UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in INGEGNERIA MECCANICA

Predizione di eventi di guasto su linee produttive tramite algoritmi di deep learning

Prediction of failure events on production lines through deep learning algorithms

Relatore: Chiar.mo

Prof. Ciarapica Filippo Emanuele

Tesi di Laurea di:

Brandolini Federico

A.A. 2019/2020

UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in INGEGNERIA MECCANICA

Predizione di eventi di guasto su linee produttive tramite algoritmi di deep learning

Prediction of failure events on production lines through deep learning algorithms

Relatore: Chiar.mo

Prof. Ciarapica Filippo Emanuele

Tesi di Laurea di:

Brandolini Federico

A.A. 2019/2020

Sommario

Introduzione

La crescente sensorizzazione dei macchinari industriali e la disponibilità di dati estratti in tempo reale, frutto della diffusione dei paradigmi dell'Industria 4.0, associati all'incremento delle capacità computazionali dei moderni PC, hanno favorito le attività di ricerca e sviluppo della manutenzione predittiva.

In particolare, la possibilità di raccogliere dati in maniera continua e strutturata, unita alla possibilità di analizzare set di dati di dimensioni sempre maggiori senza un eccessivo dispendio in termini di risorse e di tempo, sta spingendo le aziende a considerare politiche di tipo predittivo.

L'adozione di politiche di manutenzione predittiva si traduce in differenti benefici: è possibile sfruttare tutta la vita utile di un bene, viene effettuato l'intervento solo quando è necessario, è possibile programmare l'approvvigionamento dei ricambi e contestualmente pianificare gli interventi in accordo con quelle che sono le necessità legate ai piani produttivi.

Per l'analisi dei dati raccolti si sono diffuse negli ultimi anni diverse tecniche di intelligenza artificiale, in questa tesi verrà trattato nello specifico la tecnica del deep learning applicata ad una linea di produzione per la predizione di eventi di guasto.

Grazie al deep learning verranno create delle reti neurali che permetteranno di calcolare i seguenti parametri: "Oee", "Ftq" e "pezzi totali prodotti" partendo da una raccolta di dati della linea di produzione.

Nel presente lavoro viene esposta inizialmente una panoramica sul mondo del deep learning, si passerà poi a descrivere il linguaggio di programmazione usato per implementare tali tecnologie, successivamente sono riportati gli algoritmi con la descrizione di ognuno di essi ed infine i risultati ottenuti ed il confronto tra i valori ottenuti dalle previsioni degli algoritmi e i valori presenti nel set di dati fornito.

1. Deep learning

1.1. Introduzione al deep learning

Il deep learning è una tecnica di apprendimento automatico che insegna ai computer a svolgere un'attività naturale per l'uomo: imparare con l'esempio.
È una tecnologia fondamentale alla base delle automobili a guida autonoma, poiché consente loro di riconoscere un segnale di stop o di distinguere un pedone da un lampione, è il fattore chiave del controllo vocale in dispositivi quali telefoni cellulari, tablet, TV e altoparlanti vivavoce.
Questa tecnica sta ottenendo risultati che prima non sembravano possibili.
Nel deep learning, un modello computerizzato impara a svolgere attività di classificazione direttamente da immagini, testo o suoni.
I modelli di deep learning possono raggiungere una precisione allo stato dell'arte, superando talvolta le prestazioni ottenute dall'uomo. L'addestramento dei modelli viene eseguito utilizzando un grande set di dati etichettati e architetture di reti neurali contenenti più layer.
Il deep learning raggiunge una precisione di riconoscimento mai vista prima. Ciò consente all'elettronica di consumo di soddisfare le aspettative degli utenti ed è fondamentale per le applicazioni essenziali per la sicurezza, come le automobili a guida autonoma. Gli sviluppi recenti hanno consentito al deep learning di progredire a tal punto da superare l'uomo in alcune attività, come la classificazione degli oggetti nelle immagini.
La traduzione strettamente letterale di questo termine è apprendimento approfondito, ed è proprio questo il centro del suo significato, perché il deep learning, sottocategoria del Machine Learning, non fa altro che creare modelli di apprendimento su più livelli.
Scientificamente, è corretto definire l'azione del deep learning come l'apprendimento di dati che non sono forniti dall'uomo, ma sono appresi grazie all'utilizzo di algoritmi di calcolo statistico.
Questi algoritmi hanno uno scopo: comprendere il funzionamento del cervello umano e come riesca ad interpretare le immagini e linguaggio. L 'apprendimento così realizzato ha la forma di una piramide: i concetti più alti sono appresi a partire dai livelli più bassi.

Abbiamo visto come sia importante portare il calcolatore a fare esperienza su un quantitativo sempre maggiore di dati sensibili e come, fino a poco tempo fa, il tempo per ottenere tale addestramento fosse abbastanza elevato.

Il deep learning è stato teorizzato per la prima volta negli anni ‘80, ma è diventato utile soltanto di recente per due ragioni principali:

1. Il deep learning richiede una grande quantità di dati etichettati. Per esempio, per lo sviluppo delle automobili senza conducente sono necessari milioni di immagini e migliaia di ore di video.
2. Il deep learning richiede una notevole potenza elaborativa, grazie all’introduzione delle GPU, ovvero nuove unità che concorrono all’elaborazione dati, questo processo è diventato molto più snello, i team di sviluppo sono in grado di ridurre i tempi di addestramento per una rete di deep learning da diverse settimane a poche ore.

Il deep learning fa una cosa fondamentale: ci regala la rappresentazione dei dati, ma lo fa a livello gerarchico e soprattutto a livelli diversi tra loro, riuscendo a elaborarli e a trasformarli.

Questa trasformazione è stupefacente perché ci consente di assistere ad una macchina che riesce a classificare i dati in entrata (input) e quelli in uscita (output), evidenziando quelli importanti ai fini della risoluzione del problema e scartando quelli che non servono.

La rivoluzione apportata dal deep learning è tutta nella capacità, simile a quella umana, di elaborare i dati, le proprie conoscenze a livelli che non sono affatto lineari.

1.2. Differenze tra deep learning e machine learning

Il deep learning è una forma specifica di machine learning.

Un flusso di lavoro di machine learning inizia con l'estrazione manuale delle feature significative dalle immagini. Le feature vengono quindi utilizzate per creare un modello che categorizza gli oggetti nell'immagine. Con un flusso di lavoro di deep learning, le feature significative vengono estratte automaticamente dalle immagini. Inoltre, il deep learning esegue un "apprendimento end-to-end", in cui una rete apprende automaticamente come elaborare dati grezzi e svolgere un'attività, per esempio una classificazione.

Un'altra differenza fondamentale è che gli algoritmi di deep learning scalano con dati, mentre l'apprendimento superficiale utilizza la convergenza. Per apprendimento superficiale si intendono i metodi di machine learning che non consentono ulteriori sviluppi una volta raggiunto un certo livello di performance, quando si aggiungono esempi e dati di training alla rete.

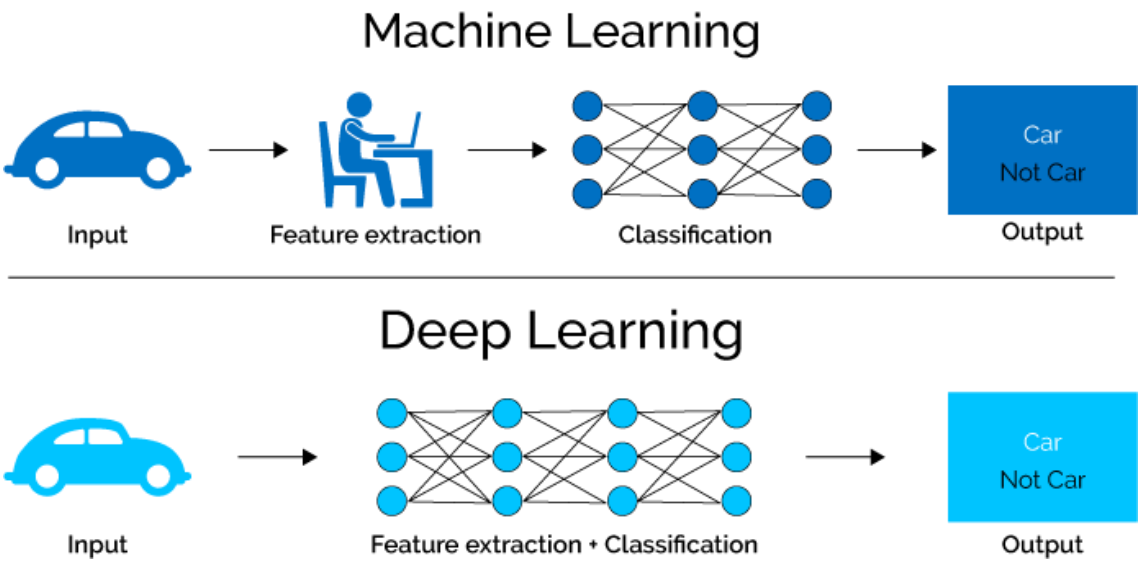
Un vantaggio fondamentale delle reti di deep learning è la possibilità di migliorare le prestazioni con l'aumentare del formato dei dati.

Il machine learning prevede la selezione manuale delle feature e di un classificatore per l'ordinamento delle immagini. Con il deep learning, l'estrazione delle feature e la procedura di modellazione sono automatiche.

Il machine learning consente di scegliere tra una varietà di tecniche e modelli in base all'applicazione utilizzata, al formato dei dati da elaborare e al tipo di problema che si desidera risolvere.

Un'applicazione di deep learning efficace richiede una grande quantità di dati (migliaia di immagini) per addestrare il modello e le GPU (Graphics Processing Units) per elaborare rapidamente i dati.

Al momento di scegliere tra machine learning e deep learning occorre considerare se di dispone di una GPU ad alte prestazioni e di una grande quantità di dati etichettati. In caso contrario, è più sensato utilizzare il machine learning anziché il deep learning. Il deep learning è generalmente più complesso, per cui è necessario disporre di almeno alcune migliaia di immagini per ottenere risultati affidabili. Con una GPU ad alte prestazioni, il modello impiega meno tempo ad analizzare tutte queste immagini.



1.3. Come creare e addestrare un modello di deep learning

I tre modi più comuni utilizzati per eseguire la classificazione di oggetti tramite deep learning sono:

Addestramento da zero

Per addestrare una rete profonda da zero, è necessario raccogliere un set di dati etichettati di grandi dimensioni e progettare un'architettura di rete in grado di apprendere le feature e il modello. Ciò è utile per le nuove applicazioni o per le applicazioni che dispongono di un grande numero di categorie di output. Si tratta di un approccio meno comune poiché, considerata la grande quantità di dati e la velocità di apprendimento, l'addestramento di queste reti richiede giorni o settimane.

Transfer Learning

La maggior parte delle applicazioni di deep learning utilizza l'approccio denominato transfer learning, un processo che consiste nell'affinamento di un modello precedentemente addestrato. Si parte da una rete esistente, come AlexNet o GoogLeNet, in cui si inseriscono nuovi dati contenenti classi precedentemente sconosciute. Una volta messa a punto la rete, è possibile svolgere una nuova attività, per esempio la semplice

categorizzazione di cani o gatti anziché 1000 oggetti diversi. Questo approccio, inoltre, presenta il vantaggio di richiedere molti meno dati (vengono elaborate migliaia di immagini anziché milioni), quindi i tempi di calcolo si riducono a pochi minuti o ad alcune ore.
Il transfer learning richiede un'interfaccia interna alla rete preesistente, che consente di apportare modifiche estremamente precise e miglioramenti per la nuova attività.

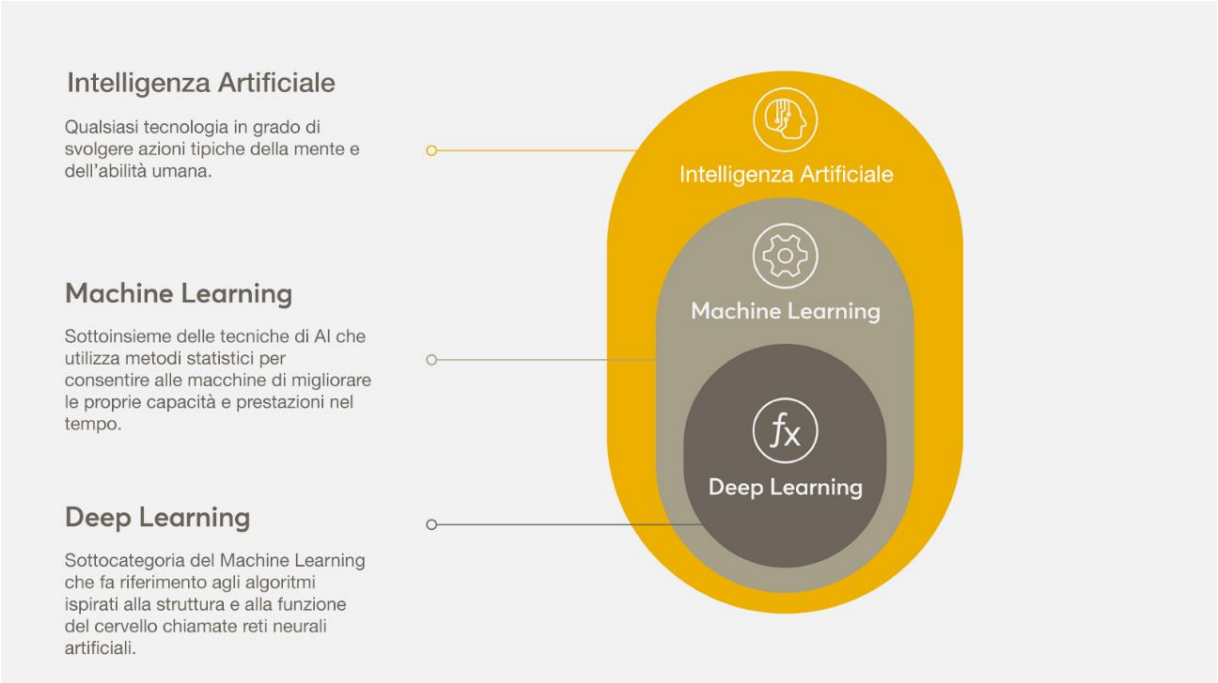
Estrazione delle feature

Un approccio di deep learning meno comune e più specializzato è l'utilizzo della rete come estrattore di feature. Poiché tutti i layer hanno il compito di apprendere determinate feature dalle immagini, è possibile estrarre queste feature dalla rete in qualsiasi momento durante il processo di addestramento.
Tali features possono quindi essere utilizzati come input per un modello di machine learning come le support vector machines (SVM).

1.4. Storia

Prima di parlare di deep learning è importante descrivere brevemente il signicato di tre importanti parole correlate a questo argomento:

- INTELLIGENZA ARTIFICIALE (IA): termine che indica un vasto insieme di tecnologie adoperate per consentire ai calcolatori elettronici di imitare il modus operandi degli esseri umani attraverso la logica, regole, alberi di decisione.
- MACHINE LEARNING: una sottocategoria delle IA che mediante varie tecniche consente ai computer di migliorare l'esecuzione di determinate operazione attraverso l'esperienza.
- DEEP LEARNING: una sottocategoria del machine learning costituita da algoritmi che permettono ai software di apprendere autonomamente l'esecuzione di determinate operazioni (riconoscimento di immagini).
  L'apprendimento avviene mediante l'analisi di ingenti quantità di dati da parti di network neurali complessi.



Nel 1958 lo psicologo Rosemblatt presentò al mondo accademico il famoso Perceptron, una rete neurale artificiale, alimentata da un computer che aveva le stesse dimensioni di una stanza. Un evento che suscitò l'entusiasmo dei media, tanto da portare un giornalista del New York Times a scrivere che presto l'umanità avrebbe assistito alla nascita di personal computer in grado di camminare, parlare e addirittura avere coscienza di sé.
Perceptron, in effetti, aveva svolto con successo il compito che gli era stato affidato: distinguere, dopo 50 tentativi, le tessere contrassegnate a destra da quelle a sinistra.
A causa dei limiti del single layer, Perceptron si fermò, incoraggiando però la ricerca a fare ulteriori passi in avanti.
Il sogno di Rosemblatt riprese vigore nei primi anni Ottanta grazie a Hinton e LeCun, i quali pubblicarono uno studio attraverso il quale veniva proposto un percorso per insegnare alle reti neurali a correggere gli errori.
Negli anni '90 si assiste ad ulteriori interessanti sviluppi nel DL: le intuizioni di LeCunn portano alla realizzazione di un dispositivo che, appoggiandosi ad una rete neurale, è in grado di comprendere la calligrafia umana (per la precisione le cifre riportate negli assegni), contemporaneamente i ricercatori tedeschi Sepp Hochreiter e Jürgen Schmidhuber creano un algoritmo che 20 anni dopo si rivelerà cruciale per le applicazioni legate al riconoscimento del linguaggio.
A metà decennio la spinta generata dai lavori di Hinton e LaCunn si esaurisce, la tecnologia di quegli anni non era sufficientemente elaborata per permettere di vedere dei miglioramenti, occorre arrivare ai giorni nostri per vedere sviluppi significativi e validi, grazie anche all'intervento di colossi del settore dell'informatica, che iniziano ad inserire il deep learning nei propri programmi commerciali.
Il DL entra nuovamente in fase di stallo.
Nel 1997 le IA attireranno comunque l'attenzione dell'opinione pubblica e dei media grazie alla clamorosa sconfitta del campione Garry Kasparov al gioco degli scacchi, fu la macchina Deep Blue di IBM a superare il noto veterano russo.
Dal 2007 circa in poi assistiamo alla terza rinascita del DL e delle reti neurali. Un importante contributo è dato in quello stesso anno da Fei-Fei Li, la professoressa si concentra su quello che sarebbe divenuto il "libro di studio" delle reti neurali.
La maggior parte del materiale utilizzabile per l'addestramento delle reti non era infatti catalogato, soprattutto le immagini.
Nasceva così ImageNet, un enorme database di immagini catalogate (milioni) pronte per essere utilizzate dai ricercatori.

Nel 2011 Microsoft implementa il deep learning nel riconoscimento del linguaggio.
L'anno successivo (2012) è il turno di Google che si rende anche protagonista del celebre "esperimento del gatto", si trattava del primo tentativo di "unsupervised learning", ovvero d'istruire una vasta rete neurale (oltre 1000 computer) senza l'impiego di dati catalogati. Il compito di analizzare milioni di immagini ed individuare eventuali pattern venne lasciato quindi alla rete stessa.
I risultati del test furono di difficile interpretazione, alcuni neuroni, situati nel livello più alto della rete, mostrarono un'energica risposta alle immagini cui erano presenti gatti (da qui il curioso nome), altri risposero alle immagini di volti umani, non venne invece trovato alcun neurone che reagisse alle immagini di auto, presenti in larga misura nel database adoperato.
Google decide di scommettere sul deep learning e tra il 2013 ed il 2016 acquista la startup DeepMind, migliora i servizi di ricerca immagini grazie alle reti neurali, rievoca la sfida al gioco degli scacchi chiamando in causa un campione cinese "Lee Sedol", che perderà 4 sfide su 5 contro il programma AlphaGo, istruito mediante reti neurali.
Anche altri importa aziende non sono rimaste certo a guardare dalla già citata Microsoft (le reti neurali sono utilizzate ad esempio per gli algoritmi Bing, riconoscimento immagini etc.) fino a Facebook (ha assoldato il veterano LeCunn che attualmente traduce grazie alle reti neurali miliardi di post al giorno).

1.5. Funzionamento

Possiamo definire il Deep Learning come un sistema che sfrutta una classe di algoritmi di apprendimento automatico che:

1) usano vari livelli di unità non lineari a cascata per svolgere compiti di estrazione di caratteristiche e di trasformazione. Ciascun livello successivo utilizza l'uscita del livello precedente come input. Gli algoritmi possono essere sia di tipo supervisionato sia non supervisionato e le applicazioni includono l'analisi di pattern (apprendimento non supervisionato) (pattern: termine inglese tradotto in "disposizione", utilizzato per descrivere la ripetizione di una determinata sequenza all'interno di un insieme di dati grezzi) e classificazione (apprendimento supervisionato);

2) sono basati sull'apprendimento non supervisionato di livelli gerarchici multipli di caratteristiche (e di rappresentazioni) dei dati. Le caratteristiche di più alto livello vengono derivate da quelle di livello più basso per creare una rappresentazione gerarchica;

3) fanno parte della più ampia classe di algoritmi di apprendimento della rappresentazione dei dati all'interno dell'apprendimento automatico (Machine Learning);

4) apprendono multipli livelli di rappresentazione che corrispondono a differenti livelli di astrazione; questi livelli formano una gerarchia di concetti.
Applicando il Deep Learning, avremo quindi una "macchina" che riesce autonomamente a classificare i dati ed a strutturarli gerarchicamente, trovando quelli più rilevanti e utili alla risoluzione di un problema (esattamente come fa la mente umana), migliorando le proprie prestazioni con l'apprendimento continuo.
Applicando il Deep Learning, avremo quindi una "macchina" che riesce autonomamente a classificare i dati ed a strutturarli gerarchicamente, trovando quelli più rilevanti e utili alla risoluzione di un problema (esattamente come fa la mente umana), migliorando le proprie prestazioni con l'apprendimento continuo.

1.6. Apprendimento

Mentre gli algoritmi di apprendimento automatico tradizionali sono lineari, gli algoritmi di apprendimento profondo sono impilati in una gerarchia di crescente complessità e astrazione.

Per capire l'apprendimento profondo, immaginiamo un bambino la cui prima parola è "cane".

Il bambino impara cos'è un cane indicando oggetti e dicendo la parola cane.

Il genitore dice "Sì, quello è un cane" o "No, non è un cane", mentre il bambino continua a puntare agli oggetti, diventa più consapevole delle caratteristiche che tutti i cani possiedono. Ciò che il bambino fa, senza saperlo, è chiarire un'astrazione complessa (il concetto di cane) costruendo una gerarchia in cui ogni livello di astrazione viene creato con la conoscenza che è stata acquisita dallo strato precedente della gerarchia».

A differenza del bambino, che impiegherà settimane o addirittura mesi per comprendere il concetto di "cane" e lo farà con l'aiuto del genitore (quello che viene definito apprendimento supervisionato), una applicazione che utilizza algoritmi di Deep Learning può mostrare e ordinare milioni di immagini, identificando con precisione quali immagini contengono i set di dati, in pochi minuti pur non avendo avuto alcun tipo di indirizzamento sulla correttezza o meno dell'identificazione di determinate immagini nel corso del training.

Solitamente, nei sistemi di Deep Learning, l'unica accortezza degli scienziati è "etichettare" i dati (con i meta tag), per esempio inserendo il meta tag "cane" all'interno delle immagini che contengono un cane ma senza spiegare al sistema come riconoscerlo: è il sistema stesso, attraverso livelli gerarchici multipli, che intuisce cosa caratterizza un cane (le zampe, il pelo, ecc.) e quindi come riconoscerlo.

Questi sistemi si basano, in sostanza, su un processo di apprendimento "trial-and-error" ma perché l'output finale sia affidabile sono necessarie enormi quantità di dati.

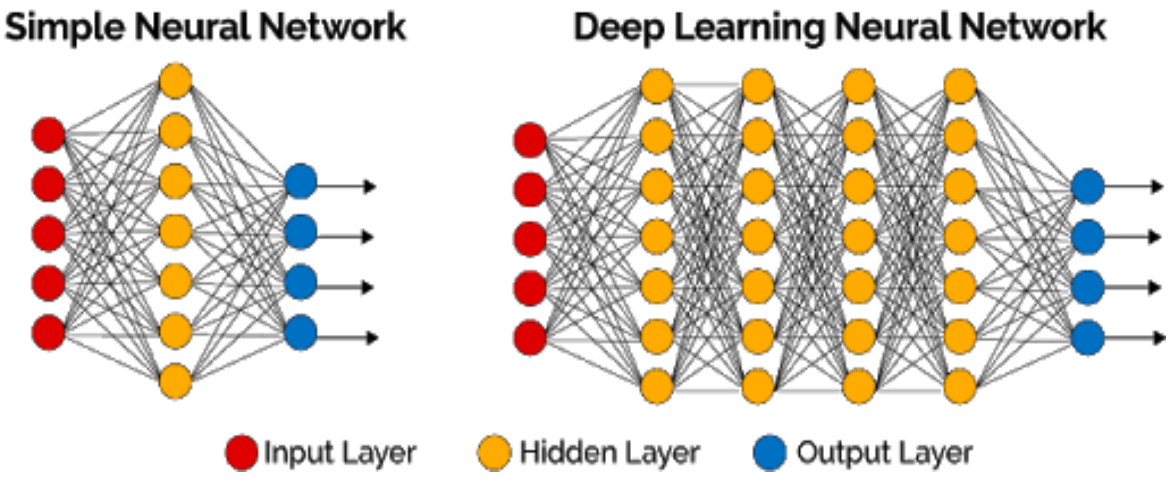
Pensare subito ai Big Data e alla facilità con cui oggi si producono e distribuiscono dati di qualsiasi forma e da qualsiasi fonte come facile risoluzione sarebbe però un errore: l'accuratezza dell'output richiede, almeno nella prima fase di addestramento, l'utilizzo di dati "etichettati" (contenenti dei meta tag) che significa che l'utilizzo di dati non strutturati potrebbero rappresentare un problema.

I dati non strutturati possono essere analizzati da un modello di apprendimento profondo una volta formato e raggiunto un livello accettabile di accuratezza, ma non per la fase di training del sistema.

Non solo, i sistemi basati su Deep Learning sono difficili da addestrare a causa del numero stesso di strati nella rete neurale.

Il numero di strati e collegamenti tra i neuroni nella rete è tale che può diventare difficile calcolare le "regolazioni" che devono essere apportate in ogni fase del processo di addestramento (un problema indicato come problema della scomparsa del gradiente); questo perché per il training comunemente si usano i cosiddetti algoritmi di retropropagazione dell'errore (backpropagation) attraverso il quale si rivedono i pesi della rete neurale (le connessioni tra i neuroni) in caso di errori (la rete propaga all'indietro l'errore in modo che i pesi delle connessioni vengano aggiornati in modo più appropriato). Un processo che continua in modo iterativo finché il gradiente (l'elemento che dà la direzione verso cui l'algoritmo deve muoversi) è nullo.

Con il Deep Learning vengono simulati i processi di apprendimento del cervello biologico attraverso sistemi artificiali (le reti neurali artificiali, appunto) per insegnare alle macchine non solo ad apprendere autonomamente ma a farlo in modo più "profondo" come sa fare il cervello umano dove profondo significa "su più livelli" (vale a dire sul numero di layer nascosti nella rete neurale – chiamati hydden layer: quelle "tradizionali" contengono 2-3 layer, mentre le reti neurali profonde possono contenerne oltre 150).



Le reti neurali profonde sfruttano un numero maggiore di strati intermedi (hydden layer) per costruire più livelli di astrazione.

Proviamo a fare un esempio concreto di funzionamento di una rete neurale profonda con il riconoscimento visivo dei pattern: i neuroni del primo strato potrebbero imparare a riconoscere i bordi, i neuroni nel secondo strato potrebbero imparare a riconoscere forme più complesse, ad esempio triangoli o rettangoli, create dai bordi.

Il terzo strato riconoscerebbe forme ancora più complesse, il quarto riconosce ulteriori dettagli e così via… i molteplici livelli di astrazione possano dare alle reti neurali profonde un vantaggio enorme nell'imparare a risolvere complessi problemi di riconoscimento di schemi proprio perché ad ogni livello intermedio aggiungono informazioni e analisi utili a fornire un output affidabile.

È abbastanza facile intuire che quanti più livelli intermedi ci sono in una rete neurale profonda (e quindi quanto più è grande la rete neurale stessa) tanto più efficace è il risultato (il compito che è "chiamata" a svolgere) ma, di contro, la scalabilità della rete neurale è strettamente correlata ai data set, ai modelli matematici e alle risorse computazionali.

Seppur la richiesta di capacità computazionali enormi possa rappresentare un limite, la scalabilità del Deep Learning grazie all'aumento dei dati disponibili e degli algoritmi è ciò che lo differenzia dal Machine Learning: i sistemi di Deep Learning, infatti, migliorano le proprie prestazioni all'aumentare dei dati mentre le applicazioni di Machine Learning (o meglio, i cosiddetti sistemi di apprendimento superficiale) una volta raggiunto un certo livello di performance non sono più scalabili nemmeno aggiungendo esempi e dati di training alla rete neurale.

Questo perché nei sistemi di Machine Learning le caratteristiche di un determinato oggetto (nel caso di sistemi di riconoscimento visivo) vengono estratte e selezionate manualmente e servono per creare un modello in grado di categorizzare gli oggetti (in base alla classificazione e al riconoscimento di quelle caratteristiche); nei sistemi di Deep Learning, invece, l'estrazione delle caratteristiche avviene in modo automatico: la rete neurale apprende in modo autonomo come analizzare dati grezzi e come svolgere un compito (per esempio classificare un oggetto riconoscendone, autonomamente, le caratteristiche).

Se dal punto di vista delle potenzialità il Deep Learning può sembrare più "affascinante" e utile del Machine Learning, va precisato che il calcolo computazionale richiesto per il loro funzionamento è davvero impattante, anche dal punto di vista economico: le CPU

più avanzate e le GPU top di gamma utili a “reggere” i workload di un sistema di Deep Learning costano ancora migliaia di dollari; il ricorso a capacità computazionali via Cloud attenuano solo in parte il problema perché la formazione di una rete neurale profonda richiede spesso l’elaborazione di grandi quantità di dati utilizzando cluster di GPU di fascia alta per molte ore.

Tra i principali metodi utilizzati nell'apprendimento non supervisionato ci sono l' analisi dei componenti principali e dei cluster .

L'analisi del cluster viene utilizzata nell'apprendimento non supervisionato per raggruppare o segmentare set di dati con attributi condivisi al fine di estrapolare le relazioni algoritmiche, è una branca dell'apprendimento automatico che raggruppa i dati che non sono stati etichettati e classificati.

Invece di rispondere al feedback, l'analisi dei cluster identifica elementi comuni nei dati e reagisce in base alla presenza o assenza di tali elementi comuni in ogni nuovo dato.

Questo approccio consente di rilevare punti di dati anomali che non rientrano in nessuno dei due gruppi.

Alcuni degli algoritmi più comuni utilizzati nell'apprendimento non supervisionato includono: 1) Clustering

2) Analisi dei componenti principali

3) Rilevazione di anomalie

Analisi dei cluster

In statistica, il clustering o analisi dei gruppi (dal termine inglese cluster analysis introdotto da Robert Tryon nel 1939) è un insieme di tecniche di analisi multivariata dei dati volte alla selezione e raggruppamento di elementi omogenei in un insieme di dati.
Le tecniche di clustering si basano su misure relative alla somiglianza tra gli elementi. In molti approcci questa similarità, o meglio, dissimilarità, è concepita in termini di distanza in uno spazio multidimensionale.
La bontà delle analisi ottenute dagli algoritmi di clustering dipende molto dalla scelta della metrica, e quindi da come è calcolata la distanza.
Gli algoritmi di clustering raggruppano gli elementi sulla base della loro distanza reciproca, e quindi l'appartenenza o meno a un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso.
Le tecniche di clustering si possono basare principalmente su due "filosofie":

Dal basso verso l'alto (metodi aggregativi o bottom-up):
Questa filosofia prevede che inizialmente tutti gli elementi siano considerati cluster a sé, e poi l'algoritmo provvede ad unire i cluster più vicini.
L'algoritmo continua ad unire elementi al cluster fino ad ottenere un numero prefissato di cluster, oppure fino a che la distanza minima tra i cluster non supera un certo valore, o ancora in relazione ad un determinato criterio statistico prefissato.

Dall'alto verso il basso (metodi divisivi o top-down):
All'inizio tutti gli elementi sono un unico cluster, e poi l'algoritmo inizia a dividere il cluster in tanti cluster di dimensioni inferiori.
Il criterio che guida la divisione è naturalmente quello di ottenere gruppi sempre più omogenei.
L'algoritmo procede fino a che non viene soddisfatta una regola di arresto generalmente legata al raggiungimento di un numero prefissato di cluster.
Esistono varie classificazioni delle tecniche di clustering comunemente utilizzate, una prima categorizzazione dipende dalla possibilità che un elemento possa o meno essere assegnato a più cluster:

Clustering esclusivo: ogni elemento può essere assegnato ad uno e ad un solo gruppo. Quindi i cluster risultanti non possono avere elementi in comune. Questo approccio è detto anche hard clustering.

Clustering non-esclusivo: in cui un elemento può appartenere a più cluster con gradi di appartenenza diversi. Questo approccio è noto anche con il nome di soft clustering.

Un'altra suddivisione delle tecniche di clustering tiene conto del tipo di algoritmo utilizzato per dividere lo spazio:

Clustering partizionale (detto anche non gerarchico, o k-clustering), in cui per definire l'appartenenza ad un gruppo viene utilizzata una distanza da un punto rappresentativo del cluster (centroide, medioide ecc.…), avendo prefissato il numero di gruppi della partizione risultato.

Clustering gerarchico, in cui viene costruita una gerarchia di partizioni caratterizzate da un numero crescente di gruppi, visualizzabile mediante una rappresentazione ad albero (dendrogramma), in cui sono rappresentati i passi di accorpamento/divisione dei gruppi.

Analisi dei componenti principali

L'analisi delle componenti principali (detta pure PCA) è una tecnica utilizzata nell'ambito della statistica multivariata per la semplificazione dei dati d'origine.
Lo scopo primario di questa tecnica è la riduzione di un numero più o meno elevato di variabili (rappresentanti altrettante caratteristiche del fenomeno analizzato) in alcune variabili latenti.
Ciò avviene tramite una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema cartesiano nel quale le variabili vengono ordinate in ordine decrescente di varianza, pertanto, la variabile con maggiore varianza viene proiettata sul primo asse, la seconda sul secondo asse e così via.
La riduzione della complessità avviene limitandosi ad analizzare le principali (per varianza) tra le nuove variabili.
Diversamente da altre trasformazioni (lineari) di variabili praticate nell′ambito della statistica, in questa tecnica sono gli stessi dati che determinano i vettori di trasformazione.
La PCA è una tecnica statistica adoperata in molti ambiti: nell'astronomia, nella medicina, in campo agro-alimentare, ecc.… fino anche alla compressione di immagini; questo perché quando ci si trova a semplificare un problema, riducendo la dimensione dello spazio di rappresentazione, si ha allo stesso tempo una perdita dell'informazione contenuta nei dati originali.
La PCA consente di controllare egregiamente il "trade-off" tra la perdita di informazioni e la semplificazione del problema (basta scegliere il numero appropriato di auto vettori).

Rilevazione di anomalie

Nel data mining , il rilevamento di anomalie è l'identificazione di elementi, eventi o osservazioni rari che sollevano sospetti differendo in modo significativo dalla maggior parte dei dati.

In genere gli articoli anomali si traducono in qualche tipo di problema come frode bancaria , un difetto strutturale, problemi medici o errori in un testo.

Le anomalie sono anche indicate come valori anomali , novità, rumore, deviazioni ed eccezioni.

In particolare, nel contesto del rilevamento di abusi e intrusioni di rete, gli oggetti interessanti spesso non sono oggetti rari, ma esplosioni inattese di attività. Questo modello non aderisce alla definizione statistica comune di un valore anomalo come oggetto raro e molti metodi di rilevamento anomalo (in particolare metodi non controllati) falliranno su tali dati, a meno che non siano stati aggregati in modo appropriato.

Al contrario, un algoritmo di analisi dei cluster potrebbe essere in grado di rilevare i micro-cluster costituiti da questi pattern.

Esistono tre grandi categorie di tecniche di rilevazione delle anomalie:

1)tecniche di rilevamento delle anomalie **senza supervisione** rilevano anomalie in un set di dati di test senza etichetta, supponendo che la maggior parte delle istanze nel set di dati sia normale cercando istanze che sembrano adattarsi meno al resto del set di dati.

2)Le tecniche di rilevazione di anomalie **supervisionate** richiedono un set di dati che è stato etichettato come "normale" e "anormale" e prevede l'addestramento di un classificatore (la differenza chiave rispetto a molti altri problemi di classificazione statistica è la natura intrinseca sbilanciata del rilevamento anomalo).

3)Le tecniche di supervisione delle anomalie **semi-supervisionate** costruiscono un modello che rappresenta il comportamento normale da una data normale set di dati di training, quindi verifica la probabilità che un'istanza di test sia generata dal modello appreso.

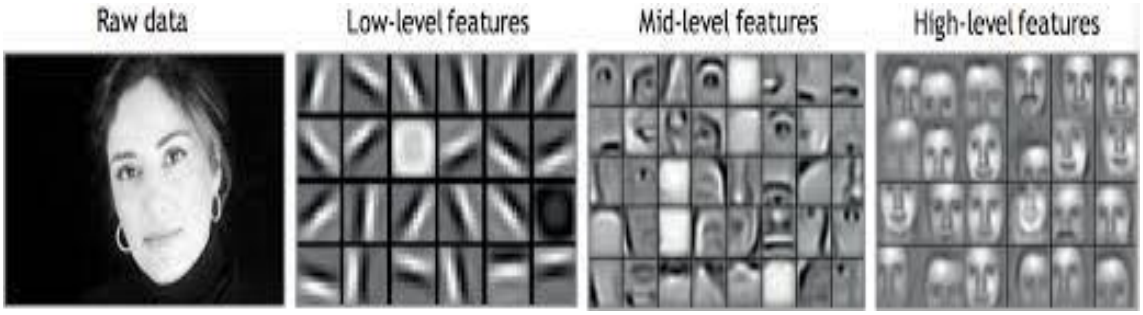
1.7. Architetture del deep learning

Il deep learning (o apprendimento strutturato profondo) fa parte di una più ampia famiglia di metodi di apprendimento automatico basati su reti neurali artificiali con apprendimento della rappresentazione.

Nel machine learning, "l'apprendimento delle caratteristiche" o "apprendimento della rappresentazione" è un insieme di tecniche che permette ad un sistema di rilevare automaticamente le features (caratteristiche) necessarie per le funzioni di rilevamento o di classificazione da dati grezzi.

Ciò sostituisce la progettazione manuale delle funzioni e consente ad una macchina di apprendere le funzionalità e di utilizzarle per eseguire un'attività specifica.

Nel campo dell'apprendimento automatico, una caratteristica (nota anche con il rispettivo termine inglese feature) è una proprietà individuale e misurabile di un fenomeno osservato.



La scelta di caratteristiche discriminanti, ad alto contenuto informativo e indipendenti fra loro è un passo cruciale per ottenere un efficiente algoritmo di riconoscimento di pattern, classificazione e regressione.

Il valore di una feature viene solitamente reso in forma numerica; esistono tuttavia delle eccezioni, come nel riconoscimento sintattico di pattern (syntactic pattern recognition), in cui vengono considerate caratteristiche strutturali come stringhe e grafi.

L'insieme (inizialmente) grezzo delle caratteristiche potrebbe essere ridondante e troppo vasto per essere gestito efficientemente.

Di conseguenza, un tipico passo preliminare in molte applicazioni dell'apprendimento automatico consiste nella selezione delle caratteristiche, nell'estrazione di caratteristiche

o, più in generale, nella riduzione della dimensionalità del cosiddetto input space ("spazio di ingresso").
Nel riconoscimento di pattern e nell'elaborazione delle immagini la selezione delle caratteristiche (in inglese: feature selection) è una forma speciale di riduzione della dimensionalità di un determinato dataset.
La selezione delle caratteristiche è il processo di riduzione degli ingressi per l'elaborazione e l'analisi o l'individuazione delle caratteristiche maggiormente significative rispetto alle altre.
Similmente esiste l'estrazione di caratteristiche (in inglese: feature extraction), dove si applica il processo di estrazione di informazioni utili dai dati esistenti.
La selezione delle caratteristiche risulta necessaria per creare un modello funzionale, ossia una riduzione della cardinalità, imponendo un limite superiore al numero di caratteristiche che devono essere considerate durante la creazione di questo.
Solitamente i dati contengono informazioni ridondanti, ovvero più di quelle necessarie (oppure possono contenere anche informazioni errate).
La selezione delle caratteristiche rende più efficiente il processo di creazione di un modello, andando ad esempio a diminuire la CPU e la memoria necessarie per l'addestramento (training), anche se vi sono casi in cui le risorse non sono un problema.
La selezione delle caratteristiche viene utilizzata per tre ragioni:

- semplificazione dei modelli per renderli più facili da interpretare da ricercatori / utenti;
- tempi di addestramento (training) minori;
- miglioramento generalizzato nella riduzione del problema di overfitting, ovvero una riduzione della varianza.

L'estrazione di caratteristiche (in inglese: feature extraction) è una forma speciale di riduzione della dimensionalità.
Quando i dati in ingresso sono troppi per l'esecuzione di un algoritmo e c'è il sospetto di ridondanza allora i dati verranno convertiti in una rappresentazione ridotta di un insieme di caratteristiche (il vettore delle caratteristiche o feature vector).
Il processo di trasformazione dei dati in ingresso in un insieme di caratteristiche è chiamato estrazione di caratteristiche.
L'estrazione di caratteristiche significa semplificare il costo delle risorse richieste per descrivere un grande insieme di dati accuratamente.

Quando si eseguono analisi di dati complessi, uno dei più grandi problemi sta nell'arginare il numero di variabili coinvolto. L'analisi di un gran numero di variabili generalmente richiede un grande uso di memoria ed elaborazione o algoritmi di classificazione che hanno bisogno di un'alta soglia di adattamento con i campioni di prova e generalizzano in modo povero nuovi campioni. L'estrazione di caratteristiche è un termine generale per metodi di costruzione di combinazione di variabili per aggirare questi problemi ma descrivendoli con una accuratezza sufficiente.

L'apprendimento delle funzioni può essere supervisionato o non supervisionato.

- Nell'apprendimento delle funzioni supervisionato, le funzioni vengono apprese utilizzando i dati di input etichettati, ad esempio reti neurali supervisionate e apprendimento del dizionario (supervisionato) .

- Nell'apprendimento delle funzioni senza supervisione , le funzioni vengono apprese con dati di input senza etichetta, gli esempi includono l'apprendimento del dizionario, l'analisi indipendente di dati e varie forme di clustering .

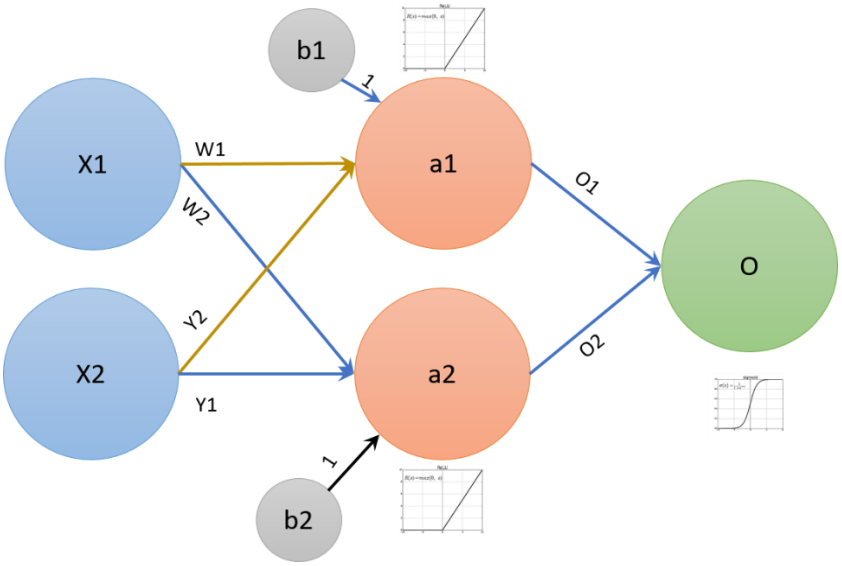
1.8. Reti neurali artificiali

Una rete neurale artificiale (in inglese artficial neural network, abbreviato in ANN) è un modello matematico composto da neuroni artificiali, che si ispira a una rete neurale biologica.
Le reti neurali artificiali (ANN) sono un algoritmo utilizzato per risolvere problemi di natura complessa non facilmente codificabili, e sono una colonna portante del machine learning come viene inteso oggi.
Sono chiamate "reti neurali" perché il comportamento dei nodi che le compongono ricorda vagamente quello dei neuroni biologici. Un neurone riceve in ingresso segnali da vari altri neuroni tramite connessioni sinaptiche e li integra.
Possiamo considerare una rete neurale come una scatola nera, con degli input, degli strati intermedi e degli output che costituiscono il risultato finale.
La rete neurale è composta da unità chiamate "neuroni", disposti su strati successivi. Ciascun neurone è collegato a tutti i neuroni dello strato successivo tramite connessioni pesate.
Una connessione non è altro che un valore numerico ("il peso" appunto), che viene moltiplicato per il valore del neurone collegato.
Ciascun neurone somma i valori pesati di tutti i neuroni ad esso collegati e aggiunge un valore di bias (questo peso è utile per «tarare» il punto di lavoro ottimale del neurone).
A questo risultato viene applicata una "funzione di attivazione", che non fa altro che trasformare matematicamente il valore prima d passarlo allo strato successivo, in questo modo i valori di input vengono propagati attraverso la rete fino ai neuroni di output.
Il succo di tutto è regolare pesi e bias in modo da arrivare ad ottenere il risultato voluto.
Una rete neurale può essere immaginata come composta di diversi "layer" (strati) di nodi, ciascuno dei quali è collegato ai nodi del layer successivo.
Tutte le reti neurali sono composte da almeno tre strati:

- Un input layer, ovvero i dati di ingresso;
- Uno o più hidden layer, dove avviene l'elaborazione vera e propria;
- Un output layer, contenente il risultato finale.

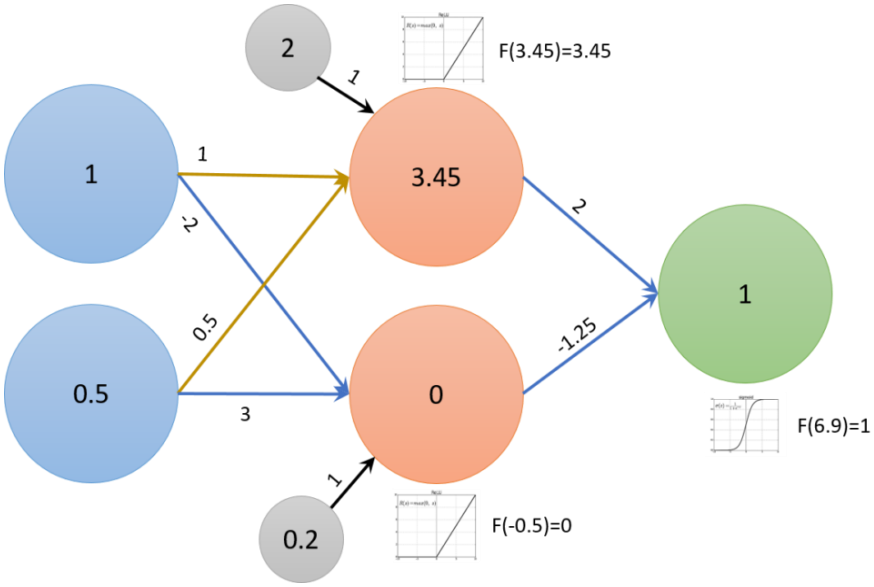
I nodi sono connessi a tutti i nodi del layer successivo, e nell'algoritmo queste connessioni vengono "pesate" tramite fattori moltiplicativi, che rappresentano la "forza" della connessione stessa.

Come per i neuroni biologici, i nodi delle ANN integrano i segnali in ingresso secondo una apposita funzione (ci torneremo più avanti). Se il valore di questa operazione supera la soglia prevista, il nodo sollecitato trasmette questo valore al layer successivo, altrimenti il valore trasmesso sarà zero.



Vediamo che il layer di input ha due nodi, X1 e X2, il layer nascosto è costituito dai nodi a1 e a2, mentre O è il nodo di output.

Ciascun nodo del secondo layer sommerà il segnale proveniente da ciascun nodo di input, moltiplicato per il "peso".



Il principio è identico per il nodo di output. Le connessioni w1 e y1 sono quelle uscenti dal nodo X1, mentre w2 e y2 sono quelle uscenti da X2.

Proviamo ad attribuire dei valori, come nella figura sotto.

I nodi "nascosti" a1 e a2 riceveranno in ingresso la somma dei nodi X1 e X2, "pesati" dalle connessioni. Quindi il valore ricevuto da a1 sarà uguale a (X1*w1+b1) + (X2*w2+b2), ovvero 1*1 + 0.5*0.5+2 = 3.25, mentre con lo stesso principio il valore ricevuto da a2 sarà -0.3.

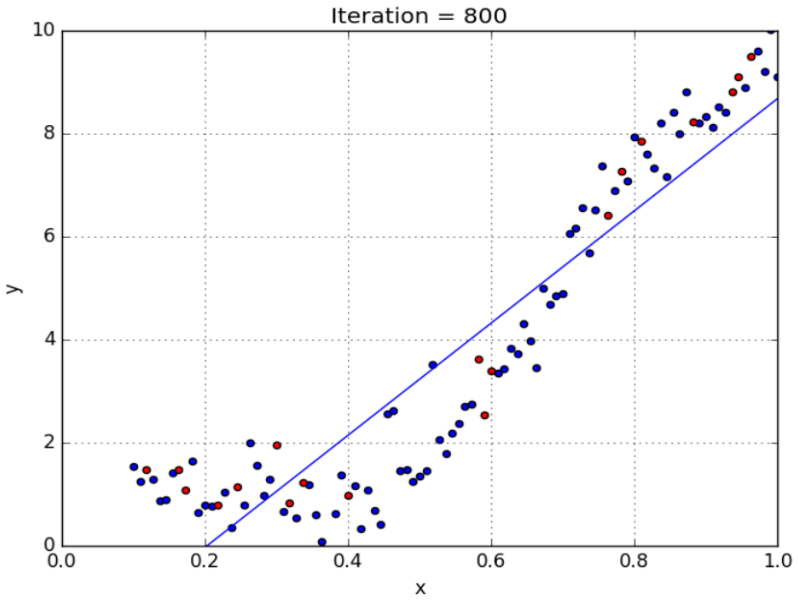
A questi valori si applicherà la funzione di attivazione, che nel caso di a1 lascerà il valore invariato, mentre per a2 ritornerà zero.

Dal layer nascosto quindi usciranno i valori 3.45 e 0, che verranno poi moltiplicati rispettivamente per 2 e -1.25 prima di venire integrati nel nodo di uscita.

Lo stesso principio si applica al nodo di uscita, che riceverà quindi un totale di 6.5, che viene trasformato in 1 dalla funzione di attivazione.

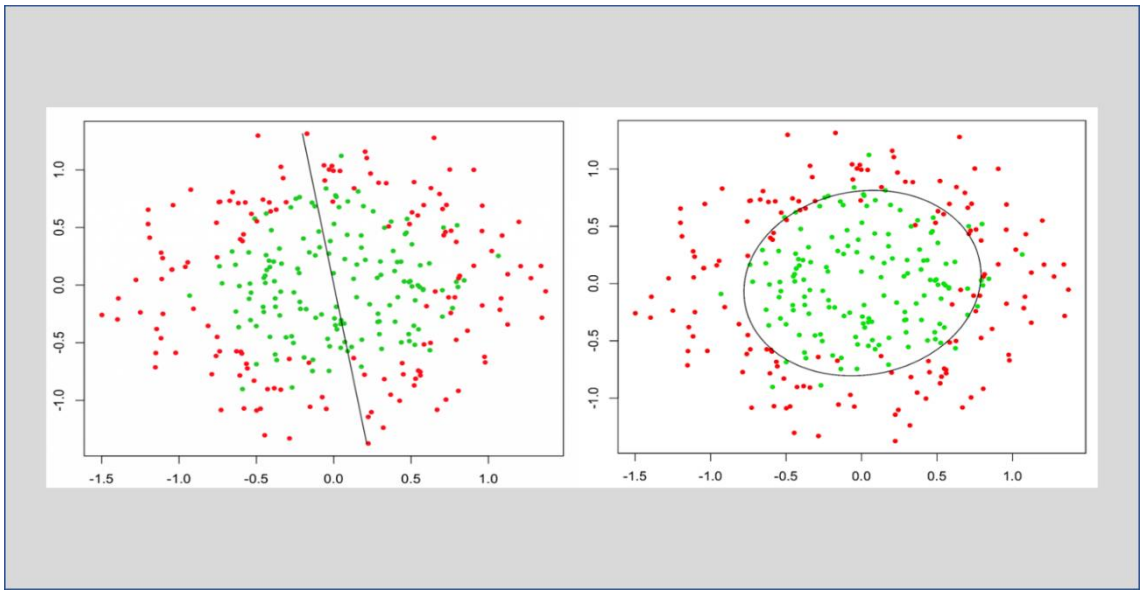
Funzione di attivazione

Nei neuroni biologici il potenziale d'azione viene trasmesso integralmente una volta che la differenza di potenziale alle membrane supera una certa soglia. In un certo senso è così anche per i neuroni "artificiali". Solo che il comportamento della risposta viene adattato a seconda delle necessità, ed è determinato dalla funzione di attivazione.
A questo punto ci si potrebbe chiedere perché applicare una funzione di attivazione. Non si potrebbe semplicemente propagare i valori attraverso la rete neurale così come sono?
Una rete neurale senza funzione di attivazione equivale semplicemente a un modello di regressione (Lo scopo dei modelli di regressione è quello di trovare un'equazione, rappresentata da un tracciato sul grafico, che rappresenti i dati in modo abbastanza preciso da spiegare il comportamento, ma anche sufficientemente "flessibile" da consentire di fare previsioni), ovvero cerca di approssimare la distribuzione dei dati con una retta.

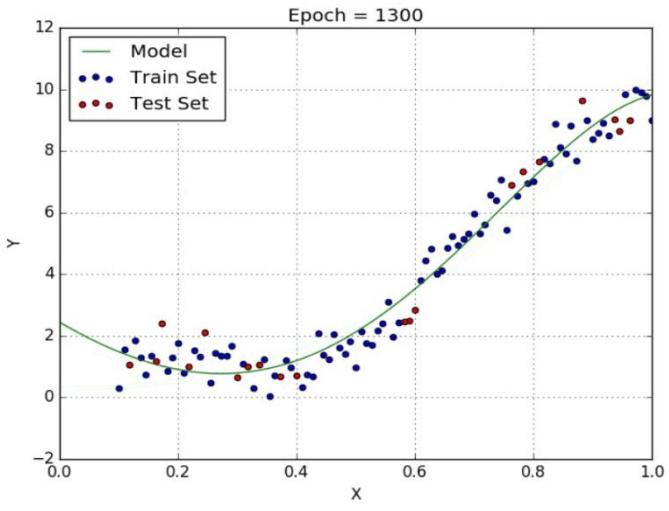


In questo esempio si può notare come la retta rappresenti la distribuzione in modo abbastanza impreciso.
Con questo modello praticamente ogni layer si comporterebbe allo stesso modo del precedente, e 100 layer sarebbero di fatto equivalenti ad averne uno soltanto: il risultato sarebbe sempre lineare.
Lo scopo delle reti neurali è quello di essere in grado di approssimare qualsiasi funzione, per far questo è necessario introdurre un fattore di non linearità, da qui la funzione di attivazione.

Inoltre, in molti casi la regressione lineare diventa non solo poco precisa ma addirittura inutilizzabile, come nel caso di distribuzione circolare.
Qui sotto una comparazione tra regressione lineare e non lineare per una distribuzione circolare.

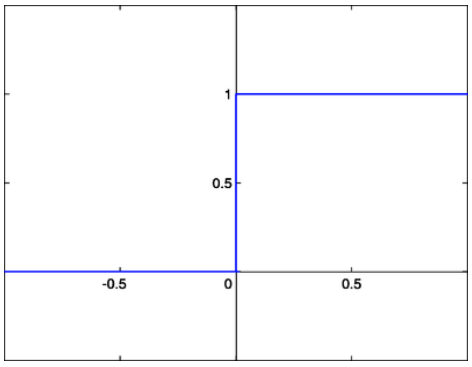


Con un modello non lineare è possibile approssimare gli stessi dati in modo molto più preciso.

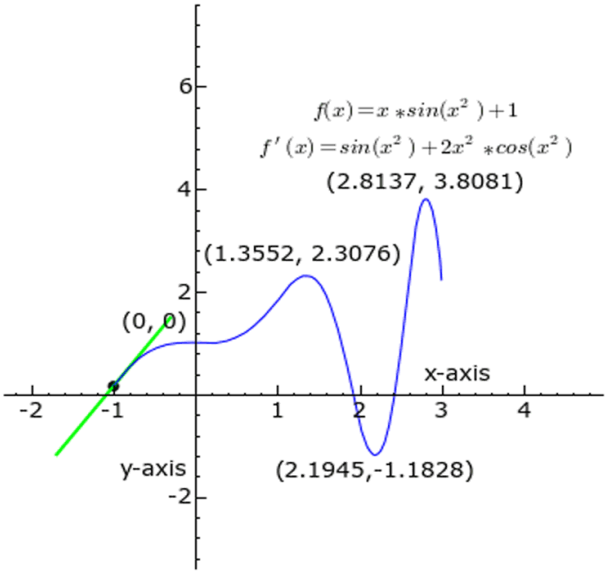


Funzione a gradino (Step Function):

La funzione più intuitiva è quella a gradino, in un certo senso più simile al funzionamento biologico. Per tutti i valori negativi la risposta rimane 0, mentre salta a +1 appena il valore raggiunge o supera anche di poco lo zero. Il vantaggio è che è semplice da calcolare, e "normalizza" i valori di output comprimendoli tutti in un range compreso tra 0 e +1. Tuttavia, questo tipo di funzione non è realmente impiegata per via della poca stabilità, e soprattutto perché non è differenziabile nel punto in cui cambia valore (non esistono derivate in quel punto). La derivata non è altro che la pendenza della retta tangente in quel punto, ed è fondamentale nel deep learning, in quanto determina la direzione verso cui orientarsi per gli aggiustamenti ai valori.



In breve, possiamo dire che questo cambio brusco di stato rende difficile controllare il comportamento della rete. Una piccola modifica su un peso potrebbe migliorare il comportamento per un determinato input ma farlo saltare totalmente per altri simili.



Funzione sigmoide:

Per ovviare al problema fu introdotta la funzione sigmoide.

Ha delle similitudini con quella a gradino, ma il passaggio da 0 a +1 è più graduale, con un andamento a forma di s appunto. Il vantaggio di questa funzione, oltre ad essere differenziabile, è di comprimere i valori in un range tra 0 e 1 e di essere quindi molto stabile anche per grosse variazioni nei valori.

La sigmoide è stata molto usata per parecchio tempo, ma ha comunque i suoi problemi.

È una funzione che presenta una convergenza molto lenta, visto che per valori di ingresso molto grandi la curva è quasi piatta, con la conseguenza che la derivata tende a zero. Questa scarsa responsività verso gli estremi della curva tende a causare problemi di

vanishing gradient (il problema è dovuto al fatto che la derivata della funzione si riduce a ogni passaggio, quindi reti con molti layer tendono a far "sfumare" il gradiente, rallentando di molto la convergenza.

Inoltre non essendo centrata sullo zero i valori in ogni step di learning possono essere solo tutti positivi o tutti negativi, il che rallenta il processo di training della rete.

Si tratta di una funzione quindi non più molto utilizzata nei layer intermedi, ma ancora decisamente valida in output per i compiti di categorizzazione.

Funzione ReLU:

La funzione ReLU (rectifier linear unit) è una funzione divenuta ultimamente molto utilizzata, specialmente nei layer intermedi. La ragione è che si tratta di una funzione molto semplice da calcolare: appiattisce a zero la risposta a tutti i valori negativi, mentre lascia tutto invariato per valori uguali o superiori a zero.

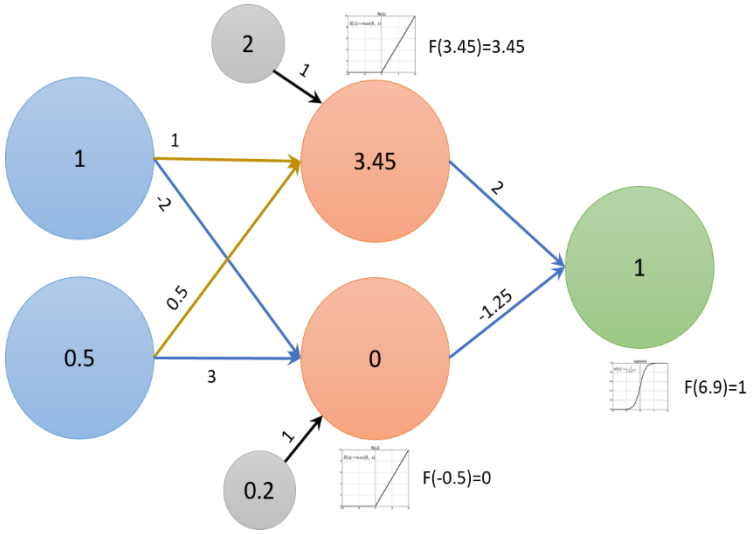
Questa semplicità, unita al fatto di ridurre drasticamente il problema del vanishing gradient, la rende una funzione particolarmente appetibile nei layer intermedi, dove la quantità di passaggi e di calcoli è importante.

Calcolare la derivata infatti è molto semplice: per tutti i valori negativi è uguale a zero, mentre per quelli positivi è uguale a 1. Nel punto angoloso nell'origine invece la derivata è indefinita ma viene comunque impostata a zero per convenzione.

Guardando di nuovo i neuroni del layer intermedio, notiamo che la loro funzione di attivazione è una ReLU, quindi nel primo caso 3.45 rimane invariato, mentre il valore del secondo da -0.45 viene considerato zero.

Il neurone di output invece ha una funzione sigmoide, e per un valore di 6.9 la risposta diventa praticamente uguale a 1.

Le funzioni di attivazioni possibili sono numerose, ma queste in questo



contesto le tre illustrate sono sufficienti a dare un'idea di cosa siano e del perché vengano usate.

Abbiamo visto come i valori di input si propagano attraverso i layer nascosti fino ai neuroni di output, passiamo ora a comprendere l'apprendimento, ovvero la regolazione di bias e pesi per ottenere il risultato voluto.

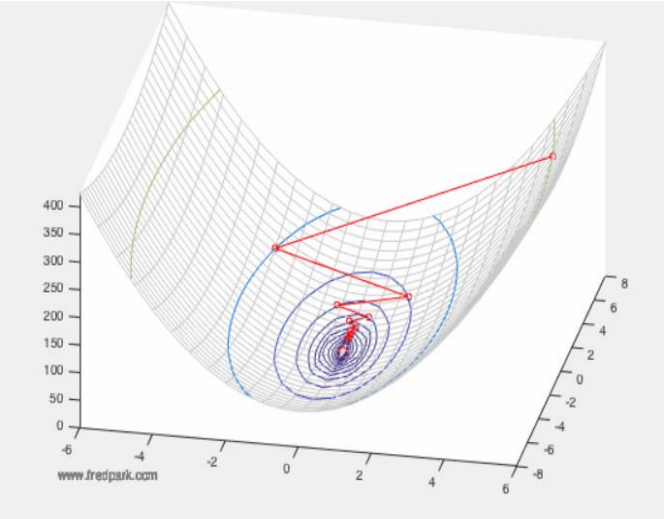
Gradient Descent

Innanzitutto, tutti i pesi e i valori di bias vengono inizialmente valorizzati in modo casuale, il che significa che nella prima passata la risposta della rete sarà casuale, e molto probabilmente completamente errata.

Il primo passo è calcolare quella che viene chiamata cost function, ovvero una funzione che rappresenti in qualche modo l'errore (come differenza tra output e valore atteso) quadratico medio di tutti gli output.

Il gradient descent è una tecnica che ha lo scopo di minimizzare quanto possibile la cost function. Immaginando la cost function come funzione di sole due variabili (per semplificare), lo scopo del nostro gradient descent è quello di trovare il minimo globale della funzione, ovvero il punto più basso.

Tutto quello che fa il gradient descent è partire da un punto casuale, e poi in base alle derivate spostarsi in una direzione o in un'altra. Una derivata elevata significa pendenza elevata, quindi ancora lontani dal minimo, e il successivo passo sarà ampio, una derivata piccola significa pendenza lieve, di conseguenza vicini al minimo, il che comporta passi di avvicinamento più piccoli.



Nella figura si può vedere come il gradiente si avvicini al minimo per passi successivi, riducendo l'ampiezza (che poi altro non è che il tasso di apprendimento) man mano che il fondo si fa più vicino.

Backpropagation

A tutto il discorso fatto finora manca ancora un tassello: abbiamo visto come i dati si propagano attraverso la rete, abbiamo visto la tecnica più utilizzata per ridurre l'errore (di fatto apprendere). Appurato che alla fine si tratta di "riaggiustare" i pesi, e che farlo a mano è fuori discussione, come si fa? È qui che interviene la backpropagation, ovvero la propagazione "all'indietro" dell'errore (differenza tra output e valore atteso).

Quello che succede in sintesi è che una volta calcolata la nostra Cost Function, abbiamo un'idea abbastanza precisa di quanto ciascun neurone di output sia lontano dal proprio valore atteso, e in che direzione (positiva o negativa).

Se il risultato atteso è 6, ci aspettiamo 1 nel neurone 6 e 0 in tutti gli altri. Se il neurone 6 presenta 0.7, allora la correzione da fare è 0.3 e riarrangeremo i pesi delle connessioni a quel neurone in modo da produrre complessivamente un valore leggermente più grande.

Se il neurone 4 invece di 0 presenta 0.8 allora la correzione sarà -0.8 e si riarrangeranno le connessioni a questo neurone in modo da abbassarne drasticamente l'output. Questo viene effettuato dall'algoritmo calcolando le derivate e moltiplicando opportunamente le matrici di valori e pesi.

Nel deep learning si utilizzano reti neurali "deep", ovvero profonde diversi layer. Lo scopo di avere diversi layer invece che uno solo è che per come funziona l'algoritmo, ciascun layer "generalizza" un po' di più rispetto al precedente.

Quindi per esempio nel caso del riconoscimento delle forme geometriche, il primo riconoscerà solamente i singoli pixel, il secondo "generalizzerà" i bordi, il terzo inizierà a riconoscere forme semplici, e così via.

La rete si muove attraverso i livelli calcolando la probabilità di ogni uscita. Ad esempio, un DNN addestrato per riconoscere le razze canine andrà oltre l'immagine data e calcolerà la probabilità che il cane nell'immagine sia una determinata razza. L'utente può rivedere i risultati e selezionare le probabilità che la rete dovrebbe visualizzare (oltre una certa soglia, ecc.) e restituire l'etichetta proposta. Ogni manipolazione matematica in quanto tale è considerata un livello e i DNN complessi hanno molti livelli, da cui il nome di reti "profonde".

1.9. Applicazioni più diffuse di deep learning

1) Macchine Automatiche di traduzione: Questa è un'attività in cui parole e frasi in una lingua vengono tradotte automaticamente in un'altra lingua. Sebbene le traduzioni automatiche ci siano già da un po' di tempo, algoritmi di Deep Learning migliorano l'apprendimento delle relazioni tra più parole e la loro mappatura in una nuova lingua.

2) Aggiunta di suoni ai video muti: altro modello di Deep Learning che associa i fotogrammi video a un database di suoni preregistrati per selezionare un suono da riprodurre che corrisponda al meglio a ciò che accade nella scena del video.

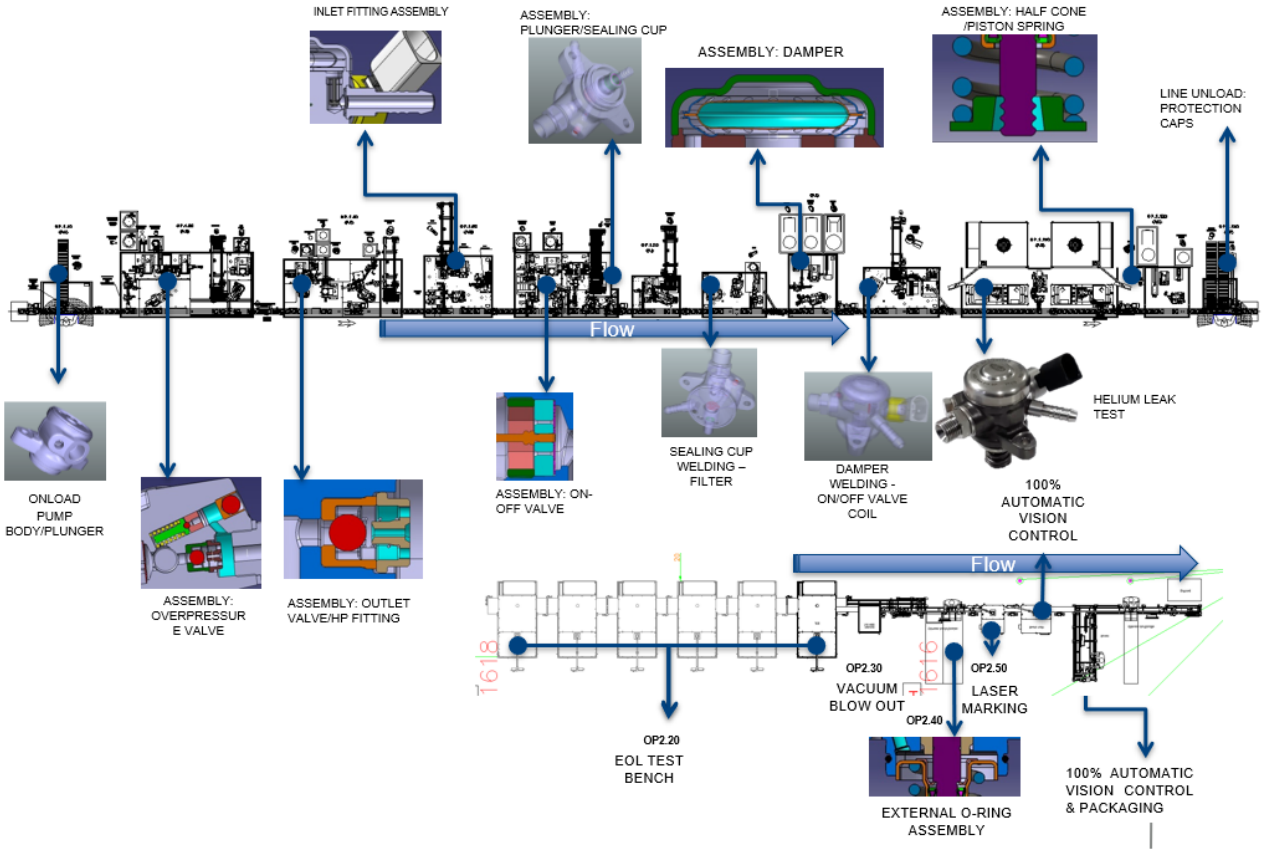
3) Classificazione di oggetti: sono stati sviluppati algoritmi in grado di classificare gli oggetti di una fotografia come uno di un insieme di oggetti precedentemente noti.
Una variante più complessa di questa attività chiamata rilevamento oggetti comporta l'identificazione specifica di uno o più oggetti all'interno della fotografia e il disegno di un riquadro attorno ad essi.

4) Generazione automatica di didascalie di immagini: Nel 2014 ci fu un'esplosione di algoritmi di deep learning che ottennero risultati molto interessanti su questo problema, sfruttando il lavoro dei migliori modelli per la classificazione e il rilevamento degli oggetti nelle fotografie.
Una volta che è possibile rilevare e generare etichette per tali oggetti nelle fotografie, è possibile utilizzarle per creare una didascalia dell'immagine. Queste tecniche sono state poi introdotte anche nei video.

5) Generazione di linguaggio naturale: applicazione che produce voce umana da una macchina, Un esempio è "Wavenet" che è in grado di generare un discorso che imita qualsiasi voce umana e che suona più naturale dei migliori sistemi di sintesi vocale esistenti, riducendo il divario con le prestazioni umane.

2. Case study

In questa tesi viene analizzata una linea di produzione della Magneti Marelli, nello specifico vengono fatte predizioni di eventi di guasto su linee produttive tramite algoritmi di deep learning.



2.1. Magneti Marelli

Magneti Marelli è una multinazionale italiana specializzata nella fornitura di prodotti e sistemi ad alta tecnologia per l'industria automobilistica con sede a Corbetta (MI), Italia.
Nel 2018, Calsonic Kansei e Magneti Marelli hanno annunciato l'intenzione di effettuare una fusione per dare vita al settimo fornitore automotive indipendente a livello globale per fatturato.
Nel corso dei suoi 80 anni di storia, Calsonic Kansei ha consolidato una reputazione di spicco per la qualità e l'eccellenza produttiva.
Dalla sua sede storica in Giappone, Calsonic Kansei ha ampliato la propria area operativa in Asia e in Europa, diventando un'azienda leader nelle soluzioni integrate per l'abitacolo (cockpit/interni per abitacolo), nei sistemi di climatizzazione, negli scambiatori di calore e nei compressori.
Fondata nel 1919, Magneti Marelli diventa sempre più conosciuta all'interno del settore automotive grazie al suo spirito pionieristico, e al contributo apportato alla mobilità intelligente e sostenibile. Durante la sua storia centenaria ha servito clienti dalla sede italiana per poi espandere le operazioni in Europa, Nord e Sud America, India e Cina, diventando un'azienda leader nel campo dell'illuminazione, dell'elettronica, della propulsione e del motorsport.
Nel 2019 è stata fondata ufficialmente MARELLI.
L'unione di questi due giganti del settore ha consentito la fusione di una straordinaria esperienza industriale e di un patrimonio storico unico. Le due aziende infatti sono tra loro altamente complementari, sia in termini di linee di prodotto che di presenza geografica. La nascita di Marelli è fondata quindi su un'unione di qualità e innovazione, per creare un nuovo attore globale.

Marelli, grazie alla sua ampia offerta rivolta ai principali player del settore, copre le principali aree di prodotto: illuminazione, comfort dell'abitacolo, propulsori elettrici, sistemi elettronici, tecnologia green, soluzioni integrate per l'abitacolo, propulsori, sospensioni, soluzioni termiche, motorsport.

Con circa 62.000 dipendenti nel mondo, il perimetro di MARELLI conta 170 fra stabilimenti e centri di Ricerca e Sviluppo in Asia, America, Europa e Africa e un fatturato di 14,6 miliardi di € (1.825 miliardi di yen) nel 2018.

2.2. Il dataset

L'obiettivo di questo studio è fare delle previsioni dei valori "OEE", "FTQ", e "Pezzi totali prodotti", partendo dall'analisi di un dataset excel contenente i dati riguardanti la linea di produzione dell'azienda.
Nelle colonne del dataset troviamo:

- Disponibilità attuale (MINUTI TURNO)
- MENSA
- BLACK OUT
- FERMI NON PROGRAMMATI \ RIGENERAZIONE UTENSILI
- FERMI PROGRAMMATI \ RIGENERAZIONE UTENSILI
- MANUTENZIONE PM PROGRAMMATA
- PULIZIE TECNICHE
- SET-UP Cambio Tipo
- MICROFERMATE
- MANCANZA MATERIALI DIRETTI
- MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE
- Fermo problemi Qualità
- GUASTI - RISOLTI PM
- GUASTI - RISOLTI AM
- Totale Min riparazione PM
- Totale Min riparazione AM
- TEMPO OPERATIVO

Questi sono alcune delle voci presenti nel dataset, nelle righe invece abbiamo le date in cui sono state fatte queste misurazioni (dal 02/01/2019 al 26/06/2019) .
Per fare tali previsioni ho usato le tecnologie di deep learning, ho quindi creato delle reti neurali diverse a seconda della voce da analizzare.
Nel dataset sono comunque presenti anche le voci "OEE", "FTQ", e "Pezzi totali prodotti", necessarie al sistema per apprendere la relazione tra i vari dati e questi ultimi.
Grazie a questo sistema possiamo inserire i valori numerici relativi ai diversi dati elencati sopra ed avere dei valori in output con una precisione nel risultato molto elevata.

2.3. OEE

L'OEE (Overall Equipment Effectiveness) è un indicatore globale di efficienza delle risorse produttive che può essere calcolato in diversi modi.
L'OEE, letteralmente "efficienza generale dell'impianto", è un indicatore percentuale che rappresenta il rendimento globale di una risorsa produttiva o di un insieme di risorse, siano esse umane o tecniche, durante il tempo nel quale queste sono disponibili a produrre.
Nella pratica, se ad esempio il reparto X nella settimana Y è disponibile a produrre, mettiamo caso, per un tempo di 5 gg * 8 ore/gg = 40 ore, allora un OEE consuntivo pari al 50% significa che il reparto ha prodotto materiali conformi in quantità pari alla metà della quantità massima teorica che da quel reparto ci si poteva attendere, a fronte delle risorse in esso presenti e delle 40 ore disponibili.
Come si può ben comprendere dall'esempio, l'OEE è l'indicatore più "esigente" ed omnicomprensivo che esista, in quanto sconta tutte le tipologie di inefficienze che portano ad una minore produttività: dalla mancanza di materiali alla cattiva pianificazione, dai setup ai tempi morti, dalle microfermate ai guasti, dalle rilavorazioni alle non conformità.
In letteratura e sul web, l'OEE viene affrontato secondo un modello "classico" che ben si adatta all'industria di processo (o comunque, laddove esistono linee di produzione automatizzate) ma che è difficile applicare in realtà organizzate per reparti, specie se queste producono per parti discrete (pezzi), e ancor più se le lavorazioni sono manuali.
Nel corso degli anni, invece, è stata introdotta una metodologia di calcolo innovativa che, pur riconducendosi allo stesso modello di base, è decisamente più adatta ai tipici contesti produttivi per reparti, oppure misti reparti/linee, con produzione di parti discrete.

La definizione di OEE "classica"

OEE = Disponibilità x Prestazione x Qualità

L'OEE, nel modello classico, è il prodotto di tre indicatori percentuali che rappresentano le tre componenti fondamentali della performance:

- Disponibilità: percentuale dell'effettivo tempo di attività rispetto a quello disponibile;
- Prestazione (o Rendimento): percentuale di parti prodotte rispetto alla potenzialità teorica, quando l'impianto è attivo (corrisponde alla velocità effettiva rispetto alla velocità nominale);
- Qualità: percentuale di parti conformi rispetto al totale delle parti prodotte.

L'OEE è quindi un numero adimensionale (per intenderci, %) che tiene quindi conto delle tre principali categorie di perdite produttive:

- Guasti, setup e attrezzaggi;
- Riduzione di velocità e microfermate;
- Scarti, rilavorazioni e perdite di resa all'avviamento.

Come detto prima, questo modello, seppure teoricamente valido, diventa di difficile applicazione nella maggioranza dei contesti produttivi.

Immaginate di avere ad esempio 6 reparti produttivi, ciascuno con macchine diverse, che realizzano un mix di prodotti variabile sia per tipologia che per quantità… riuscireste a calcolare con precisione i tre diversi fattori, pesando correttamente l'incidenza dei vari articoli/lotti/cicli di lavorazione?

Il modello innovativo

OEE = "Tempo redditizio" / Tempo disponibile

Partiamo da questo assunto: l'OEE, al pari di tutti gli indicatori di efficienza, per definizione può essere espresso attraverso un rapporto OUTPUT/INPUT.

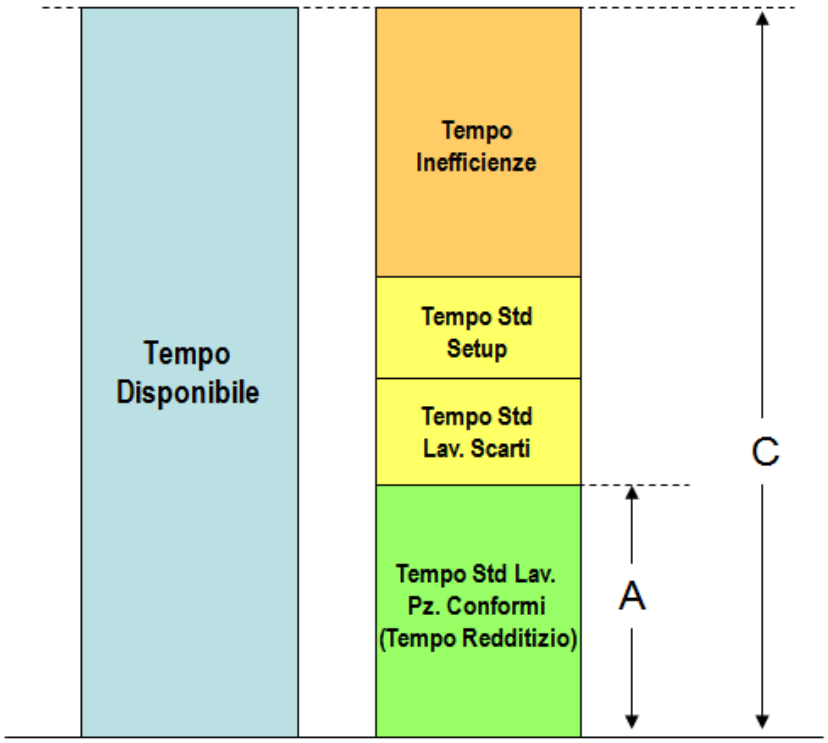
Esso dà infatti una indicazione globale sulla capacità di un insieme di risorse di produrre valore per il cliente (output) con le risorse produttive a disposizione (input).

Bene, ma come misurare output e input? Ovviamente, il numero di pezzi non va bene, a causa delle differenze nei cicli e nei tempi di produzione.

Di certo, l'input deve essere proporzionale all'impegno che l'azienda investe nel sistema produttivo, impegno che è ben rappresentato dalla disponibilità oraria delle risorse, siano esse manodopera o macchine/impianti. Non a caso, entrambe le categorie di risorse tipicamente hanno costi orari ad essi associati.

L'output deve essere espresso in una unità di misura confrontabile, e cioè temporale, in modo che l'OEE risulti un rapporto di elementi tra loro omogenei, dunque una percentuale.

In questo contesto acquista valore il concetto di tempo standard di lavoro: è il tempo necessario per l'esecuzione di una data operazione a fronte di strumenti, metodi e procedure operative stabiliti. Il tempo standard è definito dall'azienda per lo specifico ciclo di lavoro, manuale o automatico che sia: è il "tempo giusto" che serve per eseguire una lavorazione, né più, né meno. In questa accezione, il tempo standard si avvicina (anche se non è esattamente la stessa cosa) al concetto di tempo a valore aggiunto. Per non creare ambiguità, possiamo chiamarlo tempo redditizio.



OEE (Overall Equipment Effectiveness) = A / C

Il tempo standard per le attività basate su macchine o impianti a controllo numerico è tipicamente "predeterminato", in quanto dipende dal ciclo tecnologico. Per le lavorazioni manuali, invece, questo è più complesso, in quanto per la sua determinazione è necessario avvalersi di analisi sperimentali secondo le tecniche Tempi e Metodi.
Per calcolare l'oee prendiamo il tempo standard, moltiplichiamolo per il numero di pezzi processati nelle ore di lavoro considerate e otterremo il numeratore della formula (l'output). Come input, invece, useremo le ore disponibili:

$$OEE = \frac{output}{input} = \sum_{i=1}^{n} \frac{TstdLav * N°Pz.Conformi}{OreDisponibili}$$

dove i = articoli del mix produttivo realizzato

Tipicamente, un sistema produttivo che non ha mai affrontato un progetto di miglioramento dell'efficienza si attesta su valori di OEE non superiori al 50-60%.

I migliori produttori, invece, raggiungono e mantengono nel tempo un OEE pari all'85%.

Sottolineiamo il fatto che raggiungere la condizione ideale del 100% è virtualmente impossibile, in quanto rappresenterebbe un sistema che non si ferma mai e che non effettua mai attrezzaggi/setup. Se l'OEE risultasse maggiore del 100%, anzi, sarebbe sintomo di inaccuratezza del modello impostato (ad esempio, tempi standard sovradimensionati e quindi inesatti). Anche valori alti (maggiori del 70%), se rilevati in contesti che non hanno mai affrontato un processo strutturato di miglioramento dell'efficienza, devono essere validati approfonditamente.

Raggiungere un OEE dell'85% a partire, ad esempio, da una condizione di partenza del 60% (obiettivo solitamente raggiungibile in pochi mesi e senza particolari investimenti) significa aumentare l'efficienza non del 25%, bensì del 42%, in quanto la base di partenza era 60%. Nel concreto, significa produrre il 42% in più con le stesse risorse, oppure poter risparmiare il 42% di risorse a parità di produzione.

Il calcolo dell'OEE non migliora automaticamente la produttività. Esso deve essere abbinato ad una analisi dettagliata ed accurata dei motivi alla base della ridotta produttività.

Per raggiungere valori dell'85% servono non solo una buona gestione tecnica delle risorse, ma anche e soprattutto una ottima gestione organizzativa.

In questo senso, l'esperienza maturata negli anni presso numerose realtà manifatturiere può rendere molto efficace un intervento in azienda da parte di personale esterno, che può impostare correttamente il metodo, progettare le attività di miglioramento e monitorarne i risultati.

OEE (controllo e miglioramento dell'efficienza produttiva)

Essere più efficienti significa, concretamente, aumentare la produttività a parità di risorse impiegate. Questo significa che si può ottenere la stessa produttività impiegando meno risorse, oppure ottenere più produttività dalle risorse esistenti.

Spesso si è portati a pensare che il modo migliore per aumentare l'efficienza produttiva sia quello di rinnovare o potenziare gli asset tecnologici, affrontando investimenti spesso non trascurabili. L'esperienza però insegna che, in assenza di un adeguato sistema di controllo e miglioramento dell'efficienza, difficilmente un sistema produttivo esprime più del 50-60% del proprio potenziale.

Per un'azienda che voglia essere competitiva è quindi prioritario dotarsi di un efficace sistema di controllo e miglioramento dell'efficienza, in particolare sulle risorse "critiche". Questo consente all'azienda di:

- misurare l'efficienza produttiva in modo oggettivo;
- diffondere indicatori che inducono spontaneamente un aumento della produttività;
- individuare ed eliminare le fonti di inefficienza.

La tecnica universalmente riconosciuta come più efficace è il metodo OEE (Overall Equipment Effectiveness).

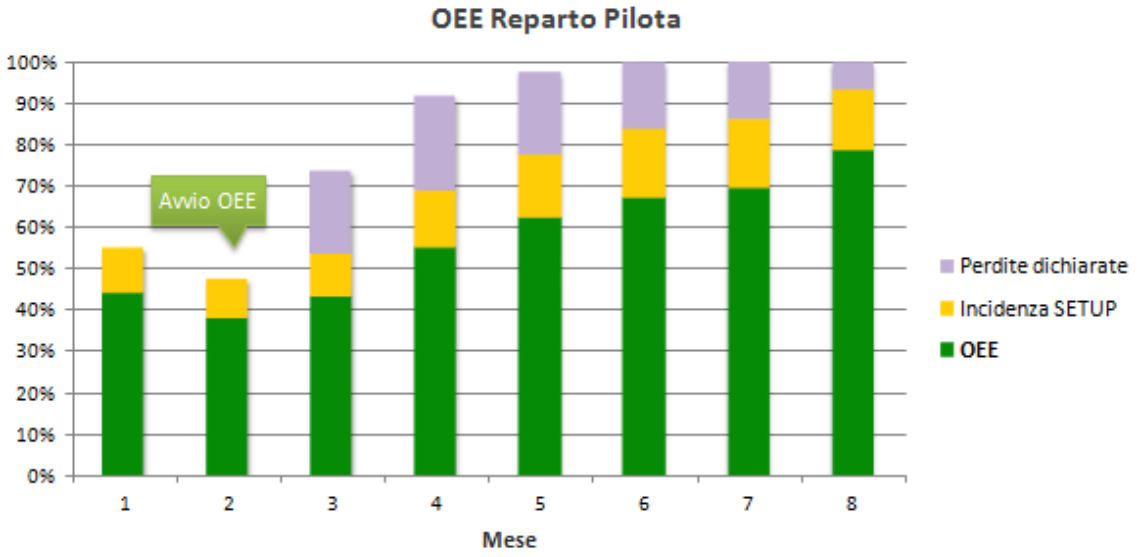
Concepita inizialmente per la produzione a flusso continuo, questa tecnica può essere implementata con successo anche in contesti produttivi per celle o per reparti, con possibilità di applicazione anche a centri di servizio che svolgono mansioni tendenzialmente ripetitive e standardizzate.

Il metodo OEE prevede alcuni step fondamentali:

- definizione/validazione degli standard (tempi e metodi)
- implementazione del sistema di registrazione delle inefficienze
- sviluppo dello strumento per il calcolo dell'efficienza e per l'analisi ABC delle inefficienze
- formazione al personale sugli obiettivi dell'attività e sul significato degli indicatori
- valutazione del livello attuale di efficienza
- individuazione delle principali fonti di inefficienza
- eliminazione delle inefficienze tramite attività di miglioramento mirate
- quantificazione del miglioramento ottenuto e feedback

La tecnica viene solitamente implementata in un'area pilota (reparto, linea produttiva o ufficio) e quindi progressivamente estesa alle altre aree. Ciò consente all'azienda di impostare un percorso sostenibile e di effettuare un fine tuning del metodo OEE sulle specificità del contesto.

La pubblicazione degli indicatori (OEE %, incidenza degli attrezzaggi, analisi delle inefficienze etc.) costituisce un riferimento condiviso per tutto il personale coinvolto.



Tipicamente, un progetto OEE consente in pochi mesi di elevare l'efficienza produttiva anche del 30-40%, a seconda della situazione di partenza, portandola alla sua massima potenzialità. Le inefficienze individuate si rivelano spesso di natura prevalentemente organizzativa, e pertanto la loro risoluzione richiede investimenti molto bassi o addirittura nulli.

Di conseguenza, l'implementazione di un progetto OEE è solitamente caratterizzato da un ritorno dell'investimento molto rapido, oltre a garantire un guadagno di efficienza significativo e stabile nel tempo.

2.4. FTQ

“First time quality” (FTQ) è definito come la percentuale di parti buone all'inizio di un ciclo produttivo, misura quindi il numero di pezzi prodotti correttamente in un processo di produzione rispetto al numero totale dei pezzi mandati in produzione.
Perché FTQ è importante?

- First time quality è un indice che permette di capire se una linea produttiva sta procedendo per il verso giusto, indicando quindi se si necessita o meno di un intervento di manutenzione per migliorare la qualità della produzione.
- Per ridurre i costi di produzione
- Eliminare gli sprechi.

Perché è necessario misurare l'FTQ?
Il motivo principale per il quale si misura il first time quality è che esso rappresenta il primo segnale di interruzione di un processo, un vero e proprio campanello di allarme che permette di intervenire cercando di limitare al minimo gli sprechi.

3. Manuale di python

Con più di un milione di siti web clienti, Python rappresenta spesso la lingua preferita dagli sviluppatori e dai data scientist che hanno bisogno di applicare tecniche statistiche o analisi dei dati nel loro lavoro.

La combinazione di sintassi coerente, i tempi di sviluppo più brevi e la flessibilità lo rendono adatto allo sviluppo di modelli di previsione sofisticati che possono essere collegati direttamente ai sistemi di produzione e all'apprendimento automatico.

Dopotutto, Python è un linguaggio di programmazione, orientato agli oggetti e di alto livello con semantica dinamica: queste caratteristiche gli permettono di esprimere idee molto potenti in pochissime righe di codice pur essendo molto leggibili.

Questo è possibile anche grazie all'ampio set di librerie, ossia insiemi di routine e funzioni scritte che svolgono un determinato compito, che possiede e può richiamare a seconda delle necessità.

Le librerie vengono spesso confuse con i termini framework e packages. Prima di vedere quali sono le più diffuse per il machine learning, vediamo di distinguere i termini tenendo conto di quanto stabilito dalla documentazione presente sul sito di Python.

3.1. Libreria

Innanzitutto, possiamo considerare la libreria come un insieme di moduli.

Ogni modulo contiene delle istruzioni e definizioni semplici. L'accorpamento di vari moduli, quindi di istruzione codice, costituisce una libreria.

Spesso i moduli sono già stati scritti da altri sviluppatori, e non c'è bisogno di ripartire da capo ogni volta. Il loro scopo è quello di semplificare le attività, aiutando gli sviluppatori a scrivere solo poche righe anziché una grande quantità di comandi.

Il codice delle librerie richiama classi e metodi che normalmente definiscono operazioni specifiche in un'area del dominio.
Ad esempio, ci sono alcune librerie di matematica che possono far sì che lo sviluppatore chiami semplicemente la funzione senza ripetere l'implementazione di come funziona un algoritmo.

3.2. Packages

Per capire cosa sono i packages si pensi alle cartelle (conosciute col nome directory) dove vengono salvati i file nel computer.
Di solito non archiviamo tutti i nostri file nella stessa posizione. Utilizziamo una gerarchia di directory ben organizzata per un accesso più semplice.
File simili sono tenuti nella stessa directory, ad esempio, potremmo conservare tutti i brani musicali nella directory "musica". Analogamente a questo, Python ha packages per directory e moduli per i file.
Dato che una directory può contenere sottodirectory e file, similmente, un pacchetto Python può avere sotto-pacchetti e moduli.
Una directory deve contenere un file chiamato __init__.py in modo che Python lo consideri come un pacchetto. Questo file può essere lasciato vuoto ma generalmente il codice di inizializzazione per quel pacchetto viene inserito in questo file.

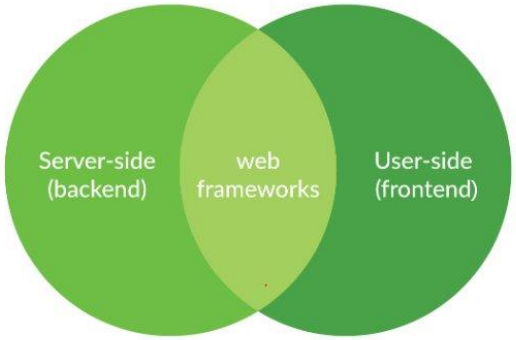
3.3. Framework

A differenza delle librerie, per framework si intende "un'astrazione", in cui il software che fornisce funzionalità generiche può essere modificato selettivamente da un ulteriore codice scritto dall'utente, fornendo così un software specifico per l'applicazione.
Si può considerare il framework come uno strumento software che fornisce un modo per creare ed eseguire applicazioni web e per farlo si avvale spesso di librerie e packages.
Utilizzando un framework web non è necessario scrivere codice per conto proprio e perdere tempo cercando possibili errori di calcolo e bug.
All'inizio dello sviluppo web, tutte le applicazioni erano codificate a mano e solo lo sviluppatore di una determinata app poteva cambiarlo o distribuirlo.
I framework Web hanno introdotto un modo semplice per uscire da questa trappola.

Dal 1995, tutte le seccature legate al cambiamento della struttura di un'applicazione sono state messe in ordine a causa della comparsa di una prestazione generale. E questo è il momento in cui sono apparse le lingue specifiche del web. La loro varietà ora funziona bene sia per pagine web statiche che dinamiche.
Possiamo avere due tipologie di Framework web:

- **Server-side**: definito anche come framework backend, sono applicazioni software che facilitano la scrittura, la manutenzione e la scalabilità delle applicazioni web. Forniscono strumenti e librerie che semplificano le comuni attività di sviluppo Web, inclusi gli URL di routing ai gestori appropriati, l'interazione con i database, le sessioni di supporto e l'autorizzazione dell'utente, l'output di formattazione (ad esempio HTML, JSON, XML) e il miglioramento della sicurezza dagli attacchi web.

- **Client-side**: definito anche framework frontend, consiste in un pacchetto costituito da una struttura di file e cartelle di codice standard (HTML, CSS, documenti JS ecc.). Si occupa essenzialmente delle parti rivolte verso l'esterno di un sito Web o di un'applicazione web. In breve, ciò che un utente vede quando apre l'app.



Esiste una terza situazione (definita Full-stack Framework) che è la combinazione di entrambe le estremità frontend e backend. Uno sviluppatore full stack è un tuttofare.
Sono responsabili per tutti i livelli di sviluppo, da come il server è impostato per il CSS relativo alla progettazione. C'è da dire che è complesso gestire entrambe le parti.

Librerie più popolari nel machine learning

Vediamo ora quali che sono le librerie più importanti di Python utilizzate nel machine learning.

Scikit-learn: fornisce una gamma di algoritmi di apprendimento supervisionato e non supervisionato tramite un'interfaccia coerente in Python. È così possibile risolvere facilmente algoritmi di regressione, di classificazione e clustering.

Anche attività quali la trasformazione di dati, la selezione delle funzionalità e i metodi di ensemble possono essere implementate in poche righe.

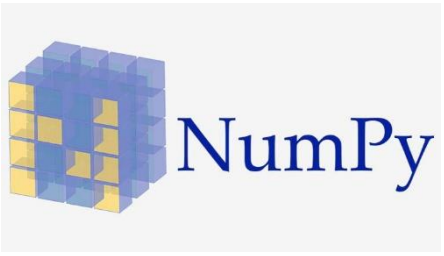
Pandas: libreria che fornisce gli strumenti per l'analisi dei dati nel linguaggio python. Il pacchetto è open source e viene fornito con diverse strutture dati che possono essere utilizzate per diverse attività di manipolazione dei dati.

Pandas è una libreria molto popolare per il recupero e la preparazione dei dati per l'uso futuro in altre librerie ML come Scikit-learn o Tensorflow.

Permette inoltre di recuperare facilmente i dati da diverse fonti: database SQL, testo, CSV, Excel, file JSON e molti altri formati meno popolari.

Una volta che i dati sono in memoria, ci sono dozzine di operazioni diverse per analizzare, trasformare, recuperare i valori mancanti, pulire il set di dati, nonché operazioni tipo SQL e un set di funzioni statistiche per eseguire anche una semplice analisi.

NumPy: sta per Numeric Python e rappresenta il pacchetto fondamentale per il calcolo scientifico con Python. NumPy è ovviamente una delle più grandi librerie di calcolo matematico e scientifico per Python.



Una delle funzionalità più importanti di NumPy è la sua interfaccia Array. Questa interfaccia può essere utilizzata per esprimere immagini, onde sonore o altri flussi binari grezzi come matrici di numeri reali con dimensione N.
La conoscenza di NumPy è molto importante per l'apprendimento automatico e la scienza dei dati.

Matplotlib: L'apprendimento automatico migliore e più sofisticato è privo di significato se non puoi comunicarlo ad altre persone.
Quindi, come si fa a trasformare effettivamente il valore da tutti questi dati che si hanno? È qui che Matplotlib viene in soccorso. È una libreria Python standard utilizzata per la creazione di diagrammi e grafici 3D. È piuttosto di basso livello, il che significa che richiede più comandi per generare grafici e figure piacevoli rispetto ad alcune librerie avanzate.
Tuttavia, il rovescio della medaglia è la flessibilità. Con abbastanza comandi, puoi creare praticamente qualsiasi tipo di grafico che desideri con Matplotlib. È possibile creare diversi grafici, da istogrammi e grafici a dispersione a grafici con coordinate non cartesiane.

Theano: rappresenta una libreria Python che consente di valutare, ottimizzare e definire espressioni matematiche che coinvolgono efficacemente gli array multidimensionali (è simile a NumPy).
Questa libreria ottimizza l'utilizzo della CPU e della GPU e migliora le prestazioni del calcolo intensivo dei dati, in quanto il codice Theano è scritto in modo tale da sfruttare il vantaggio di come funziona un compilatore del computer.
È una delle librerie di deep learning più utilizzate fino ad oggi, anche se l'ultima versione è stata rilasciata nel 2017.
TensorFlow: Secondo diversi studi Tensorflow è una delle librerie più utilizzate da chi si affaccia nel mondo del deep learning.
Tale libreria è stata sviluppato da Google, e quasi tutte le sue applicazioni utilizzano Tensorflow per l'apprendimento automatico. Se stai utilizzando le foto di Google o la ricerca vocale di Google, indirettamente stai utilizzando i modelli creati utilizzando Tensorflow.
Tensorflow è solo un framework computazionale per esprimere algoritmi che coinvolgono un gran numero di operazioni di tensori, poiché le reti neurali possono essere

espresse come grafici computazionali tramite una serie di operazioni sui tensori. I tensori sono matrici N-dimensionali che rappresentano i nostri dati.

Pytorch: rappresenta una popolare libreria di Deep Learning creata da Facebook. Oltre alla CPU, supporta calcoli accelerati dalla GPU. La libreria è focalizzata sul portare agli utenti un'esperienza di modellazione veloce e flessibile e ha ottenuto molta trazione nella comunità di Deep Learning.
Rispetto a Tensorflow, è più facile da imparare e da usare. Il rovescio della medaglia è che PyTorch è meno maturo di Tensorflow, ma la comunità sta crescendo rapidamente e ci sono già molti materiali didattici e tutorial.

Keras: è una delle librerie di apprendimento automatico più interessanti. Gli esperti consigliano di partire con questa libreria, a chi inizia a studiare il machine learning, poiché fornisce un modo più semplice per esprimere reti neurali.
Fornisce inoltre alcune utilità per l'elaborazione di set di dati, la compilazione di modelli, la valutazione dei risultati, la visualizzazione di grafici e molto altro.
Gli attuali vantaggi di Keras, a differenza di PyTorch, sono che è più maturo, ha una community più grande e un sacco di tutorial pronti all'uso. Inoltre, può utilizzare Tensorflow.
D'altra parte, PyTorch fornisce alcune caratteristiche molto interessanti (come il debug interattivo o la definizione dinamica del grafico) e sta crescendo rapidamente.
Le librerie Python appena viste semplificano di molto il machine learning (ce ne sono anche altre non citate molto utili), e hanno permesso di migliorare i risultati ottenuti dall'intelligenza artificiale.

3.4. Il linguaggio di programmazione Python

Python è un esempio di linguaggio di alto livello; altri linguaggi di alto livello sono il C, il C++, il Perl ed il Java.

Esistono anche linguaggi di basso livello, talvolta chiamati “linguaggi macchina” o “linguaggi assembly”.

In modo non del tutto corretto si può affermare che i computer possono eseguire soltanto programmi scritti in linguaggi di basso livello, i programmi scritti in un linguaggio di alto livello devono essere elaborati prima di poter essere eseguiti. Questo processo di elaborazione impiega del tempo e rappresenta un piccolo svantaggio dei linguaggi di alto livello.

I vantaggi sono d’altra parte enormi. In primo luogo, è molto più facile programmare in un linguaggio ad alto livello, questi tipi di programmi sono più veloci da scrivere, più corti e facilmente leggibili, ed è più probabile che siano corretti.

In secondo luogo, i linguaggi di alto livello sono portabili: portabilità significa che essi possono essere eseguiti su tipi di computer diversi con poche o addirittura nessuna modifica. I programmi scritti in linguaggi di basso livello possono essere eseguiti solo su un tipo di computer e devono essere riscritti per essere trasportati su un altro sistema.

Questi vantaggi sono così evidenti che quasi tutti i programmi sono scritti in linguaggi di alto livello, lasciando spazio ai linguaggi di basso livello solo in poche applicazioni specializzate.

I programmi di alto livello vengono trasformati in programmi di basso livello eseguibili dal computer tramite due tipi di elaborazione: l’interpretazione e la compilazione. Un interprete legge il programma di alto livello e lo esegue, trasformando ogni riga di istruzioni in un’azione. L’interprete elabora il programma un po’ alla volta, alternando la lettura delle istruzioni all’esecuzione dei comandi che le istruzioni descrivono:

Un compilatore legge il programma di alto livello e lo traduce completamente in basso livello, prima che il programma possa essere eseguito. In questo caso il programma di alto livello viene chiamato codice sorgente, ed il programma tradotto codice oggetto o eseguibile. Dopo che un programma è stato compilato può essere eseguito ripetutamente senza che si rendano necessarie ulteriori compilazioni finchè non ne viene modificato il codice.

Python è considerato un linguaggio interpretato perchè i programmi Python sono eseguiti da un interprete. Ci sono due modi di usare l'interprete: a linea di comando o in modo script. In modo "linea di comando" si scrivono i programmi Python una riga alla volta: dopo avere scritto una riga di codice alla pressione di Invio l'interprete la analizza subito ed elabora immediatamente il risultato, eventualmente stampandolo a video:

$ python
Python 1.5.2 (#1, Feb 1 2000, 16:32:16)
Copyright 1991-1995 Stichting Mathematish Centrum, Amsterdam
>>> print 1 + 1
2

La prima linea di questo esempio è il comando che fa partire l'interprete Python in ambiente Linux e può cambiare leggermente a seconda del sistema operativo utilizzato. Le due righe successive sono semplici informazioni di copyright del programma.
La terza riga inizia con >>>: questa è l'indicazione (chiamata "prompt") che l'interprete usa per indicare la sua disponibilità ad accettare comandi. Noi
abbiamo inserito print 1 + 1 e l'interprete ha risposto con 2.
In alternativa alla riga di comando si può scrivere un programma in un file (detto script) ed usare l'interprete per eseguire il contenuto del file. Nell'esempio seguente abbiamo usato un editor di testi per creare un file pippo.py

- print 1 + 1

Per convenzione, i file contenenti programmi Python hanno nomi che terminano con .py. Per eseguire il programma dobbiamo dire all'interprete il nome dello script:

- $ python pippo.py 2

In altri ambienti di sviluppo i dettagli dell'esecuzione dei programmi possono essere diversi.

Cos'è un programma?

Un programma è una sequenza di istruzioni che specificano come effettuare una elaborazione. L'elaborazione può essere sia di tipo matematico (per esempio la soluzione di un sistema di equazioni o il calcolo delle radici di un polinomio) che simbolico (per esempio la ricerca e sostituzione di un testo in un documento).
I dettagli sono diversi per ciascun linguaggio di programmazione, ma un piccolo gruppo di istruzioni è praticamente comune a tutti:

• input: ricezione di dati da tastiera, da file o da altro dispositivo.
• output: scrittura di dati su video, su file o trasmissione ad altro dispositivo.
• matematiche: esecuzione di semplici operazioni matematiche, quali l'addizione e la sottrazione.
• condizionali: controllo di alcune condizioni ed esecuzione della sequenza di istruzioni appropriata.
• ripetizione: ripetizione di un'azione, di solito con qualche variazione.

Cos'è il debug?

La programmazione è un processo complesso e dato che esso è fatto da esseri umani spesso comporta errori. Per ragioni bizzarre gli errori di programmazione sono chiamati bug ed il processo della loro ricerca e correzione è chiamato debug.
Sono tre i tipi di errore nei quali si incorre durante la programmazione: gli errori di sintassi, gli errori in esecuzione e gli errori di semantica. È utile distinguerli per poterli individuare più velocemente.

Errori di sintassi

Python può eseguire un programma solo se il programma è sintatticamente corretto, altrimenti l'elaborazione fallisce e l'interprete ritorna un messaggio d'errore. La sintassi si riferisce alla struttura di un programma e alle regole concernenti la sua struttura. In italiano, per fare un esempio, una frase deve iniziare con una lettera maiuscola e terminare con un punto. questa frase contiene un errore di sintassi.
Python non è così permissivo: se c'è un singolo errore di sintassi da qualche parte nel programma Python stamperà un messaggio d'errore e ne interromperà l'esecuzione, rendendo impossibile proseguire.

Errori in esecuzione

Il secondo tipo di errore è l'errore in esecuzione (o "runtime"), così chiamato perchè l'errore non appare finchè il programma non è eseguito. Questi errori sono anche chiamati eccezioni perché indicano che è accaduto qualcosa di eccezionale nel corso dell'esecuzione (per esempio si è cercato di dividere un numero per zero).

Errori di semantica

Il terzo tipo di errore è l'errore di semantica. Se c'è un errore di semantica il programma verrà eseguito senza problemi nel senso che il computer non genererà messaggi d'errore durante l'esecuzione, ma il risultato non sarà ciò che ci si aspettava, sarà qualcosa di diverso, e questo qualcosa è esattamente ciò che è stato detto di fare al computer.
Il problema sta nel fatto che il programma che è stato scritto non è quello che si desiderava scrivere: il significato del programma (la sua semantica) è sbagliato. L'identificazione degli errori di semantica è un processo complesso perché richiede di lavorare in modo inconsueto, guardando i risultati dell'esecuzione e cercando di capire cosa il programma ha fatto di sbagliato per ottenerli.

Linguaggi formali e naturali

I linguaggi naturali sono le lingue parlate, tipo l'inglese, l'italiano, lo spagnolo. Non sono stati "progettati" da qualcuno e anche se è stato imposto un certo ordine nel loro sviluppo si sono evoluti naturalmente.

I linguaggi formali sono linguaggi progettati per specifiche applicazioni.

Per fare qualche esempio, la notazione matematica è un linguaggio formale particolarmente indicato ad esprimere relazioni tra numeri e simboli; i chimici usano un linguaggio formale per rappresentare la struttura delle molecole; cosa più importante dal nostro punto di vista, i linguaggi di programmazione sono linguaggi formali che sono stati progettati per esprimere elaborazioni.

I linguaggi formali tendono ad essere piuttosto rigidi per quanto riguarda la sintassi: 3 + 3 = 6 è una dichiarazione matematica sintatticamente corretta, mentre 3 = ÷6$ non lo è, H2O ` e un simbolo chimico sintatticamente corretto contrariamente a 2Zz.

Le regole sintattiche si possono dividere in due categorie: la prima riguarda i token, la seconda la struttura. I token sono gli elementi di base del linguaggio (quali possono essere le parole in letteratura, i numeri in matematica e gli elementi chimici in chimica). Uno dei problemi con 3 = ÷6$ ` e che $ non ` e un token valido in matematica; 2Zz non ` e valido perchè nessun elemento chimico è identificato dal simbolo Zz.

Il secondo tipo di regola riguarda la struttura della dichiarazione, cioè il modo in cui i token sono disposti. La dichiarazione 3 = ÷6$ ` e strutturalmente non valida perchè un segno ÷ non può essere posto immediatamente dopo un segno =.

Allo stesso modo l'indice nelle formule chimiche deve essere indicato dopo il simbolo dell'elementi chimico, non prima, e quindi l'espressione 2Zz non è valida.

3.5. Variabili, espressioni ed istruzioni

Valori e tipi

Un valore è una delle cose fondamentali manipolate da un programmatore, come lo sono una lettera dell'alfabeto nella scrittura o un numero in matematica. I valori che vediamo sono" Hello World!" e 2, quest'ultimo il risultato ottenuto quando abbiamo sommato 1+1. Questi valori appartengono a tipi diversi: 2 è un intero, e "Hello, World!" è una stringa, così chiamata perchè contiene una serie (o "stringa") di caratteri. L'interprete può identificare le stringhe perché sono racchiuse tra virgolette.

L'istruzione print funziona sia per le stringhe che per gli interi.

Se non sei sicuro del tipo di un valore, l'interprete te lo può dire:

```
>>> type("Hello, World!")
<type 'string'>
>>> type(17)
<type 'int'>
```

Ovviamente le stringhe appartengono al tipo string e gli interi al tipo int. Non è invece intuitivo il fatto che i numeri con il punto decimale appartengano al tipo float: questi numeri sono rappresentati in un formato chiamato virgola mobile o floating-point.

```
>>> type(3.2)
<type 'float'>
```

Cosa dire di numeri come "17" e "3.2"?

Sembrano effettivamente dei numeri, ma sono racchiusi tra virgolette e questo sicuramente significa qualcosa, infatti, non siamo in presenza di numeri ma di stringhe:

```
>>> type("17")
<type 'string'>
>>> type("3.2")
<type 'string'>
```

Variabili

Una delle caratteristiche più potenti in un linguaggio di programmazione è la capacità di manipolare variabili. Una variabile è un nome che si riferisce ad un valore.
L'istruzione di assegnazione crea nuove variabili e assegna loro un valore:

```
>>> messaggio = "Come va?"
>>> n = 17
>>> pi = 3.14159
```

Questo esempio effettua tre assegnazioni. La prima assegna la stringa "Come va?" ad una nuova variabile chiamata messaggio. La seconda assegna l'intero 17 alla variabile "n" e la terza assegna il valore 3,14159 alla variabile "pi".
L'istruzione print funziona anche con le variabili:

```
>>> print messaggio
Come va?
>>> print n
17
>>> print pi
3.14159
```

ed in ogni caso il risultato è il valore della variabile.
Anche le variabili hanno il tipo; ancora una volta possiamo chiedere all'interprete a quale tipo ogni variabile appartenga:

```
>>> type(message)
<type 'string'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

Il tipo di una variabile è il tipo di valore cui essa si riferisce.

3.6. Rete neurale feed-forward e back propagation

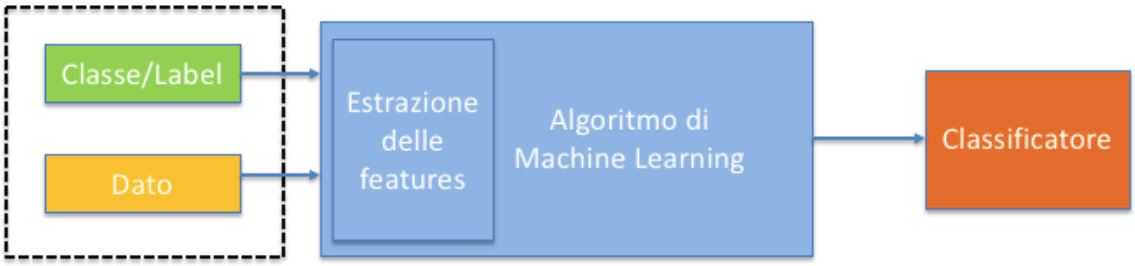
le reti neurali feed-forward sono molto importanti perché, in generale, possono essere usate per risolvere problemi in cui si ha la necessità di qualificare/classificare il dato di ingresso. Esse, infatti, costituiscono la base per creare modelli di reti neurali molto complesse, tra cui, inizio a citare per esempio, le CNN (**convolutional neural network)** usate molto spesso nella classificazione, tracciamento e confronto di immagini; queste ultime implementano differenti livelli di specializzazione e la loro caratteristica è che sono in grado di estrarre delle "features" che costituiscono, di fatto, la firma per poter distinguere un contenuto da un altro.

Le reti neurali feed-forward a più strati o multistrato, sono anche chiamate *multilayer perceptron* (**MLP**) e sono tra i modelli di rete neurale probabilmente più studiati e più utilizzati.

Utilizzare una rete neurale supervisionata significa dividere il problema in due fasi:

- Fase di training
- Fase di prediction

nella prima fase dobbiamo preoccuparci di creare un modello/classificatore addestrando la rete



mentre la seconda fase ci consente di utilizzare il modello costruito per effettuare delle predizioni o classificazioni.

Definiamo attivazione **interna** (A) del neurone la somma pesata di tutti i segnali in ingresso.

Nel caso specifico di un neurone a 4 ingressi ed una uscita si ha, ad esempio:

$$A = x1*w1 + x2*w2 + x3*w3 + x4*w4 + b$$

Ove b è una costante chiamata bias(impulso) che serve per far adattare meglio la rete e lo si può immaginare come un ulteriore neurone che non accetta valori in ingresso, mentre l'uscita Y del neurone in tal caso è

$$Y = \frac{1}{1 + e^{-A}}$$

Proviamo a capire cosa significa tutto ciò facendo qualche esempio pratico e, per semplificare ulteriormente, riduciamo ancora di più la complessità del problema utilizzando come esempio un neurone con un ingresso ed una uscita.

In tal caso, la funzione di attivazione interna (**A**) è:

$$A = x*w + b$$

mentre la Y

$$Y = \frac{1}{1+e^{-A}} = \frac{1}{1 + e^{-(x*w+b)}}$$

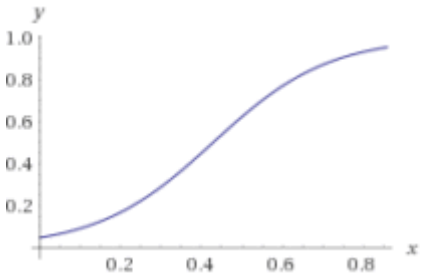
A questo punto, diventa abbastanza chiaro che, a seconda dei valori di **w** e **b** che si andranno a scegliere durante la **fase di training** (che è il nostro obiettivo) il comportamento della rete neurale tenderà a cambiare. Per capire il concetto facciamo qualche piccolo esperimento andando a scegliere randomicamente dei valori di **w** e **b**, applicandoli poi al modello del neurone. Operando per via sperimentale, possiamo apprezzare come varia la funzione di uscita del neurone e, dunque, il comportamento della rete.

Test #1

w=7

b=-3

$$\frac{1}{1+e^{-(7x-3)}}$$



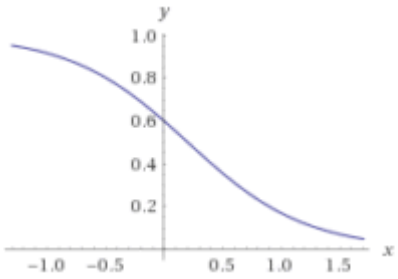
(x from 0 to 0.9)

Test #2

w=-2

b=0,4

$$\frac{1}{1+e^{-(-2x+0.4)}}$$



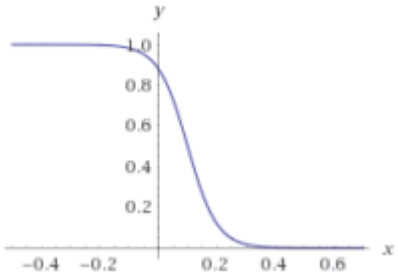
(x from −1.3 to 1.7)

Test #3

w=-20

b=2

$$\frac{1}{1+e^{-(-20x+2)}}$$



(x from −0.5 to 0.7)

Guardando i tre grafici, risulta evidente come sia diverso il comportamento del neurone al variare dei parametri **w** e **b**, la cui ricerca dei valori ottimali è "lo scopo" della fase di addestramento; in altri termini, trovare i giusti valori di **w** e **b**, significa trovare la rete neurale che si comporta meglio in fase di predizione.

Facciamo altri tre test, aumentando, però, la complessità del nostro neurone che, questa volta, avrà due ingressi ed una sola uscita.

$A = x1*w1 + x2*w2 + b$

In tal caso, la nostra Y sarà:

$$Y = \frac{1}{1 + e^{-(x_1*w_1 + x_2*w_2 + b)}}$$

Facciamo dei test anche in questo caso scegliendo valori casuali di w e b:

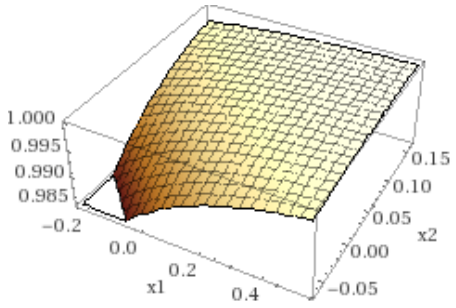
Test #4

w1=3

w2=10

b=-5

$$\frac{1}{1+e^{-(3\,x1+10\,x2+5)}}$$



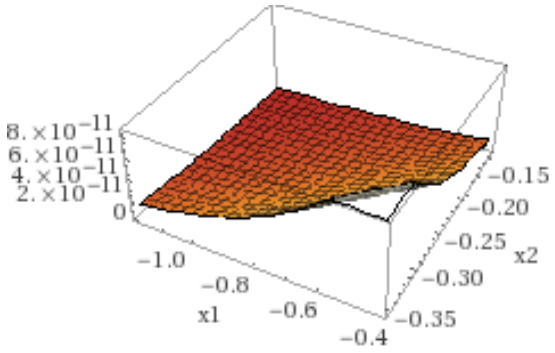
Test #5

w1=-3

w2=-10

b=-25

$$\frac{1}{1+e^{-(3\,x1-10\,x2-25)}}$$



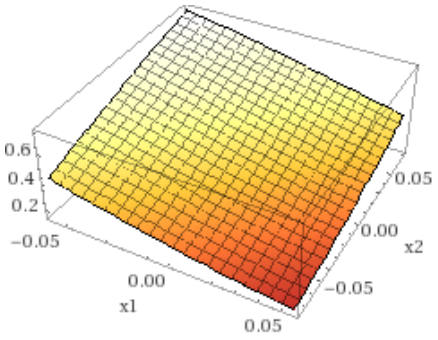
Test #6

w1=-13

w2=10

b=-0.3

$$\frac{1}{1+e^{-(-13\,x1+10\,x2-0.3)}}$$



Anche in questo caso è facile capire dalle figure come cambia sensibilmente il comportamento del neurone al variare dei pesi.

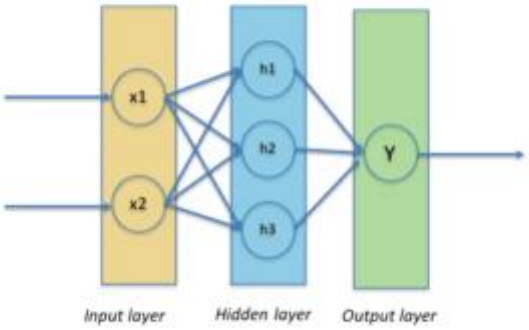
Ovviamente, utilizzare un singolo neurone non è sempre possibile e non è, al contempo, la soluzione ottimale. L'utilizzo di un singolo neurone è possibile, infatti, solo quando si devono risolvere problemi linearmente separabili, in tal caso, l'algoritmo di apprendimento tenderà a trovare una soluzione in un numero finito di passi, altrimenti, esso ciclicherà alla ricerca dei pesi senza arrivare mai ad una soluzione ottimale.

Quindi è possibile usare un singolo neurone solo quando i valori attesi possono essere descritti attraverso un piano, o comunque, curve non chiuse.

Se ciò non fosse possibile ci viene in aiuto la deep neural network, ovvero una rete neurale a più strati, con cui è possibile modellare curve più complesse.

Facciamo un esempio utilizzando una tra le più semplici deep neural network e cioè una rete neurale a due ingressi ed una sola uscita con un hidden layer a 3 neuroni.

scriviamo il modello matematico della rete neurale per la cui implementazione dobbiamo ricordarci che lo strato di input non entra nella somma pesata in quanto l'output dei neuroni dell'input layer è esattamente il vettore dei dati di ingresso.



*h1= F(x1*w11+x2*w12 + b1)*

*h2 = F(x1*w21 + x2*w22+b2)*

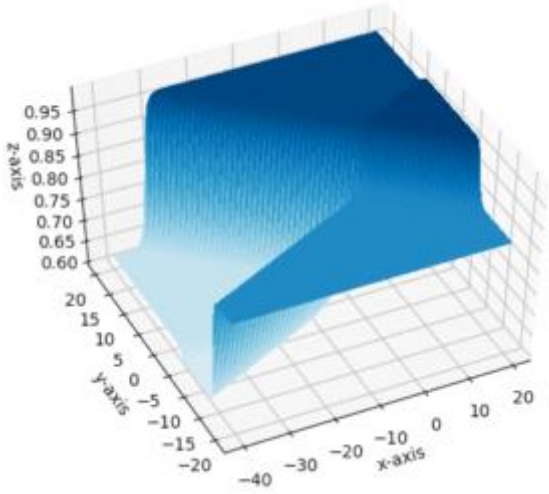
*h3 = F(x1*w31 + x2*w32+b3)*

*Y = F(h1*w1 + h2 * w2 + h3*w3 +b)*

ove F è sempre la funzione Sigmoidale, per cui, si ha che la nostra Y sarà:

$$Y = \frac{1}{1 + e^{-\left(w1 * \frac{1}{1+e^{-(x1*w11 + x2*w12 + b1)}} + w2 * \frac{1}{1+e^{-(x1*w21 + x2*w22 + b2)}} + w3 * \frac{1}{1+e^{-(x1*w31 + x2*w32 + b3)}} + b\right)}}$$

di seguito una delle possibili rappresentazioni di Y al variare dei pesi:



Tutto ciò per capire quanto complessa può diventare la rete neurale all'aumentare del numero dei neuroni e dei livelli.

A questo punto manca solo un ultimo passo, ovvero, come scegliere i pesi e con quale principio?

Anche questa volta esiste uno strumento matematico in nostro aiuto che si chiama back propagation dell'errore grazie al quale la rete è in grado di imparare e modificare in automatico i pesi confrontando, in fase di training della rete, il risultato ottenuto con il risultato atteso (quello reale).

Facciamo appello ancora una volta al nostro modello semplificato di neurone ad un solo ingresso ed una sola uscita:

$$A = x*w + b$$

In tal caso, l'algoritmo di back propagation dell'errore lavorerà secondo il seguente approccio:

$$E_{rr} = \widehat{Y} - Y$$

$$Y = \frac{1}{1 + e^{-(A)}}$$

$$Delta_y = E_{rr} * \partial(Y) = E_r * Y * (1 - Y)$$

$$w = w + (\varepsilon * Delta_y * x)$$

$$b = b + (\varepsilon * Delta_y)$$

In pratica, partendo da valori casuali di **w** e **b** si fa un test utilizzando dati di ingresso il cui valore di uscita è noto; si calcola il valore dell'errore come differenza tra il valore di uscita noto e quello ottenuto, ed il delta y come moltiplicazione tra il valore dell'errore e la derivata di Y, che, nel nostro caso, essendo una sigmoide, è proprio Y*(1-Y). A questo punto, si calcolano, semplicemente, **w** e **b** applicando la formula di cui sopra.

Il valore di epsilon, chiamato fattore di apprendimento, è a nostra scelta, ma sarà opportuno non assegnare ad esso valori troppo piccoli perché, altrimenti, si rischierebbe di non raggiungere la convergenza ottimale della rete; assegnando, altrimenti, valori troppo grandi aumenterebbe la velocità della fase di addestramento, come anche la possibilità di errori e di forti oscillazioni. I valori di epsilon dovranno essere, comunque, sempre compresi tra 0 e 1.

3.7. Definizione di classificazione

La classificazione è il processo di ricerca o scoperta di un modello (funzione) che aiuta a separare i dati in più classi categoriali. In classifica, viene identificata l'appartenenza al gruppo del problema, il che significa che i dati sono classificati in diverse etichette in base ad alcuni parametri e quindi le etichette sono previste per i dati.
I modelli derivati potrebbero essere dimostrati sotto forma di regole "IF-THEN", alberi decisionali o reti neurali, ecc. Un albero decisionale è fondamentalmente un diagramma di flusso che assomiglia a una struttura ad albero in cui ogni nodo interno rappresenta un test su un attributo, e i suoi rami mostrano l'esito del test. Il processo di classificazione affronta i problemi in cui i dati possono essere suddivisi in due o più etichette discrete, in altre parole, due o più insiemi disgiunti.
Facciamo un esempio, supponiamo di voler prevedere la possibilità della pioggia in alcune regioni sulla base di alcuni parametri. Poi ci sarebbero due etichette piove e non piove sotto le quali si possono classificare diverse regioni.

3.8. Definizione di regressione

La regressione è il processo per trovare un modello o una funzione per distinguere i dati in valori reali continui invece di utilizzare le classi. Matematicamente, con un problema di regressione, si sta cercando di trovare l'approssimazione della funzione con la deviazione minima dell'errore. Nella regressione, si prevede che la dipendenza numerica dei dati la distingua.
L'analisi di regressione è il modello statistico che viene utilizzato per prevedere i dati numerici anziché le etichette. Può anche identificare il movimento di distribuzione in base ai dati disponibili o ai dati storici.
Prendiamo l' esempio simile anche nella regressione, dove stiamo trovando la possibilità di pioggia in alcune regioni particolari con l'aiuto di alcuni parametri. In questo caso, c'è una probabilità associata alla pioggia. Qui non stiamo classificando le regioni all'interno della pioggia e non ci sono etichette per la pioggia, ma le classifichiamo con la probabilità associata.

Differenze chiave tra classificazione e regressione

Il processo di classificazione modella una funzione attraverso la quale i dati sono previsti in etichette di classi discrete. D'altra parte, la regressione è il processo di creazione di un modello che prevede la quantità continua.

Gli algoritmi di classificazione coinvolgono l'albero delle decisioni, la regressione logistica, ecc. Al contrario, l'albero di regressione (es. Foresta casuale) e la regressione lineare sono gli esempi degli algoritmi di regressione.

La classificazione prevede dati non ordinati mentre la regressione prevede i dati ordinati. La regressione può essere valutata usando l'errore quadratico medio. Al contrario, la classificazione viene valutata misurando l'accuratezza.

In conclusione, la tecnica di classificazione fornisce il modello o la funzione predittiva che prevede i nuovi dati in categorie discrete o etichette con l'aiuto dei dati storici. Viceversa, il metodo di regressione modella le funzioni a valori costanti, il che significa che predice i dati in dati numerici continui.

4. Applicazione del deep learning al caso di studio

4.1. Algoritmi per i pezzi totali prodotti

Codice per trovare gli iperparametri necessari alla previsione dei pezzi totali prodotti

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import RandomizedSearchCV
import pandas as pd

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['Pezzi totali prodotti']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
estimator=MLPRegressor(max_iter=2000, verbose=2)
param_grid = {'hidden_layer_sizes': [int(x) for x in np.linspace(start=1, stop=10,
num=5)],
        'activation': ['relu', 'tanh', 'logistic'],
        'learning_rate': ['constant', 'adaptive', 'invscaling'],
        'solver': ['adam', 'lbfgs', 'sgd']}
rs = RandomizedSearchCV(estimator, param_grid, cv=10, n_iter=20)
grid_result = rs.fit(X_train, Y_train)
best_params = grid_result.best_params_
rs_best_estimator = rs.best_estimator_
rs_score = rs.score(X_test, Y_test)
print("Iperparametri migliori: %s" % rs.best_params_)
print("Accuracy=%.4f" % rs.best_score_)
```

MLP regression per i pezzi totali prodotti

```
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['Pezzi totali prodotti']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50), max_iter=2000,
verbose=2, solver='adam', learning_rate='invscaling', activation='relu')
regr.fit(X_train, Y_train)
y_pred_train = regr.predict(X_train)
y_pred = regr.predict(X_test)
r2_score_train=r2_score(Y_train, y_pred_train)
r2_score_test=r2_score(Y_test, y_pred)
print("r2_score train=%.4f" % r2_score_train)
print("r2_score test=%.4f" % r2_score_test)
mae1=mean_absolute_error(Y_train, y_pred_train)
mae2=mean_absolute_error(Y_test, y_pred)
print("mean absolute error train=%.4f" % mae1)
print("mean absolute error test=%.4f" % mae2)
```

PCA (principal component analysis)

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['Pezzi totali prodotti']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
pca = PCA(0.90)
X_train_pca= pca.fit_transform(X_train)
X_test_pca= pca.transform(X_test)
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50), max_iter=2000,
verbose=2, solver='adam', learning_rate='invscaling', activation='relu')
regr.fit(X_train_pca, Y_train)
y_pred_train = regr.predict(X_train_pca)
y_pred = regr.predict(X_test_pca)
r2_score_train=r2_score(Y_train, y_pred_train)
r2_score_test=r2_score(Y_test, y_pred)
print("r2_score train=%.4f" % r2_score_train)
print("r2_score test=%.4f" % r2_score_test)
mae1=mean_absolute_error(Y_train, y_pred_train)
mae2=mean_absolute_error(Y_test, y_pred)
print("mean absolute error train=%.4f" % mae1)
print("mean absolute error test=%.4f" % mae2)
```

4.2. Algoritmi per l'OEE

Codice per trovare gli iperparametri necessari alla previsione dell'OEE

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import RandomizedSearchCV
import pandas as pd
dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['OEE']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
estimator=MLPRegressor(max_iter=2000, verbose=2)
param_grid = {'hidden_layer_sizes': [int(x) for x in np.linspace(start=1, stop=10,
num=5)],
        'activation': ['relu', 'tanh', 'logistic'],
        'learning_rate': ['constant', 'adaptive', 'invscaling'],
        'solver': ['adam', 'lbfgs', 'sgd']}
rs = RandomizedSearchCV(estimator, param_grid, cv=10, n_iter=20)
grid_result = rs.fit(X_train, Y_train)
best_params = grid_result.best_params_
rs_best_estimator = rs.best_estimator_
rs_score = rs.score(X_test, Y_test)
print("Iperparametri migliori: %s" % rs.best_params_)
print("Accuracy=%.4f" % rs.best_score_)
```

MLP regression per OEE

```
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['OEE']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50), max_iter=2000,
verbose=2, solver='lbfgs', learning_rate='invscaling', activation='relu')
regr.fit(X_train, Y_train)
y_pred_train = regr.predict(X_train)
y_pred = regr.predict(X_test)
r2_score_train=r2_score(Y_train, y_pred_train)
r2_score_test=r2_score(Y_test, y_pred)
print("r2_score train=%.4f" % r2_score_train)
print("r2_score test=%.4f" % r2_score_test)
mae1=mean_absolute_error(Y_train, y_pred_train)
mae2=mean_absolute_error(Y_test, y_pred)
print("mean absolute error train=%.4f" % mae1)
print("mean absolute error test=%.4f" % mae2)
```

PCA (principal component analysis)

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['OEE']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
pca = PCA(0.9)
X_train_pca= pca.fit_transform(X_train)
X_test_pca= pca.transform(X_test)
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50), max_iter=2000,
verbose=2, solver='lbfgs', learning_rate='invscaling', activation='relu')
regr.fit(X_train_pca, Y_train)
y_pred_train = regr.predict(X_train_pca)
y_pred = regr.predict(X_test_pca)
r2_score_train=r2_score(Y_train, y_pred_train)
r2_score_test=r2_score(Y_test, y_pred)
print("r2_score train=%.4f" % r2_score_train)
print("r2_score test=%.4f" % r2_score_test)
mae1=mean_absolute_error(Y_train, y_pred_train)
mae2=mean_absolute_error(Y_test, y_pred)
print("mean absolute error train=%.4f" % mae1)
print("mean absolute error test=%.4f" % mae2)
```

4.3. Algoritmi per FTQ

Codice per trovare gli iperparametri necessari alla previsione di FTQ

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import RandomizedSearchCV
import pandas as pd

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['Pezzi totali prodotti', 'Pezzi di scarto', 'Pezzi rigettati', 'FTQ', '%
scarto', 'TOTALE PEZZI BUONI', 'Disponibilità', 'PERFORMANCE', 'Qualità', 'OEE'],
axis=1)
Y = dataset['FTQ']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
estimator=MLPRegressor(max_iter=2000, verbose=2)
param_grid = {'hidden_layer_sizes': [int(x) for x in np.linspace(start=1, stop=10,
num=5)],
        'activation': ['relu', 'tanh', 'logistic'],
        'learning_rate': ['constant', 'adaptive', 'invscaling'],
        'solver': ['adam', 'lbfgs', 'sgd']}
rs = RandomizedSearchCV(estimator, param_grid, cv=10, n_iter=20)
grid_result = rs.fit(X_train, Y_train)
best_params = grid_result.best_params_
rs_best_estimator = rs.best_estimator_
rs_score = rs.score(X_test, Y_test)
print("Iperparametri migliori: %s" % rs.best_params_)
print("Accuracy=%.4f" % rs.best_score_)
```

MLP regression per FTQ

```
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error

dataset = pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['FTQ', '% scarto', 'TOTALE PEZZI BUONI', 'Disponibilità',
'PERFORMANCE', 'Qualità', 'OEE'], axis=1)
Y = dataset['FTQ']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50, 50, 50, 50),
max_iter=2000, verbose=2, solver='lbfgs',
                    learning_rate='invscaling', activation='relu')
regr.fit(X_train, Y_train)
y_pred_train = regr.predict(X_train)
y_pred = regr.predict(X_test)
r2_score_train=r2_score(Y_train, y_pred_train)
r2_score_test=r2_score(Y_test, y_pred)
print("r2_score train=%.4f" % r2_score_train)
print("r2_score test=%.4f" % r2_score_test)
mae1=mean_absolute_error(Y_train, y_pred_train)
mae2=mean_absolute_error(Y_test, y_pred)
print("mean absolute error train=%.4f" % mae1)
print("mean absolute error test=%.4f" % mae2)
```

PCA (principal component analysis)

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

dataset=pd.read_excel("venv/DATASET.1.xlsx")
X = dataset.drop(['FTQ', '% scarto', 'TOTALE PEZZI BUONI', 'Disponibilità',
'PERFORMANCE', 'Qualità',          'OEE'], axis=1)
Y = dataset['FTQ']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=20)
pca = PCA(0.9)
X_train_pca= pca.fit_transform(X_train)
X_test_pca= pca.transform(X_test)
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50, 50, 50, 50),
max_iter=2000, verbose=2, solver='lbfgs',
                learning_rate='invscaling', activation='relu')
regr.fit(X_train_pca, Y_train)
y_pred_train = regr.predict(X_train_pca)
y_pred = regr.predict(X_test_pca)
r2_score_train=r2_score(Y_train, y_pred_train)
r2_score_test=r2_score(Y_test, y_pred)
print("r2_score train=%.4f" % r2_score_train)
print("r2_score test=%.4f" % r2_score_test)
mae1=mean_absolute_error(Y_train, y_pred_train)
mae2=mean_absolute_error(Y_test, y_pred)
print("mean absolute error train=%.4f" % mae1)
print("mean absolute error test=%.4f" % mae2)
```

4.4. Descrizione codice per trovare gli iperparametri

Nella prima parte del codice ho importato le librerie e le funzioni necessarie al funzionamento del mio scrtipt, successivamente tramite la funzione pd.read è possibile accedere ai dati del dataset fornitomi, questi dati vengono divisi in due parti e vengono assegnati alle variabili X e Y.

Alla variabile X viene associato il dataset e con la funzione "drop" elimino le colonne superflue, i valori di X serviranno per la previsione del valore di Y, che nel mio caso corrisponde ai "pezzi totali prodotti", "oee" e "ftq".

La funzione train_test_split ci consente di dividere il dataset in una parte di train e una parte di test, in questo caso avendo specificato test_size=0,3, avremo che il codice usa il 70% dei dati per apprendere la relazione che c'è tra le variabili X e Y, mentre il 30% è usato per fare previsioni sul valore di Y tramite le relazioni ricavate durante l'addestramento.

Per scegliere gli iperparametri da inserire nella rete neurale utilizzo la funzione "RandomizedSearchCV", quest'ultima confronta i diversi parametri per andare poi a scegliere quali di essi mi garantiscono la migliore accuratezza.

I parametri da trovare nella random search sono :

- Numero di hidden layer (ho inserito un ciclo per andare da un minimo di 1 ad un massimo di 10 livelli, ovviamente più livelli nascosti ci sono e piu la ricerca sarà lunga, si potrebbero avere anche delle elaborazioni di alcune ore);
- Funzione di attivazione;
- Learning rate;
- Solver.

Per la funzione di attivazione possiamo scegliere tra :

- 'logistic', la funzione sigmoide logistica, restituisce f (x) = 1 / (1 + exp (-x));
- 'tanh', la funzione tan iperbolica, restituisce f (x) = tanh (x);
- 'relu', la funzione di unità lineare rettificata, restituisce f (x) = max (0, x).

Il **learning rate** indica la modifica dei pesi (weights*)* di una rete neurale profonda (Deep Neural Network) e costituisce uno degli **hyperparameters** più delicati e importanti da **regolare** (tune) per raggiungere ottime performance sul nostro problema.

- 'constant' è un tasso di apprendimento costante dato da 'learning_rate_init'(il tasso di apprendimento iniziale);
- 'invscaling' riduce gradualmente la velocità di apprendimento ad ogni passo temporale 't';
- "Adaptive" mantiene il tasso di apprendimento costante a "learning_rate_init" fino a quando la training loss continua a diminuire, la loss (perdita) viene calcolata sui set di pattern di allenamento e validazione (training e validation set) e quindi è limitata a valutare quanto il modello si sta comportando bene con questi due set. Ogni volta che due epoche consecutive non riescono a ridurre la loss, o non riescono ad aumentare il punteggio di convalida, il tasso di apprendimento corrente viene diviso per 5.

Abbiamo tre diversi risolutori:

- "lbfgs" è un ottimizzatore nella famiglia dei “metodi quasi-Newton”;
- 'sgd' si riferisce alla discesa del gradiente stocastico;
- 'adam' si riferisce a un ottimizzatore basato sul gradiente stocastico proposto da Kingma, Diederik e Jimmy Ba.

Il solutore predefinito "adam" funziona abbastanza bene su set di dati relativamente grandi (con migliaia di campioni di addestramento o più) in termini di tempo di addestramento e punteggio di convalida. Per set di dati di piccole dimensioni, tuttavia, "lbfgs" può convergere più velocemente e ottenere prestazioni migliori.

Le ultime due righe di codice mi permettono di avere in output quali sono i migliori iperparametri e quale sarà l’accuratezza delle previsioni fatte usando quest’ultimi.

4.5. Descrizione codice per la regressione

La prima parte del codice serve per importare le librerie necessarie, per importare il dataset ed assegnarlo alle due variabili.

Anche in questo caso si usa la funzione "train_test_split" necessaria a dividere i dati in una parte di train ed una di test.

Vado ora a creare la rete neurale con la funzione "MLPregressor", all'interno della funzione si devono specificare tutti gli iperparametri trovati dal codice precedente per ottimizzare al meglio la previsione, oltre al numero di hidden layer si scrivono anche il numero di neuroni per ogni livello.

Dopo aver allenato il sistema con la funzione "fit" si possono eseguire le previsioni con la funzione "predict".

Infine, introduco due parametri, che andranno a fare delle misure sia sul train che sul test per vedere quanto il mio sistema è affidabile, essi sono:

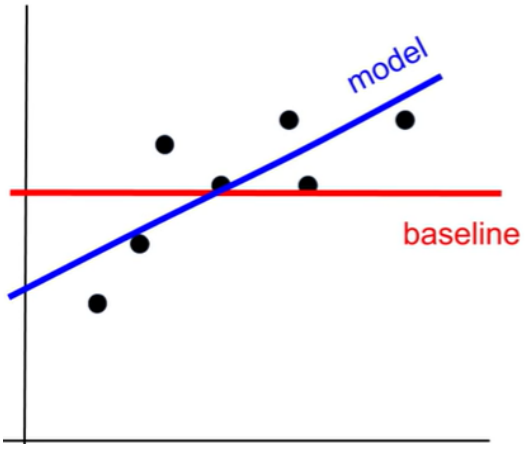
- Errore quadratico medio (MSE - Mean Squared Error).
- R2 score

L'errore quadratico medio consiste nella media della somma degli errori elevati al quadrato.

Gli errori vengono elevati al quadrato in modo da avere tutti valori positivi, se avessimo valori sia positivi che negativi rischiamo che la loro somma sia nulla, avremo quindi un'indicazione errata.

Il valore del MSE può andare da 0 a infinito, con 0 si hanno ottime prestazioni, più il valore cresce e più le prestazioni peggiorano.

R2 fa leva sul valore del MSE, infatti misura l'errore quadratico medio sia del mio modello, ma anche di un modello immaginario (baseline), questo modello fornisce sempre una risposta costante nel tempo, il cui valore è la media delle risposte desiderate.



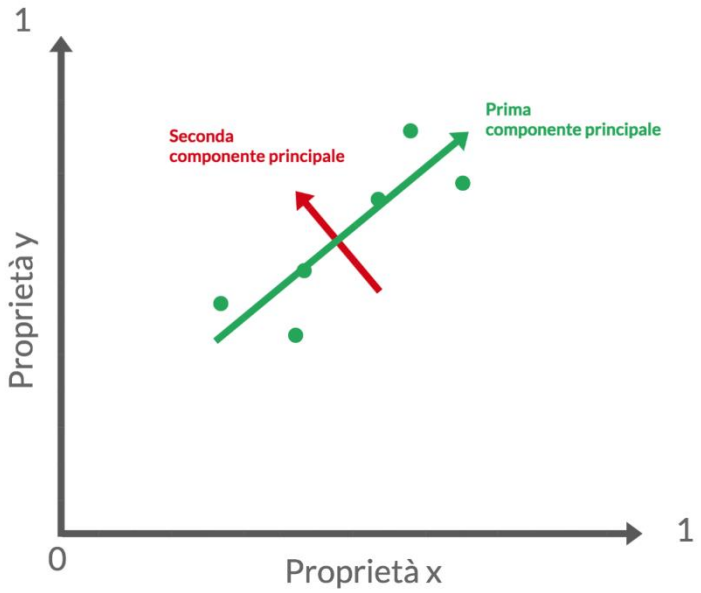
$$R^2 = 1 - \frac{MSE}{MSE_{base}}$$

Se l’errore quadratico medio del mio modello e quello della baseline fossero uguali otterrei un valore di R2 uguale a 0 (prestazione basilare), se invece l’errore del mio modello fosse inferiore rispetto a quello della baseline, otterrei un valore di R2 che si avvicina a 1 (il valore 1 equivale ad una prestazione perfetta), le prestazioni del mio modello sono migliori rispetto alla baseline.
Il valore di R2 può essere anche negativo nel caso in cui avessi un modello estremamente scarso, se l’errore del mio modello fosse maggiore di quello della baseline, ottenengo un rapporto tra i due errori maggiore di 1.

4.6. Descrizione codice per la principal component analysis (pca)

In questo codice è stata eseguita la regressione tramite rete neurale, si può notare come parte del codice sia la stessa del programma precedente ad eccezione della principal component analysis.

La PCA consiste nell'identificare le direzioni di maggiore varianza nei dati, che vengono chiamate componenti principali (dall'inglese principal components). La prima componente principale è la direzione di maggior varianza, le altre sono le direzioni di maggior varianza ortogonali alle precedenti. L'ortogonalità ci assicura che ogni componente sia indipendente dalle altre e quindi contenga informazioni differente.



La PCA, come le altre tecniche di riduzione della dimensionalità, può essere utilizzata principalmente per due scopi, visualizzare i dati in 2 o 3 dimensioni oppure velocizzare la fase di addestramento.

Nella PCA si possono selezionare il numero di componenti principali che ci permettono di ridurre la complessità del modello mantenendo la maggior quantità possibile di informazione (cioè di varianza).

Nel mio caso piuttosto che specificare il numero di componenti, specifico il numero di varianza che vogliamo mantenere.

Rinunciando al 10% di varianza abbiamo ridotto il tempo di addestramento però le metriche del nostro modello sono lievemente peggiorate, ma questo è normale avendo ridotto l'informazione disponibile.

Riducendo il valore della varianza si riduce il numero di proprietà e quindi il numero di pesi da ottimizzare, ne segue che l'addestramento è più veloce, ma allo stesso tempo si perdono informazioni e quindi le performance del nostro modello peggioreranno.

5. Conclusioni

Vengono ora riportati i risultati ottenuti dagli algoritmi prodotti e descritti nel capitolo precedente, i primi due risultati sono frutto della random search, vediamo quindi quali sono i migliori iperparametri per la costruzione delle reti neurali e qual è l'accuratezza con la quale opereranno.

Passando al secondo algoritmo, ovvero il modello usato per la previsione, ne viene misurata l'efficienza tramite l'errore quadratico medio (mae) e R2 score (entrambi descritti nel capitolo 4).

Questi due parametri sono stati valutati sia su dati di test che di training, i risultati che effettivamente ci interessano sono quelli sui dati di test, sarà infatti quella la precisione che otterremo per le previsioni.

Gli algoritmi sono stati eseguiti più volte in modo da riportare più valori in output per controllare che il risultato fosse più o meno costante nelle diverse esecuzioni.

R2 score solitamente è un valore compreso tra 0 e 1, può essere negativo nel caso di un modello estremamente scarso, tutti i valori ottenuti dai modelli sono risultati ottimi, si può quindi affermare di aver ottenuto una precisione di circa il 90%, raggiungendo valori molto elevati specialmente nel caso dei "pezzi totali prodotti".

Anche nel caso dell'errore quadratico medio i risultati ottenuti sono ottimi, ad esempio nel caso dell'Oee i valori misurati sono 0,0254 nella prima previsione e 0,0322 nella seconda, l'errore ottenuto è veramente basso.

Sono infine riportati i dati ottenuti dopo aver effettuato la principal component analysis, anche qui l'efficienza è stata misurata con i due parametri usati anche nel caso precedente.

Con la Pca sono diminuiti i tempi per l'esecuzione, ma sono peggiorati i valori ottenuti, nel caso dei pezzi totali prodotti mentre prima si aveva un'elevata precisione, ora si ha un'efficienza del 70-80%, ed anche l'errore mentre prima era di circa 10 unità (valore molto piccolo dato che le unità prodotte si aggirano sempre attorno ai 1000 pezzi), ora si ha un errore di 104,2264 nella prima previsione e di 62,4562 nella seconda.

Iperparametri per i pezzi totali prodotti

Iperparametri migliori: {'solver': 'adam', 'learning_rate': 'invscaling',
'hidden_layer_sizes': 7, 'activation': 'relu'}
Accuracy=0,9919

Risultati

r2_score train=0,9970
r2_score test=0,9967
mean absolute error train=9,1125
mean absolute error test=9,7563

r2_score train=0,9965
r2_score test=0,9955
mean absolute error train=10,5653
mean absolute error test=12,1270

Risultati dopo aver utilizzato la PCA

r2_score train=0,8710
r2_score test=0,7427
mean absolute error train=86,1551
mean absolute error test=104,2264

r2_score train=0,9367
r2_score test=0,8249
mean absolute error train=45,3232
mean absolute error test=62,4562

Iperparametri per l’OEE

Iperparametri migliori: {'solver': 'lbfgs', 'learning_rate': 'invscaling', 'hidden_layer_sizes': 7, 'activation': 'relu'}
Accuracy=0,9023

Risultati

r2_score train=0,9714
r2_score test=0,9432
mean absolute error train=0,0184
mean absolute error test=0,0254

r2_score train=0,9796
r2_score test=0,8956
mean absolute error train=0,0150
mean absolute error test=0,0322

Risultati dopo aver utilizzato la PCA

r2_score train=0,9742
r2_score test=0,8790
mean absolute error train=0,0150
mean absolute error test=0,0373

r2_score train=0,9777
r2_score test=0,8603
mean absolute error train=0,0130
mean absolute error test=0,0387

Iperparametri per FTQ

Iperparametri migliori: {'solver': 'lbfgs', 'learning_rate': 'invscaling',
'hidden_layer_sizes': 10, 'activation': 'relu'}
Accuracy=0,5740

Risultati

r2_score train=0,9916
r2_score test=0,8771
mean absolute error train=0,0041
mean absolute error test=0,0101

r2_score train=0,9963
r2_score test=0,9756
mean absolute error train=0,0030
mean absolute error test=0,0059

Risultati dopo aver usato la pca

r2_score train=0,9370
r2_score test=0,9215
mean absolute error train=0,0103
mean absolute error test=0,0121

r2_score train=0,9602
r2_score test=0,9033
mean absolute error train=0,0075
mean absolute error test=0,0136

Partendo dagli algoritmi precedenti e inserendo una variabile chiamata X_test dove poter scrivere i valori di tutti i parametri presenti nel dataset è possibile calcolare il valore di Oee, Ftq e pezzi totali prodotti, un esempio è il seguente codice.

```
from sklearn.model_selection import train_test_split
import pandas as pd
dataset=pd.read_excel("venv/DATASET.1.xlsx")
dataset1 = ['Mese','Giorno','Turno','Disponibilità attuale (MINUTI TURNO)','MENSA','BLACK OUT','FERMI NON PROGRAMMATI \ RIGENERAZIONE UTENSILI','FERMI PROGRAMMATI \ RIGENERAZIONE UTENSILI','MANUTENZIONE PM PROGRAMMATA','PULIZIE TECNICHE','SET-UP Cambio Tipo','MICROFERMATE','MANCANZA MATERIALI DIRETTI','MANCANZA ALIMENTAZIONE DA POSTAZIONE A MONTE','Fermo problemi Qualità','GUASTI - RISOLTI PM','GUASTI - RISOLTI AM','Totale Min riparazione PM','Totale Min riparazione AM','OP_OP_10 AM','OP_OP_10 PM','OP_OP_30.1 AM','OP_OP_30.1 PM','OP_OP_30.2 AM','OP_OP_30.2 PM','OP_OP_40.1 AM','OP_OP_40.1 PM','OP_OP_40.2 AM','OP_OP_40.2 PM','OP_OP_50 AM','OP_OP_50 PM','OP_OP_60.1 AM','OP_OP_60.1 PM','OP_OP_60.2 AM','OP_OP_60.2 PM','OP_OP_70 AM','OP_OP_70 PM','OP_OP_80 AM','OP_OP_80 PM','OP_OP_90 AM','OP_OP_90 PM','OP_OP_100 AM','OP_OP_100 PM','OP_OP_110 AM','OP_OP_110 PM','OP_OP_120 AM','OP_OP_120 PM','TEMPO OPERATIVO', 'Pezzi totali prodotti']
dataset=dataset[dataset1]
dataset=dataset.drop([1], axis=0)
X = dataset.drop('Pezzi totali prodotti', axis=1)
Y = dataset['Pezzi totali prodotti']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=20)
from sklearn.neural_network import MLPRegressor
regr = MLPRegressor(hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50), max_iter=2000, verbose=2, solver='adam', learning_rate='invscaling', activation='relu')
regr.fit(X_train, Y_train)
X_test=[(1,2,2,450,30,0,0,0,0,0,0,0,0,0,0,60,30,90,30,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,360)]
y_pred=regr.predict(X_test)
print(y_pred)
```

In questo codice viene fatta la previsione dei pezzi totali prodotti partendo dai valori presenti nella seconda riga del dataset, isolando prima la stessa riga in modo tale che il sistema non conoscesse il valore cercato in output, ma lo apprendesse dalle altre righe.
Il valore dei pezzi totali prodotti presente nel dataset è di 1262 unità, il codice viene eseguito diverse volte il codice in modo da vedere i risultati di più prove:

- 1276,14775939
- 1285,1234603
- 1285,22287251
- 1278,34206016
- 1280,16205347

Nel caso del calcolo dell'OEE il suo valore nella terza riga del dataset è di 0,649370833333333, i valori ottenuti tramite diverse previsioni sono:

- 0,66999341
- 0,68328367
- 0,61629336
- 0,68623151
- 0,62455496

Nel caso del calcolo di FTQ, utilizzando i valori della seconda riga dovrei ottenere secondo il dataset la cifra 0,887480190174326, con il sistema creato ho ottenuto i seguenti valori:

- 0,93518997
- 0,92218542
- 0,89834811
- 0,89122716
- 0,85836538

Nelle diverse prove fatte per questi casi si hanno valori molto vicini a quelli presenti nel dataset.
Questi codici possono essere utilizzati per ricavare i dati relativi anche ad altre date o ad altri turni di lavoro cambiando i valori dei diversi parametri presenti nella variabile X_test, inoltre se volessimo trovare i valori di altri parametri quali "Performance" o "Disponibilità" si potrebbero riutilizzare gli stessi codici opportunamente ottimizzati, cambiando quindi gli iperparametri usati per la costruzione della rete neurale con quelli ottenuti dalla random search che permetteranno di ottenere dati in output molto simili a quelli presenti nel dataset.

6. Sitografia

https://www.manutenzione-online.com/articolo/il-machine-learning-a-supporto-della-diagnostica/

www.organizzazioneaziendale.net

https://it.wikipedia.org/wiki/Magneti_Marelli

https://www.marelli.com/it/our-history/

https://lorenzogovoni.com/python-librerie-machine-learning/

https://it.wikipedia.org/wiki/Apprendimento_profondo

https://en.wikipedia.org/wiki/Deep_learning

https://www.intelligenzaartificiale.it/

https://it.mathworks.com/discovery/deep-learning.html

https://www.ai4business.it/

https://www.internetpost.it/deep-learning-storia-lunga-60-anni/#:~:text=Nel%201958%20lo%20psicologo%20Frank,computer%20grande%20come%20una%20stanza.

https://lorenzogovoni.com/

https://www.domsoria.com/2018/04/rete-neurale-feed-forward-e-back-propagation/

https://it.gadget-info.com/difference-between-classification

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

https://www.udemy.com/course/machine-learning-pratico/learn/lecture/12656414?start=540#overview

https://it.wikipedia.org/wiki/Rete_neurale_artificiale#:~:text=Nel%20campo%20dell'apprendimento%20automatico,di%20una%20rete%20neurale%20biologica.

https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/

https://www.intelligenzaartificiale.it/reti-neurali/

https://www.spindox.it/it/blog/ml1-reti-neurali-demistificate/