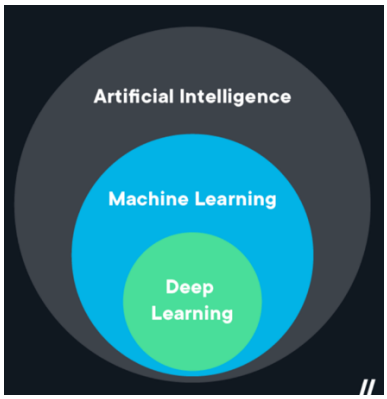
PRIME DEFINIZIONI

L'intelligenza artificiale è la scienza che mira a creare macchine che pensano e agiscono come gli esseri umani.

Il machine learning è una sottocategoria dell'AI che è incentrata sulla costruzione degli algoritmi e modelli che permettono al computer di "imparare" da dati a fare predizioni (supervised learning / unsupervised learning).

Il deep learning è un sottoinsieme del machine learning che sfrutta le reti neurali, con struttura simile al sistema neurale umano, per analizzare dati(CNNs, RNNs, GANs).



→ *SUDDIVISIONE*

STORIA

I primi calcolatori nascono negli anni 40, i relè e le valvole termoioniche. Teoria della computazione e test Turing → si chiede se le macchine possano pensare. Nel 1943 viene idealizzato il primo modello di neurone artificiale. A fine anni 60 / inizio anni 70 c'è un grande entusiasmo che porta però a predizioni fin troppo ottimistiche. Nel successivo periodo 74-80 i finanziamenti calano drasticamente per via delle aspettative alte, i problemi principali erano la scarsa capacità computazionale e dai dataset piccoli. Una volta terminato il primo inverno, nel periodo che arriva fino al 1987, avviene lo sviluppo dei sistemi esperti e dei primi algoritmi per le reti neurali (Backpropagation 1986) che coincide con la Prima Primavera, i principali finanziamenti sono offerti dal Giappone per la Quinta generazione dei calcolatori. Questa porterà al Secondo Inverno data l'insoddisfazione delle aspettative(le reti neurali non riescono a scalare a problemi di complessità maggiore), per cui di nuovo tornano a mancare i finanziamenti.

Dal 1993 gli hardware iniziano a diventare sempre più potenti grazie anche alla separazione del calcolatore con l'invenzione della prima GPU da parte di NVIDIA(1999). La raccolta di Big Data molto più semplice e ne consegue che l'allenamento delle reti neurali segue lo stesso trend. Il deep learning ha grande successo in vari campi.

Confronto tra i paradigmi programmazione tradizionale / machine learning

Programmazione Tradizionale: le regole sono definite esplicitamente dal programmatore. Diventa molto complessa la gestione di problemi con molte variabili. Richiede meno dati e si basa sulle regole definite. Cambiare o aggiornare il comportamento del programma richiede modifiche al codice che possono essere più o meno dispendiose.

Machine Learning: le regole (modello) sono apprese automaticamente dai dati. Più adatta a problemi complessi e ad alta dimensione, dove è difficile definire regole esplicite. Richiede grandi quantità di dati per addestrare modelli accurati. Il modello può essere riaddestrato con nuovi dati per adattarsi a cambiamenti nei pattern dei dati.

Preparazione dei dati

Una volta acquisiti i dati è necessaria la fase di annotazione in cui l'etichetta del dato indica l'effettivo contenuto del dato. Se ho un dataset di dati non annotati allora il learning si dice non supervisionato: l'algoritmo deve imparare a riconoscere i pattern e a etichettare i dati correttamente.

Una volta acquisiti i dati sono da separare in 3 macroset:

-Training set, è l'insieme di dati che il modello sfrutta per allenarsi a riconoscere i pattern e quindi a predire quale sarà il risultato di un dato sconosciuto.

-Validation set, è un sottoinsieme del Training set, su questo insieme vengono messi a punto gli iperparametri, che sono parametri non apprendibili dai dati che quindi necessitano di essere impostati a parte(es. learning rate, numero di neuroni per strato della rete neurale, profondità massima di una foresta decisionale).

-Testing set, l'insieme di dati su cui si effettua il test della rete(si individua la % di successo).

TASK

In machine learning ci sono 3 task principali a seconda dell'output desiderato: Classificazione, Regressione, Clustering.

Classificazione

Una classe è un insieme di dati con proprietà comuni, è un concetto legato all'etichetta. Dato un input l'algoritmo deve restituire la classe corrispondente. Se la classificazione è tra due categorie allora è un problema di classificazione binaria(0 o 1), altrimenti è detta classificazione multiclasse.

Regressione

Dato l'input, l'output desiderato è una funzione, quindi l'obiettivo è trovare la relazione tra una variabile indipendente e una o più variabili dipendenti. Da un set di valori discreti si vuole ottenere un valore continuo(es. dato peso e altezza si vuole ottenere una funzione che descriva il peso di una persona con altezza non presente nel training set).

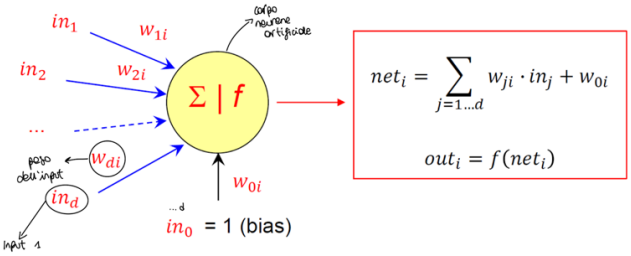
Clustering

Dato l'input voglio ottenere dei raggruppamenti dei dati stessi. Solitamente il set per questo tipo di task è NON etichettato per cui ci si trova in un apprendimento non supervisionato; quindi è mediamente più complesso rispetto agli altri task. Un esempio è dato un set di utenti di un e-commerce creare classi in base ai prodotti comprati dagli stessi per creare suggerimenti / adv personalizzate.

Neurone Biologico & Neurone Artificiale

Il neurone biologico è diviso in 4 parti: dendriti che ricevono l'input e li trasmettono al soma che li elabora per produrre un output, l'assone è la parte che si occupa di trasmettere l'output agli altri neuroni mentre la sinapsi è il collegamento effettivo neurone-neurone ed è la parte che può modificare la risposta e variare l'efficienza di trasporto.

Percettrone:



$in_1, in_2 \ldots in_d$ sono i d ingressi che arrivano ai dendriti → INPUT.

$w_{1i}, w_{2i} \ldots w_{di}$ sono i d pesi associati agli input, sono questi parametri che l'algoritmo modifica per determinare l'importanza di un dato input.

w_o anche detto BIAS è un peso associato ad un input fittizio che serve per tarare il lavoro del neurone.

$f(\cdot)$ è detta funzione di attivazione, funge da interruttore del neurone ovvero attiva il neurone solo se gli input superano una certa soglia determinata. Genericamente sono non lineari e derivabili, alcuni delle più classiche sono Sigmoid, ReLU, Leaky ReLU, Tanh, Maxout, ELU.

Allineamenti di più strati ognuno composto da più neuroni artificiali costituiscono i MultiLayer Perceptron o MLP. I singoli strati sono classificati come input layer, hidden layer, output layer. Una rete neurale è sempre costituita da un solo layer di input, un solo layer di output e uno o più hidden layer.

La prima classificazione delle reti neurali:

-FeedForward(FFNN) dove i collegamenti tra neuroni avvengono solamente ai livelli successivi, un neurone non può essere connesso a sé stesso o a neuroni dello stesso livello / livello precedente.

-Ricorrenti, i collegamenti possono essere ricorsivi sullo stesso neurone o verso neuroni appartenenti allo stesso livello, vengono dette connessioni di feedback. La rete ha una sorta di memoria che condiziona gli output successivi.

Il training di una rete consiste nella ripetizione della predizione sulla base di alcuni input in cui il ruolo chiave lo svolge la funzione di loss che indica all'algoritmo la distanza dal risultato corretto. Per capire il cambiamento dell'errore si studia la derivata della funzione di loss quindi l'andamento e l'obiettivo è quello di minimizzare questa funzione(errore tendente a 0).

Concretamente, una volta fissati il numero di neuroni per livello e il numero dei livelli stesso, bisogna aggiustare i pesi(e il bias) in modo che diano come risultato il mapping desiderato. Per trovare i minimi della loss function si usa il metodo della discesa del gradiente ad ogni iterazione della rete. Il problema più complesso è costituito dai minimi locali che danno l'apparenza di aver trovato un massimo assoluto → il valore della loss function sale in ogni direzione.

Il metodo della discesa del gradiente è basato sull'algoritmo di backpropagation che ha 4 step fondamentali:

1) **Passaggio in Avanti**:

   I dati di input vengono propagati attraverso la rete, strato per strato. Per ogni neurone, si calcola l'attivazione utilizzando una funzione di attivazione.

2) **Calcolo della Perdita**:

   Si calcola la funzione di perdita confrontando la previsione con il valore reale.

3) **Passaggio all'Indietro**:

   Si calcolano i gradienti della funzione di perdita rispetto a ciascun peso nella rete utilizzando la regola della catena del calcolo differenziale. Il gradiente indica come modificare ciascun peso per ridurre la perdita.

4) **Aggiornamento dei Pesi**:

   I pesi vengono aggiornati utilizzando il metodo della discesa del gradiente.

Loss function nel caso della classificazione binaria: Binary Cross Entropy.
$BCE = -y_i \cdot \log \hat{y}_i - (1 - y_i) \cdot \log(1 - \hat{y}_i)$

Loss function nel caso della classificazione multiclasse: Categorical Cross Entropy.
$CCE = -\sum_i y_i \cdot \log \hat{y}_i$

CONVUTIONAL NEURAL NETWORK

Il principale scopo per cui sono state disegnate le CNN è il processo delle immagini (anche perché gli MLP non hanno alcuna capcità di individuare elementi dopo semplici operazionii come una traslazione). L'idea di base è avere una struttura tridimensionale quindi con Larghezzza, Lunghezza e una Profondità variabile (1 = bianco e nero, 3 = RGB). La dimensione di un neurone si dice **Kernel** ed è 3x3x1. Per adattare il kernel all'immagine si effettua un'operazione detta convoluzione (meccanismo sliding-window), consiste nello scorrimento del kernel (anche detto filtro) su ogni pixel dell'immagine e per ogni posizione calcolare il prodotto scalare tra la maschera e la prozione dellp'input coperta.
L'output è detto **features-map**.

I layer più vicini al layer di input estraggono informazioni "semplici" e mano mano che ci si avvicina al layer di output l'informazione estratta si complica(ad esempio in un'immagine con un viso il layer di output sarà il viso stesso, o nei layer intermedi porzioni come occhi, orecchie ecc..., nei layer primari saranno semplici raggruppamenti di pixel appartenenti all'immagine).

Una CNN tradizionale è composta da più layer sequenziali:

-Pooling layer, è uno dei layer chiave che riduce la dimensione del volume in input attraverso funzioni come una media(average), max(most used) ecc... Chiaramente sono utilizzate principalmente per ridurre il costo computazionale e per evitare l'overfitting.

-Layer di attivazione, svolge una funzione similare alla funzione di attivazione, esempio RELU.

-Flatten layer, appiattisce la feature map ad un vettore unidimensionale (es. semplice srotolamento), connette il feature extractor con il classifier (che rimane un MLP fully-connected, solitamente FFNN) che risolve il task di classificazione multiclasse o binaria.

Sostanzialmente una CNN è formata da una parte convoluzionale (pooling), una parte fully-connected(MLP) e un connettore tra le due parti(flattten).

Algoritmo BACKPROPAGATION

L'algoritmo di backpropagation è uno dei più diffusi per l'addestramento di una ANN(rete neurale artificiale), è legato alla tecnica del calcolo delle derivate della funzione errore che è basata sulla derivazione delle funzioni composte e la sua funzione principale è l'aggiornamento dei pesi all'indietro.

Ricavare la formula di aggiornamento dei pesi con 1 input 3 hidden e 1 output:

Loss function: $L(y_1, \hat{y}_1)$ → misura l'errore tra l'output ottenuto e quello atteso.

Obiettivo trovare i pesi che minimizzano L → $w \in argmin(L(y_1\hat{y}_1(w)))$, per farlo è necessario utilizzare il metodo della discesa del gradiente ricordando che:

$$w^{k+1} = w^k - \eta\nabla L(w^k)$$ → il peso calcolato al passo successivo è uguale al peso attuale – il gradiente della loss function (calcolata nel peso attuale) moltiplicato per il learning rate

Nel nostro caso, il gradiente della loss function è: $\nabla L = \frac{\partial L}{\partial w_{11}^{(1)}}, \frac{\partial L}{\partial w_{11}^{(2)}}, \frac{\partial L}{\partial w_{11}^{(3)}}$

Bisogna quindi calcolare le 3 derivate parziali della loss function rispetto ai 3 pesi,

CALCOLI

Otteniamo → $\frac{\partial L}{\partial w_{11}^{(3)}} = \boldsymbol{\delta_1^{(3)} z_1^{(2)}}$

$\frac{\partial L}{\partial w_{11}^{(2)}} = \boldsymbol{\delta_1^{(2)} z_1^{(1)}}$

$\frac{\partial L}{\partial w_{11}^{(1)}} = \boldsymbol{\delta_1^{(1)} z_1^{(0)}}$

L'aggiornamento dei pesi per epoche successive dovrà quindi essere:

- $w_{11}^{(3)} = w_{11}^{(3)} - \eta\delta_1^{(3)} z_1^{(2)}$
- $w_{11}^{(2)} = w_{11}^{(2)} - \eta\delta_1^{(2)} z_1^{(1)}$
- $w_{11}^{(1)} = w_{11}^{(1)} - \eta\delta_1^{(1)} x_1$

Differenza tra Cost function e Loss function: la Loss function misura l'errore dall'output atteso per ogni singolo output di ogni neurone, la Cost function misura l'errore sull'intero dataset, è come se aggregasse tutte le loss function su tutto il dataset.

TECNICHE DI OTTIMIZZAZIONE

Batch Gradient Descent

Il calcolo della funzione costo passa per una forward propagation dell'intero training set per poi aggiornare i parametri tramite una backpropagation → i dati vengono aggiornati una singola volta

Discesa del gradiente stocastico (SGD)

Si utilizza una singola osservazione per aggiornare i parametri, supponendo di avere un dataset composto da 3 input e 5 campioni la prima osservazione farà passare il primo campione nella rete per poi effettuare la backpropagation per aggiornare i parametri, questi passaggi vengono ripetuti per ogni campione per cui si hanno N iterazioni, dove N è il numero di campioni e il concetto di "iterazione" si riferisce strettamente all'aggiornamento dei parametri della ANN.

Mini-Batch Gradient Descent

Si considera un sottoinsieme del dataset, se nel complesso ci sono N osservazioni il mini-batch sarà composto da un numero di osservazioni $1 < x < N$. → L'osservazione indica quanti campioni fai scorrere nella rete ad ogni iterazione (se il mini-batch è = 2 allora scorrerai 2 campioni alla volta, se per esempio il dataset ha 7 campioni dopo 3 iterazioni se ne eseguirà un'ultima sul campione rimanente).

Teorema che risponde alla domanda di convergenza in un punto stazionario

Il metodo di discesa del gradiente con passo finito converge se la funzione costo(C) è continua e differenziabile con gradiente continuo Lipschitz con costante L, considerando poi l'iterato successivo come $x^{k+1} = x^k - \eta \nabla C(x^k)$ se $\eta = {}^{2}/_{L}$ allora C converge a x^* dove $\nabla C(x^*) = 0$. Aggiungendo l'ipotesi che la funzione costo sia convessa riconosciamo che il punto stazionario individuato dal metodo di discesa del gradiente coincide con il massimo globale.

Non convessità della funzione costo

La funzione costo priva di hidden layer è convessa, aggiungendo degli hidden layer e quindi introducendo la non linearità nei problemi la funzione costo diventa altamente non convessa per cui è facile imbattersi in quelli che sono punti di sella o minimi locali che impediscono all'algoritmo di proseguire la ricerca del massimo globale, che ricordiamo essere il valore dei pesi che minimizza la loss function ovvero l'errore dall'output atteso.

Learning rate: il LR è uno degli iperparametri fondamentali per l'addestramento di una ANN, valori troppo alti di questo possono far si che sia impossibile per l'algoritmo giungere al minimo globale, mentre valori bassi richiedono un numero di passi estremamente elevato prima di raggiungere la soluzione ottimale. Nella ricerca del minimo globale è cruciale la scelta del learning rate e la soluzione ottimale è sceglierne uno dinamico che nelle fasi iniziali di ricerca sia medio-alto mentre nelle fasi finali sia basso.

Momentum

Il metodo della discesa del gradiente con momentum è stato studiato per ovviare al problema delle oscillazioni nell'aggiornamento dei parametri. L'introduzione del momentum accelera la convergenza e aiuta l'algoritmo a ridurre le oscillazioni.

Formula di aggiornamento dei pesi: $w_{k+1} = w_k + \eta v_k$ → dove $v_k = \beta v_{k-1} + \nabla C(w_k)$

β prende il nome di momentum e ha un valore compreso tra 0 e 1 dove i valori ottimali stanno tra 0.8 e 0.9. Se il momentum lo si pone = 0 allora il metodo ritorna nel metodo di discesa del gradiente classico. Il termine "v" indica la velocità accumulata rispetto al passo precedente, serve per aumentare le dimensioni degli aggiornamenti quando i gradienti puntano nella stessa direzione (segno +) mentre le riduce se puntano in senso opposto (segno -).

Dati i problemi legati al LR troppo basso/alto sono state progettate delle tecniche di LR decay ovvero un decadimento progressivo del learning rate che puntano a cercare un equilibrio tra velocità e stabilità di convergenza.

- Step decay, riduce il LR per ogni epoca di un fattore δ: $\eta = \eta_0 \cdot \delta^{\left\lfloor \frac{n}{s} \right\rfloor}$
- Decadimento esponenziale, varia in base alle iterazioni seguendo la regola matematica $\eta = \eta \cdot e^{-kt}$
- Decadimento basato sul tempo, divide il learning rate iniziale in base al numero di iterazioni eseguite: $\eta = \frac{\eta_0}{1+k \cdot t}$

Learning rate adattivo

1. **Adagrad**, adatta il LR ai parametri eseguendo aggiornamenti grandi ai parametri poco frequenti e viceversa.
   Aggiornamento → $s_{k+1} = s_k + (\nabla C(w_k))^2$
   da cui
   $$w_{k+1} = w_k - \frac{\eta}{\sqrt{s_k + \epsilon}} \nabla C(w_k)$$

   Risolve uno dei problemi più ostici ovvero la regolazione manuale del LR, si adatta alle caratteristiche dei dati e del modello e offre robustezza oltre che efficienza ma l'accumulo dei gradienti quadrati può portare ad un learning rate infimo, senza contare che presenta una dipendenza dai parametri iniziali che quindi devono essere calibrati nel modo corretto.

2. **RMSProp**, è un algoritmo che deriva da Adagrad e ne migliora la parte di accumulo del gradiente con una media ponderata esponenziale dei gradienti al quadrato. Adatta il LR individualmente per ciascun parametro della rete. Rispetto a Adagrad è più stabile e ne consegue che ha una convergenza più affidabile ed è meno dipendente dai parametri iniziali. Ora il parametro da ponderare è γ. La media ponderata esponenziale può accumulare più errore soprattutto per gradienti rumorosi(con perturbazione).
   Aggiornamento → $s_{k+1} = \gamma s_k + (1-\gamma)(\nabla C(w_k))^2$
   da cui
   $$w_{k+1} = w_k - \frac{\eta}{\sqrt{s_k + \epsilon}} \nabla C(w_k)$$

3. **ADADELTA**, calcola l'accumulo allo stesso modo di RMSProp ma non richiede di impostare un learning rate iniziale dato che utilizza la quantità di cambiamento stessa come calibrazione per i cambiamenti successivi.
   Aggiornamento → $s_{k+1} = \gamma s_k + (1-\gamma)(\nabla C(w_k))^2$

   $$\breve{\nabla} C = \frac{\sqrt{\Delta w_{k-1} + \epsilon}}{\sqrt{s_k + \epsilon}} \nabla C(w_k)$$

   da cui

   $$w_{k+1} = w_k - \breve{\nabla} C(w_k)$$

   dove $\Delta w_k = \gamma \Delta w_{k-1} + (1-\gamma)(\breve{\nabla} C)^2$

4. **ADAM**, vuole unire i vantaggi di RMSProp e il metodo di discesa con momentum, utilizza la media ponderata esponenziale dei gradienti dei passi precedenti, per ottenere una stima del momento del gradiente:

   $$v_k = \beta_1 v_{k-1} + (1-\beta_1)\nabla C(w_k)$$

   e del momento secondo del gradiente:

   $$s_k = \beta_2 s_{k-1} + (1-\beta_2)(\nabla C(w_k))^2$$

   per $v_0 = 0\ e\ s_0 = 0$ i momenti sono sbilanciati per cui si usano le seguenti normalizzazioni:

   $$\hat{v}_k = \frac{v_k}{1-(\beta_2)^k} \qquad \hat{s}_k = \frac{s_k}{1-(\beta_2)^k}$$

   per cui l'aggiornamento diventa: $w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{s}_k + \epsilon}} \hat{v}_k$