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 (da comporre e aggiornare manualmente) che possono mitigare alcuni di questi problemi.
- 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 e riescono a tenere traccia di dettagli dal passato della conversazione.
- L'integrazione con basi di conoscenza esterne (KB, database, API) richiede estensioni o script sviluppati ad-hoc, poiché AIML di per sé non offre costrutti semantici o query integrate, e non permette di integrare script internamente alle regole [6].

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 [7].
In alcuni ambiti ristretti (FAQ, conversazioni scriptate, assistenti vocali), 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

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.

Come vedremo, le RNN sono state fondamentali per introdurre la capacità di modellare dipendenze temporali nei dati, ma presentano limiti legati al vanishing/exploding del gradiente e alla difficoltà di mantenere informazioni a lungo termine.
Ciononostante, le RNN hanno gettato le basi per l'evoluzione delle architetture neurali, e sono state fondamentali per l'introduzione di modelli più avanzati come le Long Short-Term Memory (LSTM) che poi hanno portato all'ideazione delle tecniche che caratterizzano i Transformer.

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 quali:

- modellazione del linguaggio
- riconoscimento vocale
- descrizione di immagini
- 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 di mantenere uno «stato» interno che rappresenta il contesto storico.
Questo stato viene aggiornato in ogni passo temporale t, consentendo alla rete di tenere in considerazione gli input precedenti ($\boldsymbol{X}_0, \boldsymbol{X}_1, ..., \boldsymbol{X}_{t-1}$) 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.

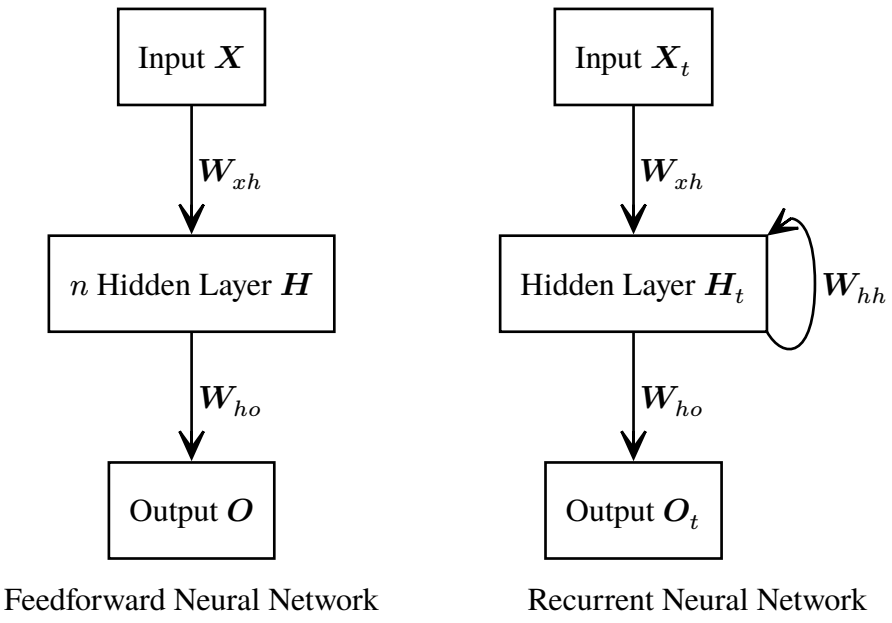

Figura 1: Schema di una rete feedforward (FFN) e di una rete ricorrente (RNN)

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 (dopo aver mostrato l'esempio

attualmente trattato per l'apprendimento), 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.

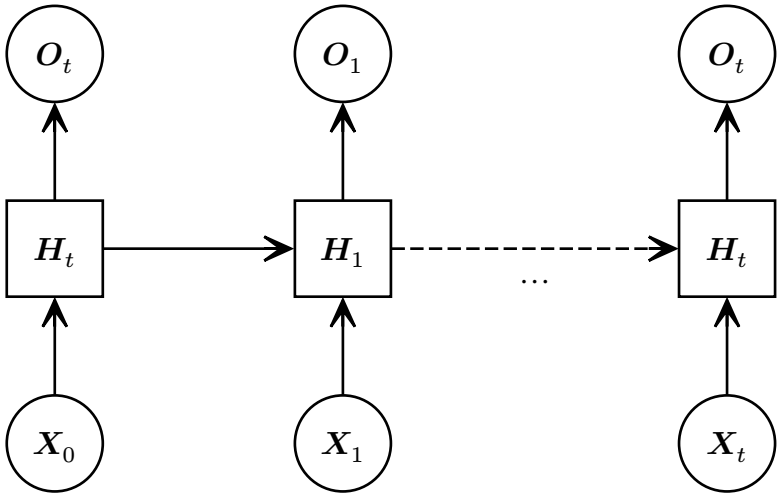

Figura 2: Una RNN srotolata nel tempo

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 Gated Recurrent Unit (GRU) e le Long Short-Term Memory (LSTM), 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 «Nel 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 intenzionalmente 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 utilizzato un solo layer, ma quattro:

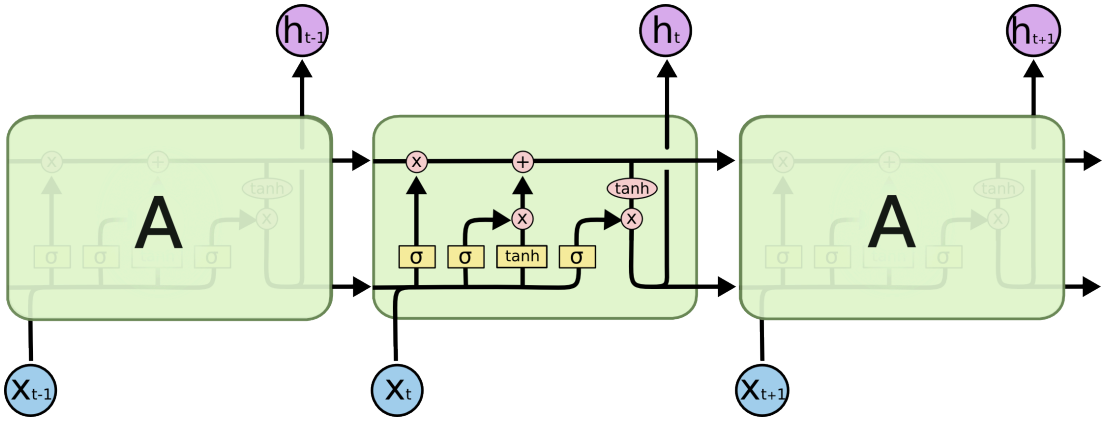

Figura 3: Schema di una cella LSTM, cortesia di Christopher Olah [9].

Rimanendo ad un livello concettuale, possiamo dire che una cella LSTM è composta da tre «gate»:

Il primo, il **Forget Gate**, decide quali informazioni scartare dallo stato nascosto (e quindi dimenticare). Coincide con un layer fully connected con funzione di attivazione sigmoide:

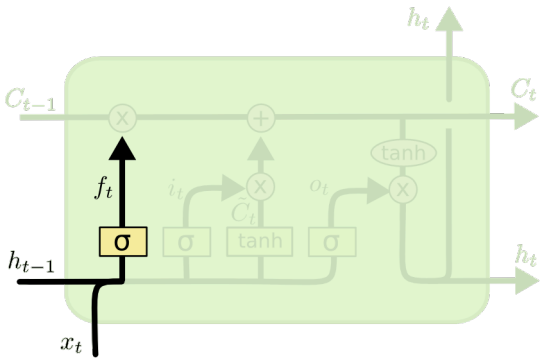

Figura 4: Forget Gate evidenziato [9].

Il passo successivo decide quali nuove informazioni aggiungere allo stato nascosto, cioè lo stato della cella LSTM. Questo passaggio è chiamato **Input Gate**. La sigmoide in questo caso decide quali valori aggiornare nello stato, mentre la funzione tangente iperbolica crea un vettore di valori candidati da aggiungere allo stato:

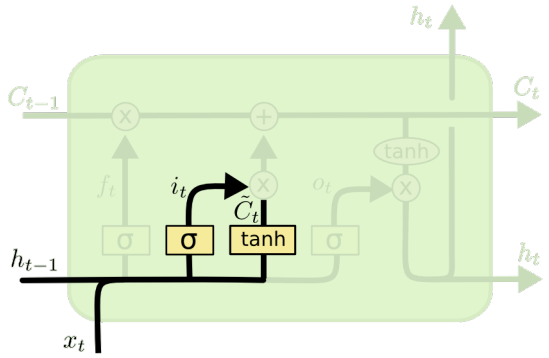

Figura 5: Input Gate evidenziato [9].

Dopo che abbiamo conservato tutti i valori che vogliamo, e scartato quelli che non riteniamo più utili, possiamo calcolare il nuovo stato della cella. Una volta fatto, possiamo decidere quali informazioni passare in output. Questo passaggio è chiamato **Output Gate**:

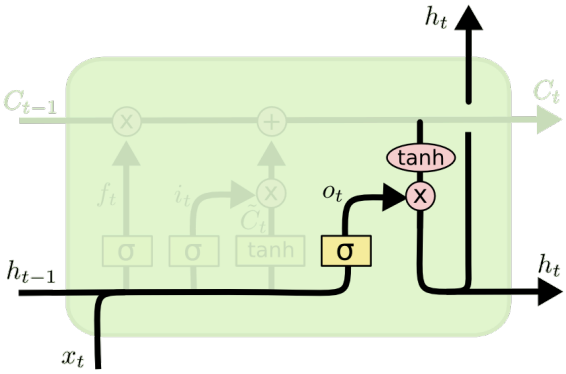

Figura 6: Output Gate evidenziato [9].

Lo stesso processo viene ripetuto per ogni passo temporale, permettendo alla LSTM di mantenere informazioni a lungo termine e di evitare i problemi di vanishing/exploding gradient.

2.2.2.1 Impatto

Da quando le LSTM sono state introdotte, sono diventate uno dei modelli più utilizzati per la modellazione di sequenze, e sono state adottate in una vasta gamma di applicazioni. Nonostante in molti casi siano state superate dai Transformer, le LSTM sono state utilizzate con grande successo in molteplici campi:

- Nel 2015 Google ha annunciato di aver implementato le LSTM in Google Voice [14], [15], riducendo gli errori di trascrizione del 49% [16].

- Nel 2016 le LSTM vengono sempre più usate:
  - Sempre Google implementa le LSTM anche in Google Translate, riducendo gli errori di traduzione del 60% [17], [18].
  - Apple le adopera per quicktype e Siri [19], mentre Amazon implementa una LSTM bidirezionale da utilizzare per le operazioni di TTS con Alexa [20].
- Nel 2017 Facebook riporta di aver utilizzato le LSTM per la traduzione automatica di più di 4.5 miliardi di testi al giorno [21], mentre Microsoft afferma di aver raggiunto un'accuratezza del 94.9% nel riconoscimento [22]
- Nel 2018 OpenAI utilizza le LSTM per creare un modello in grado di battere dei giocatori umani al videogioco MOBA Dota 2 [23], e per controllare una mano robotica in grado di manipolare gli oggetti con una precisione simile a quella umana [24]. In modo simile anche Google Deep Mind utilizza le LSTM per creare un modello in grado di battere i giocatori umani al gioco di strategia in tempo reale Starcraft II [25].

Possiamo quindi affermare che le LSTM abbiano effettivamente avuto un impatto concreto e importante sulla vita di tutti i giorni [26]. Hanno permesso di migliorare la qualità di servizi già esistenti, e di creare nuove applicazioni che prima non sarebbero state possibili. Inoltre, sui concetti alla base delle LSTM, sono state costruite le basi per la creazione dei Transformer, che hanno portato poi alla nuova età dell'oro dell'Intelligenza Artificiale generativa.

2.2.3 Transformers

Il meccanismo introdotto con le LSTM possiamo considerarlo come un processo in cui il modello impara su quali informazioni durante il passaggio del tempo sia più importante prestare attenzione.

L'architettura dei Transformer, introdotta da A. Vaswani *et al.* [27], parte da questa idea e la porta all'estremo, basandosi solamente su di essa per creare un modello in grado di catturare le dipendenze a lungo termine. Non utilizzando più reti ricorrenti, l'architettura è composta da diversi encoders e decoders che lavorano in parallelo, e che si scambiano informazioni tramite meccanismi di attenzione. Questo permette di lavorare su sequenze di dati in parallelo, riducendo i tempi di addestramento e migliorando le prestazioni.

2.2.3.1 Intuizione di base

Il meccanismo di «self-attention» è il cuore dell'architettura dei Transformer. Questo meccanismo consente al modello di decidere l'importanza di ogni parola[3] in una sequenza rispetto alle altre, senza essere limitato dalla loro distanza posizionale. Ciò

[3]In questa tesi ci concentreremo prevalentemente sull'utilizzo dei transformer con il fine di creare modelli linguistici.

permette ai Transformer di identificare e concentrarsi sulle parti più rilevanti dell'input, anche in sequenze molto lunghe.

Tuttavia, questo approccio comporta una complessità computazionale quadratica rispetto alla lunghezza della sequenza, rendendolo costoso in termini di risorse di calcolo e memoria.

Per gestire questa complessità e migliorare le prestazioni, si utilizza una context window per limitare il numero di token considerati in ogni calcolo di attenzione. Questo bilancia la capacità del modello di cogliere relazioni rilevanti senza esaurire le risorse disponibili, specialmente nei casi in cui si affrontano set di dati di grandi dimensioni.

2.2.3.2 Struttura del Transformer

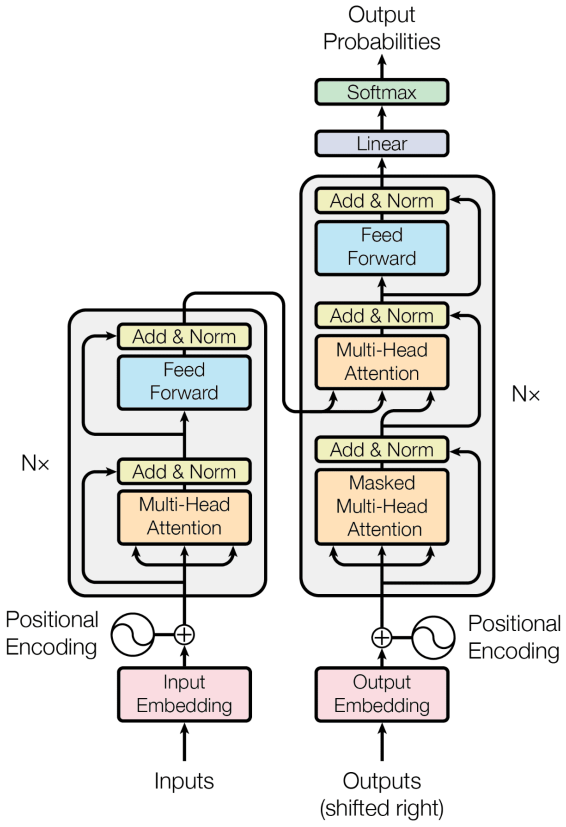

Figura 7: Architettura di un Transformer [27].

L'architettura dei Transformer è composta da due parti principali: l'encoder e il decoder. L'encoder elabora l'input, mentre il decoder genera l'output[4].

Normalmente si utilizza una sequenza di encoder e decoder, con in cima un layer di output composto da una rete neurale fully connected, che sceglie la parola successiva da generare.

[4]Spesso è utilizzato nei modelli di traduzione automatica

Vediamo le componenti più in dettaglio:

- **Positional Encoding**: Poiché i Transformer non processano sequenzialmente i dati, le informazioni sulla posizione delle parole vengono aggiunte tramite codifiche posizionali sinusoidali. In questo modo il modello è in grado di differenziare i token basandosi sulla loro posizione relativa nella sequenza.
- **Encoder**: è costituito da una serie di livelli identici, ognuno dei quali comprende un modulo di auto-attenzione multi-head e una rete feedforward.
  L'auto-attenzione permette di identificare relazioni tra le parole nella sequenza di input, mentre la rete feedforward esegue trasformazioni non lineari per migliorare la rappresentazione dei dati. Ogni livello è dotato di dropout per prevenire l'overfitting e di normalizzazione layer-wise per stabilizzare l'addestramento.
- **Decoder**: Simile all'encoder, il decoder utilizza un modulo di auto-attenzione per elaborare il contesto generato in precedenza e un modulo di *cross-attention* per incorporare le informazioni elaborate dall'encoder.
  Questo processo consente al modello di generare sequenze di output con un forte legame semantico con l'input originale.
- **Multi-Head Attention**: Questo componente consente al modello di applicare meccanismi di attenzione paralleli su diverse sotto-rappresentazioni dei dati.
  Gli output di queste *attention head* vengono combinate e trasformate, migliorando la comprensione del contesto a più livelli.
- Ogni livello del Transformer contiene una *Rete Neurale Feedforward* composta da due strati completamente connessi e una funzione di attivazione intermedia, solitamente la ReLU[5].

$$\mathrm{ReLU}(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

  Questa rete aggiunge complessità non lineare al modello.

2.2.3.3 Impatto

Dal loro sviluppo, i Transformer hanno rivoluzionato il panorama dell'AI, portando alla nascita delle LLM.
Modelli come GPT e BERT hanno dimostrato le loro potenzialità in numerosi compiti, dal completamento del testo alla traduzione automatica, fino alla generazione di contenuti creativi. Inoltre, i Transformer hanno trovato applicazione in campi come la bioinformatica, la visione artificiale e l'apprendimento multimodale.

Negli ultimi anni, i Transformer sono diventati una tecnologia fondamentale per molti prodotti utilizzati quotidianamente.

[5]Rectified Liear Unit

Grandi aziende come Google, OpenAI e Microsoft li hanno integrati nei loro servizi[6], dai motori di ricerca come Google Search ai sistemi di assistenza virtuale come ChatGPT, Claude e Gemini.

Parallelamente, i Transformer hanno trasformato l'interazione tra utenti e macchine, rendendo più intuitivi e umanizzati i servizi basati sull'intelligenza artificiale. La capacità di questi modelli di comprendere e generare linguaggio naturale ha migliorato significativamente l'accessibilità tecnologica, consentendo un'interazione più fluida anche per utenti con conoscenze tecniche limitate.

2.2.4 LLM: Large Language Model

2.2.4.1 Introduzione

Come abbiamo visto nel capitolo precedente, i Transformer hanno rivoluzionato l'approccio all'elaborazione del linguaggio naturale (NLP), consentendo di cogliere relazioni a lungo raggio all'interno di sequenze di token e aprendo la strada a nuove applicazioni su vasta scala. La loro capacità di gestire contesti lunghi, unita all'efficienza (relativa) del calcolo in parallelo e ai meccanismi di attenzione, ha reso possibili modelli sempre più performanti.

Questa evoluzione ha condotto alla nascita e alla diffusione di modelli di grandi dimensioni, i cosiddetti Large Language Model (LLM), che sono tutt'ora al centro di numerose ricerche e applicazioni nel campo dell'Intelligenza Artificiale.

2.2.4.2 Che cosa sono i Large Language Model

Un Large Language Model è un modello di apprendimento automatico, tipicamente basato sull'architettura Transformer, addestrato su grandi quantità di testo non supervisionato (o debolmente supervisionato) per imparare strutture linguistiche e conoscenze generali sul mondo[7].

L'idea alla base degli LLM è che, addestrando una rete neurale con miliardi (o addirittura trilioni) di parametri su enormi dataset testuali, il modello riesca a catturare pattern e relazioni semantiche che ne abilitano un vasto range di applicazioni, dal riassunto automatico alla traduzione, fino al dialogo naturale e alla generazione di codice.

Tra i principali esempi di LLM troviamo:

- GPT (Generative Pre-trained Transformer) di OpenAI, in diverse versioni dal GPT-2 fino agli attuali GPT-4 e successivi;
- BERT (Bidirectional Encoder Representations from Transformers) di Google e i suoi successori come RoBERTa, DistilBERT, ecc.;

[6] Anche se a volte con grossi problemi, si veda l'articolo della BBC scritto da L. McMahon e Z. Kleinman [28].

[7] Alcuni modelli, come GPT-3, sono stati addestrati su centinaia di miliardi di token provenienti da fonti quali libri, articoli, siti web, forum, ecc.

- PaLM di Google, specializzato in compiti di generazione e ragionamento testuale;
- LLaMA di Meta, incentrato su efficienza di addestramento e inferenza;
- Claude di Anthropic, progettato per una gestione delle conversazioni più «sicura» e con minore propensione a generare contenuti problematici.
- Deep-Seek R1 che, rilasciato da pochissimo sotto licenza MIT, ha raggiunto le performance di GPT o1 con una frazione del numero di parametri [citazioni da aggiungere]

2.2.4.3 Componenti chiave di un LLM

Analogamente ai Transformer presentati in precedenza, gli LLM fanno largo uso di:

- **Self-Attention**: l'elemento chiave che permette al modello di catturare le relazioni tra token a qualsiasi distanza all'interno di una sequenza.
- **Architettura multi-head**: l'attenzione viene calcolata in parallelo su diverse «teste» per estrarre informazioni da prospettive diverse.
- **Feedforward Network**: uno o più livelli di reti neurali fully connected per introdurre non linearità e combinare le informazioni estratte dai blocchi di attenzione.
- **Positional Encoding o analoghe tecniche posizionali**: per incorporare la nozione di ordine dei token, cruciale nel linguaggio naturale.
- **Meccanismi di dropout e layer normalization**: utili a stabilizzare e rendere più robusta la fase di addestramento.

A differenza dei Transformer di dimensioni moderate, però, gli LLM:

- Presentano un numero di parametri molto maggiore (**dai miliardi ai trilioni**);
- Necessitano di **dataset estesi e variegati**, spesso raccolti da diverse fonti testuali;
- Richiedono infrastrutture di calcolo complesse ed estremamente costose (GPU/TPU di ultima generazione, cluster ad alta parallelizzazione) per l'addestramento.

2.2.4.4 Addestramento su larga scala

L'addestramento di un LLM si articola principalmente in due fasi:

2.2.4.4.1 Pre-training:

Il modello viene esposto a enormi quantità di testo non etichettato tramite task di previsione del token successivo (es. «mascherare» una parola e chiederne la ricostruzione, o prevedere direttamente il token futuro in una sequenza). In questa fase il modello impara una rappresentazione generale della lingua, acquisendo conoscenze sintattiche, semantiche e persino di tipo enciclopedico.

Ad esempio, GPT si basa su un causal language modeling, dove a ogni passo viene predetto il token successivo dato il contesto precedente; BERT, invece, utilizza una strategia masked language modeling, mascherando una porzione di token e chiedendo al modello di recuperarli.

2.2.4.4.2 Fine-tuning:

Dopo il pre-training, l'LLM può essere specializzato su un compito specifico (traduzione, Q&A, classificazione, ecc.) utilizzando dataset etichettati più piccoli. Questo processo richiede meno dati di quanti ne servano per l'intero addestramento «da zero» e sfrutta la conoscenza «generica» acquisita in precedenza.

In alcuni casi si usa anche un meccanismo di «Instruction Tuning», dove al modello vengono date istruzioni esplicite e esempi di dialogo, allo scopo di renderlo più conforme alle esigenze dell'utente (ad esempio, generare risposte più coerenti e rispettose dei vincoli di sicurezza).

Un'ulteriore evoluzione è il RLHF (Reinforcement Learning from Human Feedback), che affina il modello basandosi su giudizi umani riguardo alla qualità delle risposte, garantendo interazioni più naturali e utili[8].

2.2.4.5 Potenzialità e applicazioni

Gli LLM hanno dimostrato di eccellere in numerose aree, tra cui spiccano:

- **Traduzione automatica**: grazie alla loro profonda conoscenza del lessico e delle strutture sintattiche, possono produrre traduzioni di qualità paragonabile — se non superiore — a molte soluzioni tradizionali.
- **Completamento del testo**: tool come Copilot (che fa uso sia di GPT sia di Claude) possono generare segmenti di testo coerenti dato un prompt iniziale, con applicazioni che vanno dalla scrittura creativa alla redazione di email e documenti, per finire allo sviluppo di frammenti di codice.
- **Riepilogo e sintetizzazione**: le LLM possono riassumere articoli o lunghi documenti, evidenziando i punti chiave e riducendo il carico cognitivo dell'utente.
- **Dialogo e virtual assistant**: i modelli addestrati a rispondere alle richieste degli utenti in modo contestuale (ad esempio ChatGPT, Claude e Gemini) hanno già rivoluzionato l'interazione uomo-macchina, online o sui dispositivi degli utenti.

2.2.4.6 Sfide e limiti

Nonostante l'impressionante potenziale, dobbiamo ricordare che nulla ha solo lati positivi. Gli LLM presentano alcune criticità e limiti che ne condizionano l'utilizzo e la diffusione, tra cui possiamo evidenziare:

- **Allucinazioni e inesattezze**: il modello può generare informazioni errate o inesistenti (le cosiddette hallucination), specie se non ha ricevuto istruzioni o feedback umano adeguato.
  Spesso è un problema legato all'addestramento: è difficile che un modello ad una domanda risponda dicendo «Non lo so». Questi modelli in fondo sono addestrati

[8] Alcuni modelli di ChatGPT sono stati ottimizzati proprio attraverso RLHF, valutando l'utilità delle risposte generate e penalizzando contenuti inappropriati o indesiderati dagli utenti.

per generare risposte e interagire nelle conversazioni, non per riconoscere quando non hanno abbastanza informazioni per rispondere.
- **Bias e discriminazioni**: i dati di addestramento possono contenere pregiudizi (di genere, razza, ecc.) che il modello rischia di perpetuare nelle sue risposte. Questo è un problema che riguarda l'intero campo dell'AI, ma che negli LLM è particolarmente critico per via degli impatti sociali e culturali delle loro applicazioni che già ora stanno avendo
- **Costi di addestramento e impronta ecologica**: preparare modelli di dimensioni elevate richiede risorse computazionali notevoli, con un impatto rilevante in termini di consumi energetici e di infrastrutture necessarie.
- **Sicurezza e uso improprio**: modelli potenti possono generare contenuti estremamente realistici, il che solleva questioni etiche e potenziali rischi (disinformazione, spam, deepfake testuali, ecc.). [Aggiungere citazioni]

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] «The neural networks behind Google Voice transcription». Consultato: 11 agosto 2015. [Online]. Disponibile su: https://research.google/blog/the-neural-networks-behind-google-voice-transcription/

[15] H. Sak, A. Senior, K. Rao, F. Beaufays, e J. Schalkwyk, «Google voice search: faster and more accurate», *Google Speech Team*. Consultato: 24 settembre 2015. [Online]. Disponibile su: https://research.google/blog/google-voice-search-faster-and-more-accurate/

[16] «Neon prescription... or rather, New transcription for Google Voice». Consultato: 23 luglio 2015. [Online]. Disponibile su: https://blog.google/products/google-voice/neon-prescription-or-rather-new/

[17] Y. Wu *et al.*, «Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation». [Online]. Disponibile su: https://arxiv.org/abs/1609.08144

[18] Quoc V. Le e Mike Schuster, «A Neural Network for Machine Translation, at Production Scale», *Google Brain Team*. Consultato: 27 settembre 2016. [Online]. Disponibile su: https://research.google/blog/a-neural-network-for-machine-translation-at-production-scale/

[19] «Can Global Semantic Context Improve Neural Language Models?», *Frameworks Natural Language Processing Team*. [Online]. Disponibile su: https://machinelearning.apple.com/research/can-global-semantic-context-improve-neural-language-models

[20] «Bringing the Magic of Amazon AI and Alexa to Apps on AWS». [Online]. Disponibile su: https://www.allthingsdistributed.com/2016/11/amazon-ai-and-alexa-for-all-aws-apps.html

[21] Alexander Sidorov, «Transitioning entirely to neural machine translation». [Online]. Disponibile su: https://engineering.fb.com/2017/08/03/ml-applications/transitioning-entirely-to-neural-machine-translation/

[22] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, e A. Stolcke, «The Microsoft 2017 Conversational Speech Recognition System», in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, apr. 2018, pp. 5934–5938. doi: 10.1109/ICASSP.2018.8461870.

[23] «The Science Behind OpenAI Five that just Produced One of the Greatest Breakthrough in the History of AI». [Online]. Disponibile su: https://web.archive.org/web/20191226222000/https://towardsdatascience.com/the-science-behind-openai-five-that-just-produced-one-of-the-greatest-breakthrough-in-the-history-b045bcdc2b69?gi=24b20ef8ca3f

[24] «Learning Dexterity». [Online]. Disponibile su: https://openai.com/index/learning-dexterity/

[25] [Online]. Disponibile su: https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

[26] J. Schmidhuber, «Deep Learning: Our Miraculous Year 1990-1991». [Online]. Disponibile su: https://arxiv.org/abs/2005.05744

[27] A. Vaswani *et al.*, «Attention Is All You Need». [Online]. Disponibile su: https://arxiv.org/abs/1706.03762

[28] L. McMahon e Z. Kleinman, «Glue pizza and eat rocks: Google AI search errors go viral». [Online]. Disponibile su: https://www.bbc.com/news/articles/cd11gzejgz4o