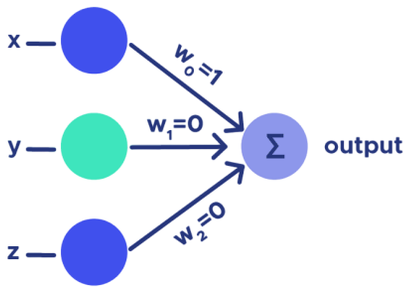
Concetti generali

Il percettrone

> Il **percettrone** è uno dei modelli più semplici e fondamentali delle reti neurali artificiali, ideato negli anni '50 da Frank Rosenblatt. È un modello matematico che simula il comportamento di un neurone biologico.

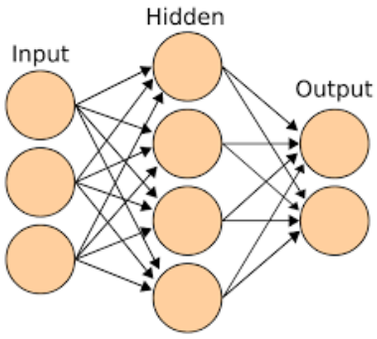


- **Ingressi**: Sono i dati in entrata, che potrebbero essere valori numerici.
- **Pesi**: Ogni ingresso ha un peso associato, che rappresenta l'importanza di quell'ingresso nel determinare l'output. I pesi vengono modificati durante l'addestramento del modello.
- **Funzione di Attivazione**: Dopo aver calcolato la somma pesata degli ingressi, il risultato viene passato attraverso una funzione di attivazione, che decide se il neurone genera un output o no. La funzione di attivazione originale del percettrone è una **funzione a soglia**, che produce 1 se la somma pesata è maggiore di un certo valore (soglia) e 0 altrimenti.
- **Bias**: Un termine aggiuntivo che serve a spostare la funzione di attivazione, permettendo al modello di adattarsi meglio ai dati.

Abbiamo quindi una combinazione lineare

Shallow Network

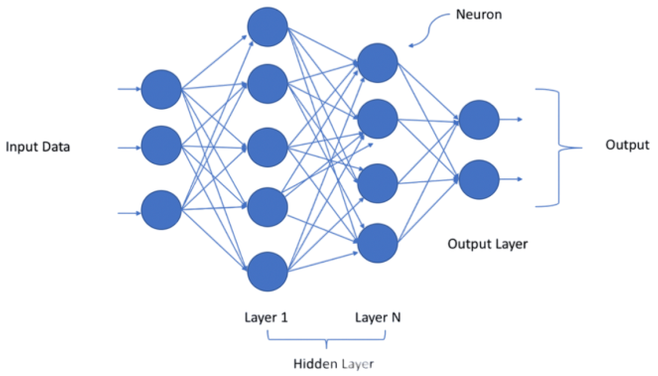
> Una **shallow network** è una rete neurale che contiene **uno o pochi strati nascosti** (in genere uno solo) tra il livello di input e quello di output. Il percettrone semplice è un esempio classico di rete superficiale con un solo strato nascosto.



- **Nell'hidden layer**, sono presenti un numero **n** di neuroni che decidono insieme quale può esse l'output riducendo cosi l'errore.

Deep Network

> Una **deep network** è una rete neurale con **molti strati nascosti** (solitamente più di 3). Questa profondità permette alla rete di apprendere rappresentazioni più complesse e astratte dei dati.



- **Struttura complessa**: Contiene molti strati nascosti tra l'input e l'output, spesso chiamati **deep layers**. Questi strati possono essere composti da centinaia o migliaia di neuroni.
- **Ogni scelta** di un layer influenza il successivo.

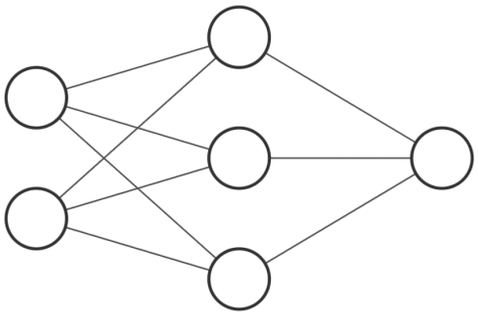
Machine Learning

> Il **machine learning** permette alle macchine di **apprendere dai dati** ed effettuare previsioni o decisioni senza essere esplicitamente programmate per eseguire ogni singolo compito. In altre parole, attraverso il machine learning, i computer imparano automaticamente da esperienze passate e migliorano le loro prestazioni nel tempo.

Esempio:

Un esempio classico di rete neurale con **2 input**, **3 neuroni** nello strato nascosto, e **1 output** potrebbe essere utilizzato per classificare dei punti su un piano cartesiano in due categorie

distinte (ad esempio, **punti blu** e **punti rossi**)



- **Input layer**
  - I **2 input** della rete corrispondono alle coordinate x_1 e x_2 dei punti sul piano cartesiano.
  - Ogni punto $P_{(x_1,x_2)}$ rappresenta un esempio di input alla rete.
- **Hidden layer**
  - Lo **strato nascosto** ha **3 neuroni**, ciascuno con i propri **pesi** che moltiplicano gli input. Ogni neurone somma gli ingressi pesati, aggiunge un termine di **bias**, e poi passa il risultato attraverso una **funzione di attivazione.**
- **Output layer**
  - Lo **strato di output** ha **1 neurone**, che produce un valore di output. Questo valore può essere interpretato come una **probabilità** che il punto appartenga alla classe "rossa" o "blu".
  - **L'output** può essere compreso tra 0 e 1, dove 0 indica un punto blu e 1 indica un punto rosso, usando una funzione di attivazione `sigmoide` o `softmax` per la classificazione binaria.

Encoder Decoder

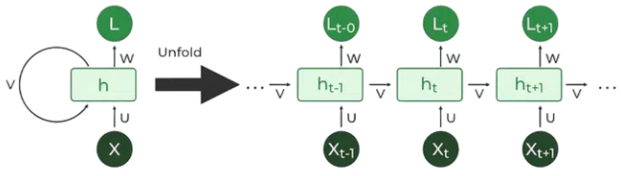
> L'architettura **Encoder-Decoder** è ampiamente utilizzata nei modelli di deep learning per compiti che coinvolgono sequenze, come la traduzione automatica, il riassunto di testi, o la generazione di testo.
>
> - **L'encoder** prende una sequenza di input e la trasforma in una rappresentazione compatta o vettoriale, spesso chiamata **embedding** o **contesto**. Questa rappresentazione è il riassunto di tutte le informazioni rilevanti presenti nella sequenza di input.
> - **Il decoder** prende l'output prodotto dall'encoder (il vettore di contesto) e genera una nuova sequenza di output. Di solito, questa nuova sequenza è in una forma diversa rispetto all'input.

Dal 1986 (RNN) al 2022 (ChatGPT)

RNN (Recurrent Neural Network)

> Una **RNN (Recurrent Neural Network)** è un tipo di rete neurale progettata per lavorare con dati sequenziali. A differenza delle reti neurali tradizionali (feedforward), che processano input indipendenti l'uno dall'altro, le RNN mantengono una "memoria" degli input precedenti utilizzando uno stato nascosto, il che le rende particolarmente efficaci per l'elaborazione di sequenze e dipendenze temporali come sequenze di parole.

- **Stato ricorrente**: Ogni cella h utilizza l'informazione proveniente dal passo precedente, consentendo di "ricordare" sequenze di dati e di mantenere una sorta di memoria temporale.
- **Unfolding nel tempo**: Le RNN possono essere "espanse" lungo la dimensione temporale, il che permette di elaborare sequenze di lunghezza arbitraria.
- **Problema del gradiente**: Le RNN tradizionali soffrono di problemi noti come il "vanishing gradient", che rende difficile l'apprendimento di dipendenze a lungo termine.



- **Input** X_t: Ogni input della sequenza viene fornito in un momento t. Questo può essere, ad esempio, una parola di una frase o un fotogramma di un video.
- **Stato Nascosto** h_t: Lo stato nascosto h_t contiene informazioni sull'input corrente X_t e lo stato precedente h_{t-1}. È come una memoria che si evolve man mano che la sequenza procede.
- **Output** L_t: La RNN produce un output L_t che può essere utilizzato per fare previsioni o classificazioni in tempo reale, o conservato per elaborazioni successive.
- **Backpropagation Through Time (BPTT)**: Durante l'addestramento, il gradiente di errore viene propagato all'indietro attraverso il tempo, aggiornando i pesi della rete affinché essa possa migliorare le previsioni successive

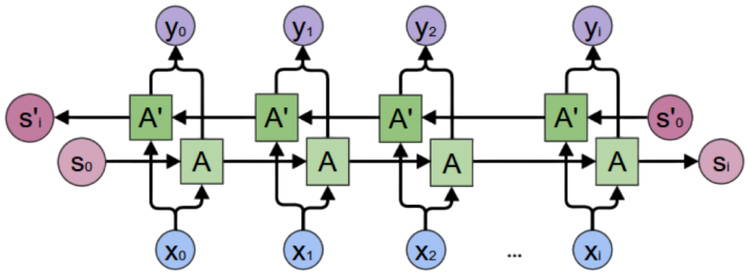
RNN Bidirezionali

> Le **RNN bidirezionali** sono una variante delle RNN tradizionali che processano le sequenze di dati in entrambe le direzioni: **dal passato al futuro** e **dal futuro al passato**. Questo tipo di rete è utile per migliorare la capacità di apprendimento delle dipendenze a lungo termine, soprattutto nei casi in cui le informazioni future sono altrettanto rilevanti quanto quelle passate.

In una **RNN bidirezionale**, ci sono due strati di RNN:

1. **RNN forward**: Elabora la sequenza in avanti (dal primo elemento all'ultimo).
2. **RNN backward**: Elabora la sequenza all'indietro (dall'ultimo elemento al primo).

Questi due strati sono combinati per fornire una rappresentazione completa che tiene conto sia delle informazioni passate che di quelle future.



- A rappresenta il livello di RNN che processa la sequenza in avanti (forward).
- A' rappresenta il livello di RNN che processa la sequenza all'indietro (backward).
- $x_0, x_1, x_2, \ldots, x_i$ sono gli input della sequenza che vengono forniti alla rete.
- $y_0, y_1, y_2, \ldots, y_i$ sono gli output generati dalla combinazione delle informazioni forward e backward.
- S_0 e S_0' rappresentano gli stati nascosti iniziali delle due direzioni, forward e backward.

Esempio di Applicazione

Consideriamo il caso di tradurre una frase da una lingua all'altra (ad esempio, dall'inglese all'italiano):

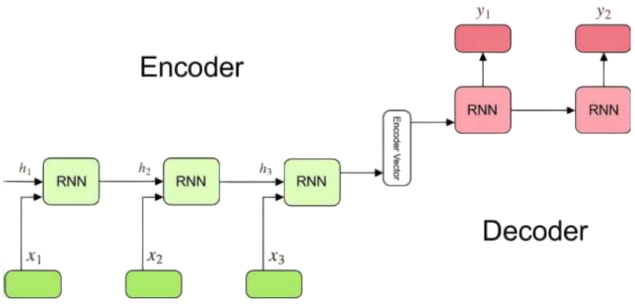
- **Input**: Una frase come "I am going to the store".
- **Forward RNN** (A): Processa la frase dall'inizio alla fine. In ogni passo, memorizza lo stato nascosto in base alle parole precedenti.
- **Backward RNN** (A'): Processa la frase dalla fine all'inizio, generando uno stato nascosto che tiene conto delle parole future.

In un punto come "going to the", l'informazione forward potrebbe indicare che la traduzione dovrebbe essere "sto andando a", mentre l'informazione backward potrebbe indicare che dopo "store" si aspetta "il negozio". Combinando queste informazioni, la RNN bidirezionale può produrre una traduzione più accurata.

Encoder Decoder RNN

> L'architettura **Encoder-Decoder** nelle **RNN** (Recurrent Neural Networks) è un modello ampiamente utilizzato per compiti di **traduzione sequenza-sequenza** (sequence-to-sequence), come la traduzione automatica, il riconoscimento vocale e il riassunto di testi. È progettata per gestire compiti in cui l'input e l'output sono sequenze di lunghezza variabile.

- **Encoder**:
  - L'encoder prende in input una sequenza e la comprime in una rappresentazione densa, spesso chiamata **vettore di contesto** o **embedding**.
  - In una RNN, l'encoder elabora l'input sequenziale, producendo uno **stato nascosto finale** che rappresenta l'intera sequenza di input.
  - Questa rappresentazione riassuntiva viene poi passata al decoder.
- **Decoder**:
  - Il decoder utilizza lo stato nascosto finale dell'encoder (il **vettore di contesto**) per generare una nuova sequenza di output.
  - Come l'encoder, anche il decoder è una RNN, che elabora la sequenza di output passo dopo passo.
  - A differenza dell'encoder, il decoder genera output una parola o **token** alla volta, utilizzando il proprio stato nascosto e il contesto fornito dall'encoder.



- **Parte sinistra: Encoder**:
  - X_1, X_2, X_3 rappresentano gli input di una sequenza (ad esempio, parole in una frase).
  - Ogni input viene elaborato da un blocco **RNN**. I blocchi sono collegati l'uno all'altro in sequenza, e ciascun blocco aggiorna uno **stato nascosto** h_1, h_2, h_3, che rappresenta il riassunto delle informazioni processate fino a quel punto.
  - Lo stato nascosto finale h_3 è il **vettore di contesto** o **encoder vector**, che condensa tutte le informazioni della sequenza di input.
- **Parte destra: Decoder**:
  - Il vettore di contesto (lo stato nascosto finale dell'encoder) viene passato al decoder, che inizia a generare la sequenza di output.
  - Il decoder, come l'encoder, è una serie di blocchi **RNN** che producono gli output y_1, y_2, e così via.
  - Ogni blocco RNN del decoder genera un output basato sul proprio stato nascosto e sull'output generato nel passo precedente.

Transformers

> I **Transformers** sono un'architettura di rete neurale introdotta per la prima volta nel 2017 con il lavoro "Attention is All You Need". Questa architettura ha rivoluzionato molti campi del deep learning, in particolare il **Natural Language Processing (NLP)**, per la sua capacità di gestire sequenze di dati in parallelo e di cogliere le dipendenze a lungo raggio, il che ha portato alla sostituzione delle RNN in molti contesti.

I principi chiave dei transformers sono:

Attention Mechanism

- La caratteristica principale del Transformer è il meccanismo di **attenzione**, in particolare il **multi-head self-attention**.
- A differenza delle RNN, che elaborano sequenze in ordine, l'attenzione consente al modello di concentrarsi su tutte le parole della sequenza contemporaneamente, attribuendo un peso diverso a ciascuna parola in base alla sua rilevanza per quella posizione.
- L'attenzione calcola la dipendenza di ogni token rispetto agli altri token nella sequenza.

Stacking

- I Transformers utilizzano **più livelli impilati** (stacking) di encoder e decoder. Questo permette di arricchire progressivamente le rappresentazioni degli input man mano che attraversano i vari strati, consentendo al modello di apprendere gerarchie di informazioni.

Multi-Head Attention

- Per catturare diversi tipi di relazioni tra le parole, l'architettura di Transformer utilizza **più teste di attenzione**. Ogni "testa" può concentrarsi su diverse parti della sequenza, permettendo al modello di cogliere diversi aspetti delle dipendenze tra le parole.

BERT

> **BERT** (Bidirectional Encoder Representations from Transformers) è un modello di linguaggio sviluppato da Google nel 2018 che ha rivoluzionato il **Natural Language Processing (NLP)**. A differenza dei modelli tradizionali unidirezionali, BERT è **bidirezionale**, il che significa che considera sia il contesto alla sinistra che alla destra di una parola in una frase. Questa capacità gli permette di comprendere meglio il significato delle parole in base al contesto completo.

BERT viene pre-addestrato su enormi dataset non supervisionati (come Wikipedia e BooksCorpus) utilizzando due compiti principali: **Masked Language Model (MLM)** e **Next Sentence Prediction (NSP)**.

Semi-Supervised Learning

- Questa parte descrive il **pre-training** del modello BERT.
- Il modello viene addestrato su grandi quantità di dati testuali non etichettati, come Wikipedia e libri (libri sono rappresentati graficamente).
- Il compito di **pre-training** è il **Masked Language Modeling**: alcune parole del testo vengono nascoste, e BERT deve predire la parola mancante utilizzando il contesto bidirezionale.
- In questo stadio, BERT impara le strutture linguistiche di base e le relazioni tra le parole.

Supervised Learning

- Una volta completato il pre-training, BERT viene **fine-tuned** su un compito specifico supervisionato.
- L'esempio mostra un compito di classificazione: identificare se un'email è spam o meno.
- Il dataset utilizzato per questo fine-tuning è etichettato, quindi le classi (spam e non spam) sono definite e fornite al modello.
- Il modello pre-addestrato viene quindi adattato a questo compito specifico con l'obiettivo di massimizzare le prestazioni su quel dataset.

Architettura di BERT

- Ogni token di input (parola o pezzo di parola) viene convertito in un embedding.
- Un token speciale `[CLS]` viene aggiunto all'inizio della sequenza per la classificazione e un token `[SEP]` viene utilizzato per separare le due frasi in input.
- Gli embedding di input vengono poi elaborati attraverso vari strati del modello BERT, che applica il meccanismo di **self-attention** per apprendere le relazioni tra i token.
- L'output finale associato al token `[CLS]` viene usato per predire l'etichetta di classe (ad esempio, "spam" o "non spam" nel caso di classificazione).

BART

> **BART** (Bidirectional and Auto-Regressive Transformers) è un modello sviluppato da Facebook AI che combina le migliori caratteristiche degli encoder bidirezionali (come BERT) e dei decoder autoregressivi (come GPT). BART è stato progettato per eccellere in compiti di generazione di testo e correzione, sfruttando una potente architettura **encoder-decoder**.

- **Encoder Bidirezionale**:
  - L'encoder di BART è simile a quello di BERT, cioè **bidirezionale**, il che significa che esamina l'intero contesto di una sequenza di input (sia le parole precedenti che quelle successive) per comprendere meglio ogni parola.
  - Questo consente di catturare informazioni di contesto sia a sinistra che a destra della parola in esame.
- **Decoder Autoregressivo**:
  - Il decoder di BART funziona in modo simile al **modello GPT**, cioè **autoregressivo**. Ciò significa che genera l'output un token alla volta, utilizzando i token precedenti per predire il successivo.
  - Questo permette a BART di generare testo in maniera fluida e sequenziale, come nei modelli di completamento di testo.

Cosa è un prompt?

Prompt: È il testo iniziale che viene dato in input a un modello di linguaggio per indirizzare il modello a produrre un'uscita (output) rilevante. Può essere una frase, una domanda, una richiesta o un contesto specifico. Il prompt può essere più o meno dettagliato, a seconda del livello di controllo che si desidera esercitare sull'output.

GPT-3

> **GPT-3 (Generative Pre-trained Transformer 3)**, rilasciato da OpenAI nel 2020, è una delle più avanzate e potenti reti neurali mai sviluppate per la **generazione del linguaggio naturale**. Si basa sull'architettura **Transformer**, come i suoi predecessori (GPT e GPT-2), ma è caratterizzato da una scala e una capacità enormemente superiori.

- **Architettura basata sui Transformer**:
  - GPT-3 utilizza la stessa architettura di base dei modelli Transformer: il **self-attention** meccanismo per catturare relazioni contestuali tra le parole.
  - È un **modello solo decoder**, autoregressivo, che predice il prossimo token basandosi su quelli precedenti, rendendolo estremamente efficace nella generazione del testo.
- **Generazione del testo autoregressiva**:
  - GPT-3 genera testo **token per token**, prevedendo ogni parola basandosi sulle parole precedenti. Questa capacità autoregressiva gli consente di produrre testo coerente e ben strutturato in una varietà di contesti.
  - La generazione è fluida, il che lo rende adatto a compiti che richiedono creatività, come scrittura di racconti, articoli, risposte conversazionali, o persino codici di programmazione.
- **Pre-training su dati enormi**:
  - GPT-3 è stato addestrato su un vasto corpus di dati testuali raccolti da fonti diverse, come libri, articoli, siti web e altro. Il pre-training ha permesso al modello di imparare una vasta gamma di conoscenze e strutture linguistiche.

In un **modello solo decoder**, ogni passaggio di generazione considera solo i token generati fino a quel momento. Questo consente a GPT-3 di continuare il testo in modo sequenziale, predicendo ogni nuovo token in modo fluido, come nel caso del completamento di frasi, la scrittura creativa, o la risposta a domande in modo conversazionale.

ChatGPT (2022)

> **ChatGPT** è una versione specializzata di **GPT-3**, progettata e ottimizzata da OpenAI per gestire conversazioni in linguaggio naturale. ChatGPT è in grado di comprendere il contesto delle conversazioni, rispondere in modo coerente e adattarsi al tono e allo stile dell'interlocutore. Pur essendo basato sull'architettura GPT-3, ChatGPT è ulteriormente raffinato per eccellere in **scenari conversazionali**, rendendolo particolarmente adatto a interazioni con utenti, assistenti virtuali, customer service, e altri contesti di dialogo.

- **Supervisione Umana e RLHF (Reinforcement Learning with Human Feedback)**:
  - Uno degli approcci principali utilizzati per addestrare **ChatGPT** è il **Reinforcement Learning with Human Feedback (RLHF)**. In questo processo, gli annotatori umani interagiscono con il

modello, valutano le risposte e forniscono feedback su cosa dovrebbe essere migliorato. Il modello viene quindi aggiornato per allinearsi meglio alle preferenze umane.

    - GPT-3 è stato addestrato principalmente attraverso apprendimento non supervisionato su grandi quantità di testo e non è stato soggetto allo stesso tipo di supervisione dettagliata per quanto riguarda i dialoghi umani come ChatGPT.

- **Sicurezza e Filtro sui Contenuti**:
    - ChatGPT ha un **maggiore controllo sui contenuti** rispetto a GPT-3. È stato progettato per evitare di generare contenuti inappropriati, dannosi o fuorvianti. Grazie al processo di RLHF, ChatGPT ha appreso a evitare risposte offensive, violente o potenzialmente problematiche, il che lo rende più sicuro per l'interazione con gli utenti.
    - GPT-3, essendo un modello di linguaggio generale, può occasionalmente produrre risposte inappropriate o fuori contesto, poiché non ha un livello di filtro equivalente a quello applicato a ChatGPT.
- **Uso di Prompt**:
    - **GPT-3** si basa pesantemente sull'uso dei **prompt**, ovvero il contesto fornito dall'utente prima della generazione di testo. L'utente deve specificare chiaramente il contesto in cui GPT-3 deve operare.
    - **ChatGPT** è meno dipendente dai prompt espliciti, poiché è stato addestrato a seguire una struttura di conversazione. Gli utenti possono interagire con ChatGPT in modo più naturale, senza dover fornire dettagli espliciti ad ogni passo.

Pre-addestramento e Fine-tuning

- I **LLM** sfruttano la tecnica del **pre-addestramento** per imparare da grandi quantità di dati non supervisionati (come testi generici). Successivamente, viene applicato il **fine-tuning** per specializzare il modello su un compito specifico.
- Per esempio, BERT viene pre-addestrato su grandi corpus di testo e poi fine-tuned su specifici task come la classificazione, la risposta alle domande o il riconoscimento di entità. Il pre-addestramento sfrutta l'apprendimento per trasferimento, consentendo al modello di generalizzare meglio ai nuovi compiti.

Zero-shot e Few-shot Learning tramite Prompting

- **Zero-shot learning** si riferisce alla capacità del modello di gestire nuovi compiti senza esempi specifici, basandosi solo su un **prompt** che descrive il compito o la domanda.
- **Few-shot learning** invece si verifica quando al modello vengono forniti solo pochi esempi (da 1 a 5) per apprendere un nuovo compito. Questo permette al modello di adattarsi a nuovi task con pochissime informazioni aggiuntive, sfruttando le sue conoscenze precedentemente apprese.
- Il concetto di **prompting** è cruciale: fornire al modello un contesto testuale ben formulato permette di ottenere risultati specifici senza la necessità di ulteriori addestramenti.