un sottoinsieme discreto come $\{-1,+1\}^d$ con $d \gg 1$.

Formalmente, uno spazio vettoriale V su un campo K è un insieme i cui vettori possono essere sommati tra loro e moltiplicati per scalari in K, rispettando le proprietà dell’algebra lineare. Nel nostro caso delle Vector Symbolic Architectures (VSA), lo spazio vettoriale iperdimensionale è un caso particolare in cui i vettori sono generati casualmente e distribuiti uniformemente su uno spazio ad alta dimensionalità (ad esempio, d=10,000) e associati arbitrariamente ad un concetto. Inoltre, a differenza di uno spazio vettoriale semplice, le componenti dei vettori di uno spazio iperdimensionale VSA non sono numeri reali continui, ma bipolari discreti[9], cioè appartengono all’insieme[10] {−1,+1}, e sono campionati da una distribuzione di Rademacher (ogni componente ha pari probabilità di essere −1 o +1) al fine di essere più robusti al rumore e di essere più semplici nelle operazioni tra loro. L’insieme $X := \{x_i\}_{i=1}^m$, detto codebook, è l’elemento che raccoglie questi vettori atomici tra loro: per d sufficientemente grande, la similarità coseno tra due vettori distinti tende a zero con alta probabilità, rendendoli pseudo-ortogonali rispetto alla metrica angolare. Più la similarità coseno tra i due vettori scelti è simile, minori sono le componenti che li separano.

Sebbene tali vettori non formino una base nel senso canonico dell'algebra lineare (poiché sono generati casualmente, con un ordine solo arbitrario e funzionale, e non garantiti come linearmente indipendenti), l’alto numero di dimensioni consente comunque di trattare ciascun vettore come distintamente identificabile grazie alla rarefazione geometrica dello spazio: in $\{-1,+1\}^d$, a partire da una certa soglia di dimensionalità, quasi ogni coppia di vettori risulta sufficientemente dissimile da ogni altra da poter fungere da simbolo distinto e manipolabile. È altissima la probabilità che due vettori, generati dagli stessi strumenti e al medesimo istante di tempo, possiedano componenti estremamente diverse tra loro. Lo spazio vettoriale iperdimensionale, infine, non si distingue per la natura delle sue operazioni algebriche, che restano formalmente e nella pratica delle operazioni le stesse di uno spazio vettoriale ordinario, ma per la sua struttura probabilistica e arbitraria, per l’uso di rappresentazioni distribuite e per la semantica simbolica associata alle configurazioni vettoriali. In questo contesto, il significato non è codificato da singole dimensioni, ma dalla combinazione distribuita di tutte le dimensioni, ossia dalla struttura intera dei singoli vettori, facendo leva sull’effetto geometrico dell’iperdimensionalità per assicurare robustezza, disgiunzione e tolleranza al rumore.

Nel contesto della logica formale e delle rappresentazioni distribuite di linguaggio naturale – poiché la logica che soggiace questi spazi vettoriali è una logica distribuita della rappresentazione – potremmo descrivere lo spazio vettoriale iperdimensionale come una *struttura semantica* (nel senso di modello matematico che significato a dei simboli) che permette di rappresentare atomi proposizionali o termini logici come vettori in $\{-1,+1\}^d$ o R^d, dove d è un numero sufficientemente grande da rendere, con altissima probabilità, quasi ortogonali tra loro i simboli rappresentati,

9 Nel caso dell’IA connessionista sono le funzioni soglia a trasdurre una prima rappresentazione vettoriale a numeri reali continui affinché attivi le funzioni di attivazione (Trasnformers). Nel nostro caso, ci limitiamo a far notare come, anche se possedendo uno spazio vettoriale con dimensioni a valori reali e continui, le funzioni soglia possano essere applicate per riportare la situazione ad un’architettura VSA, per le ragioni computazionali indicate.

10 Ma molto spesso appartengono all’ insieme {0, 1}, specialmente nelle architetture sparse.

affinché questi possano essere tra loro distinti e considerati indipendenti. Asseriamo ad una struttura semantica poiché l'insieme di partenza su cui vogliamo operare è pur sempre un linguaggio logico, quello della logica dei predicati derivanti da una formalizzazione dell'input testuale, i cui oggetti devono essere rappresentati come vettori quasi ortogonali. Infatti, ogni simbolo primitivo della logica (es. costanti, predicati, variabili) è mappato a un vettore atomico, generato in modo casuale ma gestito al fine di garantire bassa correlazione con altri simboli (eccezione fatta per pochi casi di errore). Le operazioni logiche (congiunzione, disgiunzione, negazione, implicazione) vengono simulate mediante operazioni algebriche tra vettori, secondo le regole definite dal sistema VSA (binding, superposizione, permutazione, etc.), che esploreremo più nel dettaglio a seguire.

Formalmente, possiamo considerare un dominio simbolico *L*, costituito da un linguaggio logico finito Σ (che include tutti i simboli proposizionali e i connettivi logici) e un insieme Φ di formule ben formate (FBF), dunque composte unicamente attraverso il linguaggio Σ . L'insieme Φ viene elaborato da una funzione di codifica $\tau:\Phi \rightarrow V$, dove V è un insieme costituito da vettori $v_1,\ldots,v_n$ che hanno come dimensione un valore $\{-1,+1\}^d$ oppure R^d, che assegna a ciascuna formula una rappresentazione vettoriale: funzione che possiamo attribuire all'encoder semantico.

Potremmo concludere, riassumendo, che, dal punto di vista della matematica e della logica matematica, uno spazio vettoriale iperdimensionale è definito come una semantica distribuita su R^d per un linguaggio logico, in cui simboli e formule sono rappresentati come vettori quasi ortogonali, e le operazioni logiche sono simulate da operazioni algebriche continue, garantendo robustezza e computabilità neurale dell'inferenza.

Lo spazio iperdimensionale comprende anche delle operazioni proprie a struttura logica, radicate, ovviamente, in operazioni matematiche vettoriali.

Similarità coseno: per raffrontare la similarità semantica (ossia quanti componenti due vettori condividano l'uno con l'altro), si utilizza la similarità coseno (sim), data da:

$$\mathrm{sim}(u,v)=\frac{u\cdot v}{\|u\|\cdot\|v\|}.$$

una similarità vicina a 1 significa che due formule hanno rappresentazioni simili (condividono molte componenti), mentre una similarità vicina a 0 implica indipendenza o ortogonalità semantica (tipico per simboli atomici casuali).

Bundling: l'operazione logica di *bundling*[11] svolge un ruolo cruciale nella rappresentazione distribuita: consente la composizione di concetti attraverso una somma vettoriale che aggrega più vettori in una singola entità $A \oplus B = C$. Il vettore *C* risultante mantiene una somiglianza non trascurabile con ciascuno dei vettori originali, rendendo parzialmente accessibili le componenti da cui è derivato. Tale operazione può essere intesa come una forma di *superimposizione*, in cui le informazioni vengono sovrapposte senza una struttura gerarchica esplicita, realizzando quella che Plate[12] definisce *unstructured similarity preservation*: non è necessario che i simboli siano ordinati o etichettati in modo strutturato, purché siano definiti all'interno dello spazio semantico. Il vettore aggregato risultante dalla combinazione di A e B sarà quindi simile sia ad A che a B, ma anche ad altri vettori che, a loro volta, li includano come componenti. Inoltre, l'operazione di bundling gode

11 Cfr. Schlegel, K., Neubert, P., & Protzel, P. (2021). *A comparison of vector symbolic architectures*. *Artificial Intelligence Review*, 55, 2717–2752. https://doi.org/10.1007/s10462-021-10110-3

12 Plate, T. A. Holographic reduced representations. IEEE Transactions on Neural Networks 6, 623–641 (1995); ma anche Plate, T. A. Holographic Reduced Representations: Distributed Representation for Cognitive Structures (Center for the Study of Language and Information,Stanford, 2003)

di una proprietà di associatività approssimata, tale che (A+B)+C≈A+(B+C), facilitando la costruzione incrementale di rappresentazioni complesse. Ciò, per le regole assiomatiche dell'algebra lineare. Inoltre, nelle architetture a codifica sparsa, caratterizzate da un numero ridotto di componenti attive, il bundling può essere efficacemente implementato mediante l'operazione logica OR, compatibile con la semantica distribuita di tipo binario o bipolare. Questo perché nella rappresentazione vettoriale a codifica sparsa, la posizione dei valori attivi conta moltissimo, ed è quindi necessario che questa operazione sia il più possibile accurata.

Qualora vi siano delle operazioni di bundling reiterate, c'è però la possibilità, come descrivavamo nella sezione precedente, che si insedino delle ambiguità nell'operazione di bundling o ocmunque degli errori da gestire quando i vettori diventano, a seguito di molte operazioni di OR, estremamente densi (pensiamo a vettori con valore attivo nel novanta percento delle posizioni: significherebbe che solo il dieci percento di queste può essere decisiva nell'identificazione rispetto altri vettori). La densità va quindi ridotta al fine di mantenere la codizione di quasi ortogonalità con gli altri vettori dello spazio. Arbitrariamente, dei valori attivi vengono annullati, ripristinando, in parte, la condizione normale di ortogonalità: questa è detta operazione di thinning[13].

Un'esempio esemplificativo, tratto dalla tradizione filosofica. Nel suo *Tractatus Logico Philosophicus*, L. Wittgenstein si occupa di comporre la realtà attraverso proposizioni atomiche che, mano a mano, progrediscono in concetti più complessi. Asserisce ad esempio che la proposizione è un'immagine della realtà, composta da oggetti, fatti, stati di cose. Potremmo considerare la proposizione come un vettore $\vec{P}$ composto dai vettori semplici oggetti ($\vec{o}$), fatti ($\vec{f}$), stati di cose ($\vec{c}$). In una rappresentazione classica, il concetto di proposizione avrebbe potuto essere definito come composizione. Nella logica distribuita dei sistemi VSA, invece, ci basterà aggregarli attraverso un bundling di questi tre vettori:

$$\vec{P} = \vec{o} + \vec{f} + \vec{c}.$$

Il vettore $\vec{P}$ non ha una struttura sintattica esplicita, ma costituisce una rappresentazione distribuita e fedele del concetto di *proposizione atomica*: una configurazione logica che esprime un possibile stato del mondo. Questo tipo di *bundling* consente di mantenere, come detto, una certa similarità tra $\vec{P}$ e ciascuna delle sue componenti. Non esiste una gerarchia nelle posizioni dei vettori componenti, ma ciò che conta è la presenza definita di ciascun concetto nel vettore complessivo. In questo modo, la semantica emerge dalla combinazione, non dall'ordine di combinazione, rispettando gli assiomi dell'algebra lineare.

Poiché vi è similarità tra il vettore composto e le sue parti, è possibile anche un'operazione inversa detta di *unbundling*, che consiste nel risalire, dal vettore composto, ad un vettore componente. Avendo ad esempio il vettore composto $\vec{P}$ e le sue componenti $\vec{o}$ ed $\vec{f}$, possiamo attraverso l'unbunding risalire al suo componente $\vec{c}$ (in realtà, con una componente inevitabile di rumore).

Binding: a differenza del *bundling*, che aggrega concetti preservando la similarità con le loro componenti in modo non strutturato, il binding realizza una combinazione strutturata tra simboli, capace di codificare relazioni sintattiche. L'operazione è non commutativa: legare A con B (cioè A⊗B) non equivale a legare B con A, e ciò permette di rappresentare ruoli, relazioni o strutture gerarchiche all'interno di una rappresentazione distribuita.

13 Il thinning non riguarda in alcun modo le tecniche di riduzione dimensionale (appunto delle dimensioni dei vettori), ma si concentra sull'azzeramento di alcune delle dimensioni. La dimensionalità rimane invariata, sono i valori ad essere appiattiti.

$$A \otimes B \neq B \otimes A$$

Possiamo fare un esempio elementare per chiarificare questo concetto: *cane*⊗*colore* = “colore del cane”. Qui il concetto di *cane* è legato al concetto di *colore* mediante un’operazione di binding (simbolizzata come prodotto), ottenendo una rappresentazione composita che esprime la relazione tra i due concetti. Il risultato non è simile, in termini di similarità coseno, né a *cane* né a *colore:* infatti la nuova entità semantica è qualitativamente distinta. Tuttavia, strutture analoghe, come *gatto*⊗*colore* o *cane*⊗*taglia*, condivideranno proprietà strutturali, rendendo possibile il confronto tra relazioni. Anche in questo caso l’*unbinding* è possibile, ma è molto più complesso dal punto di vista informatico[14].

14 Cfr. la parte introduttiva di Hersche, M., Zeqiri, M., Benini, L., Sebastian, A., & Rahimi, A. (2023). *A neuro-vector-symbolic architecture for solving Raven’s progressive matrices*. IBM Research – Zurich; ETH Zürich. Retrieved from https://github.com/IBM/neuro-vector-symbolic-architectures

Formalizzazione logica

Degenerazione logica di un sistema distribuito: riduzione epistemica delle dimensioni degli spazi vettoriali.

Sia L_D un sistema di logica distribuita implementato mediante un spazio di rappresentazione vettoriale iperdimensionale V con $V \subseteq \mathbb{R}^n$ (es. sistemi informatici VSA – Vector Symbolic Architectures), dotato di una struttura semantica topologica T (che determina la vicinanza concettuale dei vettori tramite similarità coseno) e di una logica inferenziale $\vdash_D$, ossia una struttura logica ternaria del tipo $L_D = (V, T, \vdash_D)$, ed L_P un sistema di logica proposizionale classico avente un insieme di formule logico-simboliche S e $\vdash_P$ una logica inferenziale proposizionale, allora esiste una mappa di proiezione della degenerazione

$$\pi: L_D \to L_P$$

tale che:

L_P è una logica proposizionale discreta e binaria, degenerata dal sistema logico distribuito L_D, dove le relazioni inferenziali $\vdash_D$ vengono preservate, ma la topologia semantica P propria dello spazio vettoriale e le caratteristiche metriche vengono sospese. Dunque non si ritengono rilevanti, in L_P, caratteristiche quali la similarità vettoriale (similarità semantica) o la distanza dei vettori nello spazio. L'assunto è analogo al principio geometrico di degenerazione: come una curva algebrica può degenerare in una retta con punti doppi, così uno spazio semantico logico può degenerare in una struttura proposizionale binaria. Questo modello predilige la trasparenza inferenziale della logica proposizionale, ovvero del modello L_P, sacrificando la densità semantica ai fini dell'interpretabilità.

Operazioni distribuite come relazioni: mappatura e corrispondenze dell'insieme di operazioni.

Le operazioni logiche che avvengono all'interno dello spazio distribuito sono reinterpretabili, per alcuni aspetti, come relazioni logiche tra i simboli. La strada che abbiamo qui deciso di intraprendere è stata quella di scomporre le operazioni degli spazi vettoriali nelle loro principali caratteristiche, rappresentando queste caratteristiche come relazioni tra simboli. Chiameremo quindi due vettori $\vec{v}_1 e \vec{v}_2$ rispettivamente A e B. Questo ci peremtterà di trattarli come simboli proposizionali, permettendoci, attraverso le regole che andremo a definire, di cogliere quali siano le proprietà implicite nelle proprietà delle strutture vettoriali dei sistemi VSA.

Bundling. Vanno inizialmente definite quali siano le proprietà dell'operazione di bundling su cui operare la nostra verifica formale. Abbiamo detto essere queste principalmente (1) l'aggregazione di concetti (che si distingue dalla composizione del binding), (2) la commutatività e (3) la preservazione della similarità con i vettori di composizione.

(1) Possiamo esprimere una relazione binaria forte che espliciti l'aggregazione dei due vettori con la formula $(A \wedge B) \Leftrightarrow C$ e aggiungere che la similarità del vettore composto con i due compositori è preservata. Nel suo insieme la prima proprietà potrebbe essere costruita come:

$$R_1 := ((A \wedge B) \Leftrightarrow C) \wedge (C \Leftrightarrow A) \wedge (C \Leftrightarrow B).$$

(2) La proprietà commutativa è molto forte se dimostrata attraverso la logica proposizionale, poiché questa, per sua natura, considera commutative le operazioni di congiunzione e disgiunzione,

rendendo l'ordine dei termini irrilevante nella valutazione di verità. Possiamo, per formalizzare, esprimere la commutatività del bundling in questo modo:

$$R_2 := (A \wedge B) \Leftrightarrow (B \wedge A).$$

Il vantaggio in questo senso è il suo valore tautologico, che deriva dalla sua struttura sintattica simmetrica nella logica proposizionale classica: la congiunzione $A \wedge B$ è definita in modo tale da essere logicamente equivalente a $B \wedge A$, indipendentemente dal contenuto semantico di A e B. Questo significa che la proprietà non richiede dimostrazione: è interna al sistema assiomatico della logica proposizionale, e quindi garantisce per costruzione che l'ordine degli operandi sia irrilevante nella valutazione di verità dell'espressione. Un gran bel vantaggio per l'operazione di bundling.

(3) La similarità tra vettori – simboli - può essere esplicitata attraverso la seconda parte della R_1, ossia dicendo che $(C \Leftrightarrow A) \wedge (C \Leftrightarrow B)$. Soffermarci su questa regola ci permette di notare quanto questa regola sia costrittiva. Infatti, qui è necessario che A e B siano logicamente equivalenti a C. Possiamo riformulare la prima regola allo attraverso delle implicazioni, dicendo per esempio che se c'è C, allora posso inferire A e B: $(C \rightarrow A) \wedge (C \rightarrow B)$. Sarebbe ridondante una seconda regola che specifichi questo, quindi riformuliamo piuttosto la prima entro questi termini, ossia evitando che A e B debbano necessariamente condividere la forma logica di C, poiché non necessaria:

$$R_1 := ((A \wedge B) \rightarrow C) \wedge (C \rightarrow A) \wedge (C \rightarrow B)$$

Binding. Abbiamo nei paragrafi precedenti esplorato il binding come operazione con tre principali proprietà: (1) l'operazione di binding tra due vettori definisce un nuovo vettore nuovo e diverso dai due precedenti, (2) l'operazione non è commutativa e (3) conoscendo solo uno dei due vettori componenti e il vettore composto dall'unione posso dedurre il vettore componente sconosciuto. La relazione logica che ognuna di queste proprietà esplica è definibile attraverso operazioni tra simboli.

(1) $(A \wedge B) \Leftrightarrow C$. In questi termini si sta esplicitando una relazione binaria forte tra la congiunzione di A e B e il simbolo C. Il simbolo C è definito dall'unione dei simboli A e B. Dobbiamo però aggiungere che il simbolo C è nuovo nella struttura, regola che può esssere così esplicitata: $C \neq A \wedge C \neq B$, o meglio, non vi è una relazione binaria forte diretta tra A e C o tra B e C senza che vi sia il secondo simbolo: $\neg(C \Leftrightarrow A) \wedge \neg(C \Leftrightarrow B)$. Nel suo insieme, la prima proprietà che definirà il nostro operatore risulta come:

$$R_1 := ((A \wedge B) \Leftrightarrow C) \wedge (\neg(C \Leftrightarrow A) \wedge \neg(C \Leftrightarrow B)).$$

Si noti che qui, a differenza della regola R_1 propria del bundling, la relazione diretta coi simboli che compongono C non è mantenuta, poiché il concetto composto è nuovo completamente.

(2) In una formula ben formata che componga i simboli A e B, come ad esempio una formula $A \wedge B$, la proprietà commutativa è garantita. Infatti $A \wedge B \equiv B \wedge A$. L'idea principale qui, per dichiarare l'opposto di questa proprietà, è negarla nel nostro modello. Diremo quindi che associare in una determinata posizione i simboli A e B per risultare C non è come scambiarli tra loro e risultare C:

$$R_2 := \neg(((A \wedge B) \Leftrightarrow C) \Leftrightarrow (B \wedge A) \Leftrightarrow C).$$

In questo modo neghiamo che associare A e B per comporre C sia come associare B e A.

(3) La deduzione di uno dei tre simboli conoscendo un componente e il composto è facilmente riconducibile ad una formula in cui, dati A e C, posso dedurre B, mentre dati B e C, posso dedurre a

$$R_3 := ((A \wedge C) \rightarrow B) \wedge ((B \wedge C) \rightarrow A).$$

L'*operatore* $\otimes$ *che* andremo a creare dovrà combinare, congiungere assieme, tutte queste regole. Potremmo infatti dire che $A \otimes B = R_1 \wedge R_2 \wedge R_3$, ossia:

$$[((A \wedge B) \Leftrightarrow C) \wedge (\neg(C \Leftrightarrow A) \wedge \neg(C \Leftrightarrow B))] \wedge [\neg(((A \wedge B) \Leftrightarrow C) \Leftrightarrow (B \wedge A) \Leftrightarrow C)] \wedge [((A \wedge C) \rightarrow B) \wedge ((B \wedge C) \rightarrow A)]$$

Eppure, una rapida verifica della seconda regola mostra che non esistono combinazioni di valori di verità che la rendano vera. Infatti, la formula:

$$R_2 := \neg(((A \wedge B) \Leftrightarrow C) \Leftrightarrow (B \wedge A) \Leftrightarrow C)$$

risulta sempre falsa nella logica booleana classica, poiché la commutatività dell'operatore AND garantisce che le due implicazioni siano logicamente equivalenti in ogni caso. Di conseguenza, la loro equivalenza è tautologica, e negandola si ottiene una contraddizione costante, cioè una formula mai vera. Ovviamente, non potendo essere vero il valore della seconda regola, è inutile la verifica delle altre, in quanto tutte e tre devono essere vere contemporaneamente.

Questo esito non è strutturale, e non dipende esplicitamente dalla scelta dell'operatore congiuntivo AND, che è commutativo. Per dimostrarlo, vogliamo convertire la seconda regola in una logicamente equivalente senza però servirci di operatori come AND o OR, al fine di dimostrare come questo errore sia inevitabile.

La formula di non commutatività (R_2) va riconvertita utilizzando operazioni di implicazione:

$$\neg((A \rightarrow (B \rightarrow C)) \Leftrightarrow (B \rightarrow (A \rightarrow C))).$$

Utilizziamo delle tavole di verità per verificare che:

$$\neg(((A \wedge B) \Leftrightarrow C) \Leftrightarrow (B \wedge A) \Leftrightarrow C) \equiv \neg((A \rightarrow (B \rightarrow C)) \Leftrightarrow (B \rightarrow (A \rightarrow C))).$$

NOT(((	A	AND	B)	IF and only IF	C)	IF and only IF	(B	AND	A)	IF and only IF	C)
F	T	T	T	T	T	T	T	T	T	T	T
F	T	T	T	F	F	T	T	T	T	F	F
F	T	F	F	F	T	T	F	F	T	F	T
F	T	F	F	T	F	T	F	F	T	T	F
F	F	F	T	F	T	T	T	F	F	F	T
F	F	F	T	T	F	T	T	F	F	T	F
F	F	F	F	F	T	T	F	F	F	F	T
F	F	F	F	T	F	T	F	F	F	T	F

NOT ((	A	IF...THEN	(B	IF...THEN	C))	IF and only IF	(B	IF...THEN	(A	IF...THEN	C)))
F	T	T	T	T	T	T	T	T	T	T	T
F	T	F	T	F	F	T	T	F	T	F	F
F	T	T	F	T	T	T	F	T	T	T	T
F	T	T	F	T	F	T	F	T	T	F	F
F	F	T	T	T	T	T	T	T	F	T	T
F	F	T	T	F	F	T	T	T	F	T	F
F	F	T	F	T	T	T	F	T	F	T	T
F	F	T	F	T	F	T	F	T	F	T	F

Ci ritroviamo quindi in una situazione in cui non è possibile uscire dal valore negativo del segno $A \otimes B$ così come lo avevamo rappresentato, poiché in qualsiasi caso la seconda regola dell'operatore che stavamo tentando di costruire risulterà di valore di verità negativo. Ebbene, questo che sembrerebbe essere il fallimento del nostro modello logico, porta in realtà alla luce un'evidenza filosofica importante. La seconda regola, infatti, non rappresenta una semplice negazione formale, ma costituisce il vincolo strutturale che formalizza la non-commutatività del binding semantico. Essa esprime che l'inversione degli argomenti compromette la validità inferenziale, rendendo esplicita l'asimmetria semantica tra A e B. L'interpretazione di questa regola è quindi epistemologica: non si tratta di un errore logico da eliminare, ma di una condizione negativa che garantisce il rispetto dell'ordine argomentativo e che distingue, non solo filosoficamente ma anche in modo logico-rigoroso, i due livelli entro quali stiamo operando. Questa forma negativa impedisce, in sostanza, l'equivalenza tra due strutture proposizionali che, pur sintatticamente simmetriche, non condividono lo stesso significato semantico. In questo modo, la non-commutatività non è imposta per via sintattica, ma emerge come proprietà semantica dedotta negativamente dalla rottura dell'equivalenza implicativa tra i due ordinamenti, spartendo logica distribuita e proposizionale in modo rigoroso.

Piuttosto che affermare che la seconda regola sia "sempre falsa", è più corretto sostenere che la simmetria tra i due modelli logici viene sistematicamente rifiutata. Questo non implica, in termini computazionali, l'impossibilità della degenerazione dell'operazione di binding in un operatore logico simmetrico, ma indica piuttosto che in ogni assegnazione booleana la negazione dell'equivalenza tra le due implicazioni risulta vera. La regola, dunque, non seleziona specifici casi, bensì conferma costantemente che binding-rispettivo operatore booleano non è simmetrico: essa agisce come una condizione ontologica necessaria per la distinzione dei due modelli, non come una condizione variabile. Va per questo considerata "vera" nel sistema logico, proprio perché conduce all'assurdo. Questo assurdo sistematico diventa il sigillo logico che garantisce che l'ordine dei due modelli distinti. Essa è strutturalmente sempre falsa, ma va assunta per vera perché epistemologicamente indispensabile. Questo ci porta a concludere che le due regole da verificarsi affinché la degenerazione dell'operazione di binding sia da dirsi attendibile sono la prima (rispetto la generazione di nuovi simboli) e la tera, che riguarda la deduzione dei simboli compositivi.

Architettura del sistema: passaggi logici e strumenti di implementazione

Questa sezione è dedicata alla costruzione di una mappa orientativa contenente i nodi concettuali del programma per la degenerazione dello spazio vettoriale iperdimensionale attraverso un linguaggio di programmazione basico come Python.

Poiché il framework adempia del tutto alle sue mansioni, non è solo necessario che mappi i vettori singoli come simboli unici, ma anche che sappia gestire in tempo reale la casistica in cui nello spazio vettoriale avvengano delle operazioni logiche (riconducibili ad operazioni dell'algebra lineare, come abbiamo visto), al fine di mapparle e reinterpretarle come operazioni tra simboli nella logica proposizionale, più accessibile ad occhio nudo.

Il framework mappa inizialmente ogni vettore di un sistema VSA in un simbolo logico, costruendo una tabella che associa ogni simbolo ad un vettore. Quando vengono eseguite operazioni di binding o bundling tra vettori, il framework non si limita a calcolare il risultato vettoriale derivante dall'operazione: parallelamente, deduce le regole logiche che descrivono l'operazione (ad esempio, che il risultato di un'associazione di binding è una congiunzione non commutativa tra simboli). Ogni simbolo composto è dunque corredato da una rappresentazione logica della sua derivazione, tracciata in una struttura ad albero o grafo, che conserva la storia compositiva del vettore e aiuta il programmatore a vedere in modo trasparente quali siano i passaggi logici di ognuna delle associazioni. In fase di interrogazione, un vettore simbolico può essere spiegato restituendo la sequenza delle operazioni che l'hanno generato come espressione logica. Inoltre, una seconda partizione del framework può analizzare le regole che governano ciascuna operazione, verificandone la coerenza logica, la validità rispetto a proprietà strutturali (commutatività, associatività, simmetria, ecc.) e il rispetto delle relazioni epistemiche stabilite nel dominio. Questo consente non solo di interpretare vettori complessi, ma anche di controllare la correttezza dei processi di composizione simbolica in un sistema VSA. Le funzioni principali del programma sono le seguenti:

1. Impostazione, dichiarazione delle librerie, acquisizione del modello VSA, adattamento del modello al programma.
2. Mappatura dei vettori semplici in simboli.
3. Trasduzione, a seguito delle operazioni di binding o binding, dei vettori composti come regole logiche derivanti dalle proprietà delle operazioni.
4. Verifica della correttezza delle operazioni logiche.

1) Impostazione e acquisizione del modello.

Inizialmente dovranno essere dichiarate le librerie che utilizzeremo, quali: `numpy scipy networkx pandas.` Scelto il modello ad architettura vettoriale VSA che vorremo utilizzare (Holographic Reduced Representations, Binary Spatter Code o altri), poi, dovremo implementare una parte di codice che sia in grado di comprendere quali sono le strutture, gli oggetti e le variabili del codice del modello VSA, al fine di poterle convertire in oggetti interpretabili dal nostro programma.

Esempio di struttura del codice:

```
import numpy as np # operazioni su vettori e algebra lineare
```

```
from scipy.spatial.distance import cosine  # calcolo cosine similarity
import networkx as nx # rappresentazione di strutture ad albero
import pandas as pd # gestione dizionario vettore->simbolo

# Parametri del modello VSA
vector_dim = 10000  # Dimensione vettori iperdimensionali
vsa_model = "HRR"   # Modello VSA scelto (es. Holographic Reduced
Representations) ovviamente considerati gli scopi del nostro framework

# Inizializzazione vettori casuali
def init_vector():
    return np.random.choice([-1, 1], size=vector_dim)  # Crea vettori binari per
HRR con parametri casuali
```

2) Mappatura dei vettori semplici.

Definiremo poi una classe LogicMap che, fungendo da interfaccia logico-formale ponte tra rappresentazioni vettoriali e simboli proposizionali, definisce un sistema di mappatura che associa a ciascun vettore univocamente identificato (anche per via della sua complessità matematica) un simbolo atomico della logica proposizionale, secondo una soglia di similarità (basata tipicamente su cosine similarity o distanza euclidea normalizzata, in base al modello VSA utilizzato). Il modulo gestisce un dizionario semantico in cui ogni simbolo proposizionale viene ancorato a un vettore di riferimento, e fornisce strumenti per inferire o recuperare il simbolo corrispondente a un vettore osservato, consentendo così la decodifica dei contenuti distribuiti in forma simbolica. Questa transizione da vettore a simbolo è fondamentale per rendere interpretabili e verificabili i processi di inferenza nei sistemi neuro-simbolici, e per abilitare l'analisi logica della struttura interna dei binding e dei bundling operati nello spazio vettoriale. Si tratta, lo sottolineamo, di un'analisi logica delle strutture che compongono i vettori attraverso le operazioni logiche di bundling e binding, ma non hanno a che vedere con l'analisi semantica dei concetti rappresentati.

Esempio di codice:

```
class LogicMap:
    def __init__(self):
        self.vector_to_symbol = {}  # Dizionario vettore -> simbolo
        self.symbol_to_vector = {}  # Dizionario simbolo -> vettore

    def add_mapping(self, vector, symbol):
        # Aggiunge una mappatura vettore-simbolo.
        self.vector_to_symbol[tuple(vector)] = symbol
        self.symbol_to_vector[symbol] = vector

    def get_symbol(self, vector, threshold=0.9):
        # Recupera il simbolo più simile al vettore dato.
        for ref_vector, symbol in self.vector_to_symbol.items():
            similarity = 1 - cosine(vector, ref_vector)
            if similarity > threshold:
                return symbol
        return None  # Nessun simbolo trovato sopra la soglia
```

3) Trasduzione.

Le operazioni di composizione tramite le operazioni di bundling e binding rimangono, in un certo senso, simboliche per il programmatore, che si affida al programma per la somma algebrica dei vettori e altre operazioni matematiche, smarrendo il senso logico-simbolico delle operazioni stesse.

Prima di una qualsiasi operazione, il framework individua la composizione dei simboli. Se i simboli sono semplici, ossia definiti da un solo vettore, allora li identifica e procede all'operazione, se invece sono composti, ossia definiti da operazioni precedenti di bundling e binding, sarà necessario individuare consequenzialmente questi precedenti passaggi. In questo modo, rappresentando ad albero le formazioni dei vettori, sarà più intuitivo per l'ingegnere di sistema non perdere il controllo del senso delle operazioni.

Per quanto concerne i simboli logici creati mediante il processo di bundling, il framework dovrà scomporre il simbolo nel rispetto delle proprietà dell'operazione, ossia l'aggregazione di concetti, la commutatività e la preservazione della similarità con i vettori di composizione. Quando invece il sistema VSA compie un'operazione di binding e crea un nuovo vettore $\vec{c}$, crea anche un nuovo simbolo C. Il framework, quando andrà a recuperare quel simbolo, dovrà dedurre anche le regole di composizione dello stesso, avvenute, ad esempio, con i due simboli A e B. Si occuperà di riprendere le regole di composizione (viste nei precedenti paragrafi) di crazione di nuovo simbolo, non commutatività (sempre vera) e possibilità di deduzione dei simboli. In questo modo il modulo di controllo logico del programma potrà seguire passo passo le combinazioni tra vettori e, nel caso vi fossero delle allucinazioni o degli addensamenti di dati – che portano inevitabilmente a degli errori –, procede a rilevarlo, permettendo al programmatore di spiegare e sistemare questa allucinazione informatica.

Possiamo provare a definire la bozza di una semplice funzione che rappresenta i simboli come nodi di un grafo dato il simbolo stesso, l'operazione a cui è stato sottoposto e il suo simbolo genitore utilizzando la funzione DiGraph della libreria networkx (nx):

```
def build_composition_tree(symbol, operation, parent_symbols):
    G = nx.DiGraph()
    G.add_node(symbol, type="symbol")
    for parent in parent_symbols:
        G.add_edge(parent, symbol, operation=operation)
    return G
```

4) Verifica.

È necessario un modulo di verifica poi, affinché la struttura (da vettoriale a logica) sia letta da uno strumento terzo che si preoccupi della coerenza di tutte le operazioni. In questo modo, qualora dei vettori perdessero la loro aderenza logica per via di rumore o densità, il lettore logico se ne accorgerebbe subito, permettendo al programmatore di intervenire sul problema.

Esistono in questo momento diversi progetti attivi o comunque già implementati di lettori della logica formale. Di seguito ne illustriamo alcuni, prendendoli ad esempio per creare il nostro personale:

- PyEDA: Una libreria Python per l'EDA (Electronic Design Automation), ma racchiude al suo interno un potente modulo per la verifica della logica proposizionale: supporta la creazione di espressioni logiche, tabelle di verità, semplificazione di espressioni e verifica di validità. Github: https://github.com/cjdrake/pyeda.
- SymPy: un potente framework per l'algebra scritto interamente in python, con un modulo dedicato anche al calcolo simbolico. Non estremamente utile per la verifica, ma per lo meno capace di coprire sia le operazioni algebriche che quelle simboliche. Github: https://github.com/sympy/sympy.
- Z3 (Microsoft): un potentissimo solver SMT (Satisfiability Modulo Theories) che supporta anche la logica proposizionale. È in grado di verificare la soddisfacibilità e validità di espressioni logiche, anche complesse, molto adatto ai nostri scopi. Forniamo anche un esempio di utilizzo, perché lo riteniamo efficace e utile al nostro progetto. Github: https://github.com/Z3Prover/z3.

```
from z3 import * #importa il modulo Z3
A, B, C = Bools('A B C') #definisce i simboli
expr = Implies(And(A, B), C) #definisce l'operazione
s = Solver() #definisce una variabile s che richiama la funzione di
                  soluzione della regola logica
s.add(Not(expr)) #richiama add dalla funzione Solver() e fornisce
                  l'espressione expr
print(s.check())  # richiama il print per mostrare l'output

#ci aspettiamo un output che chiarisca la validità dell'espressione.
```

Bibliografia

Bechtel, W., & Abrahamsen, A. (2002). *Connectionism and the Mind: An Introduction to Parallel Processing in Networks*. Oxford: Blackwell.

Besold, T. R., Garcez, A. d., & Bader, S. (2017). *Neural-Symbolic Learning and Reasoning: A Survey and Interpretation*. In *arXiv preprint arXiv:1711.03902*.

Churchland, P. S., & Sejnowski, T. J. (1992). *The Computational Brain*. Cambridge, MA: MIT Press.

Clark, A. (1993). *Associative Engines: Connectionism, Concepts, and Representational Change*. MIT Press.

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1–2), 3–71.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.

Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd ed.). Upper Saddle River, NJ: Pearson.

Hersche, M., Zeqiri, M., Benini, L., Sebastian, A., & Rahimi, A. (2023). *A neuro-vector-symbolic architecture for solving Raven's progressive matrices*. IBM Research – Zurich; ETH Zürich. Recuperato da: https://github.com/IBM/neuro-vector-symbolic-architectures

Kanerva, P. (2009). Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2), 139–159. https://doi.org/10.1007/s12559-009-9009-8

Marcus, G. (2001). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press.

Neubert, P., Schubert, S., & Protzel, P. (2019). An Introduction to Hyperdimensional Computing for Robotics. *KI – Künstliche Intelligenz*. https://doi.org/10.1007/s13218-019-00623-z

Plate, T. A. (1995). Holographic Reduced Representations. *IEEE Transactions on Neural Networks*, 6(3), 623–641.

Plate, T. A. (2003). *Holographic Reduced Representations: Distributed Representation for Cognitive Structures*. Stanford: CSLI Publications.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Schlegel, K., Neubert, P., & Protzel, P. (2021). A comparison of vector symbolic architectures. *Artificial Intelligence Review*, 55, 2717–2752. https://doi.org/10.1007/s10462-021-10110-3

Wittgenstein, L. (1921). *Tractatus Logico-Philosophicus*. (Trad. it. *Tractatus logico-philosophicus*, a cura di A. G. Conte, Einaudi, Torino, varie edizioni).