Introduzione alle moderne tecniche dell'Intelligenza Artificiale

Giorgio **Dimino**
Rai - Centro Ricerche, Innovazione Tecnologica e Sperimentazione

L'intelligenza artificiale (o IA, dalle iniziali delle due parole, in italiano) è una disciplina appartenente all'informatica che studia i fondamenti teorici, le metodologie e le tecniche che consentono la progettazione di sistemi hardware e sistemi di programmi software capaci di fornire all'elaboratore elettronico prestazioni che, a un osservatore comune, sembrerebbero essere di pertinenza esclusiva dell'intelligenza umana.

(Marco Sommalvico)
Da Wikipedia, the free encyclopedia

Introduzione

In questo articolo verrà fornito uno spaccato introduttivo sulle *tecniche di Intelligenza Artificiale* allo stato dell'arte, ovvero le *reti neurali profonde*, che in meno di dieci anni hanno profondamente cambiato le aspettative di una tecnologia che sino ad ora non era riuscita, se non in alcuni ambiti verticali, a diventare pervasiva nella vita quotidiana delle persone. I recenti avanzamenti nel campo delle reti neurali profonde hanno portato le prestazioni di sistemi afferenti ai temi classici dell'intelligenza artificiale (*classificazione di testi o immagini*, *riconoscimento di volti*, *traduzione automatica*, *guida autonoma*) a livelli prossimi a quelli richiesti per un utilizzo industriale su larga scala. Questo articolo si propone di fornire le nozioni di base per capire il funzionamento delle reti neurali profonde e le problematiche ad esse connesse, lasciando ad altri articoli in questo stesso numero della rivista il compito di analizzare ambiti applicativi specifici e le relative peculiarità.

Sino a non più di dieci anni fa le reti neurali profonde (DNN) *erano guardate con sospetto nel mondo accademico, poi improvvisamente tutto è cambiato: nel giro di pochissimi anni la situazione si è ribaltata e oggi nelle conferenze di intelligenza artificiale fanno la parte del leone. Che cosa è successo? L'avanzamento tecnologico nel campo della videografica ha reso disponibili piattaforme di calcolo parallelo a bassissimo costo, su cui sono stati realizzati framework open source per la prototipazione di reti neurali, rendendo accessibile anche a piccoli gruppi di ricerca le risorse di calcolo necessarie a sperimentare una tecnologia così computazionalmente intensiva.*

Le reti neurali oggi rappresentano lo stato dell'arte in molti ambiti applicativi nel campo della visione artificiale e del processamento automatico del linguaggio. La pervasività della tecnologia è resa possibile dal fatto che le DNN hanno la caratteristica desiderabile di essere in gran parte indipendenti dal dominio di applicazione, per cui nuove tecniche e ottimizzazioni sperimentate per la risoluzione di un particolare problema spesso risultano immediatamente estendibili ad altri domini. Una rete è caratterizzata da un modello che determina le interconnessioni tra i neuroni che compongono la rete, una funzione di errore che ne permette l'addestramento ed un insieme di iperparametri che ne determinano la dimensione.

Questo contributo fornisce i fondamenti per comprenderne il funzionamento, le principali caratteristiche e le problematiche associate.

201_01.1_2.00

Definizione di Intelligenza Artificiale

In generale parliamo di *Intelligenza Artificiale* riferendoci a una macchina che è in grado di eseguire compiti che richiedono un'elaborazione dei dati non algoritmicamente predefinita allo scopo di prendere delle decisioni ad alto livello (es. giocare a scacchi).

Questi sono alcuni dei campi di applicazione dell'IA:

- Problem solving
- Ragionamento autonomo
- Interpretazione semantica dei dati
- Autoapprendimento
- Modellazione della conoscenza
- Visione artificiale
- Riconoscimento del parlato

Le principali tecniche che fanno capo all'Intelligenza Artificiale possono essere schematicamente raggruppate in due grandi filoni di ricerca. Il primo, particolarmente in auge negli anni '80 e '90 del secolo scorso, si basa sulla *modellazione esplicita della conoscenza* e ha portato alla realizzazione dei cosiddetti *sistemi esperti*, basati su un'organizzazione gerarchica della conoscenza (ad esempio insiemi di ontologie e regole) per trarre conclusioni logiche a partire dai dati. Il secondo, comunemente conosciuto col nome di *Machine Learning*, è focalizzato sull'apprendimento automatico basato sulla ricerca di metodi di regressione che permettano di approssimare un certo fenomeno di interesse con una distribuzione stocastica, stimata a partire da un numero limitato di osservazioni, in modo da poterne predire gli stati futuri.

Le *reti neurali profonde* (*Deep Neural Networks* - ***DNN***) sono una derivazione del machine learning che si basa sulla realizzazione di reti neurali formate da un gran numero di celle elementari identiche, ispirate dalla conformazione del cervello umano.

Cenni storici

I tentativi di impiegare i calcolatori per compiti che emulano il comportamento umano risalgono agli anni '50. Già Alan Turing ipotizzava la realizzazione di una macchina in grado di apprendere autonomamente e definiva il concetto di intelligenza artificiale tramite la famosa metafora dell'*imitation game* [1], detto anche *test di Turing*, secondo cui una macchina può dirsi dotata di intelligenza se un interlocutore dietro ad un tendone non è in grado di distinguere se stia parlando con un essere umano oppure con un automa. La prima formulazione di rete neurale, denominata *multilayer perceptron* [2], risale agli anni '60, ispirata dall'osservazione della anatomia del cervello umano, ma solo negli anni '80 viene introdotto un meccanismo di apprendimento basato sulla propagazione all'indietro del gradiente dell'errore [3], principio fondamentale dell'apprendimento delle reti moderne. Negli stessi anni vengono definite le *reti convoluzionali per la visione artificiale* [4], anch'esse ispirate all'anatomia della corteccia visiva [5].

Gli studi sulle DNN procedono negli anni senza particolare impulso, principalmente ad opera di pochi studiosi tra cui ***Yan LeCun***, sino al 2012, quando un gruppo di ricercatori dell'università di Toronto presenta alla conferenza internazionale **NIPS** (*Neural Information Processing Systems*) un articolo [6] in cui dimostra di essere in grado di surclassare lo stato dell'arte in un noto benchmark di classificazione di immagini, chiamato *ImageNet Classification*, con un sistema basato su reti neurali convoluzionali, denominato **AlexNet**. Tale sistema era a sua volta ispirato da un articolo del 1998 di Bengio e LeCun [7], in cui viene definita l'architettura della prima rete convoluzionale per la classificazione di immagini, chiamata **LeNet**. Da allora, e nel giro di pochissimi anni, le DNN sono state impiegate per migliorare lo stato dell'arte in praticamente tutti i campi dell'IA, dal riconoscimento del parlato, alla traduzione automatica, alla classificazione di immagini.

Tra i fattori che hanno determinato una evoluzione così rapida di una tecnologia rimasta dormiente per così tanti anni vi sono due condizioni esterne fondamentali. La prima è l'introduzione di *acceleratori* per il calcolo parallelo nelle schede grafiche dei personal computer (**GPU**), fondamentali per il rendering in tempo reale degli scenari complessi

dei videogiochi moderni. ***NVIDIA***, uno dei maggiori produttori di chip grafici, intuendo la potenzialità dell'utilizzo delle GPU nella simulazione numerica, ha sviluppato e reso disponibile gratuitamente l'**SDK CUDA** [8] che permette l'utilizzo degli acceleratori per eseguire calcoli vettoriali all'interno di programmi utente, principalmente a scopi di simulazioni scientifiche. Questa tecnologia si sposa perfettamente con le esigenze delle reti neurali ed ha reso possibile ridurre di due ordini di grandezza [9] il tempo necessario ad effettuare l'addestramento di una rete, per di più utilizzando hardware di costo limitato. I ricercatori hanno quindi potuto accedere a piattaforme hardware di basso costo adatte a sperimentare architetture di reti neurali più complesse di quelle utilizzate in precedenza. La seconda condizione è legata alla diffusione del *software open source*. Quando sono apparsi i primi articoli scientifici che dichiaravano risultati oltre lo stato dell'arte sui benchmark di machine learning più popolari, lo scetticismo della comunità scientifica era piuttosto alto. A volte gli articoli venivano addirittura rigettati dai revisori in quanto i risultati riportati erano considerati inverosimili.

La diffusione di librerie open source basaste sul **CUDA** di ***NVIDIA*** per l'esecuzione di reti neurali (ad es. **TensorFlow** di ***Google*** [10] o **PyTorch** di ***Facebook*** [11]) ha permesso ai ricercatori di corredare gli articoli con i programmi software utilizzati per ottenere i risultati dichiarati ed alla comunità scientifica di validare i suddetti risultati e di sperimentare con quanto realizzato da altri [12]. Oggi qualsiasi articolo scientifico sulle reti neurali che aspiri ad un certo grado di credibilità viene corredato da un'implementazione software del modello proposto e ciò ha generato un circolo virtuoso in cui tutti i risultati ottenuti sono immediatamente messi a disposizione dell'intera comunità scientifica che può quindi sperimentare velocemente nuove soluzioni partendo dallo stato dell'arte più recente. Inoltre, l'utilizzo di GPU commerciali, disponibili anche in molte piattaforme di cloud computing commerciali, ha reso la sperimentazione delle reti neurali accessibile a chiunque abbia le conoscenze necessarie, senza necessità di investimenti rilevanti in infrastruttura.

Modello di base

Le prime formulazioni del modello di rete neurale hanno origine dal tentativo di imitare il funzionamento del cervello umano realizzando modelli simili alle reti di neuroni che costituiscono il cervello. Alla base di questo approccio viene postulato che, definito un modello semplificato del cervello, sia possibile addestrarlo ad eseguire dei compiti cognitivi senza dover programmare passo a passo lo svolgimento dell'attività. Sebbene non sia mai stato provato che questi modelli rappresentino un'approssimazione realistica delle facoltà cerebrali, i risultati ottenuti sono stati incoraggianti e hanno spinto la ricerca a sperimentare reti via via più complesse. Il modello si basa su di un elemento detto *perceptron* [13] che implementa la funzionalità di un singolo neurone (Fig. 1). Si costruisce poi una rete composta da un numero elevato di perceptron (da qui in avanti definiti *neuroni*) in comunicazione tra loro.

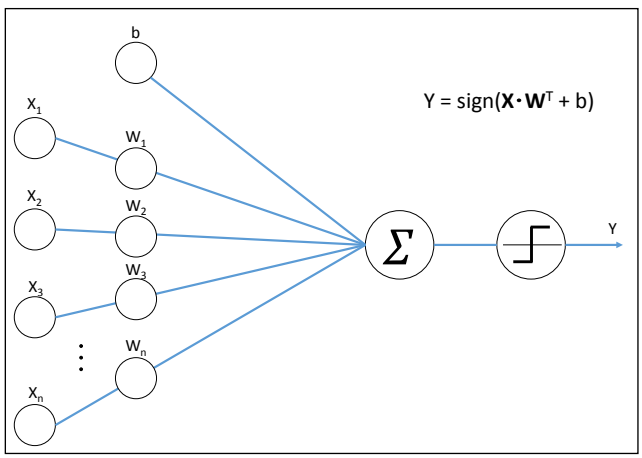


Fig. 1 – Perceptron

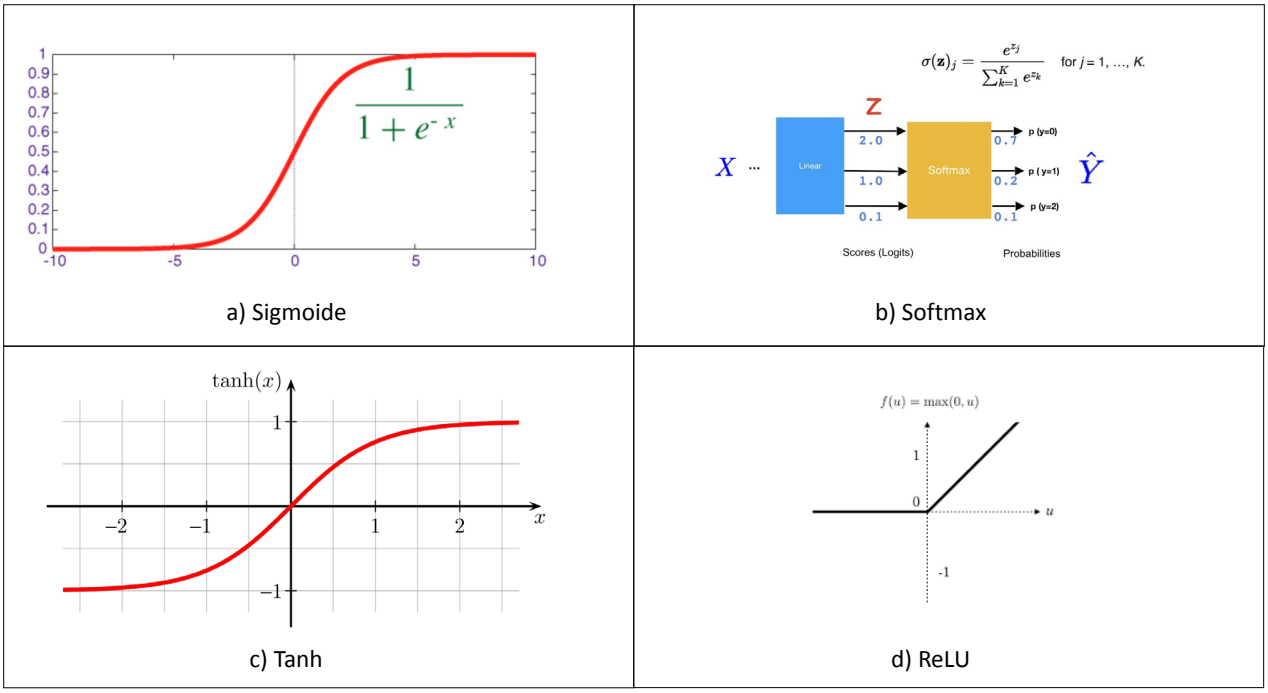
- ***X*** è il vettore di ingresso (es, i pixel di un'immagine oppure l'uscita di altri neuroni)
- ***W*** è un vettore che contiene i parametri del neurone che vengono *appresi* durante il training
- *b* è il termine di bias
- *sign()* è la funzione di attivazione
- *Y* è l'uscita del neurone, detta anche *attivazione*

Quindi la funzione di trasferimento del modello consiste nel prodotto scalare tra il vettore di ingresso ***X*** e il vettore ***W*** (detto anche *campo di ingresso*), modulato dalla *funzione di attivazione* ***f()***, in questo caso realizzata dalla funzione *sign()*, che restituisce *1* se l'input è positivo oppure *–1* se l'input è negativo. Il vettore ***W*** e il *termine di bias* ***b*** (collettivamente chiamati *pesi*) vengono appresi dalla rete autonomamente durante la *fase di training*, che consiste in un processo iterativo in cui si calcola l'errore tra l'uscita attesa e quella prodotta dalla rete su di una serie di esempi il cui risultato è noto, e si utilizza l'errore per modificare progressivamente il valore dei pesi sino a convergere su di un valore di errore minimo. Si noti come nel caso in cui la funzione di attivazione sia un'applicazione lineare del campo di ingresso, il perceptron è analiticamente identico ad un *classificatore lineare*.

La caratteristica che rende potenti le *reti neurali* risiede nell'introduzione di una nonlinearità nel sistema attraverso l'utilizzo di particolari funzioni di attivazione, e nel sovrapporre più livelli di neuroni realizzando così delle reti profonde in analogia col sistema cognitivo umano. Infatti, se il modello del neurone fosse puramente lineare, per il principio della sovrapposizione degli effetti dei sistemi lineari sarebbe sempre possibile collassare la rete in un'unica funzione lineare. Si veda al riguardo il teorema di approssimazione universale che stabilisce che una funzione continua può essere approssimata da una rete del tipo appena descritto di ampiezza o profondità arbitrariamente grande [14][15].

In Fig. 2 sono illustrate le funzioni di attivazione utilizzate più frequentemente nei sistemi.

Fig. 2 – Principali *funzioni di attivazione*:
a) *Sigmoide*: fornisce un valore compreso tra 0 e 1; può quindi essere usata per esprimere una probabilità. Può essere vista come una versione stocastica della funzione gradino utilizzata nei sistemi binari.
b) *Softmax*: agisce su un insieme di attivazioni in ingresso e fornisce un valore tra 0 e 1 ma in più la somma di tutti gli elementi fa sempre 1, quindi è adatta a modellizzare la probabilità di appartenenza ad una classe (ed una sola) fra N classi date.
c) *Tangente iperbolica*: fornisce un valore tra -1 e 1 e viene utilizzata nei livelli interni della rete.
d) *ReLU*: fornisce un valore tra 0 e 1 e viene utilizzata nei livelli interni della rete.



a) Sigmoide

b) Softmax

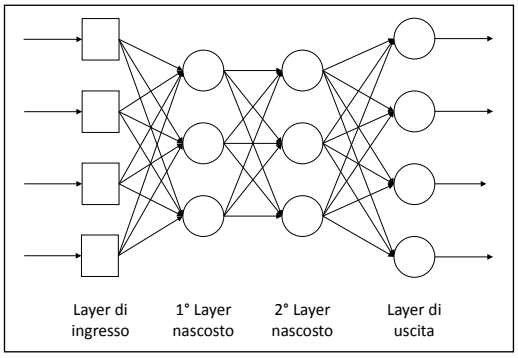
c) Tanh

d) ReLU

Nella configurazione di Fig. 1, il singolo neurone si comporta quindi come un *classificatore binario*, ovvero indica se il vettore di ingresso appartiene o meno ad una data classe. È evidente che è possibile estendere questa rete mettendo più neuroni in parallelo per realizzare un classificatore a *n* classi. Una rete di questo tipo è concettualmente molto simile ai più comuni classificatori utilizzati nel Machine Learning, come **Naive Bayes Classifier** [16] o **Support Vector Machine** [17]. La potenza delle reti neurali però diventa evidente quando vengono sovrapposti più layer di neuroni come schematizzato in Fig. 3.

Si può allora ipotizzare che i layer interni operino delle trasformazioni sui dati di ingresso via via a più alto livello in modo da agevolare il lavoro di classificazione dell'ultimo layer.

Fig. 3 – Multilayer Perceptron

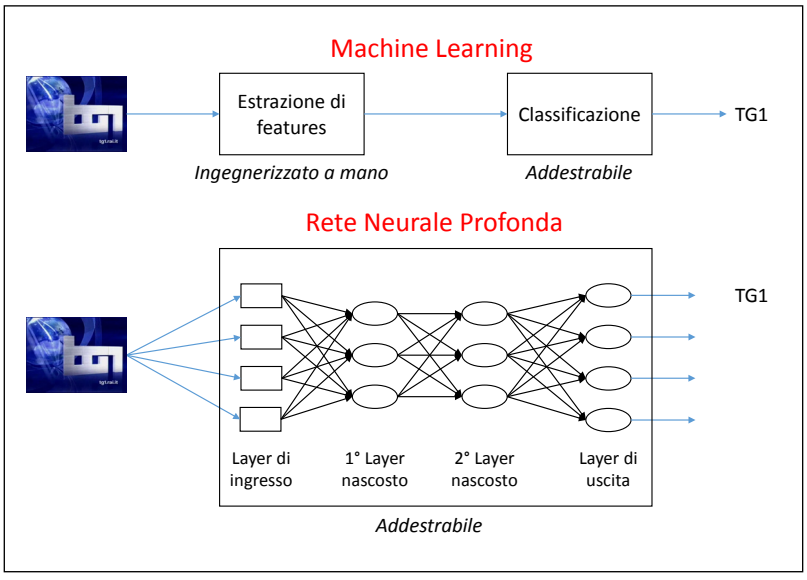


Nel *Machine Learning classico* il processo di progettazione della rete può essere logicamente diviso in due fasi: nella prima vengono individuate, principalmente manualmente, le trasformazioni dei dati di ingresso (chiamate *processi di estrazione di features*) necessarie a portare i dati in uno spazio facilmente separabile, nella seconda invece viene definito un *classificatore* idoneo (che può essere o meno supervisionato). L'efficacia della rete dipende quindi pesantemente dalla scelta delle trasformazioni della prima fase, che solitamente richiedono approfondita conoscenza del dominio dei dati che si devono trattare. Il processo nel suo complesso è scarsamente ingegnerizzabile e non generalizza facilmente. Al contrario in una *DNN*[Nota 1] tutti i layer vengono addestrati contemporaneamente a partire dal set di dati di ingresso e non è richiesta una conoscenza specifica del dominio in quanto la rete *scopre* autonomamente durante il processo di addestramento quali sono le trasformazioni dei dati di ingresso più adatte ad eseguire il compito assegnato.

La Fig. 4 illustra schematicamente il confronto tra questi due approcci.

Nota 1 - Da questo punto in poi useremo il termine *rete neurale* per indicare una DNN

Fig. 4 – Confronto tra *algoritmo di Machine Learning* e *Rete Neurale Profonda* (DNN)

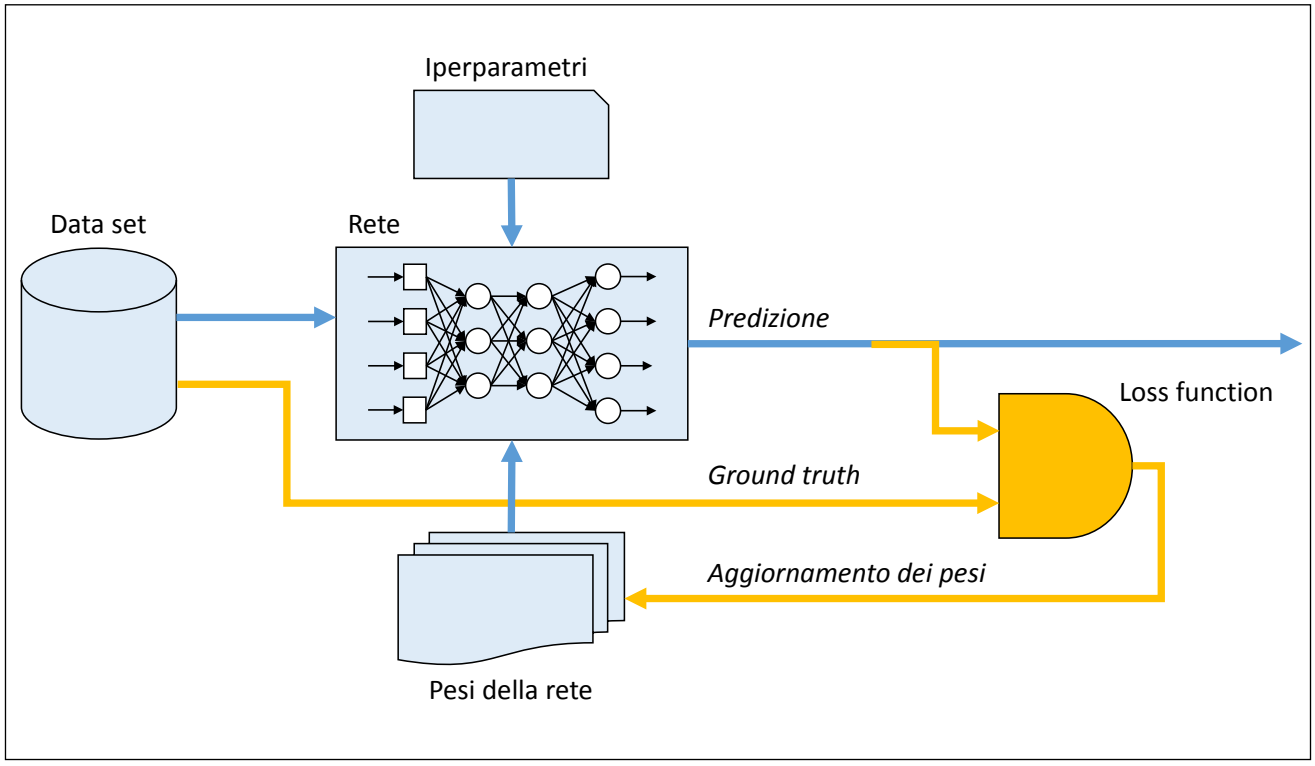


Il progetto di una rete neurale in grado di assolvere un determinato compito richiede la definizione dei seguenti elementi, che verranno illustrati nelle prossime sezioni:

- l'*architettura della rete* ed i relativi *iperparametri*
- il *data set* annotato su cui effettuare l'addestramento
- la *loss function*, ovvero la funzione che misura lo scostamento del comportamento della rete da quello atteso

Una rete neurale ha due modalità di funzionamento: la prima effettua l'addestramento modificando gradualmente i parametri dei neuroni che la compongono sino a minimizzare l'errore calcolato tramite la loss function (Fig. 5), la seconda ne permette l'uso in ambiente di produzione.

Fig. 5 – Schema di principio di addestramento di una Rete Neurale. La parte in giallo sottende all'addestramento della rete e viene rimossa quando la rete viene utilizzata in produzione



Addestramento

Generalità

Le reti neurali fanno parte della famiglia di algoritmi cosiddetti *supervisionati*, ovvero che vengono condizionati ad eseguire un dato task tramite un processo di addestramento in cui i parametri del sistema vengono iterativamente modificati in modo da riprodurre il comportamento atteso su di un insieme di esempi pre-annotati (il *data set di addestramento*). Per effettuare l'addestramento è necessario definire una funzione, detta *loss function*, che ha il compito di stimare l'errore medio effettuato dalla rete nel valutare un certo set di dati di ingresso rispetto ai valori attesi e che funge quindi da guida per l'addestramento. La caratteristica principale richiesta alla *loss function* è quella di essere derivabile rispetto ai pesi della rete, infatti il meccanismo di addestramento si basa sulla propagazione all'indietro (*backpropagation*) del gradiente dell'errore medio [18], calcolato rispetto ai pesi della rete, mediante la funzione

$$w_i \Leftarrow w_i - \lambda \frac{\partial L\left(f(\mathbf{x}; \mathbf{w})\right)}{\partial w_i}$$

o in forma vettoriale

$$\mathbf{w} \Leftarrow \mathbf{w} - \lambda \nabla L(f(\mathbf{x}; \mathbf{w}))$$

Il termine λ è detto *coefficiente di apprendimento* e governa la quantità massima di variazione ammessa per ciascun peso ad ogni iterazione, w_i sono i *pesi della rete*, $f(\boldsymbol{x}; \boldsymbol{w})$ è l'uscita della rete neurale presa a riferimento per l'apprendimento e $L()$ è la *loss function*.

In pratica l'apprendimento avviene nel seguente modo:

- si inizializza la rete inizializzando i parametri secondo un determinato criterio (ad esempio seguendo una distribuzione normale);
- si procede effettuando un certo numero di cicli (detti *epoche*) in cui tutto il training set, suddiviso in mini-batch per ragioni di praticità di calcolo, viene processato dalla rete;
- per ogni mini-batch si calcola l'errore medio della rete tramite la loss function $L()$ e si utilizza il ∇L calcolato rispetto a $\boldsymbol{w}$ per aggiornare ciascun peso w_i;
- l'aggiornamento finisce dopo un numero predefinito di epoche o quando l'errore calcolato su di un data set di validazione (una partizione del data set non utilizzata per il training) non decresce più.

Ottimizzare l'addestramento

La scelta dell'iper-parametro λ è cruciale, infatti valori troppo piccoli determinano una convergenza della rete verso il minimo dell'errore molto lenta e sono quindi necessarie elevate risorse di calcolo, per contro un valore troppo elevato porta ad un andamento irregolare della loss function e questo potrebbe comportare la divergenza della rete e quindi il fallimento dell'addestramento. È evidente che per come il processo di apprendimento è definito, non vi sia alcuna garanzia di riuscire a raggiungere il minimo assoluto della loss function, anzi è molto frequente il caso in cui il processo si arresti in un minimo locale o addirittura non converga verso un valore stabile. Per questo in letteratura vengono proposti numerosi accorgimenti per stabilizzare l'apprendimento [19], detti *di ottimizzazione*. Citeremo qui solo alcuni dei più utilizzati.

Stocastic Gradient Descent (***SGD***): l'aggiornamento dei pesi dovrebbe avvenire al termine di ogni epoca, quando cioè è stata calcolata la media della loss function sull'intero data set di training. In pratica, poiché il data set è solitamente troppo grande per essere contenuto nella memoria dell'unità di calcolo, si preferisce partizionarlo in mini-batch più piccoli, generati campionando in modo casuale il data set, e si aggiornano i pesi ad ogni mini-batch, nell'ipotesi che i dati contenuti nel mini-batch siano sufficienti a rappresentare l'intero set. A questo modo la convergenza della rete è anche più rapida.

Cosine annealing: il coefficiente di apprendimento λ viene modulato nel corso delle epoche secondo una funzione coseno in modo da ridurne progressivamente il valore man mano che il training procede. Più ci si avvicina al minimo più ci si muove per piccoli passi per evitare di scavalcarlo.

Momento del gradiente: in ciascuna iterazione la correzione è data da una media pesata del gradiente attuale e di quello del passo precedente [20]. Di fatto viene inserita un'inerzia nel gradiente per evitare variazioni di direzione troppo repentine magari in una fase in cui la rete è all'inizio del training e ancora instabile.

RMSprop: il gradiente calcolato viene diviso per una quantità proporzionale alla radice quadrata della media pesata del quadrato del gradiente attuale e del quadrato del gradiente precedente. In questo modo si cerca di normalizzare la quantità di variazione tra i diversi pesi della rete.

Weight Decay: nella loss function viene aggiunto un termine che penalizza la complessità della rete, che si riflette nello smorzamento del gradiente di un termine proporzionale al valore del peso. Reti troppo complesse tendono più facilmente a generare *overfitting*.

Con il termine *overfitting* si intende un fenomeno patologico comune nei sistemi supervisionati in cui un addestramento troppo prolungato o realizzato in modo scorretto rende la rete simile a una memoria associativa. Ovvero, la rete anziché costruire un modello che generalizza il comportamento condizionato durante il training, impara così bene a riconoscere gli esempi del training set da rigettare ogni altro input. Pertanto, è necessario prendere ogni precauzione per evitare di generare questa situazione durante l'addestramento. Il primo accorgimento che viene solitamente adottato consiste nel comparare durante il training il valore della loss function applicata al training set col valore calcolato sul set di validazione. Quando il primo risulta essere sensibilmente più basso del secondo (ricordiamo che l'obiettivo del training è la minimizzazione della loss function) vuol dire che la rete sta perdendo la capacità di generalizzare e che non è possibile spingere oltre l'addestramento.

Per mitigare il rischio di overfitting è possibile introdurre varie tecniche dette di regolarizzazione della rete.

Il ***dropout*** è una di queste tecniche in cui parte dell'informazione durante il training viene cancellata di proposito per evitare che la rete apprenda dagli esempi troppi dettagli perdendo in generalità. Si applica solitamente sui dati del training set, mascherando parte dei dati in modo casuale. Ad esempio, se l'ingresso è un'immagine, ne viene mascherata o distorta una porzione. Ovviamente ad ogni epoca il dropout deve agire su dati diversi. È possibile anche mascherare le attivazioni dei layer interni oppure, anche se utilizzato più di rado, eliminare temporaneamente alcuni collegamenti tra i neuroni della rete (ovvero azzerarne i pesi).

Un'altra tecnica utilizzata è la ***data augmentation*** in cui i dati del training set vengono processati per ottenere in modo automatico ulteriori esempi su cui effettuare l'addestramento. Ad esempio, nel caso delle immagini, si possono applicare trasformazioni che ne varino il valore dei pixel senza per questo modificarne il contenuto semantico o generare situazioni inverosimili. Le trasformazioni tipiche sono traslazione, rotazione, zoom, variazione di luminosità e contrasto, distorsione, ma quali siano appropriate o meno dipende dal caso specifico.

Un'altra situazione patologica che vanifica l'addestramento è il cosiddetto *annullamento (o esplosione) del gradiente*. Il gradiente calcolato assume valori così piccoli o così grandi da non poter più essere rappresentato con la notazione floating point di un computer. Per molti anni la profondità delle reti neurali è stata mantenuta bassa a vantaggio del parallelismo dei neuroni nello stesso layer proprio per problematiche legate all'annullamento del gradiente. Per ovviare al problema, solitamente si inseriscono livelli di normalizzazione delle attivazioni tra un layer e l'altro utilizzando una tecnica detta **BatchNorm** [21], che consiste nel normalizzare le attivazioni di uscita di ciascun layer mediante una

formula basata su media e varianza calcolate sul minibatch attuale.

La loss function

Come abbiamo visto la *loss function* è un componente fondamentale della rete, che ha il compito di valutare durante il training quanto il comportamento della rete si discosti da quello atteso. Poiché l'addestramento si basa sulla propagazione del gradiente della loss function, è evidente che questa deve essere derivabile rispetto ai parametri della rete oggetto dell'apprendimento.

A seconda del tipo di uscita della rete, la loss function potrà assumere formulazioni completamente differenti. Un primo caso è quando la rete implementa un classificatore. L'uscita sarà un vettore, la cui dimensione è pari al numero di classi tra cui si deve discriminare, e dove ciascuna componente del vettore può essere associata ad una stima della probabilità che i dati di ingresso appartengano ad una data classe. La misura che viene generalmente utilizzata in questo caso è detta *cross-entropy* ed è espressa dalla formula:

$$CE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\left(y_{i,j}\cdot\log\left(p\left(y_{i,j}\right)\right)+\left(1-y_{i,j}\right)\cdot\log\left(1-p\left(y_{i,j}\right)\right)\right)$$

dove $y_{(i,j)}$ è un indicatore binario che dice se il campione i-esimo appartiene o meno alla classe j-esima e $p(y_{(i,j)})$ è la probabilità stimata dal sistema che il campione i-esimo appartenga alla classe j-esima.

Se invece l'uscita rappresenta una grandezza numerica, viene normalmente usata una misura che esprime una distanza, come l'errore quadratico medio (MSE) oppure la sua radice quadrata (RMSE)

$$MSE = \frac{1}{N}\sum_{1}^{N}(y_i^2 - \hat{y}_i^2)$$

$$RMSE = \frac{1}{N}\sum_{1}^{N}\sqrt{(y_i^2 - \hat{y}_i^2)}$$

dove y_i è il valore vero corrispondente al campione x_i e $\hat{y}_i$ la sua predizione elaborata dalla rete.

Quando è necessario mediare tra più esigenze, è possibile comporre la loss function come somma di più termini, dove ciascun termine esprime un errore da minimizzare. Nei casi più complessi la loss function può contenere una seconda rete neurale addestrata in precedenza o addirittura in tandem con la rete principale, come nel caso delle *reti GAN* che vedremo più avanti.

Il data set per l'addestramento

A seconda della complessità del problema da risolvere il set di dati da utilizzare per l'addestramento, *data set*, può essere più o meno esteso, ma in generale è costituito da un insieme di dati pre-annotati (ad es. immagini ed etichette corrispondenti) scelti in modo da caratterizzare tutti gli aspetti salienti delle classi tra cui si vuole discriminare. Per fare un esempio pratico, il data set più utilizzato in ambiente accademico per effettuare benchmark sulla classificazione di immagini, **ImageNet** [22], contiene circa *14.000.000* di immagini suddivise in *2000* classi diverse. Generalmente una parte del data set (ad esempio il 20%) viene utilizzato come *validation set*, ovvero viene usato come set di riscontro per verificare la capacità della rete di generalizzare, cioè di rispondere correttamente a fronte di dati di ingresso mai visti in precedenza, e non contribuisce al training.

La selezione di un data set appropriato è un'operazione critica nella progettazione di un sistema basato su reti neurali. Innanzi tutto, non esistono formule per calcolare quanto debba essere esteso il data set, in linea di massima più è grande e meglio è. La difficoltà non risiede solo nella possibilità di reperire un gran numero di esempi, ma nel fatto che questi devono essere annotati con la risposta attesa (ad es. la categoria di appartenenza di un'immagine). In casi come la traduzione automatica ciò significa che è necessario disporre di un corpus di testi allineato sulle due lingue tra cui si vuole effettuare la traduzione. Essendo molto oneroso produrre un corpus simile appositamente si ricorre di solito a sorgenti preesistenti, come gli *atti della Commissione Europea*, che sono pubblici e disponibili nelle lingue ufficiali dell'Unione.

Per ottenere dalla rete prestazioni elevate è necessario che i dati annotati che compongono il corpus siano il più possibile attinenti al dominio in cui la rete verrà impiegata, ed è evidente che questo limita notevolmente la disponibilità di dati provenienti da sorgenti esistenti.

Negli ultimi anni si è andata ad affermare la tecnica del *transfer learning*, ovvero del *trasferimento dell'apprendimento*. Questa tecnica si basa sull'ipotesi che in una rete i primi livelli siano sostanzialmente invarianti al variare del compito della rete. Il training viene quindi effettuato partendo da una rete che non viene inizializzata con parametri casuali ma che è già stata addestrata su di un corpus esteso e il più possibile generico. Ad es. nel caso delle immagini, si parte solitamente da modelli pre-addestrati su **ImageNet**.

Il training viene effettuato facendo variare in una prima fase solo i parametri relativi agli ultimi layer che implementano il classificatore vero e proprio, lasciando fissi quelli relativi alla parte convoluzionale, che descriveremo più sotto. Raggiunto un certo grado di stabilità si procede poi ad addestrare anche gli altri layer con un learning rate molto basso. In questo modo si riesce ad addestrare una rete ottimizzata su di un particolare dominio con un numero di iterazioni (epoche) molto basso, utilizzando un data set di diversi ordini di grandezza più ridotto rispetto a quello che sarebbe necessario partendo da una rete inizializzata in modo casuale.

Il problema più grande da affrontare quando si compila un data set di addestramento (ma che risulta evidente solo al momento dell'utilizzo della rete addestrata) è quello del cosiddetto *bias* [23], ovvero *pregiudizio*. Se la distribuzione dei dati nel training set non copre uniformemente il dominio in cui la rete si troverà ad operare, c'è il rischio concreto che la rete risulti affetta da bias, cioè che in presenza di dati afferenti a regioni della distribuzione reale non adeguatamente rappresentati nel data set di addestramento la rete si comporti in modo aprioristico con risultati poco attinenti ai dati puntuali di ingresso. È osservabile che se i dati di ingresso sono affetti da qualche pregiudizio la rete non solo lo emulerà ma tenderà ad amplificarlo. Questo pone molti problemi etici nell'impiego di queste tecnologie in campi che possono avere conseguenze rispetto ai diritti degli individui.

Architetture principali

In questa sezione verranno descritte alcune tra le principali architetture di DNN, comunemente utilizzate sia in ambiente accademico che industriale. Dato lo scopo introduttivo dell'articolo, si illustrerà solo il principio di funzionamento di massima delle reti.

Multilayer perceptron (dense network)

Questa architettura, già descritta sommariamente in precedenza, è conosciuta con diversi nomi tra cui *dense network* e *fully connected network*. È stata la prima architettura proposta di rete neurale ed è una rete di tipo *feed forward*, ovvero in cui il segnale fluisce dagli ingressi verso le uscite senza mai creare recursioni. Si caratterizza per il fatto che ciascun neurone di un layer è connesso a tutti i neuroni del layer successivo. Il numero di neuroni del layer di ingresso è pari alla dimensione del vettore di dati di ingresso, mentre i layer interni possono avere dimensioni diverse a seconda del tipo di utilizzo della rete.

Questo schema viene spesso usato come classificatore. In tal caso il numero di celle nel layer di uscita è uguale al numero di classi tra cui si vuole discriminare. La funzione di attivazione di questo layer è normalmente *softmax* se le scelte sono esclusive oppure *sigmoide* nel caso più classi possano essere valide contemporaneamente. Nel primo caso quindi, il valore di ciascun elemento è associabile alla probabilità che l'insieme dei dati di ingresso appartenga alla classe corrispondente mentre nel secondo caso è assimilabile al grado di appartenenza a ciascuna delle classi. Il numero di layer interni e la loro ampiezza dipende in modo empirico dalla complessità del problema che si vuole risolvere. La funzione di attivazione di questi layer è solitamente *ReLU* oppure *tanh* (si veda la Fig. 2).

Reti convoluzionali

Le reti di tipo *multilayer preceptron* sono computazionalmente troppo pesanti per essere impiegate nell'elaborazione di immagini, in quanto sarebbe necessario prevedere un neurone per ogni pixel dell'immagine. Poiché ciascun neurone viene connesso a tutti i neuroni del layer successivo, il numero di parametri da stimare sarebbe eccessivo per i casi di interesse pratico. Si preferisce quindi utilizzare un'architettura basata su *reti convoluzionali*, che permettono di ridurre pesantemente il numero di parametri da stimare per una data profondità della rete.

L'elemento base è descritto nella Fig. 6 e consiste in una serie di *nuclei di convoluzione* (detti anche *filtri*), tipicamente di dimensione *3x3* o *5x5*, i cui coefficienti vengono appresi durante il training.

Nei casi più in uso, la rete si compone di un numero elevato (sino a qualche decina) di layer di questo tipo intercalati con layer che applicano una funzione non lineare di attivazione (solitamente la *ReLU*) e layer che applicano un sottocampionamento degli elementi del layer attuale per generare un'uscita verso il layer successivo di dimensione ridotta. L'operazione effettuata in questi layer è chiamata *max pooling* e consiste nel condensare un blocchetto *2x2* di dati in un unico valore corrispondente al valore massimo nel blocchetto. Alternando layer convoluzionali a layer di pooling si riduce progressivamente la dimensione dei dati in uscita, sino ad arrivare ad un vettore di dimensione indicativamente dell'ordine delle migliaia di elementi.

La rete in questa operazione impara ad estrarre dall'immagine di ingresso le caratteristiche salienti che utilizzerà per risolvere il problema su cui è stata addestrata (Fig. 7). Visualizzando la forma che assumono i filtri generati durante l'addestramento si nota che questi rappresentano strutture via via più complesse procedendo dall'ingresso verso l'uscita [24].

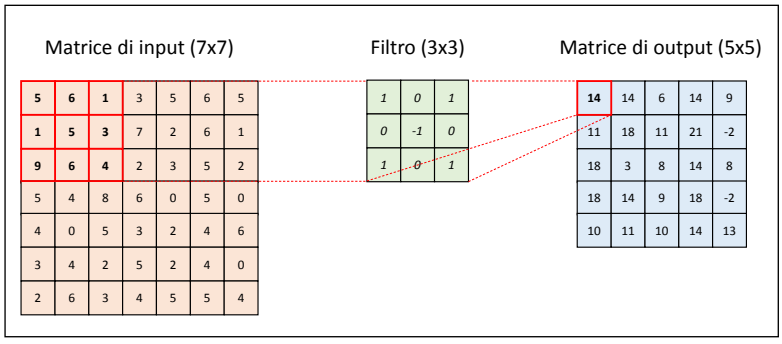


Fig. 6 – Operazione di *convoluzione bidimensionale*. Sull'immagine arancio viene utilizzato un filtro (verde) di dimensione 3x3. Il risultato è la matrice celeste di dimensione 5x5.

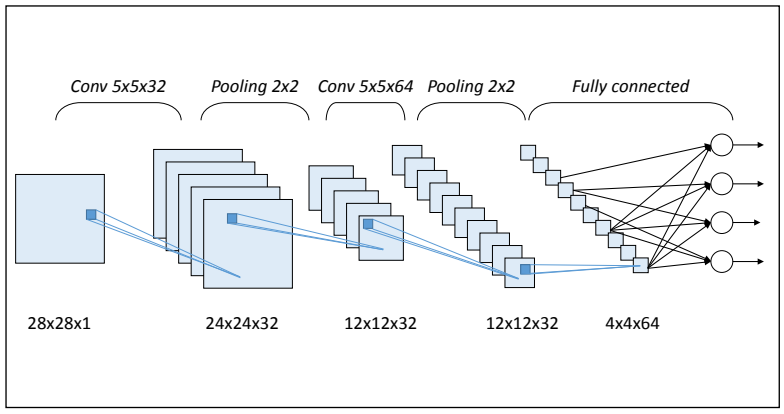


Fig. 7 – Esempio di rete convoluzionale

Se la rete viene impiegata come classificatore di immagini, il layer finale della rete convoluzionale viene connesso ad una *rete dense*, del tipo visto nel paragrafo precedente. Le reti **LeNet** e **AlexNet**, citate in precedenza, sono di questo tipo.

RESNET

Una importante limitazione dell'architettura di rete convoluzionale vista sopra è che è molto difficile addestrare reti con un numero di layer superiore a qualche decina per problemi di annullamento del gradiente già citati.

L'idea alla base della **RESNET** [25] consiste nell'inserire un cortocircuito tra coppie di layer convoluzionali adiacenti, come mostrato in Fig. 8. Di fatto a questo modo viene propagato il residuo tra il segnale di ingresso ed una sua elaborazione effettuata dai layer convoluzionali. Questo schema risulta più stabile e meno soggetto a problemi di regolarizzazione e permette di addestrare agevolmente reti con centinaia di layer. Le reti **RESNET34** e **RESNET50** sono due esempi di configurazioni di **RESNET** spesso utilizzate nei casi pratici, in quanto rappresentano un buon compromesso tra performance e complessità.

L'architettura **RESNET** ha ispirato la realizzazione di diverse altre reti tra cui la **EfficientNet** [26], che allo stato attuale rappresenta tra le reti convoluzionali quella maggiormente efficiente, ovvero quella che riesce ad ottenere risultati allo stato dell'arte nei principali benchmark scientifici con una complessità di calcolo sensibilmente inferiore rispetto alle architetture precedenti.

Reti recursive

Le *reti recursive*, **RNN** (*Recurrent Neural Network*), sono indicate per processare serie ordinate di dati, in cui l'ordine di apparizione dei simboli è importante, come ad es. una sequenza audio oppure dei testi.

La rete ha una struttura molto semplice, basata su di una cella recursiva dove l'uscita di ciascun layer viene rimessa in ingresso al layer stesso. Questa struttura può essere sviluppata come una serie lineare di celle uguali (con gli stessi parametri) che rappresentano lo stato della rete al tempo *t*, *t+1*, *t+2* e così via (si veda l'esempio di Fig. 9).

Nelle reti recursive viene normalmente usata come funzione di attivazione la *tanh* in quanto essendo limitata tra *-1* e *+1*, riduce il problema dell'esplosione del gradiente durante il training. È possibile comporre reti a più layer dove l'uscita di una cella viene posta in ingresso ad un'altra cella dello stesso tipo ma con pesi diversi, così come è possibile creare reti bidirezionali affiancando due celle che vengono alimentate con i dati in ordine inverso l'una rispetto all'altra. L'uscita in questo caso sarà calcolata utilizzando gli stati nascosti (h_t) di entrambe le celle.

Fig. 8 – Elemento base di una architettura RESNET

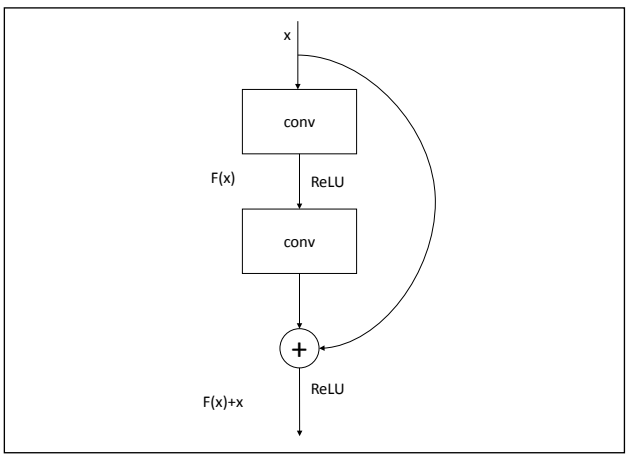
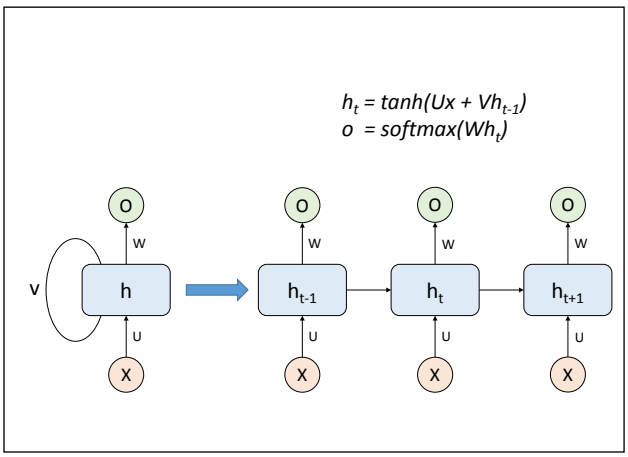


Fig. 9 – Rete Neurale Recursiva



In realtà viene spesso utilizzata una versione modificata della rete descritta qui, chiamata ***Long Short Term Memory*** o ***LSTM*** [27], che risolve alcuni problemi tipici di questa architettura ricorsiva tra cui la difficoltà ad apprendere relazioni tra dati di ingresso che compaiono distanziati nel tempo e il rischio di annullamento del gradiente durante il training dovuto al fatto che la rete, per effetto della recursione, risulta essere molto profonda.

Transformer

Le reti recursive ricevono i dati di ingresso sequenzialmente seguendo l'ordine temporale di arrivo. Per questa ragione le dipendenze tra dati distanti nel tempo risultano difficili da modellare. L'architettura *Transformer* è stata concepita proprio per affrontare questo problema [28]. In questo caso non viene utilizzato uno schema recursivo bensì i dati entrano a pacchetti in una rete di tipo feed forward. Il blocco principale, preposto all'analisi della interdipendenza dei dati, è detto *Attention* ed è costituito da un layer di celle di tipo *dense* che hanno il compito di confrontare ogni dato di ingresso con tutti gli altri del pacchetto e di generare un valore che indica la forza della connessione tra i dati in questione.

Più blocchi di tipo *Attention* possono essere inseriti in parallelo per modellare diversi tipi di dipendenze (*Multi-Head Attention*). L'architettura del Transformer contiene un encoder e un decoder, in quanto viene normalmente utilizzata per risolvere task dove i dati di uscita hanno una struttura simile a quelli di ingresso, ad es. traduzione di un testo tra due lingue o sommarizzazione di un testo, ma è possibile utilizzare solo l'encoder per task che richiedono una risposta globale come la *sentiment analysis* o la *classificazione*. L'encoder si basa sulla sovrapposizione di due blocchi, la *Multi-Head Attention* seguita da un blocco *dense* (chiamato *Feed Forward* in Fig. 10). Si notino in Fig. 10 le connessioni di bypass in ogni blocco analogamente a quanto avviene nelle **RESNET**. Più elementi di questo tipo possono essere sovrapposti.

Il decoder è sostanzialmente simile all'encoder, si differenzia per l'aggiunta di un secondo blocco di *Multi-Head Attention* che permette di utilizzare anche i dati di uscita elaborati dalla rete sino al momento attuale. A differenza dell'encoder che processa i dati in un unico passo, il decoder deve effettuare tante iterazioni quanti sono i dati presenti nel pacchetto dove and ogni iterazione i dati

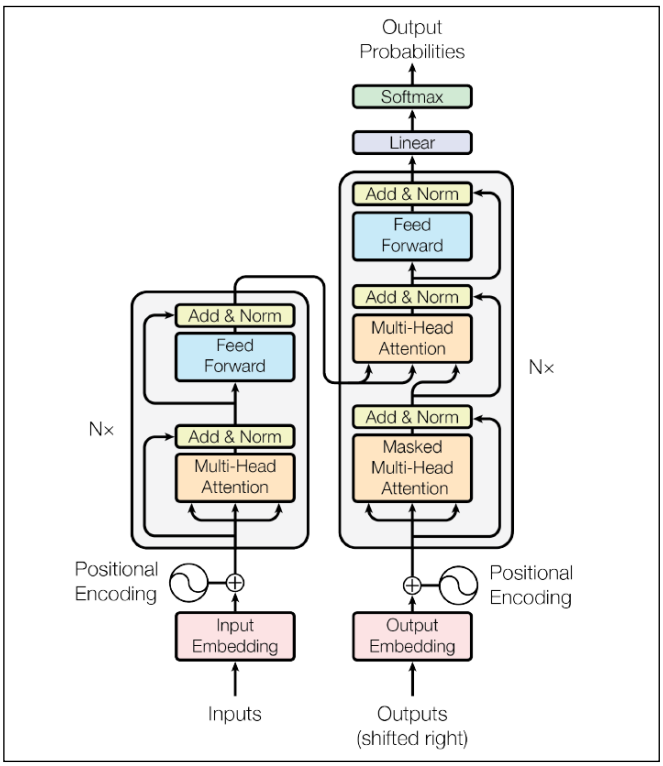


Fig. 10 – Elemento base (encoder e decoder) di una rete Transformer. Tratto da [28]

di uscita generati vengono riportati all'ingresso del decoder scalati di una posizione sino al completamento della sequenza.

Con l'architettura Transformer sono stati migliorati praticamente tutti i benchmark scientifici standard nel campo del *Natural Language Processing*[Nota 2], al costo però di un significativo aumento delle risorse di calcolo richieste rispetto alle soluzioni precedenti. Le reti Transformer più complesse al momento della stesura del presente articolo necessitano dell'apprendimento di un numero di pesi dell'ordine di 10^{11}, non molto distante dal numero di connessioni del cervello umano che è di circa 10^{14}.

Generative Adversarial Network (GAN)

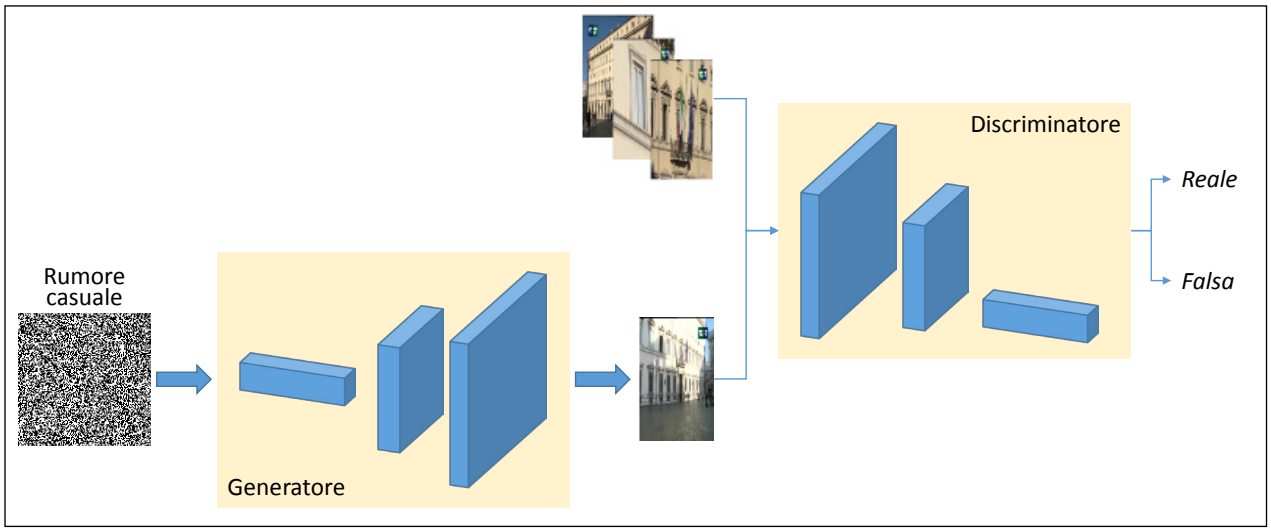
Concludiamo questa breve introduzione alle architetture DNN più diffuse citando le *Generative Adversarial Network*, meglio conosciute con l'acronimo **GAN**. Queste architetture, di cui ne esistono diverse varianti, hanno lo scopo di generare dati originali (cioè non osservati tra i dati di input) ma verosimili.

Durante il training la rete viene condizionata a produrre dati (ad esempio immagini) appartenenti ad una data categoria (ad es. contenenti determinati oggetti) di cui vengono forniti numerosi esempi.

Tecnicamente le **GAN** sono costituite da due reti distinte e speculari che vengono addestrate contestualmente. La prima funge da generatore, la seconda invece ha il compito di discriminare se l'immagine prodotta dal generatore è reale o falsa. La rete discriminante è solitamente di tipo convoluzionale (ad es. una **RESNET**) e viene addestrata a riconoscere le immagini reali contenute nel training set da immagini sintetiche prodotte dalla rete generatrice, la quale viene invece addestrata a produrre immagini che superino il controllo della rete discriminante.

L'architettura della rete generatrice è sostanzialmente inversa a quella di una rete convoluzionale: a partire da un vettore di dati arbitrario, tramite operazioni di convoluzione ed interpolazione si genera la struttura di un'immagine.

Fig. 11 – Generative Adversarial Network



Nota 2 - Il *Natural Language Processing* (NLP) è una branca dell'intelligenza artificiale che si occupa del trattamento automatico di informazioni scritte o parlate espresse in linguaggio naturale. Tra i problemi più comuni affrontati citiamo la trascrizione del parlato in testo, la comprensione di testi scritti, l'analisi grammaticale/sintattica/semantica di testi e la generazione automatica di testi.

L'addestramento avviene utilizzando una loss function di questo tipo [29]:

$$\text{Discriminator Loss} = \text{Max}(D(x) - D(G(z)))$$
$$\text{Generator Loss} = \text{Max}(D(G(z)))$$

Il *discriminatore* cerca di massimizzare la differenza tra la sua uscita in presenza di immagini reali *x* e quella in presenza di immagini generate *G(z)*, mentre il *generatore* cerca di ingannare il discriminatore cercando di generare immagini che ne massimizzino l'uscita. L'addestramento procede a fasi alterne in cui si addestra un componente per volta mantenendo costante l'altro.

Nel caso più semplice di utilizzo, la rete generatrice viene alimentata con un vettore di dati casuali e l'immagine generata viene valutata dalla rete discriminante. Si procede a generare immagini variando il vettore di ingresso sino a che un'immagine generata non ottiene un punteggio sufficientemente alto dalla rete discriminante. In altre varianti di implementazione è invece possibile fornire in ingresso alla rete generatrice un'immagine ed ottenerne in uscita una versione modificata. In questo caso il vettore di ingresso alla rete generatrice non sarà più una sequenza casuale ma l'uscita di una rete convoluzionale applicata all'immagine di ingresso. È possibile a questo modo realizzare reti che effettuano trasformazioni interessanti delle immagini di ingresso, ad es. colorazione di immagini monocromatiche, aumento della risoluzione o rimozione di disturbi.

Conclusioni

In questa trattazione ci si è posti lo scopo di fornire al lettore le nozioni di base per comprendere il funzionamento delle reti neurali profonde, elencando le principali tecniche di ottimizzazione del processo di apprendimento e le principali architetture utilizzate in alcuni dei campi di applicazione più comuni e promettenti. Nei prossimi articoli verranno illustrate alcune applicazioni di particolare interesse per il settore radiotelevisivo e multimediale che, come vedremo, possono ricevere un nuovo impulso da questa promettente tecnologia.

Bibliografia

[1] M. Turing, *Computing Machinery and Intelligence*, in "MIND", vol. LIX, n. 236, 1950, pp. 433-460, **DOI**: 10.1093/mind/LIX.236.433

[2] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, 1961

[3] D. E. Rumelhart, G. E. Hinton e R. J. Williams, *Learning Internal Representations by Error Propagation*, in D. E. Rumelhart e J. L. McClelland (ed.), "Parallel Distributed Processing : Explorations in the Microstructure of Cognition, vol. 1: Foundations", MIT Press,1986, pp. 318-362, https://ieeexplore.ieee.org/servlet/opac?bknumber=6276825

[4] K. Fukushima, *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*, in "Biological Cybernetics", vol. 36, 1980, pp. 193-202, **DOI**: 10.1007/BF00344251

[5] D. H. Hubel e T. N. Wiesel, *Receptive fields of single neurones in the cat's striate cortex*, in "The Journal of Physiology", vol. 148, n. 3, 1959, pp. 574-591, **DOI**: 10.1113/jphysiol.1959.sp006308

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, *ImageNet Classification with Deep Convolutional Networks*, in "Advances in Neural Information Processing Systems 25 (NIPS 2012)", 2012, pp. 1097-1105, https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[7] Y. Lecun ed altri, *Gradient Based Learning Applied to Document Recognition*, in "Proceedings of the IEEE", vol. 86, n. 11, 1998, pp. 2278-2324, **DOI**: 10.1109/5.726791

[8] *CUDA Toolkit home page*, NVIDIA. Developer (web), https://developer.nvidia.com/cuda-toolkit (ultimo accesso 27/10/2020)

[9] *State of AI Report 2020*, stateof.ai (web), https://www.stateof.ai/ (ultimo accesso 27/10/2020)

[10] *TensorFlow home page*, TensorFlow (web), https://www.tensorflow.org/ (ultimo accesso 27/10/2020)

[11] PyTorch home page, PyTorch (web), https://pytorch.org/ (ultimo accesso 27/10/2020)

[12] *The latest in Machine Learning*, paperswithcode.com (web), https://paperswithcode.com/ (ultimo accesso 27/10/2020)

[13] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton*, Cornell Aeronautical Laboratory, Report n. 85-460-1, 1957

[14] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, in "Mathematics of Control, Signals and Systems", vol. 2, n. 4, 1989, pp. 303-314, **DOI**: 10.1007/BF02551274

[15] M. Leshno ed altri, *Original Contribution: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, in "Neural Networks", vol. 6, n. 6, 1993, pp. 861-867, **DOI**: 10.1016/S0893-6080(05)80131-5

[16] H. Zhang, *The Optimality of Naive Bayes*, in "Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference", 2004

[17] C. Cortes e V. Vapnik, *Support-vector networks*, in "Machine Learning ", vol. 20, n. 3, 1995, pp. 273–297, **DOI**: 10.1007/BF00994018

[18] Robbins e S. Monro, *A Stochastic Approximation Method*, in "Annals of Mathematical Statistics", vol. 22, n. 3, 1951, pp. 400-407, https://projecteuclid.org/euclid.aoms/1177729586

[19] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint , 2017, arXiv:1609.04747v2

[20] N. Qian, *On the momentum term in gradient descent learning algorithms*, in "Neural networks", vol. 12, n. 1, 1999, pp. 141-151, **DOI**: 10.1016/S0893-6080(98)00116-6

[21] S. Ioffe e C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, in "Proceedings of the 32nd International Conference on Machine Learning", PMLR, vol. 37, 2015, pp. 448-456, http://proceedings.mlr.press/v37/ioffe15.html

[22] *About ImageNet*, ImageNet (web), http://imagenet.stanford.edu/about-overview (ultimo accesso 27/10/2020)

[23] P. Krishnamurthy, *Understanding data bias*, in "towards data science" (web), 2019, https://towardsdatascience.com/survey-d4f168791e57 (ultimo accesso 27/10/2020)

[24] A. Nguyen, J. Yosinski e J. Clune, *Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks*. in ICML Visualization Workshop, 2016, https://icmlviz.github.io/icmlviz2016/assets/papers/5.pdf

[25] Kaiming He, *Deep Residual Learning for Image Recognition*, in "2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", 2016, **DOI**: 10.1109/CVPR.2016.90

[26] Mingxing Tan e Quoc V. L, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, in "Proceedings of the 36th International Conference on Machine Learning", PLMR, vol. 97, 2019, pp. 6105-6114, http://proceedings.mlr.press/v97/tan19a.html

[27] S. Hochreiter e J. Schmidhuber, *Long Short-term Memory*, in "Neural Computation", vol. 9, n. 8, 1997, pp. 1735-1780, **DOI**: 10.1162/neco.1997.9.8.1735

[28] A. Vaswani ed altri, *Attention Is All You Need*, in "Advances in Neural Information Processing Systems 30 (NIPS 2017)", 2017, pp. 5998-6008, https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[29] M. Arjovsky, S. Chintala, L. Bottou, *Wasserstein Generative Adversial Networks*, in "Proceedings of the 34th International Conference on Machine Learning", PMLR, vol. 70, 2017, pp. 214-223, http://proceedings.mlr.press/v70/arjovsky17a.html