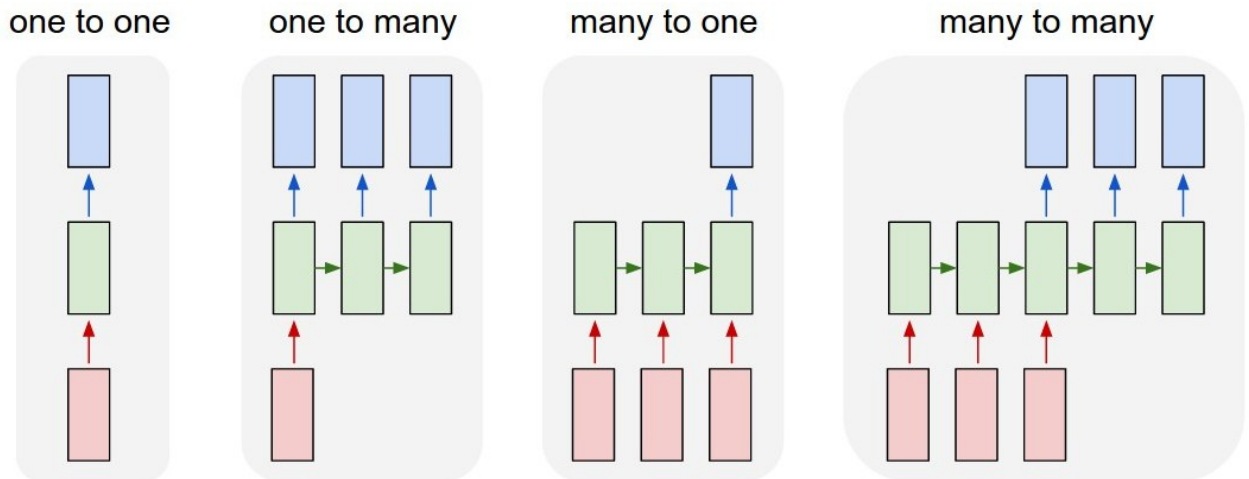


Deep Learning

- Introduzione
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
 - Sequence to Sequence
 - Unfolding in Time
 - Basic Cells, LSTM, GRU
 - Time Series, Dati statici e dinamici
- Natural Language Processing e LLM
 - Applicazioni
 - Tokenization – Embedding
 - Neural Machine Translation
 - Transformers
 - Large Language Models
 - Prompt Engineering e Few-shot Learning
 - Embedding, Indexing, Tuning
- Reinforcement Learning
 - Q-Learning
 - Deep Reinforcement Learning

Sequence to Sequence

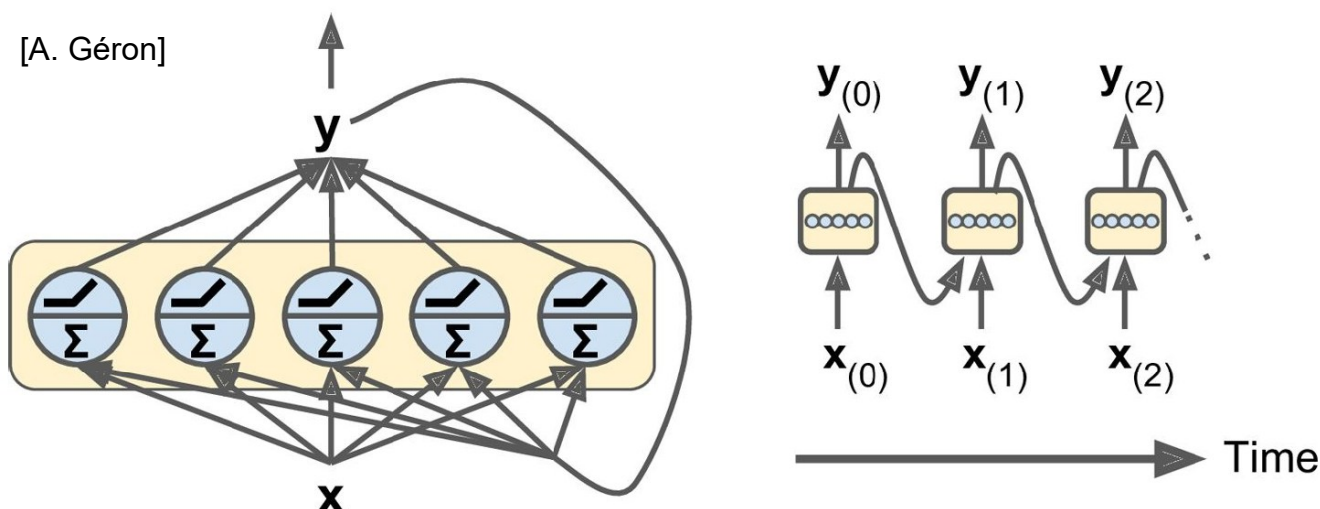


Le reti **feedforward** (es. MLP, CNN) studiate fino a questo momento operano su **vettori di dimensione prefissata**. Ad esempio l'input è un vettore immagine e l'output un vettore di probabilità delle classi. Questo caso è rappresentato dalla prima colonna della figura come sequenza **One to one**.

Applicazioni diverse richiedono che l'input e/o l'output possano essere sequenze (anche di **lunghezza variabile**).

- **One to many.** Es. **image captioning** dove l'input è un'immagine e l'output una frase (sequenza di parole) in linguaggio naturale che la descrive.
- **Many to one.** Es. **sentiment analysis** dove l'input è un testo (es. recensione di un prodotto) e l'output un valore continuo che esprime il sentiment o giudizio (positivo o negativo).
- **Many to many.** Es. **language translation** dove l'input è una frase in Inglese e l'output la sua traduzione in Italiano.

Reti Ricorrenti (RNN)



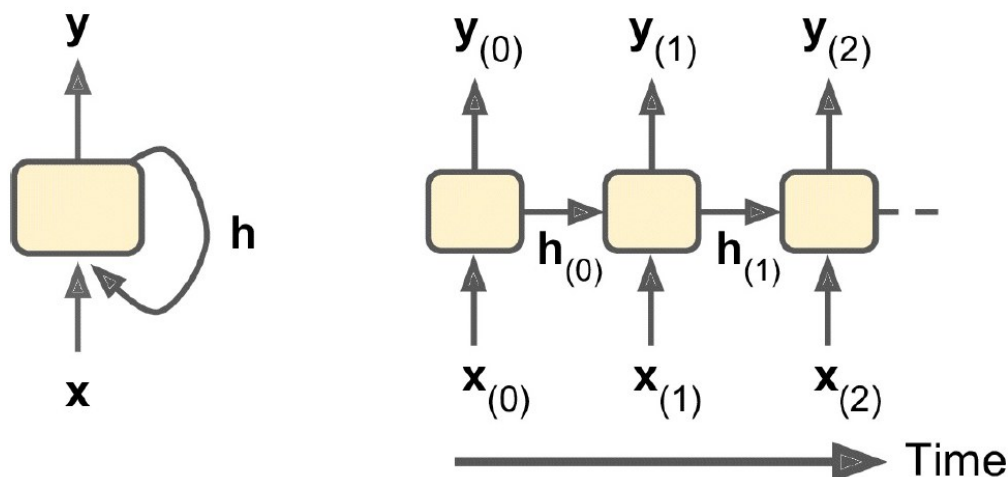
Le **reti ricorrenti** prevedono «anche» collegamenti all'indietro o verso lo stesso livello. I modelli più comuni e diffusi (es. LSTM, GRA) prevedono collegamenti **verso lo stesso livello**.

A ogni step della sequenza (ad esempio a ogni istante temporale t) il livello riceve oltre all'**input** $x_{(t)}$ anche il suo **output dello step precedente** $y_{(t-1)}$. Questo consente alla rete di basare le sue decisioni sulla **storia passata** (effetto memoria) ovvero su tutti gli elementi di una sequenza e sulla loro **posizione reciproca**.

- In una frase non è rilevante solo la presenza di specifiche parole ma è importante anche come le parole sono tra loro legate (posizione reciproca).
- La classificazione di video (sequenze di immagini) non necessariamente si basa sulle interrelazioni dei singoli frame. Esempio:
 - *per capire se un video è un documentario su animali è sufficiente la detection di animali in qualche frame (non occorre RNN).*
 - *per comprendere il linguaggio dei segni, è necessario analizzare le interrelazioni dei movimenti delle mani nei frame (utile RNN).*

Unfolding in Time

[A. Géron]



Una **cella** è una parte di rete ricorrente che preserva uno **stato** (o memoria) interno $h_{(t)}$ per ogni istante temporale. È costituita da un numero prefissato di neuroni (può essere vista come un layer).

- $h_{(t)}$ dipende dall'input $x_{(t)}$ e dallo stato precedente $h_{(t-1)}$

$$h_{(t)} = f(h_{(t-1)}, x_{(t)})$$

Per poter addestrare (con **backpropagation**) una RNN è necessario eseguire il cosiddetto **unfolding** o **unrolling in time** (parte destra della figura), stabilendo a priori il numero di passi temporali su cui effettuare l'analisi.

- Di fatto una RNN unfolded su 20 step equivale a una DNN feedforward con 20 livelli. Pertanto addestrare RNN che appaiono relativamente semplici può essere molto costoso e critico per la convergenza (problema del **vanishing gradient**).
- Da notare che in una RNN unfolded: l'input e l'output sono in generale collegati a tutte le istanze della cella e i pesi della cella (nascosti in f) **sono comuni a tutte le istanze della cella**.

Basic Cell

In una **cella base** di RNN:

Lo stato $\mathbf{h}_{(t)}$ dipende dall'input $\mathbf{x}_{(t)}$ e dallo stato precedente $\mathbf{h}_{(t-1)}$

$$\mathbf{h}_{(t)} = \phi(\mathbf{x}_{(t)}^T \cdot \mathbf{W}_x + \mathbf{h}_{(t-1)}^T \cdot \mathbf{W}_h + b)$$

dove:

- \mathbf{W}_x e \mathbf{W}_y sono i vettori dei pesi e b il bias da apprendere.
- ϕ è la funzione di attivazione (es. Relu).

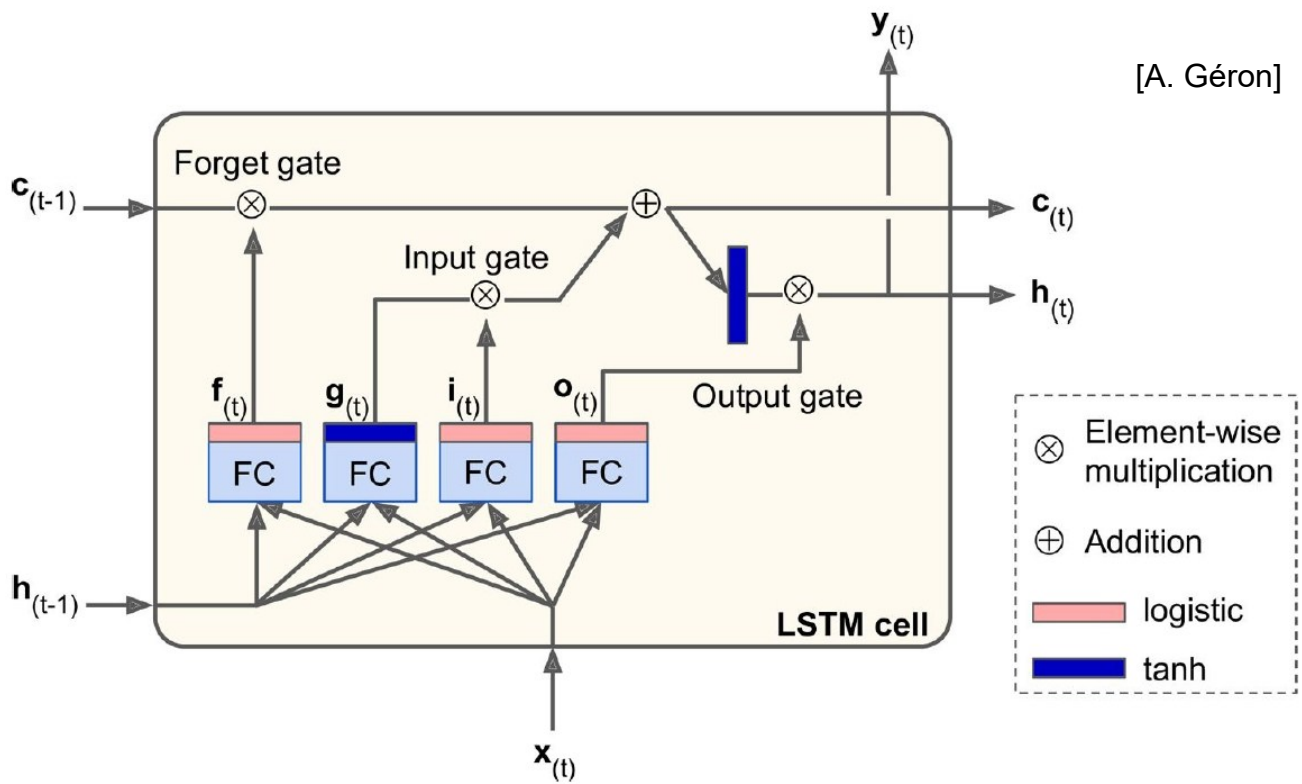
e l'output $\mathbf{y}_{(t)}$ corrisponde allo stato $\mathbf{h}_{(t)}$:

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)}$$

Le celle base hanno difficoltà a ricordare/sfruttare input di step lontani: la memoria dei primi input **tende a svanire**. D'altro canto sappiamo che in una frase anche le prime parole possono avere un'importanza molto rilevante.

Per risolvere questo problema e facilitare la convergenza in applicazioni complesse, sono state proposte celle più evolute dotate di un **effetto memoria a lungo termine**: **LSTM** e **GRU** sono le più note tipologie.

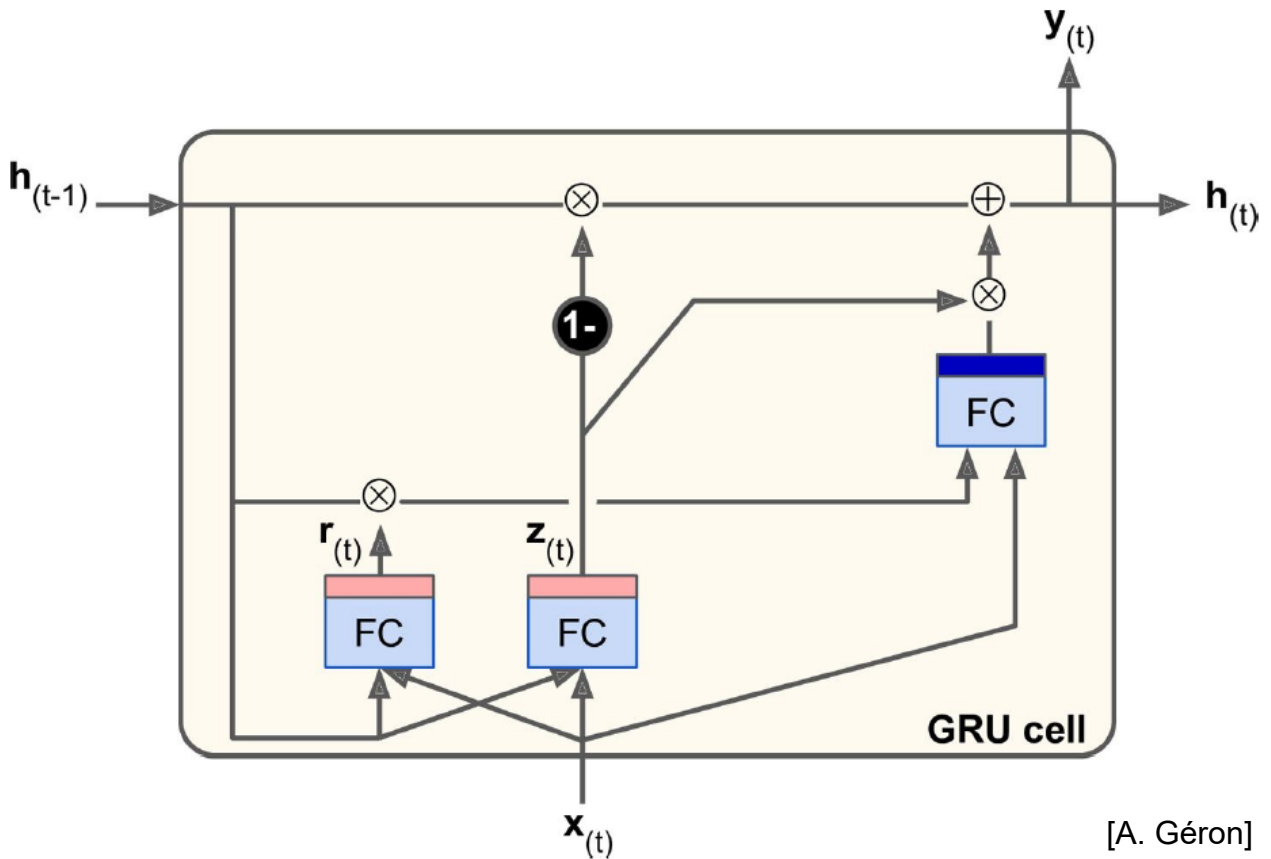
LSTM



In **LSTM** (Long Short-Term Memory) lo stato $h_{(t)}$ è suddiviso in due vettori $h_{(t)}$ e $c_{(t)}$:

- $h_{(t)}$ è uno stato (o memoria) a **breve** termine; anche in questo caso uguale all'output della cella $y_{(t)}$.
- $c_{(t)}$ è uno stato (o memoria) a **lungo** termine.
- la cella apprende durante il training cosa è importante dimenticare (**forget gate**) dello stato passato $c_{(t-1)}$ e cosa estrarre e aggiungere (**input gate**) dall'input corrente $x_{(t)}$.
- per il calcolo dell'output $y_{(t)} = h_{(t)}$ si combina l'input corrente (**output gate**) con informazioni estratte dalla memoria a lungo termine.

GRU



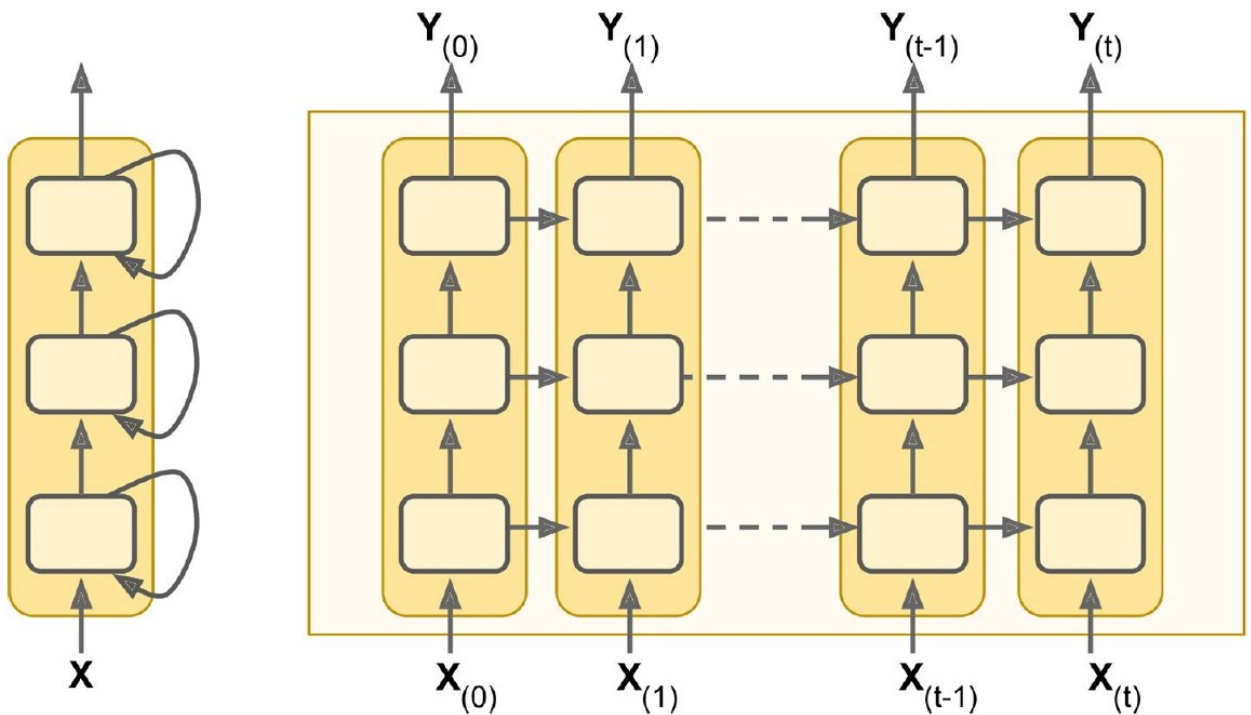
GRU (**G**ated **R**ecurrent **U**nit) è una versione semplificata di LSTM, che cerca di mantenerne i vantaggi, riducendo parametri e complessità. Le principali semplificazioni sono:

- uno solo stato di memoria $h_{(t)}$.
- uno solo gate (con logica invertita) per quantificare quanto dimenticare e quanto aggiungere: se necessario aggiungere prima devo dimenticare.

Per maggiori approfondimento su LSMT e GRU:

- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Deep RNN



[A. Géron]

In figura una RNN costituita da **tre layer** (stacked): a destra la versione unfolded (su $t+1$ step).

- gli output di un livello sono gli input del livello successivo
- con più livelli possono essere apprese relazioni più complesse dei dati.
- l'addestramento (supervisionato) consiste sempre nel fornire gli input e gli output (desiderati) per il solo ultimo livello.
- se l'input sono immagini, i vettori $x_{(t)}$ in genere non corrispondono ai pixel ma a feature di alto livello estratte da una CNN.
- se l'input sono parole, i vettori $x_{(t)}$ sono il risultato di word embedding (vedi nel seguito).

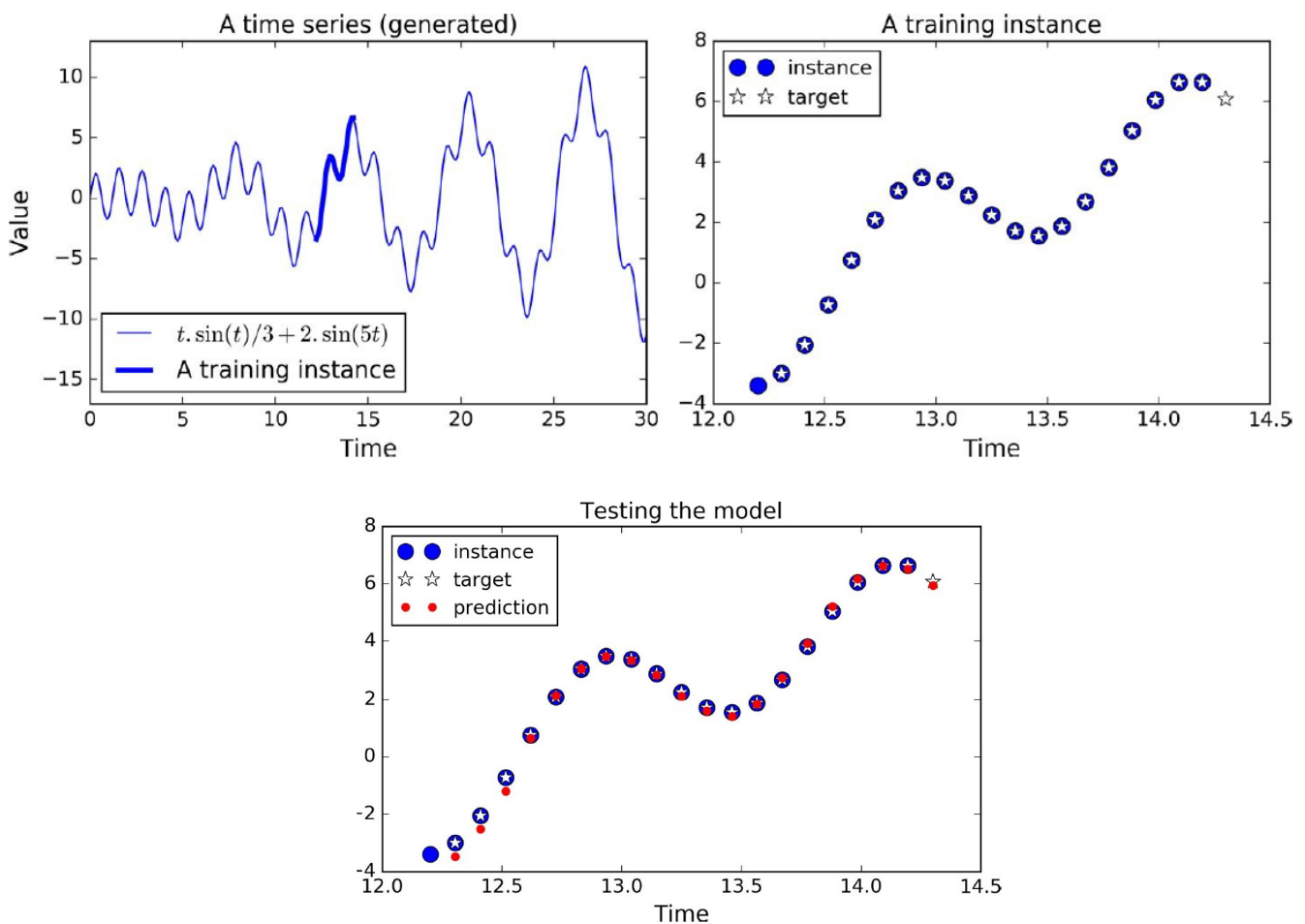
Esempio: Time Series

Una serie temporale (**time serie**) può essere relativa all'andamento nel tempo: di un **titolo di borsa**, della **temperatura di un ambiente**, del **consumo energetico** di un impianto, ecc.

Possiamo considerare una serie temporale come una funzione campionata in più istanti temporali. Conoscendo i valori fino all'istante t siamo interessati a predire il valore a $t+1$. Nota bene: non solo input 1D.

In [A. Géron] **esempio in Tensorflow/Keras di RNN** per la predizione di una funzione matematica (combinazione di sinusoidi).

■ <https://github.com/ageron/handson-ml2>



Note su Time Series

Prima di proporre modelli complessi (con il serio rischio di overfitting) provare semplici baseline, valutandone le prestazioni e la generalizzazione:

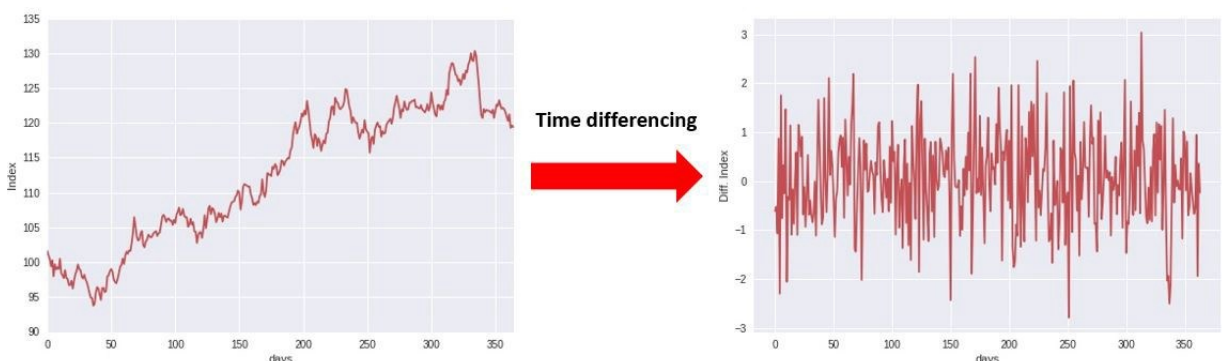
- **Persistence model**: la predizione al tempo $t+1$ corrisponde al valore al tempo t .
- **Linear model**: un semplice regressore lineare.

Modelli statistici classici: le serie temporali sono state studiate per decenni in ambito statistico ed esistono molti metodi (relativamente semplici) e talvolta preferibili a reti ricorrenti:

- <https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/>

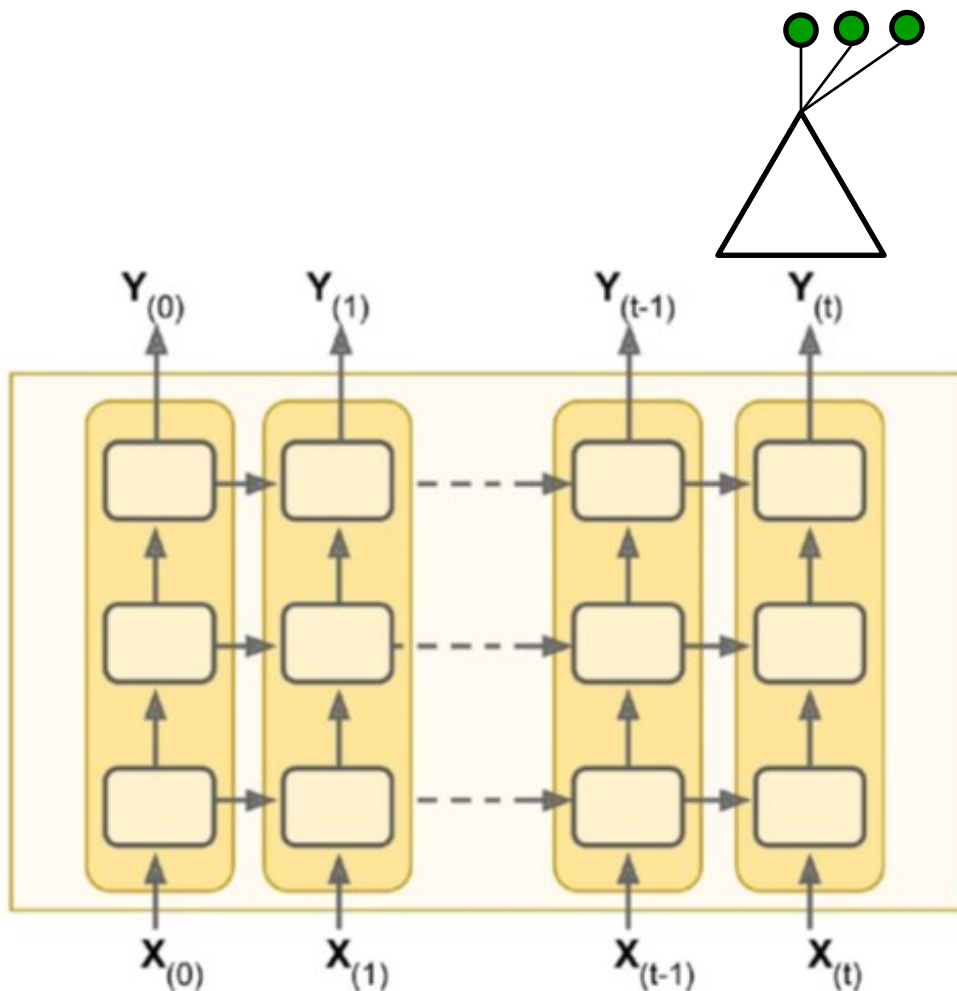
Decorrelazione dei dati della sequenza: talvolta, se i valori di una sequenza sono correlati nel tempo, si ha l'impressione di riuscire a predire con grande successo.

- Esempio: **trend di un titolo** di borsa. Anche il persistence model sembra performare bene rispetto alle metriche classiche (es. MSE). Proviamo a decorrelare i dati sostituendo al valore in t la differenza tra il valore in t e quello a $t-1$.



<https://medium.com/data-science/how-not-to-use-machine-learning-for-time-series-forecasting-avoiding-the-pitfalls-19f9d7adf424>

Classificazione/Regressione su serie temporali con Deep RNN



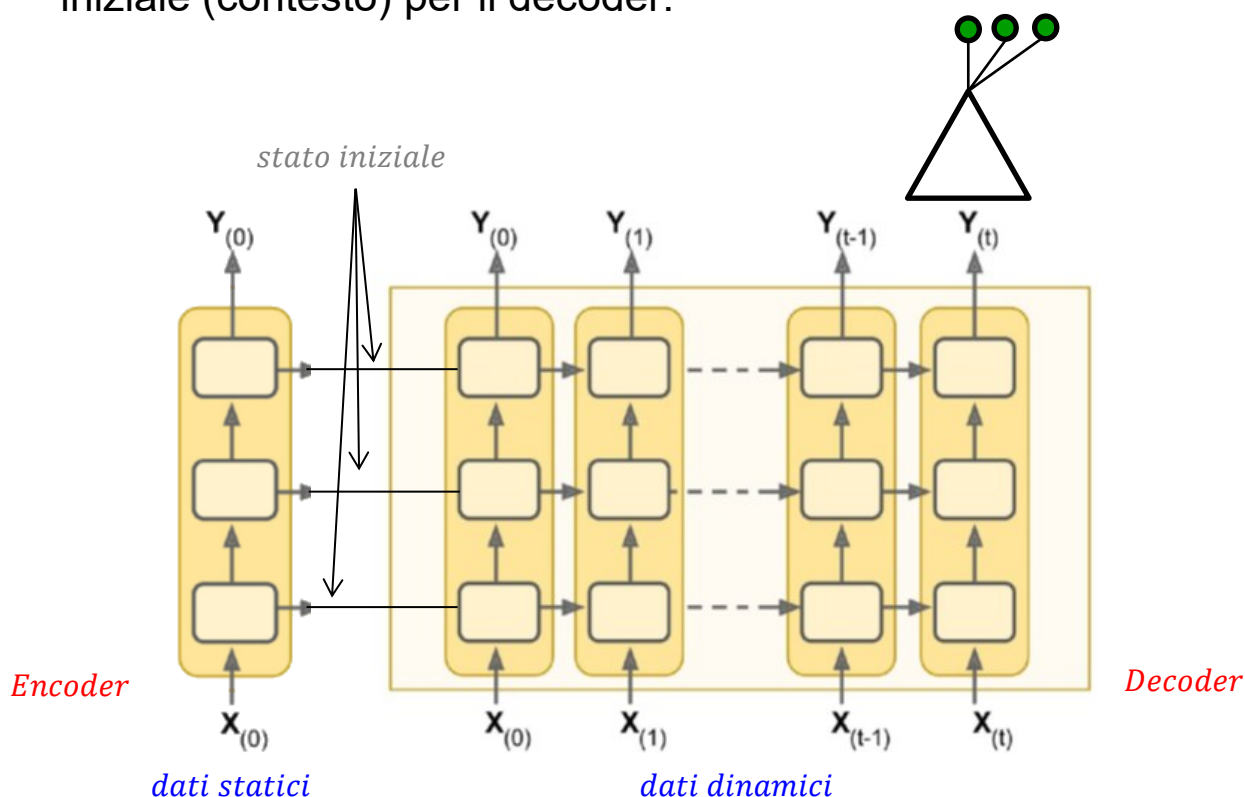
Un «semplice» **MLP** (con uno o pochi layer) è collegato al vettore $Y_{(t)}$.

- Nel caso di classificazione tanti neuroni di output quante sono le classi
- Nel caso di regressione, tanti neuroni quante sono le prediction ahead (es. uso tre neuroni per predire a $t+1$, $t+2$ e $t+3$).

Combinazione dati Statici e Dinamici

Talvolta solo un sottoinsieme dei dati disponibili varia nel time-frame considerato (dati **dinamici**). Altri dati (**statici**) definiscono il contesto. **Esempio**: Informazioni di base di un paziente (dati statici) e evoluzione suoi livelli di glucosio (dati dinamici). Come modellare il sistema:

1. Si replicano i dati statici in tutte le osservazioni (da $\mathbf{X}_{(0)}$ a $\mathbf{X}_{(t)}$) → poco efficiente.
2. Si forniscono i dati statici come input al MLP finale → il modello ricorrente non è influenzato da essi (non ottimale).
3. Si utilizzano i dati statici per definire lo stato hidden \mathbf{h} di ingresso (non più inizializzato a 0) → di fatto si crea un modello **encoder-decoder** dove l'encoder opera su un solo istante temporale (vettore dei dati statici) e produce lo stato iniziale (contesto) per il decoder.



Natural Language Processing (NLP)

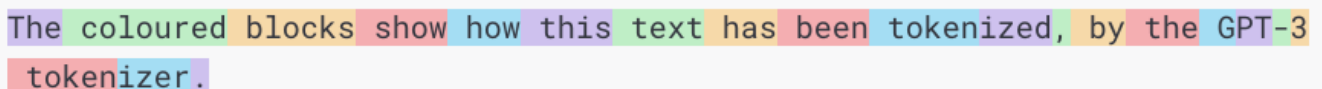
L'elaborazione del linguaggio naturale è uno dei settori cui il deep learning ha dato i maggiori contributi. Tra le principali applicazioni:

- **Text Classification**: es. Sentiment Analysis
- **Entity Extraction** (a.k.a. **Named Entity Recognition**): estrarre dal testo non strutturato informazioni rilevanti (es. parametri vitali del paziente da una cartella clinica).
- **Summarization**: es. generare riassunti concisi di documenti
- **Generation**: generare testi in diversi ambiti specificando lo stile (applicazioni in ambito giornalistico e/o marketing).
- **Machine Translation**: es. traduzione tra lingue, ma anche tra linguaggi di programmazione.
- **Coding**: completamento intelligente del codice, creazione funzioni da specifiche in linguaggio naturale, commenti automatici.
- **Question Answering**: es. rispondere in linguaggio naturale a domande poste in linguaggio naturale.
- **Digital Assistant e Chatbots**: dagli assistenti digitali (es. Amazon Alexa, Google assistant, Apple Siri) a conversational AI come ChatGPT.
- **Automatic Speech Recognition** (and **Synthesis**): es. traduzione parlato in testo e sintesi vocale.
- **Multimodal Extensions**: oltre al testo si utilizzano input/output di diversa natura (es. immagini, suoni). Ad esempio data un'immagine generare una descrizione in linguaggio naturale.

Tokenization

È il processo di **suddivisione di un testo** in unità più piccole (dette **token**) che costituiscono la sequenza di input fornita ai modelli di NLP e quella prodotta in output dagli stessi.

- I token possono essere **interi parole**, **parti di parole** (ad esempio nelle parole composte), **singoli caratteri** o **simboli** (es. emoticon).
- Utilizzare **token invece di parole** consente di lavorare simultaneamente con più lingue anche basate su alfabeti diversi, con i simboli usati nei linguaggi di programmazione, ecc. Consente inoltre di creare nuove parole unendo più token.
- Il numero totale di token utilizzabili da un modello definisce il suo **vocabolario**. GPT-2 e GPT-3 hanno un vocabolario di circa **50K** token.
- La tecnica BPE (**Byte Pair Encoding**) utilizzata per la creazione del vocabolario consente a GPT-3 di supportare fino a 46 lingue senza aumentare il numero di token (cosa che sarebbe necessaria utilizzando le parole).
- In GPT-3, dato un testo in inglese la lunghezza media di un token è di 4 caratteri.



The coloured blocks show how this text has been tokenized, by the GPT-3 tokenizer.

Esempio creazione dizionario con BPE (fonte: GPT-4)

Suppose we have a small corpus containing the following words and their respective frequencies:

low: 5
lower: 2
newest: 6
widest: 3

We start by initializing the vocabulary with **individual characters** and their frequencies.

Step 1: Find the most frequent pair of characters and merge them.

In this case, the most frequent pair is 'e' and 's' (occurring in 'newest' and 'widest'), so we merge them into a single token 'es'.

New vocabulary: l, o, w, e, r, n, s, t, d, i, es

Step 2: Repeat the process for the next most frequent pair.

Now, the most frequent pair is 'lo', so we merge them into a single token 'lo'.

New vocabulary: l, o, w, e, r, n, s, t, d, i, es, lo

Step 3: Continue this process until a defined stopping criterion is met, such as a fixed vocabulary size or a minimum frequency threshold.

After several iterations, we might end up with a vocabulary like this:

l, o, w, e, r, n, s, t, d, i, es, lo, ne, we, st, wi, de

Now, if we want to **represent the word "lowest"** using this vocabulary, it would be **tokenized** as ['lo', 'we', 'st'].

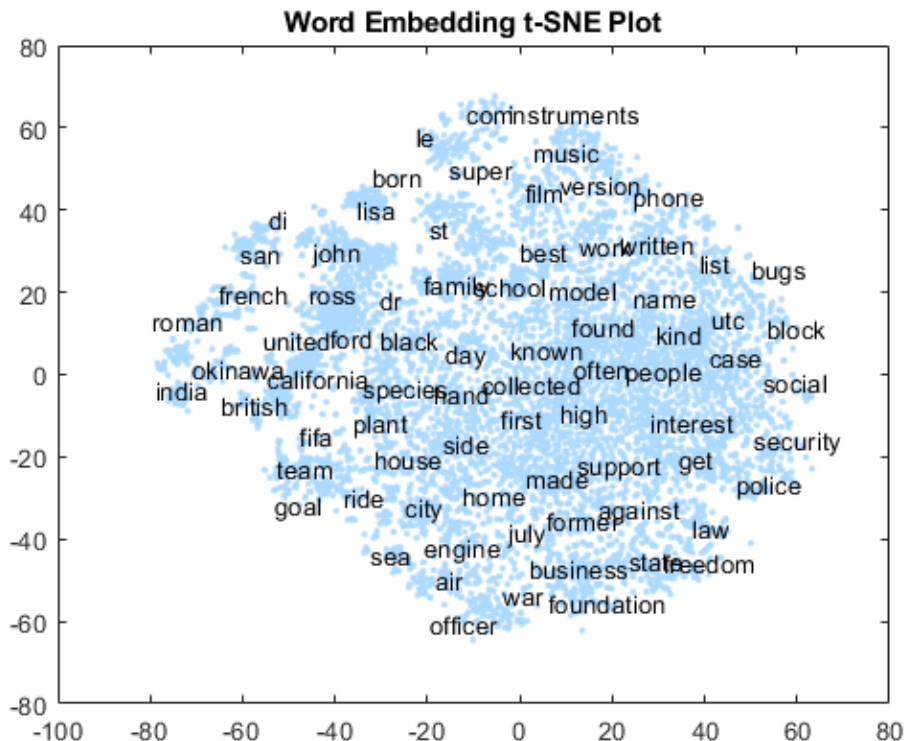
BPE helps models like GPT-2 and GPT-3 efficiently represent a wide range of words while keeping the vocabulary size manageable. By breaking words into smaller subword units, the model can handle rare words, out-of-vocabulary terms, and even different languages without significantly increasing the vocabulary size.

Word (token) Embedding

Fornire ai modelli gli **ID** dei token o una rappresentazione **one-hot-vector** derivata dagli stessi non è efficace perché **non codifica** la semantica e la similarità tra parole (es. sinonimi).

Le tecniche di **word** (o **token**) **embedding** associano a ogni parola/token del vocabolario un vettore di dimensionalità contenuta (es. 512). Parole/token di **significato simile** sono associate a **vettori vicini** nello spazio.

- **Word2Vec** è uno dei primi modelli di rete neurale (a due livelli) che può essere addestrato (**unsupervised**) per produrre l'embedding a partire da un corpus di testi.

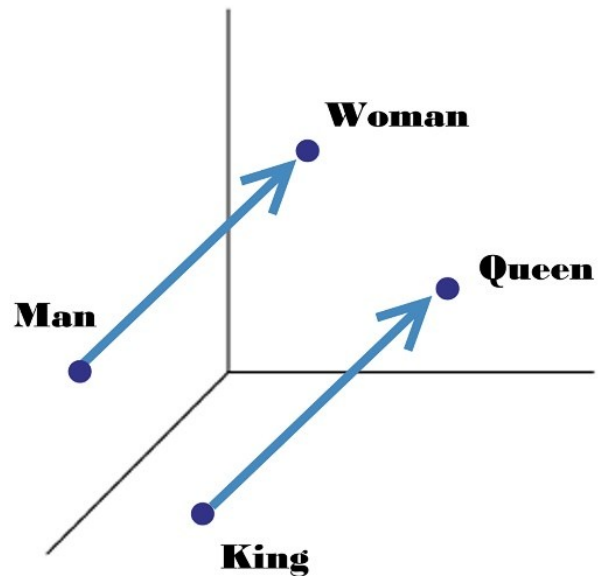


- Nei modelli recenti (GPT 3 e successivi) l'embedding viene **appreso** insieme al resto dei parametri durante il training.

Embedding e Algebra dei Vettori

La rappresentazione delle parole con **vettori** nello spazio multidimensionale (embedding) consente di calcolare similarità (semantica) tramite distanza Euclidea e di **manipolare concetti con operazioni algebriche**:

$$\text{King} - \text{Man} + \text{Woman} \approx \text{Queen}$$



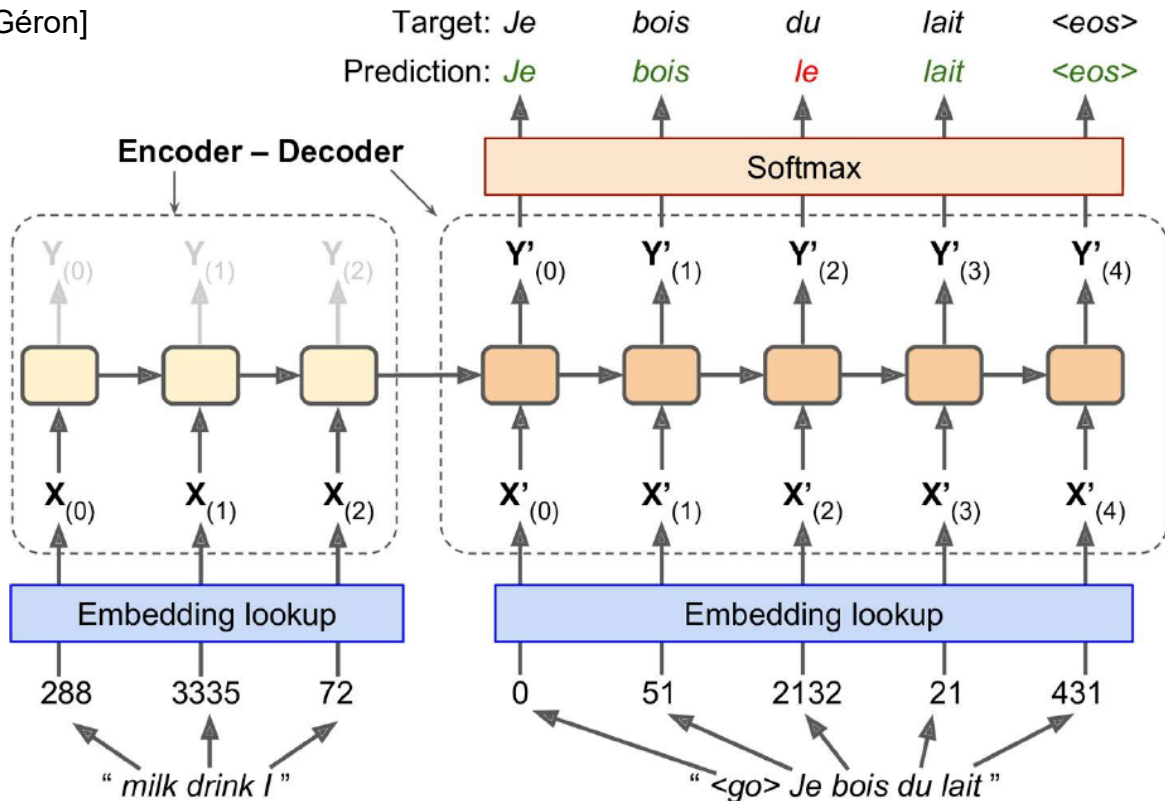
Possibile anche in contesti più complessi manipolando **embedding di input multimediali** nello spazio latente:



https://github.com/Newmu/dcgan_code

Neural Machine Translation

[A. Géron]



Elaborazione di sequenze in modalità **many-to-many** (lunghezza variabile). La **traduzione** da una **lingua** a un'altra è l'applicazione più nota.

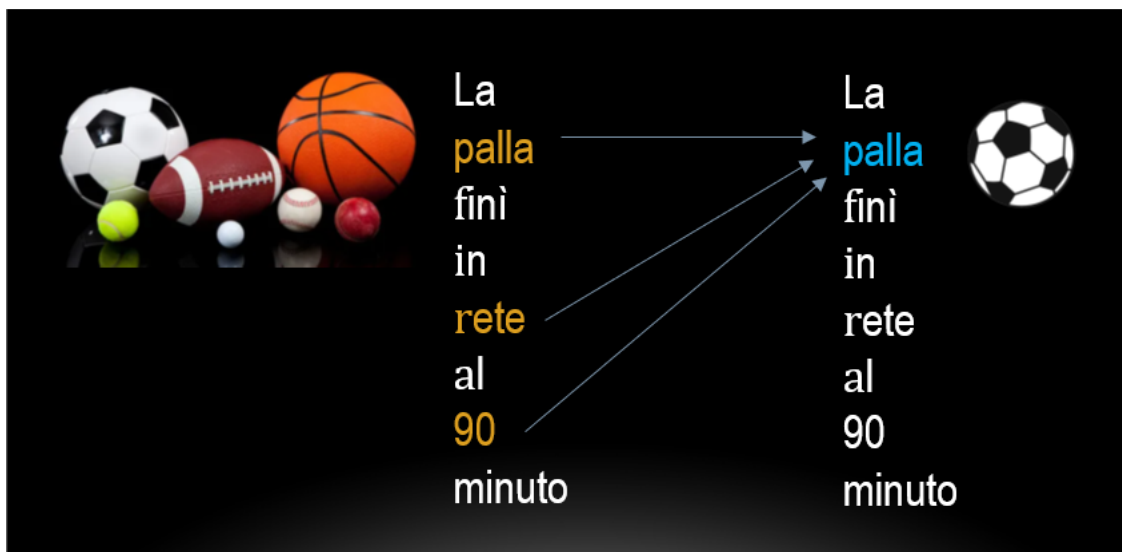
- I modelli (storicamente) più utilizzati sono architetture RNN di tipo **Encoder-Decoder** con meccanismi di **attenzione** (per ogni parola assegna peso alle altre parole per essa importanti nella frase). Vedi **Seq2Seq** per TensorFlow.
- Nel decoder si opera in modo **autoregressivo**, ovvero si genera un token per volta che viene riportato in input per successiva generazione.

Transformers

LSTM rappresenta un'importante evoluzione di basic RNN cells ma la modellazione di relazioni tra **porzioni lontane** dell'input **non è ottimale** e la computazione è prevalentemente **sequenziale** (difficile addestrare grandi modelli).

L'evoluzione dei **meccanismi di attenzione** (quando valuto un token a quali altri token devo far attenzione e quali posso invece ignorare) ha portato all'architettura **transformer** ([Google Brain, 2017](#)).

L'elemento chiave sono i layer di **self-attention** che mettono in relazione tra loro le parole nella frase per meglio **specificarne la semantica**.



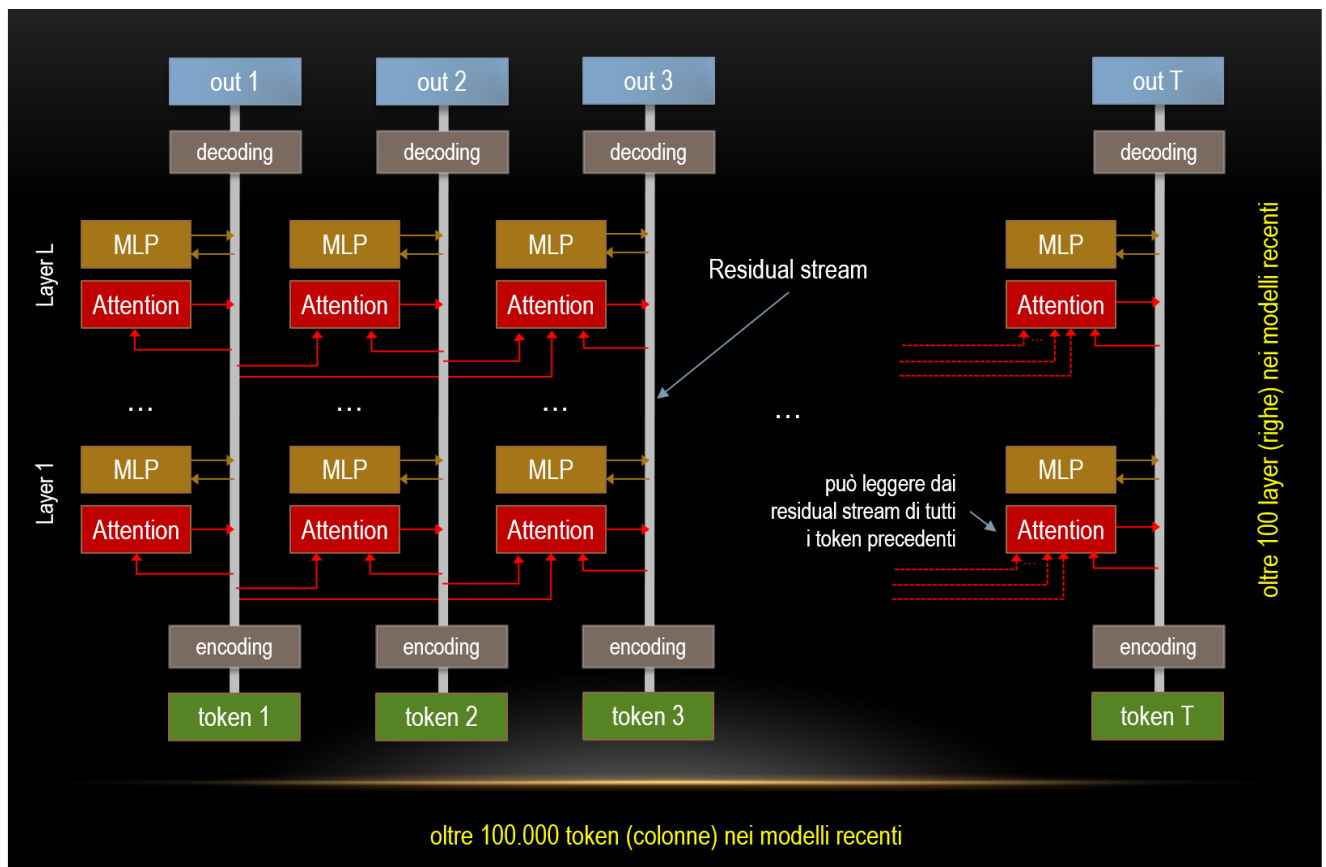
I transformers sono oggi utilizzati nella **maggior parte di applicazioni di NLP**, ma hanno dimostrato buone prestazioni anche in altri settori (es. Visione e Speech). *Architettura unificata del futuro?*

Ref: [Why Transformers are Slowly Replacing CNNs in Computer Vision?](#)

Transformers (2)

I modelli più recenti per la gestione di sequenze sono basati su architettura transformer semplificata **decoder-only**:

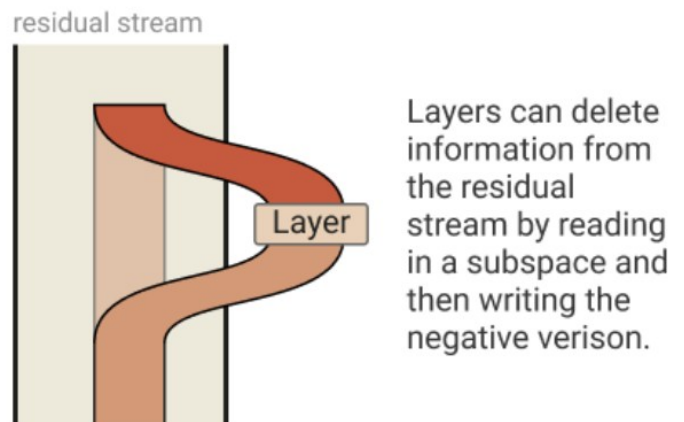
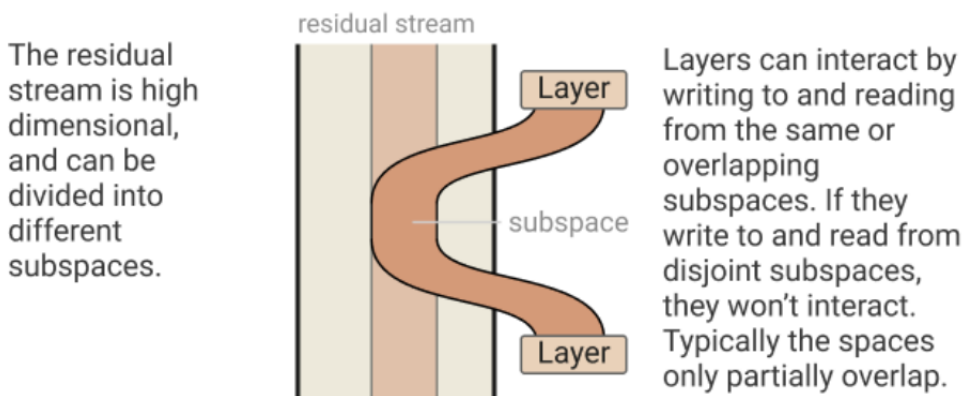
- architettura **colonnare**; come in tutte le reti ricorrenti utilizzate per sequenze: i) i pesi sono **condivisi** da tutte le colonne; ii) la generazione è autoregressiva (un token per volta).
- Per ogni colonna il **residual stream** (spazio multidimensionale) veicola le informazioni dal livello di input a quello di output.
- Per ogni layer, il blocco:
 - **MLP** è un semplice multilayer perceptron che può leggere e scrivere solo sul proprio residual stream.
 - **Attention** può leggere informazioni dai residual stream di tutti i token precedenti e scrivere sul proprio residual stream.



Transformers (3)

Ogni **residual stream** può essere visto come un **canale di informazione ad elevata capacità** (spazio di dimensione $d_{\text{model}} = 12288$ in GPT 3)

I moduli di **Attention** e **MLP** possono leggere/scrivere da/su **sottospazi** dei residual stream anche senza interferire con l'informazione già presente (message passing).



Immagini da: A Mathematical Framework for Transformer Circuits by N. Elhage et al. (Anthropic), Dicembre 2021.

Large Language Models (LLM)

Si tratta di modelli linguistici **autoregressivi** di **enormi** dimensioni (**miliardi** di parametri) addestrati su **grandi corpus** di testo (inclusi news, libri, articoli, codice, ecc.).

L'efficacia dell'**architettura transformers**, della tecnica di **addestramento** (**predici il token successivo di una sequenza**) e il fattore **scala** (sia nei parametri che nel dataset di training) hanno portato a **risultati ben oltre le aspettative**, non solo nelle classiche applicazioni di NLP ma anche nel problem solving, reasoning, coding, ...

Sparks of Artificial General Intelligence: Early experiments with GPT-4

Alcuni numeri di **modelli allo stato dell'arte** nel 2025:

- Numero **parametri**: 100-2000 miliardi (B)
- **Corpus** addestramento: 10000-100000 miliardi token (B)
Studi indicano rapporto di almeno 20:1 rispetto parametri
- **Costo** addestramento: 100-500M\$ (decine di migliaia di GPU impiegate per mesi)



LifeArchitect.ai/models-table

Next token prediction

La parte più corposa nell'addestramento di un LLM è basata sulla **predizione del prossimo token** (ottenuta facendo scorrere sul testo una finestra a lunghezza fissa). È detto **self-supervised** perché **non richiede etichettatura** manuale (il testo è tutto noto).

1	Nel	Nel	Nel	Nel	Nel
2	mezzo	mezzo	mezzo	mezzo	mezzo
3	del	del	del	del	del
4	cammin	cammin	cammin	cammin	cammin
5	di	di	di	di	di
6	nostra	nostra	nostra	nostra	nostra
7	vita	vita	vita	vita	vita
8	mi	mi	mi	mi	mi
9	ritrovai	ritrovai	ritrovai	ritrovai	ritrovai
10	per	per	per	per	per
11	una	una	una	una	una
12	selva	selva	selva	selva	selva
13	oscura	oscura	oscura	oscura	oscura
14	ché	ché	ché	ché	ché
15	la	la	la	la	la

In **ARANCIONE** i token forniti in input,
In **ROSSO** il token da predire.

Il training basato su **next token prediction** è stato criticato e considerato troppo semplicistico; ma probabilmente è uno dei punti di forza perché consente di apprendere:

- non solo **proprietà linguistiche** e **statistiche** del linguaggio
- forza il modello a fare previsioni corrette anche in problemi di matematica, logica, coding, senso comune, dove (in assenza di un oracolo o memorizzazione esaustiva di tutti i casi) **è possibile fare previsioni corrette solo apprendendo un modello del problema sottostante**.

Costruzione di prompt efficaci

La costruzione di prompt efficaci (prompt engineering) è attualmente “more of an art than a science”.

È consigliabile inserire nel prompt elementi che possono dirigere la risposta del modello nella direzione desiderata:

- Ad esempio se termino il prompt con un «punto elenco» il modello comprende che mi aspetto una lista puntata.
- Posso esplicitamente inserire nel prompt istruzioni tipo: nella risposta «elenca i singoli passi che ti hanno portato alla soluzione».
- Questo comportamento può essere ulteriormente potenziato inserendo nel prompt stesso esempi (Chain-of-Thoughts). In questo modo si facilita ovviamente anche la verifica di correttezza della risposta.

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅

Zero, One, Few-shot prompting

Per istruire un LLM sul tipo di **estrazione di informazione** (ed elaborazione) che ci aspettiamo, possiamo aggiungere uno o più esempi «Q: ... A: ...» direttamente nel prompt.

- **Zero-shot** non include esempi e pone la domanda direttamente.
- **One-shot** include un solo esempio.
- **Few-shot** include alcuni esempi (in genere meno di 5).

Esempio (3-shots):

Review: This movie sucks. Sentiment: negative.

Review: I love this movie. Sentiment: positive.

Review: This is not the kind of movie I love. Sentiment: negative.

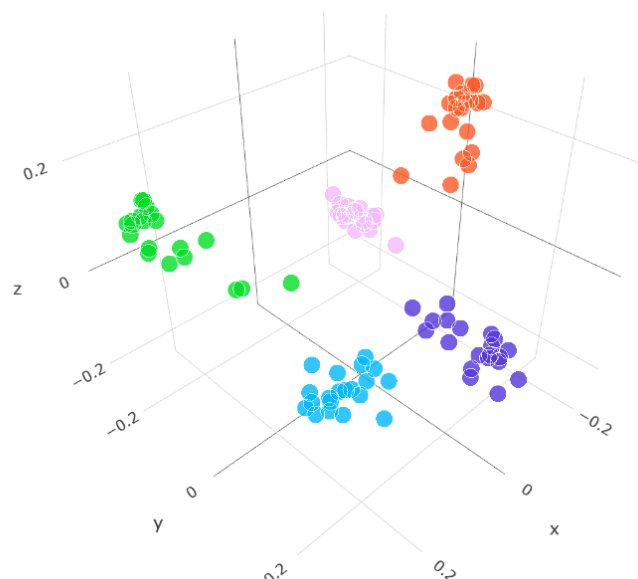
Review: This is the best movie I have ever seen. Sentiment:

Document Embedding

Utile in applicazioni che richiedono di individuare analogie o raggruppare testi per similarità (es. classificare per categorie le richieste inoltrate a un call center).

<https://openai.com/blog/introducing-text-and-code-embeddings>

- Si utilizza un LLM per estrarre un vettore di feature semanticamente significative da ciascun testo (**embedding**). *Attenzione a non confondere con il token embedding applicato in ingresso ai singoli token del prompt.*
- Si possono calcolare distanze (Euclidea o coseno) e/o applicare algoritmi di clustering agli embedding.
- **Operativamente:** dato un archivio di documenti posso utilizzare l'LLM per pre-calcolare l'embedding e memorizzarli in archivio locale. Dato un nuovo documento posso eseguire ricerche di similarità rispetto a quelli in archivio, estraendo l'embedding del solo nuovo documento e confrontandolo con quelli memorizzati.



● animal ● athlete ● film ● transportation ● village

Interrogare una knowledge base proprietaria

Immaginiamo di voler utilizzare un LLM potente per rispondere a domande poste in linguaggio naturale su un argomento molto specifico che l'LLM non conosce. Esempio: *funzionamento di un macchinario aziendale per il quale esiste un manuale d'uso di 1000 pagine*.

- Una possibilità è il **fine tuning** del modello sul testo del manuale (slide successiva), ma non è semplice/efficiente.
- Una seconda possibilità potrebbe essere quella di **inserire nel prompt** tutto il manuale (troppo lungo, poco efficiente!).
- Un'alternativa efficace si basa su **data indexing** attraverso embedding (vedi [LLamaIndex](#)):
 - Si suddivide il manuale in chunk (pezzi di testo) parzialmente sovrapposti
 - Di ciascuno si calcola l'embedding utilizzando l'LLM **A** e si crea un indice.
 - Data una query in linguaggio naturale si calcola l'embedding della query sempre con LLM **A** e si accede all'indice per identificare i chunk simili.
 - Si sottopone all'LLM **B** la query inserendo nel prompt (come contesto) il testo relativo ai soli chunk simili

A può essere un LLM leggero (anche open source in locale) per garantire privacy e confidenzialità), mentre **B** un LLM stato dell'arte da interrogare in cloud.

Fine Tuning

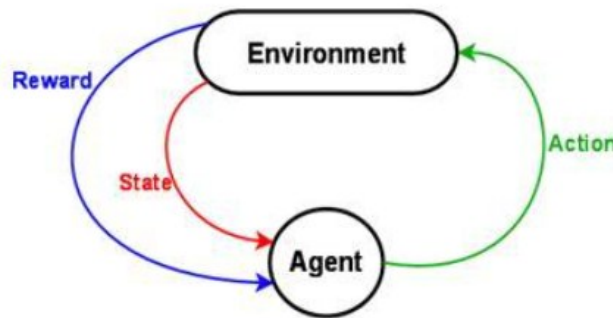
Quando un LLM deve fornire risposte su un dominio specifico sul quale non è stato addestrato e non è possibile (per vincoli di lunghezza e costi) inserire la nuova conoscenza nel prompt, un'alternativa è il fine-tuning del modello stesso (o meglio di un suo clone). Durante il fine-tuning si adattano «una parte» dei parametri per ottimizzare le risposte sul nuovo dominio.

- A tal fine è necessario predisporre un set «etichettato» (Prompt – Output) di almeno qualche centinaia di esempi (da suddividere poi in training e validation). Le prestazioni tendono ad aumentare solo linearmente con il raddoppio del numero di esempi.
- **Attenzione:** il modello può «dimenticare» cose che conosceva o sapeva fare. È tollerato?
- Di certo non è immaginabile eseguire tuning di miliardi di parametri con poche centinaia di esempi, e sono ipotizzabili le seguenti alternative:
 - La maggior parte dei livelli sono bloccati e il tuning è eseguito solo sull'ultimo o gli ultimi layer.
 - Si aggiungono adapters (layer laterali) inizializzati in modo da non modificare le risposte iniziali del modello. Durante il tuning il comportamento viene «lentamente» modificato. La tecnica LORA (Low-Rank Adaptation) è una delle più usate.

Reinforcement Learning

L'obiettivo è **apprendere un comportamento** ottimale a partire dalle esperienze passate.

- Un agente esegue **azioni** (a) che modificano l'**ambiente**, provocando passaggi da uno **stato** (s) all'altro. Quando l'agente ottiene risultati positivi riceve una ricompensa o **reward** (r) che però può essere temporalmente ritardata rispetto all'azione, o alla sequenza di azioni, che l'hanno determinata.



- Un **episodio** (o game) è una sequenza finita di stati, azioni, reward:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$$

- In ciascun stato s_{t-1} , l'obiettivo è scegliere l'azione ottimale a_{t-1} , ovvero quella che massimizza il **future reward** R_t :

$$R_t = r_t + r_{t+1} + \dots + r_n$$

- In molte applicazioni reali l'ambiente è **stocastico** (i.e., non è detto che la stessa azione determini sempre la stessa sequenza di stati e reward), pertanto applicando la logica del «*meglio un uovo oggi che una gallina domani*» si pesano maggiormente i reward temporalmente vicini (**discounted future reward**):

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^{n-t} \cdot r_n \quad (\text{con } 0 \leq \gamma \leq 1)$$

Q-Learning

- Il **discounted future reward** può essere definito ricorsivamente:

$$R_t = r_t + \gamma \cdot R_{t+1}$$

- Nel Q-learning la funzione $Q(s, a)$ indica l'**ottimalità** (o **qualità**) dell'azione a quando ci si trova in stato s . Volendo massimizzare il discounted future reward si pone:

$$Q(s_t, a_t) = \max R_{t+1}$$

- Nell'ipotesi che la funzione $Q(s, a)$ sia **nota**, quando ci si trova in stato s , si può dimostrare che la **policy** ottimale è quella che sceglie l'azione a^* tale che:

$$a^* = \arg \max_a Q(s, a)$$

- Il punto cruciale è dunque l'**apprendimento della funzione Q** . Data una **transizione** (quaterna) $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ possiamo scrivere:

$$Q(s_t, a_t) = \max R_{t+1} = \max(r_{t+1} + \gamma \cdot R_{t+2}) =$$

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot \max R_{t+2} = r_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

L'azione a_{t+1} (che non fa parte della quaterna) sarà scelta con la policy ottimale precedente, ottenendo:

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a)$$

nota come **Equazione di Bellman**.

Q-Learning (2)

L'algoritmo di apprendimento di Q sfrutta l'equazione di Bellman:

inizializza $Q(s, a)$ in modo casuale

esegui m episodi

$t = 0$

do

seleziona l'azione ottimale $a_t = \arg \max_a Q(s_t, a)$

esegui a_t e osserva r_{t+1} e s_{t+1}

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

$t = t + 1$

while (episodio corrente non terminato)

end episodi

Dove α è il learning rate: se $\alpha = 1$ l'aggiornamento di $Q(s_t, a_t)$ è eseguito esattamente con l'equazione di Bellman, se $\alpha < 1$, la modifica va nella direzione suggerita dall'equazione di Bellman (ma con passi più piccoli).

Valori tipici iniziali: $\gamma = 0.9$, $\alpha = 0.5$ (α è in genere progressivamente ridotto durante l'apprendimento)

■ Problema (pratico): quanto è grande Q ?

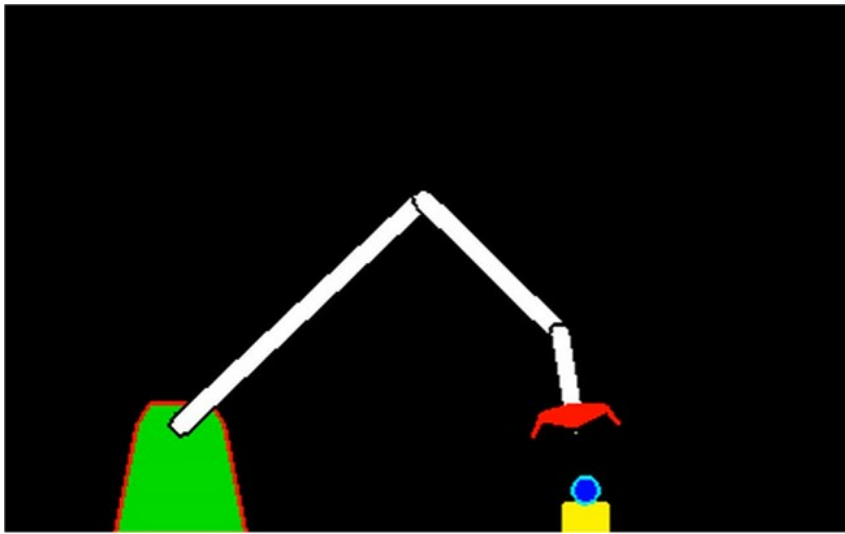
Tanti valori quanti sono i possibili stati \times le possibili azioni.

Q-Learning: esempio Grasping

Sviluppato con OpenAI Gym + Box2D (<https://gym.openai.com/>)

[Credits: Giammarco Tosi]

Un braccio robotico con **tre giunti** e **pinza**, deve prendere una pallina posta su un piedistallo di **altezza** (Y) e **posizione** (X) casuali.



Stato: codificato con Δx e Δy della pinza rispetto alla pallina + **stato della pinza** (aperto/chiuso).

Azioni: ruota a destra/sinistra su ognuno dei tre giunti, inverti stato pinza (chiudi se aperta e viceversa) → **7 azioni**.

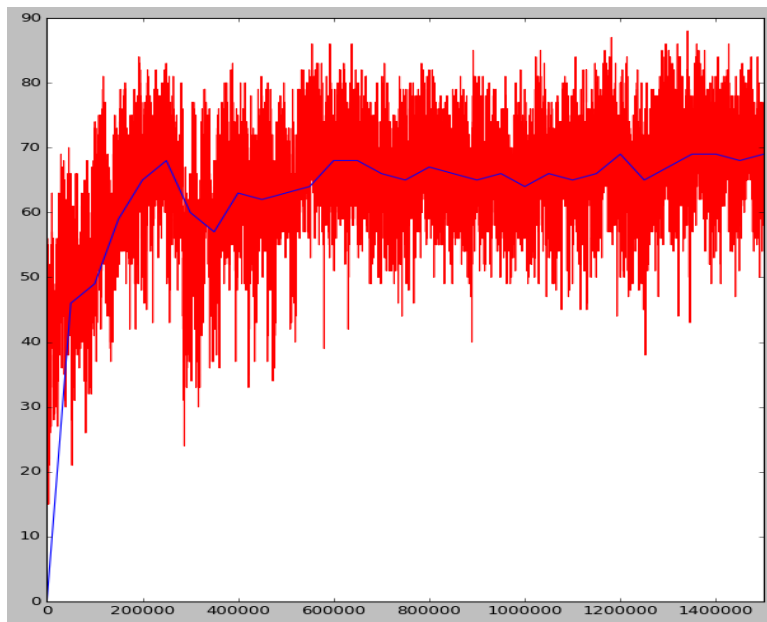
Reward:

- Avvicinamento alla pallina: +1
- Allontanamento dalla pallina: -1
- Movimento a pinza chiusa = -0.2
- Cattura pallina = **+100**

Q-Learning: esempio Grasping (2)

Q-learning:

- Memorizzazione esplicita della tabella **Q**
- Parametri addestramento: $\varepsilon = 0.1$, $\gamma = 0.6$, $\alpha = 0.2$



Il grafico rappresenta (in **rosso**) la percentuale di episodi vinti (pallina catturata) durante l'addestramento ogni 100 episodi. In **blu** è riportata la percentuale di episodi vinti ogni 5000 episodi.

Video di esempio:

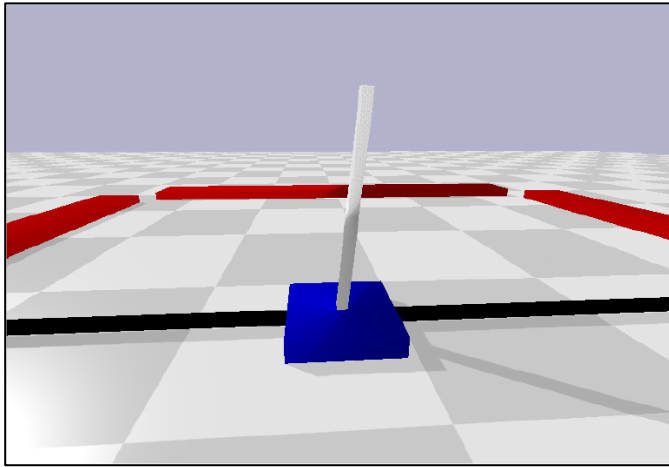
- Pre-training: http://bias.csr.unibo.it/maltoni/ml/Demo/Qarm_pre.wmv
- Post-training: http://bias.csr.unibo.it/maltoni/ml/Demo/Qarm_post.wmv

...altri esempi

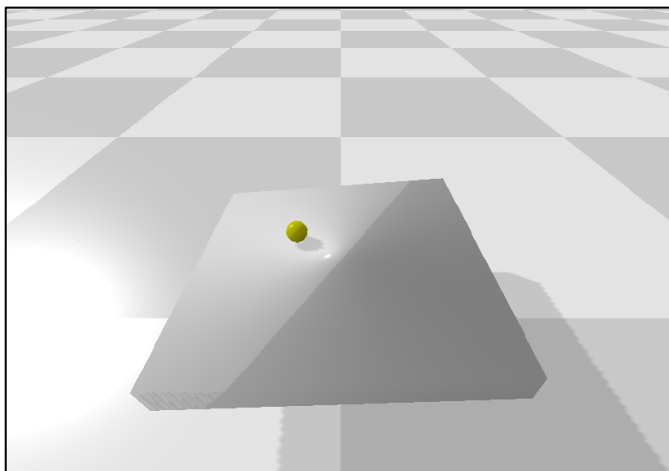
Balancing:

[Credits: Michele Buzzoni]

- Problemi di bilanciamento
- Simulatore 3D: Bullet Physics



Cart-Pole (Carrello Palo)



Mobile plane (Piano mobile)

Manipolazioni di oggetti (con reinforcement learning):

<https://bair.berkeley.edu/blog/2018/08/31/dexterous-manip>

Deep Reinforcement Learning

Nel 2013 ricercatori della società Deep Mind (immediatamente acquisita da Google) pubblicano l'articolo [Playing Atari with Deep Reinforcement Learning](#) dove algoritmi di reinforcement learning sono utilizzati con successo per addestrare un calcolatore a giocare (a livello super-human) a numerosi giochi della consolle Atari.

- La cosa di per sé non sarebbe [straordinaria](#), se non per il fatto che lo stato s osservato dall'agente non consiste di feature numeriche game-specific (es. *la posizione della navicella, la sua velocità*), ma di semplici immagini dello schermo ([raw pixel](#)). Questo permette tra l'altro allo stesso algoritmo di apprendere giochi diversi semplicemente giocando.
- Considerando lo stato s formato da 4 immagini dello schermo (a 256 livelli di grigio) e risoluzione 86×86 , il numero di stati è $256^{84 \times 84 \times 4} \approx 10^{67970}$, più del numero di atomi nell'universo conosciuto! [Impossibile gestire esplicitamente](#) una tabella Q di tali dimensioni.
- L'idea consiste nell'approssimare la funzione Q con una [rete neurale deep](#) (CNN) che, per ogni stato di input, fornisce in output un valore di qualità per ogni possibile azione.
- Ulteriori raffinamenti hanno portato allo sviluppo di [AlphaGo](#) che nel 2016 ha battuto a [Go](#) il campione umano Lee Sedol.
- Dopo oltre un decennio di studi sono stati messi a punto approcci ancora più efficienti e robusti come [Proximal Policy Optimization](#) ([PPO](#)) e [Group Relative Policy Optimization](#) ([GRPO](#)), oggi ampiamente utilizzati anche per l'ottimizzazione del comportamento di LLM.