Università degli Studi di Torino
Scuola di Scienze della Natura
Corso di Laurea Magistrale in Informatica

Tesi di Laurea Magistrale

Design, realizzazione e ingegnerizzazione di un sistema di dialogo basato su LLM nel dominio delle tecnologie assistive

RELATORE
Prof. Alessandro Mazzei

CANDIDATO
Stefano Vittorio Porta
859133

Anno Accademico 2023/2024

Dichiarazione di Originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Ringraziamenti

Todo

Abstract

Todo.

Parole chiave

«classificazionexNLU», «data annotation/augmentation», «NLGBasataSuParafrasi», «ingegnerizzazione»

Indice

1 Introduzione

1.1 Contesto generale

1.2 Motivazioni e obiettivi della tesi

1.3 Struttura del documento

2 Origini e Stato dell'Arte

In questo capitolo verrà fornita una panoramica sui principali approcci per la realizzazione di chatbot e sistemi di interrogazione automatica basati sul linguaggio naturale. Si partirà dall'analisi dei sistemi **rule-based**, che hanno segnato le prime tappe nella ricerca nel campo degli agenti conversazionali, per poi passare ai moderni approcci basati sull'apprendimento automatico con le reti neurali, prestando particolare attenzione ai modelli basati sull'architettura dei transformer.

Il Natural Language Understanding è un campo di ricerca in continua evoluzione, con tecniche e algoritmi costantemente aggiornati e nuovi modelli che vengono proposti e testati.

2.1 Chatbot rule-based

Fin dalle origini dello studio dell'elaborazione del linguaggio naturale, i cosiddetti chatbot *rule-based* (basati su regole) hanno costituito il primo approccio per simulare un interlocutore con cui gli utenti possano interagire col linguaggio naturale.
In questi sistemi le risposte del chatbot sono generate sulla base di pattern di input definiti in modo esplicito e di regole o template codificati dall'autore del programma. Tutte le possibili ramificazioni delle conversazioni sono previste e scelte in anticipo, al momento della progettazione.
Nonostante le moderne tecniche di apprendimento automatico abbiano guadagnato terreno, i chatbot rule-based mantengono un ruolo significativo, soprattutto quando è cruciale avere un controllo totale sulle risposte, mantenendo il flusso logico esplicito e operando in un dominio ristretto.

2.1.1 Le Origini con ELIZA

Uno dei primi e più noti esempi di chatbot rule-based è ELIZA, sviluppato da Joseph Weizenbaum al MIT nel 1966 [1], [2].
ELIZA simulava le risposte di uno psicoterapeuta rogersiano[1], basandosi su un semplice algoritmo di riscrittura (rewrite) delle frasi in input.
L'effetto era sorprendentemente convincente in alcuni casi, poichè ELIZA trasformava abilmente le frasi dell'utente in domande di ritorno, mantenendo un'apparenza di comprensione.

[1]Carl Rogers è stato uno psicologo statunitense, noto per gli studi sul counseling e fondatore della «psicoterapia incentrata sulla persona». Questo approccio portava alla costruzione di una relazione collaborativa e di fiducia tra terapista e paziente (per Rogers «cliente» dal momento che secondo lui non aveva un ruolo passivo ma attivo nella guarigione), basata sull'empatia, ritenuta essenziale per poter raggiungere il benessere psicologico [3].

Gli uomini sono tutti uguali.
IN CHE MODO
Ci infastidiscono sempre di una cosa o dell'altra.
PUOI PENSARE A UN ESEMPIO SPECIFICO?
Beh, il mio ragazzo mi ha fatta venire qui.
IL TUO RAGAZZO TI HA FATTA VENIRE QUI.
Mi dice sono depressa molto spesso. MI DISPIACE CHE TU SIA DEPRESSA.

Listato 1: Un esempio di conversazione con ELIZA da J. Weizenbaum [1]

2.1.1.1 Composizione

ELIZA si basava su **script di regole** scritte originariamente in SLIP (un dialetto di Lisp, un linguaggio basato sulla manipolazione dei simboli [4]), organizzate in una serie di **pattern** (decomposition rules) e **risposte template** (reassembly rules).
L'idea fondamentale era che ogni regola si attivasse se l'input dell'utente conteneva una certa parola chiave (keyword); ad ogni keyword era associato un insieme di "trasformazioni" più o meno generali, che permettevano di riformulare la frase dell'utente.

```
(HOW DO YOU 00. PLEASETELL ME YOUR PROBLEM)
START
(SORRY ( ( 0 ) (PLEASE DON'T A P O L I G I Z E )
(APOLOGIES ARE NOT NECESSARY) (WHAT FEELINGS
DO YOU HAVE WHEN YOU APOLOGIZE) ( l I V E TOLD YOU
THAT APOLOGIES ARE NOT REQUIRED)))
(DONT = DON'T)
(CANT = CAN'T)
(WONT = WON'T)
(REMEMBER S
( ( 0 YOU REMEMBER O) (DO YOU OFTEN THINK OF 4)
(DOES THINKING OF ~ BRING ANYTHING ELSE TO MINO)
(WHAT ELSE OO YOU REMEMBER)
(WHY DO YOU REMEMBER 4 JUST NOW)
(WHAT IN THE PRESENT SITUATION REMINDS YOU OF ~)
(WHAT IS THE CONNECTION BETWEEN ME AND ~))
((0 DO I REMEMBER 0) (DID YOU THINK I WOULD FORGET 5)
(WHY DO YOU THINK I SHOULD RECALL S NOW)
(WHAT ABOUT 5) (=WHAT) (YOU MENTIONED S))
((0) (NEWKEY)))
```

Codice 1: Un frammento delle regole che compongono ELIZA da J. Weizenbaum [1]

2.1.1.1.1 Parole chiave e priorità

Le regole di ELIZA erano raggruppate attorno a delle keyword, che definivano l'argomento o l'elemento su cui il sistema doveva focalizzare l'attenzione. Ad esempio, se la keyword era "REMEMBER", tutte le regole associate si occupavano di reinterpretare le frasi in cui l'utente accennava a un ricordo.

- Ogni keyword poteva avere una priorità numerica, indicando quanto fosse importante rispetto alle altre. In caso di input multiple che attivassero più keyword, veniva scelta quella con la priorità più alta.

- Se l'utente menzionava "IO NON RICORDO" ("I DON'T REMEMBER"), ELIZA cercava innanzitutto la keyword "REMEMBER" per attivare le regole di corrispondenza più specifiche.

2.1.1.1.2 Decomposition Rules

Ogni keyword era legata a una serie di decomposition rules che descrivevano come scomporre la frase dell'utente in componenti rilevanti. Questo avveniva tramite pattern contenenti placeholder e occasionalmente wildcard. Ad esempio, una decomposition rule per la keyword "REMEMBER" poteva apparire così (in forma semplificata)

```
( 0 YOU REMEMBER 0 )
```

Qui, il simbolo 0 rappresenta dei placeholder che potevano catturare qualunque parte del testo. Se la frase dell'utente era «I think **you remember** when I was a child», il sistema isolava le porzioni di testo prima e dopo la stringa "YOU REMEMBER".

Queste regole permettevano a ELIZA di individuare la struttura della frase e di memorizzare le parti variabili (ad esempio, tutto ciò che seguiva "YOU REMEMBER").

2.1.1.1.3 Reassembly Rules

Una volta trovato un pattern corrispondente, ELIZA applicava una corrispondente reassembly rule, ovvero un template che generava la risposta. Proseguendo con l'esempio della keyword "REMEMBER", se la decomposition rule catturava una parte di frase dopo "YOU REMEMBER", la reassembly rule selezionata poteva essere una delle seguenti:

```
(DO YOU OFTEN THINK OF 4?)
(WHY DO YOU REMEMBER 4 JUST NOW?)
```

e il sistema rispondeva sostituendo dinamicamente il placeholder con la parte di testo catturata.

Così, se l'utente diceva «Io penso che tu ricordi il mio compleanno», ELIZA poteva rispondere «Perché pensi che io ricordi il tuo compleanno proprio ora?»

2.1.1.1.4 Regole di Fallback

Se nessuna keyword né pattern specifico viene attivato, ELIZA passava a una serie di regole di fallback, spesso domande generiche come:

```
(TELL ME MORE)
(WHY DO YOU SAY THAT?)
```

Queste tornano utili a mantenere viva la conversazione anche quando non vi è corrispondenza con le regole definite, evitando il silenzio o un evidente messaggio di errore.

2.1.1.2 Conclusioni

È evidente comunque come ELIZA non fosse realmente in grado di comprendere il significato delle frasi: si limitava a riconoscere e riformulare le frasi usando parole chiave ottenute dalla conversazione. Un utente che ne è consapevole può potenzialmente «ingannare» il sistema con frasi prive di senso, ottenendo comunque risposte coerenti.

```
Il mio gatto è morto.
MI DISPIACE PER IL TUO GATTO.
Ora che è morto non so cosa fare.
COSA TI FA PENSARE DI NON SAPERE COSA FARE?
```

Listato 2: Un listato dal poco senso. (Povero gatto)

Ad ogni modo Weizenbaum riportò (con non poca preoccupazione per ciò che potesse implicare) che diversi utenti iniziarono a fare riferimento al programma umanizzandolo, attribuendogli sentimenti e personalità.
Una personalità che era, se non proprio di un terapeuta, perlomeno di un interlocutore percepito come utile.

Esiste un episodio celeberrimo che dimostra l'efficacia del sistema: anche la segretaria di Weizenbaum aveva iniziato ad interagire con esso. In un'occasione, gli chiese al ricercatore di lasciarla sola nella stanza per poter terminare privatamente la propria consulenza con ELIZA [2].

2.1.2 AIML

Negli anni '90 iniziò a guadagnare popolarità il Loebner Prize, una competizione ispirata al Test di Turing [5].
Nella competizione, chatbot e sistemi conversazionali cercavano di «ingannare» giudici umani, facendo credere loro di essere persone reali. Molti sistemi presentati alla competizione erano basati su pattern matching e rule-based, a volte integrando euristiche per la gestione di sinonimi o correzione ortografica.

Tra questi, uno dei più celebri è ALICE (Artificial Linguistic Internet Computer Entity), sviluppato da Richard Wallace utilizzando il linguaggio di markup AIML (Artificial Intelligence Markup Language) da lui introdotto [6], [7]. ALICE vinse per la prima volta il Loebner Prize nel 2000, e in seguito vinse altre due edizioni, nel 2001 e 2004.

2.1.2.1 Struttura di un chatbot AIML

Basato sull'XML, di base l'AIML fornisce una struttura formale per definire regole di conversazione attraverso "categorie" di pattern e template:

- `<pattern>` : la frase (o le frasi) a cui il chatbot deve reagire.
- `<template>` : la risposta (testuale o con elementi dinamici) che il chatbot fornisce quando si verifica il match del pattern.

Considerando ad esempio la seguente configurazione:

```
<category>
  <pattern>CIAO</pattern>
  <template>Ciao! Come posso aiutarti oggi?</template>
</category>
```

se l'utente scrivesse «Ciao»[2], il sistema risponderebbe con «Ciao! Come posso aiutarti oggi?».

Grazie all'uso di wildcard (*, _) e a elementi di personalizzazione (`<star/>`), AIML può gestire un certo grado di variabilità linguistica:

```
<category>
  <pattern>MI CHIAMO *</pattern>
  <template>
    Ciao <star/>, piacere di conoscerti!
  </template>
</category>
```

In questo caso, se l'utente scrivesse «Mi chiamo Andrea», il sistema risponderebbe con «Ciao Andrea, piacere di conoscerti!».

Esistono anche tag per permettere di memorizzare informazioni e utilizzarle in seguito, come `<set>` e `<get>`, e per gestire la conversazione in modo più dinamico, come `<think>` e `<condition>`:

```
<category>
  <pattern>CHE TEMPO FA</pattern>
  <template>
    <condition name="stagione">
      <li value="inverno">Fa piuttosto freddo, in questa stagione.</li>
      <li value="estate">Fa molto caldo, bevi tanta acqua!</li>
    </condition>
  </template>
</category>
```

È inoltre possibile raggruppare categorie simili con il tag `<topic>`, e riindirizzare la conversazione verso un argomento specifico con il tag `<srai>`:

```
<topic name="saluti">
  <category>
    <pattern>SALUTA *</pattern>
    <template>
      <srai>CIAO</srai>
    </template>
  </category>
</topic>
```

[2]Caratteri maiuscoli e minuscoli sono considerati uguali dal motore di riconoscimento.

2.1.2.2 Conclusioni su AIML

Grazie ai tag previsti dallo schema, AIML riesce a gestire conversazioni piuttosto complesse. Ciononostante, presenta comunque alcune limitazioni:

- Le strategie di wildcard e pattern matching restano prevalentemente letterali, con limitata capacità di interpretare varianti linguistiche non codificate nelle regole. Se una frase si discosta dal pattern previsto, il sistema fallisce il matching. Sono disponibili comunque alcune funzionalità per la gestione di sinonimi, semplificazione delle locuzioni e correzione ortografica, che devono essere costruite manualmente.
- La gestione del contesto (via `<that>, <topic>`, ecc.) è rudimentale, soprattutto se paragonata a sistemi moderni di NLU con modelli neurali che apprendono contesti ampi.
- L'integrazione con basi di conoscenza esterne (KB, database, API) richiede estensioni proprietarie o script custom, poiché AIML di per sé non offre costrutti semantici o query integrate, e non permette di integrare script internamente alle regole.

Nonostante questi limiti, AIML ha rappresentato un passo importante nell'evoluzione dei chatbot, offrendo un framework standardizzato e relativamente user-friendly per la creazione di agenti rule-based.
In alcuni ambiti ristretti (FAQ ripetitive, conversazioni scriptate), costituisce ancora una soluzione valida e immediata. In domini più complessi, in cui la varietà del linguaggio e l'integrazione con dati dinamici sono essenziali, diventa indispensabile affiancare o sostituire AIML con tecniche di Natural Language Understanding basate su machine learning e deep learning.

2.2 Natural Language Understanding e Stato dell'Arte

Con **Natural Language Understanding** (NLU) si fa riferimento a un insieme di tecniche e modelli che mirano a comprendere il testo in ingresso a un livello più semantico, superando la semplice analisi di pattern o keyword. Negli ultimi anni, la ricerca si è orientata verso modelli di machine learning, e in particolare di deep learning, capaci di catturare caratteristiche sintattiche, semantiche e contestuali.

La transizione dai sistemi rule-based verso metodi data-driven nel Natural Language Processing (NLP) è stata inizialmente guidata dall'impiego di Reti Neurali Ricorrenti (RNN), e solo recentemente è stata rivoluzionata dall'introduzione dei Transformer.

Queste architetture hanno permesso di passare da un'analisi del linguaggio ancorata a pattern e regole scritte a mano a un'interpretazione basata su features apprese automaticamente dai dati, in grado di cogliere sfumature sintattiche e semantiche molto più complesse.

2.2.1 Recurrent Neural Networks

Le reti neurali ricorrenti (RNN) sono un tipo di architettura di rete neurale progettata per analizzare sequenze di dati. Questa caratteristica le rende particolarmente adatte per compiti come modellazione del linguaggio, riconoscimento vocale, descrizione di immagini e tagging video. Introdotte negli anni '80, le RNN hanno rivoluzionato l'elaborazione sequenziale, superando i limiti delle reti feedforward: mentre una FFN trasporta informazioni solo in avanti attraverso la rete, le RNN funzionano a cicli, e passano le informazioni di nuovo a se stesse ad ogni step [8].

2.2.1.1 Intuizione di base

La principale differenza tra le RNN e le reti neurali tradizionali feedforward è la capacità delle prime di mantenere uno «stato» interno che rappresenta il contesto storico. Questo stato viene aggiornato in ogni passo temporale t, consentendo alla rete di considerare input precedenti ($\boldsymbol{X}_{0\ldots t-1}$) quando elabora quello corrente $\boldsymbol{X}_t$.

In termini pratici, le RNN sono progettate per «ricordare» informazioni rilevanti nel tempo, modellando così dipendenze sequenziali. Questo è rappresentato matematicamente come:

$$\boldsymbol{H}_t = \phi_h(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h) \tag{1}$$

Dove:

- $\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$ è lo stato nascosto al tempo t, che rappresenta un sommario dell'input corrente e della storia precedente.
- $\boldsymbol{X}_t \in \mathbb{R}^{n \times d}$ è l'input al tempo t.
- $\boldsymbol{W}_{xh} \in \mathbb{R}^{d \times h}$ e $\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$ sono matrici di peso rispettivamente per l'input e lo stato nascosto, addestrate durante il processo di apprendimento.
- $\boldsymbol{b}_h$ è il bias, un vettore che introduce flessibilità nella rappresentazione.

- ϕ_h è una funzione di attivazione, spesso una funzione non lineare come la tangente iperbolica o la sigmoide, utilizzata per garantire la continuità e stabilità dei gradienti.

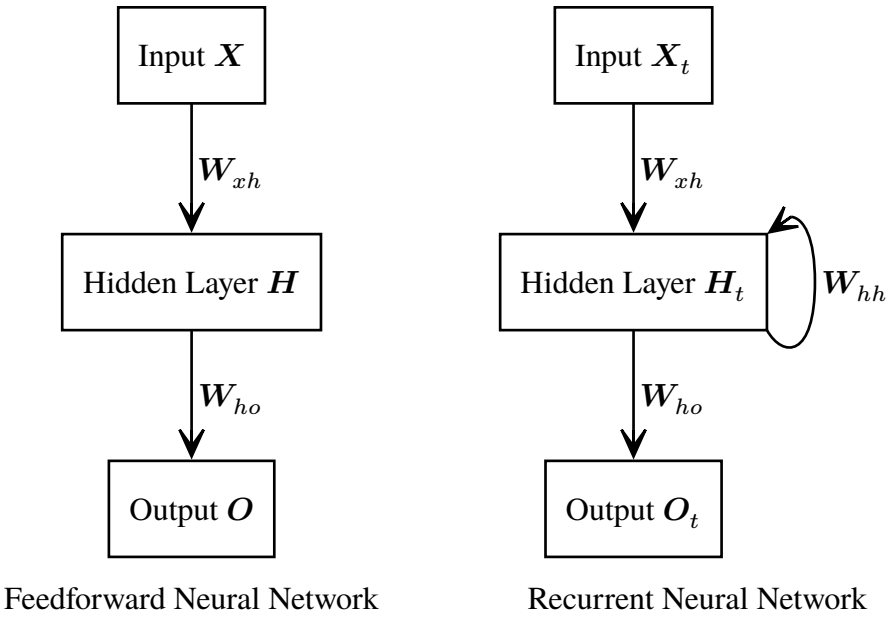

Figura 1: Schema di una rete feedforward (FFN) e di una rete ricorrente (RNN)

L'output al tempo t è poi calcolato come:

$$\boldsymbol{O}_t = \phi_o(\boldsymbol{H}_t \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{2}$$

Come per le FFN, l'addestramento viene effettuato tramite la backpropagation.
In questo caso tuttavia, il calcolo del gradiente deve tener conto della dipendenza temporale, e viene effettuato attraverso l'algoritmo di *backpropagation through time* (BPTT). Questo metodo «srotola» la rete nel tempo, trattando ogni passo temporale come un layer separato, e calcola i gradienti per ogni passo.

La funzione di loss viene calcolata sommando le loss di ogni passo temporale

$$\mathcal{L}(\boldsymbol{O}, \boldsymbol{Y}) = \sum_{t-1}^{T} \ell_t(\boldsymbol{O}_t, \boldsymbol{Y}_t) \tag{3}$$

dove ℓ_t è la funzione di loss al tempo t, O_t è l'output predetto e Y_t è l'output atteso.

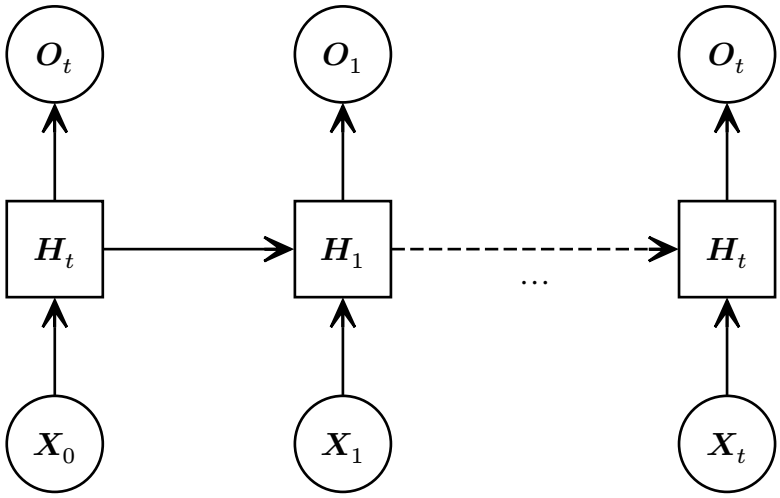

Figura 2: Una RNN srotolata nel tempo

2.2.1.2 Problemi delle RNN

Le RNN nonostante la capacità di catturare dipendenze temporali, soffrono di problemi di esplosione o vanishing del gradiente in modo particolarmente acuto.

Infatti, dal momento che durante il training si può lavorare con stringhe (potenzialmente molto lunghe), se gestiamo valori di gradiente molto piccoli o molto grandi, la moltiplicazione di molti di questi valori può portare a valori di gradiente troppo piccoli o troppo grandi, compromettendo la convergenza del modello.

Questo problema ha spinto alla ricerca di architetture alternative, come le Long Short-Term Memory (LSTM) e le Gated Recurrent Unit (GRU), che introducono meccanismi di regolazione del flusso di informazione.

2.2.2 Long Short-Term Memory

Il problema del vanishing/exploding del gradiente non si ferma solo alla fase di apprendimento, ma può influenzare anche la capacità della rete di memorizzare informazioni a lungo termine [9].

La lunghezza del testo da considerare è uno dei fattori: con una frase corta come «In cielo ci sono le nuvole» è facile per una RNN predire che la parola successiva sarà «nuvole» dopo aver visto «cielo».

Ci sono però anche casi in cui per poter effettuare la nostra predizione avremo bisogno di informazioni che si trovano molto più lontane nel testo. Man mano che la distanza temporale tra le informazioni necessarie e il punto in cui ci troviamo aumenta, le RNN non riescono più a creare collegamenti tra le informazioni [10], [11].

Per questo motivo sono state concepite le Long Short-Term Memory (LSTM), progettate nel ’97 da S. Hochreiter e J. Schmidhuber [12]. La loro architettura è inten-

zionalmente progettata per evitare i problemi descritti nella Sezione 2.2.1.2, e i risultati sono stati molto positivi [9], [13].

Esattamente come per le RNN, le LSTM sono reti ricorrenti, per cui si passano i dati in output anche come input. La differenza principale è che le LSTM hanno un meccanismo di controllo che permette di decidere quali informazioni mantenere e quali scartare: per far ciò non viene più utilizzata una rete, ma quattro:

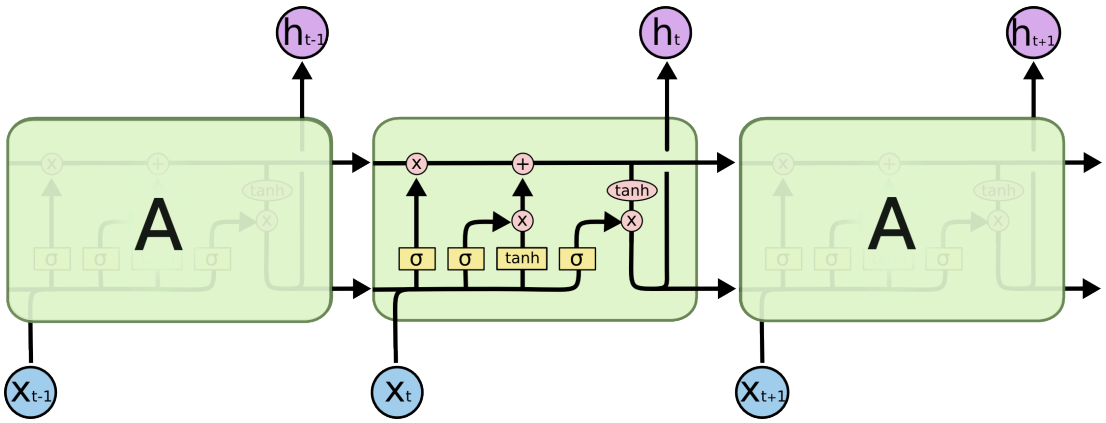

Figura 3: Schema di una cella LSTM

2.2.3 Transformers

2.2.4 LLM

2.3 Framework e strumenti moderni

2.3.1 LangChain e Haystack

https://haystack.deepset.ai/ (https://www.reddit.com/r/LocalLLaMA/comments/1dxj1mo/langchain_bad_i_get_it_what_about_langgraph/)

2.4 Conclusioni e gap da colmare

Bibliografia

[1] J. Weizenbaum, «ELIZA—a computer program for the study of natural language communication between man and machine», *Commun. ACM*, vol. 9, fasc. 1, pp. 36–45, gen. 1966, doi: 10.1145/365153.365168.

[2] C. Bassett, «The computational therapeutic: exploring Weizenbaum's ELIZA as a history of the present», *AI & SOCIETY*, vol. 34, fasc. 4, pp. 803–812, dic. 2019, doi: 10.1007/s00146-018-0825-9.

[3] C. ROGERS, «The Necessary and Sufficient Conditions of Psychotherapeutic Personality Change», *Psychotherapy (Chicago, Ill.)*, vol. 44, pp. 240–248, 2007, doi: 10.1037/0033-3204.44.3.240.

[4] P. W. Abrahams, «Symbol Manipulation Languages», vol. 9. in Advances in Computers, vol. 9. Elsevier, pp. 51–111, 1969. doi: https://doi.org/10.1016/S0065-2458(08)60311-3.

[5] A. M. TURING, «I.—COMPUTING MACHINERY AND INTELLIGENCE», *Mind*, vol. 59, fasc. 236, pp. 433–460, 1950, doi: 10.1093/mind/LIX.236.433.

[6] «Artificial Intelligence Markup Language». [Online]. Disponibile su: http://www.aiml.foundation/doc.html

[7] R. Wallace, «The anatomy of A.L.I.C.E», 2009, pp. 181–210. doi: 10.1007/978-1-4020-6710-5_13.

[8] R. M. Schmidt, «Recurrent Neural Networks (RNNs): A gentle Introduction and Overview». [Online]. Disponibile su: https://arxiv.org/abs/1912.05911

[9] Christopher Olah, «Understanding LSTM Networks». [Online]. Disponibile su: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] S. Hochreiter, «Untersuchungen zu dynamischen neuronalen Netzen», p. , 1991.

[11] Y. Bengio, P. Simard, e P. Frasconi, «Learning long-term dependencies with gradient descent is difficult», *IEEE Transactions on Neural Networks*, vol. 5, fasc. 2, pp. 157–166, mar. 1994, doi: 10.1109/72.279181.

[12] S. Hochreiter e J. Schmidhuber, «Long Short-Term Memory», *Neural Computation*, vol. 9, fasc. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[13] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, e M. Weyrich, «A survey on long short-term memory networks for time series prediction», *Procedia CIRP*, vol. 99, pp. 650–655, 2021, doi: https://doi.org/10.1016/j.procir.2021.03.088.

[14] *UniTO typst template*. [Online]. Disponibile su: https://github.com/eduardz1/UniTO-typst-template

[15] «Typst». [Online]. Disponibile su: http://typst.app/

[16] D. Jurafsky e J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed. 2025. [Online]. Disponibile su: https://web.stanford.edu/~jurafsky/slp3/

[17] D. E. Rumelhart, G. E. Hinton, e R. J. Williams, «Learning internal representations by error propagation», in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, 1986, pp. 318–362.

[18] J. Devlin, M.-W. Chang, K. Lee, e K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». [Online]. Disponibile su: https://arxiv.org/abs/1810.04805

[19] V. Sanh, L. Debut, J. Chaumond, e T. Wolf, «DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter». [Online]. Disponibile su: https://arxiv.org/abs/1910.01108

[20] S. Gururangan *et al.*, «Don't Stop Pretraining: Adapt Language Models to Domains and Tasks». [Online]. Disponibile su: https://arxiv.org/abs/2004.10964

[21] T. Wolf *et al.*, «HuggingFace's Transformers: State-of-the-art Natural Language Processing». [Online]. Disponibile su: https://arxiv.org/abs/1910.03771

[22] M. Mosbach, M. Andriushchenko, e D. Klakow, «On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines». [Online]. Disponibile su: https://arxiv.org/abs/2006.04884

[23] *Ollama - LLM local runner*. [Online]. Disponibile su: https://github.com/ollama/ollama

[24] G. Team *et al.*, «Gemma 2: Improving Open Language Models at a Practical Size», 2024. [Online]. Disponibile su: https://arxiv.org/abs/2408.00118

[25] A. Grattafiori *et al.*, «The Llama 3 Herd of Models». [Online]. Disponibile su: https://arxiv.org/abs/2407.21783

[26] J. Bai *et al.*, «Qwen Technical Report». [Online]. Disponibile su: https://arxiv.org/abs/2309.16609

[27] P. Rajpurkar, J. Zhang, K. Lopyrev, e P. Liang, «SQuAD: 100,000+ Questions for Machine Comprehension of Text», in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, e X. Carreras, A c. di, Association for Computational Linguistics, nov. 2016, pp. 2383–2392. doi: 10.18653/v1/D16-1264.

[28] P. Rajpurkar, R. Jia, e P. Liang, «Know What You Don't Know: Unanswerable Questions for SQuAD», in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, I. Gurevych e Y. Miyao, A c. di, Association for Computational Linguistics, lug. 2018, pp. 784–789. doi: 10.18653/v1/P18-2124.

[29] R. Caruana, «Multitask Learning», *Machine Learning*, vol. 28, fasc. 1, pp. 41–75, 1997, doi: 10.1023/A:1007379606734.

[30] C. N. Silla e A. A. Freitas, «A survey of hierarchical classification across different application domains», *Data Mining and Knowledge Discovery*, vol. 22, fasc. 1, pp. 31–72, 2011, doi: 10.1007/s10618-010-0175-9.

[31] H. Nakayama, T. Kubo, J. Kamura, Y. Taniguchi, e X. Liang, «doccano: Text Annotation Tool for Human». [Online]. Disponibile su: https://github.com/doccano/doccano

[32] James Saryerwinnie, «JMESPath is a query language for JSON». [Online]. Disponibile su: https://jmespath.org/

[33] Stefan Goessner, «JSONPath - XPath for JSON». [Online]. Disponibile su: https://goessner.net/articles/JsonPath/

[34] «SPARQL Query Language for RDF». [Online]. Disponibile su: https://www.w3.org/TR/sparql11-query/

[35] «Graphviz - Graph Visualization Software». [Online]. Disponibile su: https://graphviz.org/

[36] «Handlebars - Minimal Templating on Steroids». [Online]. Disponibile su: https://handlebarsjs.com/

[37] «FastAPI - Fast (high-performance) API framework for Python». [Online]. Disponibile su: https://fastapi.tiangolo.com/