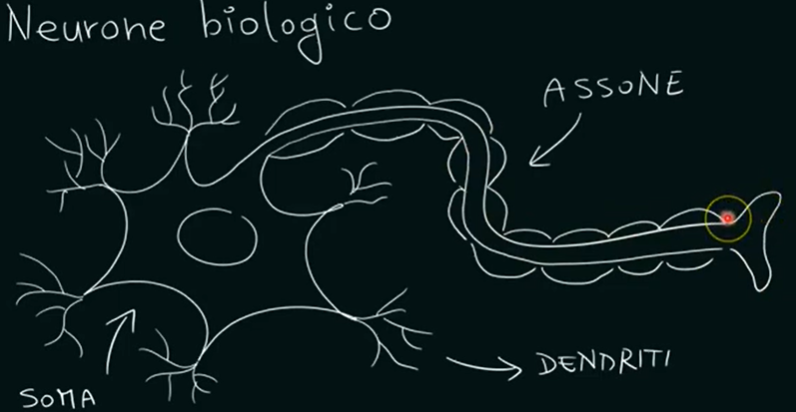
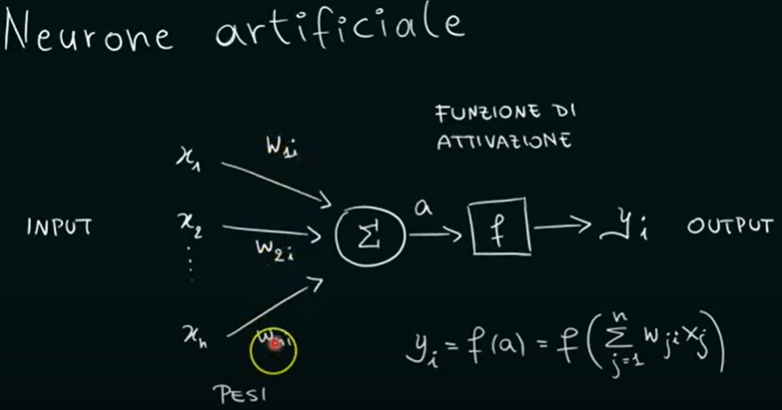
**RETI NEURALI**

Per comprendere una rete neurale dobbiamo partire dal concetto partire dalla rappresentazione biologica del neurone che ci porterà a definire un neurone della rete neurale(conosciuto come percettrone).



Un neurone biologico è composto dal soma il corpo centrale e dai dendriti terminazioni nervose da cui il neurone riceve input che andrà ad elaborare, l’assone serve per trasmettere informazioni ad altri neuroni in particolare quando le informazioni ricevute attivano il neurone esso scarica lungo l’assone (uno spike ovvero un impulso elettrico).

Partendo da questa struttura che permette begli uomini l’apprendimento vogliamo imitare una struttura artificiale che permetta alle macchine di apprendere. Andiamo a definire un neurone artificiale in termini matematici come:

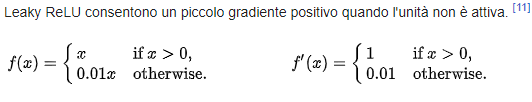


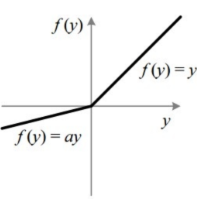
Gli input sono rappresentati dalle varie xi ogni input serve per effettuare un calcolo tramite una combinazione lineare con dei pesi wi presenti sugli archi che collegano gli input ad un nodo (interno di attivazione) di fatto il nodo va ad effettuare la così detta somma pesata ovvero una combinazione lineare tra gli input e i pesi ad essi associati il risultato di questa combinazione lineare sarà passato alla funzione di attivazione la quale schiaccerà il risultato ottenuto in un range desiderato [0,1] {tra le funzioni di attivazione normalmente utilizzate ritroviamo funzione a gradino, funzione lineare, sigmoide, ReLU, leakyReLU e tangente iperbolica

}:

**Leaky ReLU**

**Leaky Rectified Linear Unit** , o **Leaky ReLU** , è un tipo di funzione di attivazione basata su una [ReLU](https://paperswithcode.com/method/relu) , ma ha una piccola pendenza per i valori negativi invece di una pendenza piatta. Il coefficiente di pendenza viene determinato prima dell'allenamento, cioè non viene appreso durante l'allenamento. Questo tipo di funzione di attivazione è popolare nelle attività in cui possiamo soffrire di gradienti sparsi, ad esempio l'addestramento di reti avversarie generative.





**ReLU**

Nel contesto [delle reti neurali artificiali](https://en.wikipedia.org/wiki/Artificial_neural_network) , la funzione di attivazione **raddrizzatore** o **ReLU (unità lineare rettificata)**[[1]](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#cite_note-brownlee-1)[[2]](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#cite_note-medium-relu-2) è una [funzione di attivazione](https://en.wikipedia.org/wiki/Activation_function) definita come la parte positiva del suo argomento:



dove *x* è l'input di un neurone. Questa è anche nota come [funzione di rampa](https://en.wikipedia.org/wiki/Ramp_function) ed è analoga alla rettifica a [semionda](https://en.wikipedia.org/wiki/Half-wave_rectification)[nell'ingegneria elettrica](https://en.wikipedia.org/wiki/Electrical_engineering) . Nel 2011 si è scoperto che consente un migliore addestramento di reti più profonde, [[7]](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#cite_note-glorot2011-7) rispetto alle funzioni di attivazione ampiamente utilizzate prima del 2011, ad esempio il [sigmoide logistico](https://en.wikipedia.org/wiki/Logistic_function) (che si ispira alla [teoria della probabilità](https://en.wikipedia.org/wiki/Probability_theory) ; vedi [regressione logistica](https://en.wikipedia.org/wiki/Logistic_regression) ) e la sua controparte più pratica [[8]](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#cite_note-8)[, la tangente iperbolica](https://en.wikipedia.org/wiki/Hyperbolic_tangent). Il raddrizzatore è, a partire dal 2017 , la funzione di attivazione più popolare per [le reti neurali profonde](https://en.wikipedia.org/wiki/Deep_learning) .

Le unità lineari rettificate, rispetto alla [funzione sigmoidea](https://en.wikipedia.org/wiki/Sigmoid_function) o funzioni di attivazione simili, consentono un addestramento più rapido ed efficace delle architetture neurali profonde su set di dati ampi e complessi.

**Potenziali problemi della ReLU**

* Non differenziabile a zero; tuttavia, è differenziabile altrove e il valore della [derivata](https://en.wikipedia.org/wiki/Derivative) a zero può essere arbitrariamente scelto tra 0 o 1.
* Non centrato sullo zero.
* Illimitato.
* Problema ReLU morente: i neuroni ReLU (unità lineare rettificata) a volte possono essere spinti in stati in cui diventano inattivi essenzialmente per tutti gli input. In questo stato, nessun gradiente scorre all'indietro attraverso il neurone, quindi il neurone rimane bloccato in uno stato perennemente inattivo e "muore". Questa è una forma del [problema del gradiente evanescente](https://en.wikipedia.org/wiki/Vanishing_gradient_problem) . In alcuni casi, un gran numero di neuroni in una rete può rimanere bloccato in stati morti, diminuendo di fatto la capacità del modello. Questo problema si verifica in genere quando il tasso di apprendimento è troppo alto. Può essere mitigato utilizzando invece ReLU che perdono, che assegnano una piccola pendenza positiva per *x*  <0; tuttavia, le prestazioni sono ridotte.

GeLU Unità lineare dell’errore Gaussiano GELU è un'approssimazione regolare al raddrizzatore:

dove Φ( *x* ) è la funzione di [distribuzione cumulativa della distribuzione](https://en.wikipedia.org/wiki/Cumulative_distribution_function)[normale](https://en.wikipedia.org/wiki/Normal_distribution) standard . Questa funzione di attivazione è illustrata nella figura all'inizio di questo articolo. Ha un "urto" non monotono quando *x* < 0 e funge da attivazione predefinita per modelli come [BERT](https://en.wikipedia.org/wiki/BERT_(language_model)) .

La SiLU (unità lineare sigmoidea) o [funzione swish](https://en.wikipedia.org/wiki/Swish_function) è un'altra approssimazione regolare, coniata per la prima volta nel documento GELU:

Dove sigmoid(x){\displaystyle \operatorname {sigmoide} (x)} è la [funzione sigmoidea](https://en.wikipedia.org/wiki/Sigmoid_function) .

**SIGMOIDE**

Una **funzione sigmoidea** è una [funzione matematica](https://en.wikipedia.org/wiki/Mathematical_function) con una caratteristica curva a forma di "S" o **curva sigmoidea** .

Un esempio comune di funzione sigmoidea è la [funzione logistica](https://en.wikipedia.org/wiki/Logistic_function) mostrata nella prima figura e definita dalla formula:

****

**Chart, histogram

Description automatically generated**

Una funzione sigmoidea è una funzione reale [limitata](https://en.wikipedia.org/wiki/Bounded_function) , [differenziabile che è definita per tutti i valori di input reali e ha una derivata non negativa in ogni punto](https://en.wikipedia.org/wiki/Differentiable_function)[[1]](https://en.wikipedia.org/wiki/Sigmoid_function#cite_note-Han-Morag_1995-1) ed esattamente un [punto di flesso](https://en.wikipedia.org/wiki/Inflection_point) .

**Funzione a gradino (conosciuta dagli informatici come binary treshold)**

 la **funzione gradino di Heaviside** o **funzione a gradino unitaria**, il cui nome si deve a [Oliver Heaviside](https://it.wikipedia.org/wiki/Oliver_Heaviside), è una [funzione](https://it.wikipedia.org/wiki/Funzione_(matematica)) [discontinua](https://it.wikipedia.org/wiki/Funzione_continua) che ha valore [zero](https://it.wikipedia.org/wiki/0_(numero)) per argomenti negativi e [uno](https://it.wikipedia.org/wiki/1_(numero)) per argomenti positivi. Può essere definita sia come una [funzione continua a tratti](https://it.wikipedia.org/wiki/Funzione_definita_a_tratti) o come una [distribuzione](https://it.wikipedia.org/wiki/Distribuzione_(matematica)).

**Chart, shape, rectangle

Description automatically generated**

La [derivata](https://it.wikipedia.org/wiki/Derivata) distribuzionale della funzione di Heaviside  {\displaystyle H} è la [delta di Dirac](https://it.wikipedia.org/wiki/Delta_di_Dirac) {\displaystyle \delta (x)} :

****

mentre la [funzione rampa](https://it.wikipedia.org/wiki/Funzione_rampa) {\displaystyle R}R ne è la [primitiva](https://it.wikipedia.org/wiki/Primitiva_(matematica)):

Text

Description automatically generated

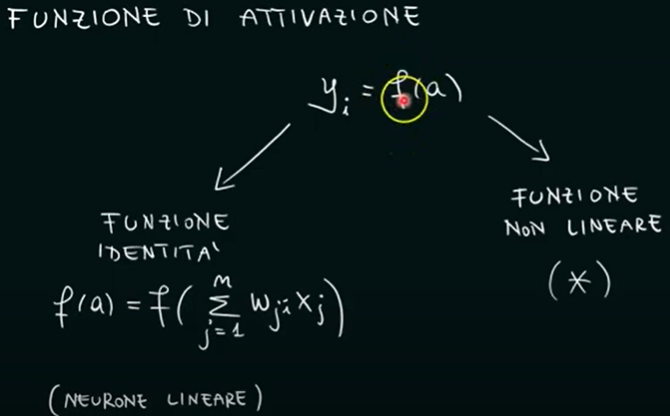
L’output di un neurone artificiale può essere o un numero reale [0,1] oppure reale tra [-1,1] o discreto {0, 1}, il caso più semplice di output di neurone è quello in cui la funzione di attivazione coincide con la funzione identità e in tal caso l’output sarà proprio la combinazione lineare dei pesi con gli input.

**TOPOLOGIE DELLE RETI NEURALI**

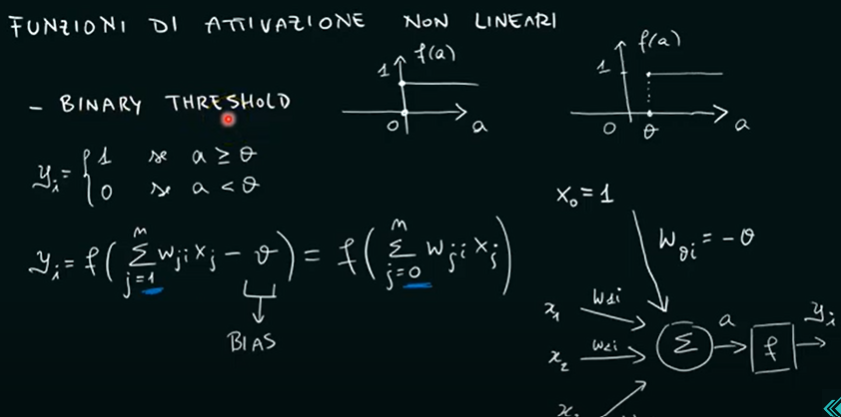
****

La prima sulla destra è una topologia totalmente connessa ovvero i neuroni sono tutti connessi tra di loro (è la meno usata tra le 3 tipologie). La seconda tipologia (centrale) è la classica **rete feed forward**, nella quale di base almeno abbiamo 3 strati uno di input layer input i cui neuroni ricevono gli input (le features dei dati) , uno nascosto hidden layer (questi sono da 1 obbligatorio a + strati) i quali neuroni sono detti neuroni nascosti e processano le informazioni ad un livello intermedio, infine abbiamo il livello di output che produce il risultato finale (la predizione della rete). La terza tipologia è detta **rete ricorrente**  e la caratterristica è che uno o più output della rete sono mandati nuovamente in input alla rete stesse tramite un meccanismo di retroazione del feedback.

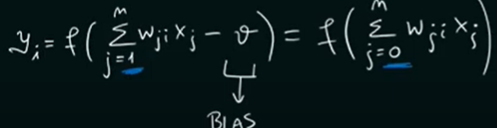
**FUNZIONI DI ATTIVAZIONE**



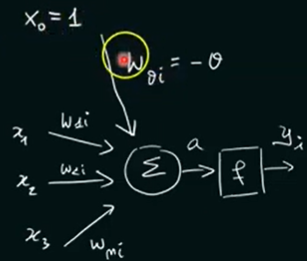
La funzione di attivazione ci permette di trovare l’output del neurone ovvero yi essa f(a) riceve la somma pesata(combinazione lineare di pesi e input). La prima distinzione che possiamo fare tra le funzioni di attivazione consiste in separare la funzione di identità dalle funzioni non lineari. La funzione identità risulta essere la combinazione lineare ricevuta, nel caso invece di funzioni non lineari abbiamo le funzioni precedentemente descritte:



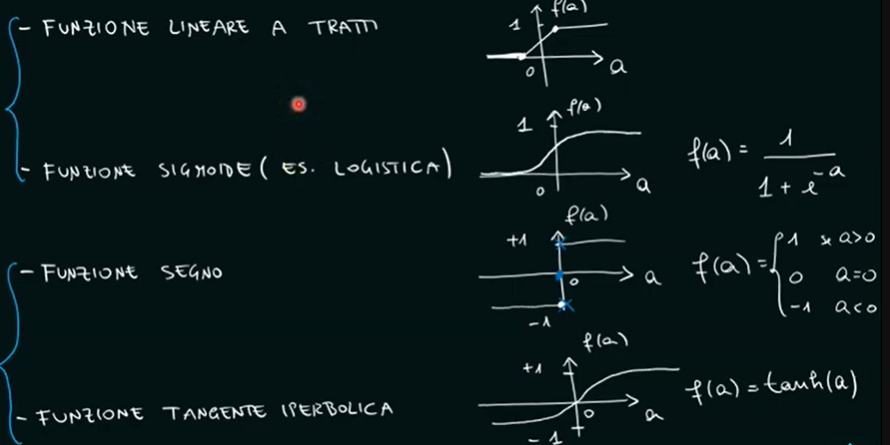
Il bias rappresenta il valore fissato per lo switch tra 0 e 1 come messo in evidenza in questa immagine:



Ora si porta il Bias all’interno della sommatoria dei pesi da associare agli input della rete, ma per farlo abbiamo bisogno di uno stratagemma che consiste nell’avere un neurone che rappresenti una costante che sarà l’input che si andrà a combinare linearmente con il peso w0 ( il quale sarà – il Bias)che rappresenterà proprio il Bias questo comporterà che il layer di input della rete assuma questa forma includendo l’input x0 = 1:



Il problema principale della funzione a gradino ( binary treshold) è che in zero non è derivabile (questo come vedremo per il funzionamento della rete è un aspetto molto importante)



Si preferiscono le funzioni iperboliche quando l’output che vogliamo è in [-1, 1].

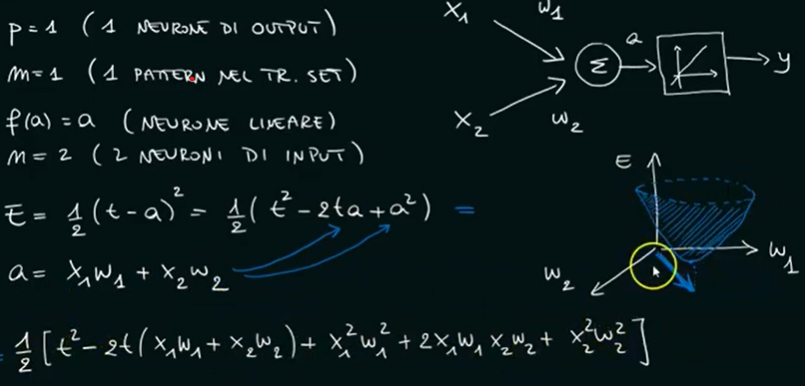
**DELTA RULE o REGOLA DI WINDROW-HOLF**

Prima di introdurre la regola della Delta Rule ci serve capire cosa intendiamo per apprendimento di una rete artificiale e in particolare ci soffermeremo su di un tipo di apprendimento detto APPRENDIMENTO SUPERVISIONATO. Il paradigma supervisionato comporta fornire degli input alla rete sotto forma di un data set(training set) del quale si conoscono gli output associati agli input dati e quello che si fa e darli in pasto alla rete affinché essa capisca come migliorare i meccanismi interni al fine di riprodurre lo stesso output in futuro quando le sarò dato un nuovo dato senza fornirgli l’output associato.

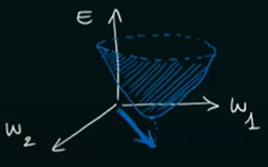


Apprendere per la rete significa cambiare il pesi sugli archi delle connessioni tra i neuroni artificiali.

L’output della rete è un vettore di p componenti, esattamente come il dati del training set sono tuple di p componenti (dette features).



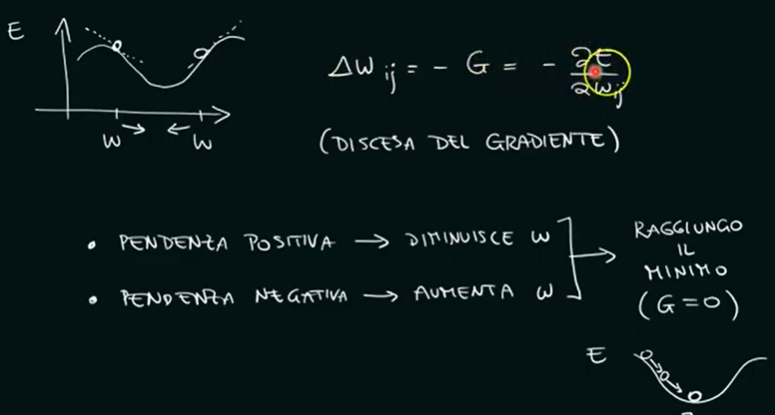
In questo esempio molto basilare di rete neurale abbiamo un solo neurone che riceve 2 input e la funzione di attivazione usata è la semplice funzione lineare, obbiettivo sarà quello di andare a minimizzare l’errore commesso in particolare l’errore quadratico definito come E, dall’immagine possiamo vedere che i calcoli di E portano a definire un Paraboloide:



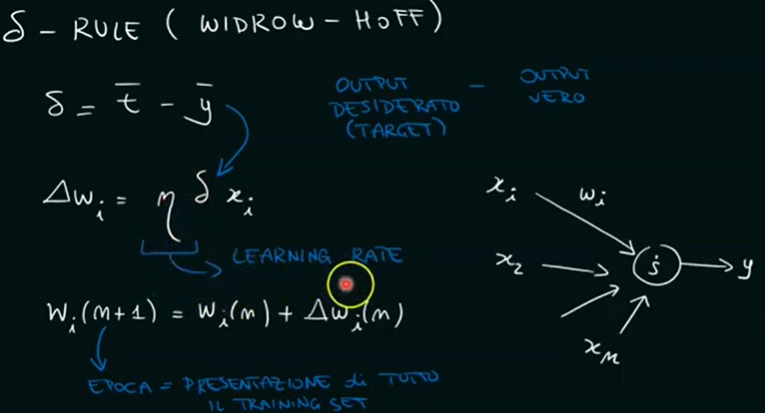
Dato dall’equazione seguente:



Per diminuire l’errore si utilizza l’algoritmo del greedy descent il quale ci dice che la variazione del peso i-esimo non è altro che meno il gradiente della funzione di errore, ovvero meno la derivata della funzione errore fratto la derivata rispetto al peso considerato. Il meno nella formala della variazione o aggiornamento del peso è fondamentale perché se la pendenza sarà positiva per raggiungere il minimo dato che l’input non lo si può cambiare dovremmo lavorare sul peso di modo che l’output prodotto sia minore del precedente (perché la pendenza è positiva indica che sia ad un valore maggiore rispetto al nostro obbiettivo) per cui la pendenza positiva con il meno diventa una quantità che sarà sottratta al precedente pese per ottenere il suo aggiornamento, viceversa se la pendenza è negativa con il meno fornirà una quantità positiva da sommare al vecchio peso per aggiornarlo.



La regola di widrow-hoff o delta rule ci dice che se fisso delta pari al vettore desiderato ovvero il target da cui sottraggo quello che ho ottenuto l’output vero, allora potrò dire che la variazione del peso sarà uguale al LR\*delta\*per l’input associato al peso come si può vedere dalla seguente immagine:



Dove quindi avremo che per aggiornare i pesi della nostra rete potremmo usare questa semplice formula:



Ove n sta a rappresentare una epoca di addestramento dove per epoca si intende l’intero training set dato in input una volta alla rete neurale.

Ora vogliamo porci la seguente domanda ovvero se la delta rule e l’algoritmo del greedy descent dicono la stessa cosa?

**DELTA-RULE GRADIENTE DECRESCENTE**

Text

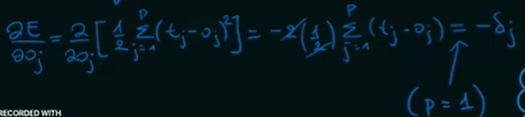
Description automatically generated 

Per poter rispondere a questa domanda dobbiamo prendere in considerazione un processo di derivazione parziale chiamato CHAIN RULE(regola di derivazione a catena): questa regola ci permette di riscrivere la derivata dell’errore fatta rispetto a wij come prodotto fra la derivata dell’errore fatta rispetto all’output oj per la derivata dell’output fatta rispetto al peso wij:

Graphical user interface, text

Description automatically generated

Ora andiamo a considerare il primo termine ovvero l’errore derivato rispetto all’output e andiamo a sviluppare la derivata:



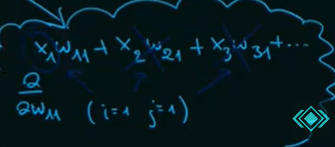
Come possiamo vedere dallo sviluppo dei passaggi matematici la derivata è pari a  ovvero il delta di cui parlavamo nella delta rule infatti è data nell’ultimo passaggio dal target – output ottenuto che risulta essere il delta.

La seconda parte del prodotto è data dalla derivata dell’output rispetto al peso ovvero:

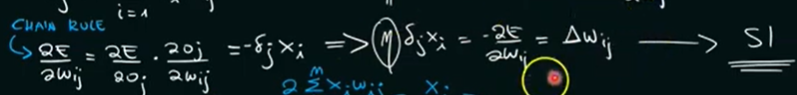
A picture containing text

Description automatically generated

Vediamo come sviluppare la derivata della sommatoria:



Questa sommatoria porta a trovare xi e quindi avremo che la delta rule è equivalente al gradiente discendente:



Ove in questo particolare case il learnig rate eta è pari ad 1.

**LEARING RATE & MOMENTUM**

Tramite il traning set fornito in pasto alla rete neurale essa è capace di modificare i pesi per ottenere in output il miglior risultato possibile. La prima difficoltà che si incontra nell’addestrare una rete è capire quale deve essere il numero che diamo ad eta ovvero il parametro di LR=learning rate detto anche passo di apprendimento ove se esso è troppo grade questo aiuta a convergere più velocemente, ma al contempo potrebbe non riuscire a convergere al minimo desiderato poi i passi così grandi tendono ad allontanarsi dal nostro obiettivo, se invece i passi sono troppo piccoli abbiamo certezza di convergere, ma la convergenza è molto lenta e servono molte iterazioni. In molte rete neurali questo LR viene appreso e quindi aggiornato al meglio.

A picture containing graphical user interface

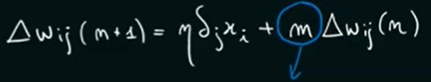
Description automatically generated

Un problema che si può riscontrare è quello dei minimi locali che non permettono la convergenza, per superarlo ci sono diverse tecniche (random restart, momentum… etc) iniziamo vedendo la tecnica del momentum.

A picture containing diagram

Description automatically generated

Il momentum è un nuovo parametro globale m compreso tra 0 < m < 1 che viene ad essere determinato tramite un approccio TRIAL & ERROR ovvero non c’è una algoritmo preciso capace di determinarlo per ogni problema, ma bisogna effettuare dei tentativi e testare il valore dato.



Quindi in questo caso la variazione o l’aggiornamento fatto all’epoca n + 1 sarà uguale a eta per delta per l’input ovvero alla formula della delta rule e in più si aggiunge una quantità data dal momentum che moltiplica la variazione di peso subita nella epoca precedente n in sostanza teniamo conto nella scelta della nuova variazione di peso di che stiamo effettuando di quale è stata la variazione fatta in precedenza, questo fa si che quando il gradiente continua a puntare nella stessa direzione dell’epoca precedente (ovvero la pendenza è la stessa) allora il passo che si copie è ampio (perché vogliamo convergere velocemente), quindi l’idea è che in caso di concordanza del gradiente attuale con quello precedente il momentum tende ad accelerare il processo di convergenza facendo compiere dei passi più ampi questo fa si che in caso di minimi locali questi possano essere superati senza rimanerne intrappolati.

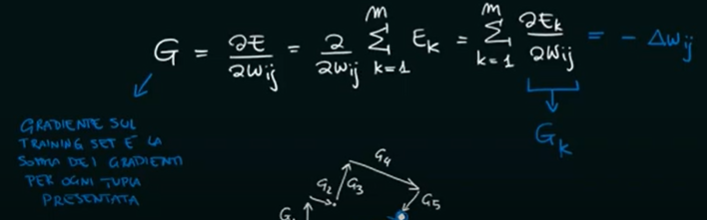


In questi caso quindi il momentum deve essere accordato ad eta in modo tale che se il momentum cresce rapidamente eta dovrà cedere il passo ovvero diminuire per non compiere passi eccessivamente grandi e non riuscire a convergere.

Nel caso in cui il gradiente sia discordante al precedente allora il passo dovrà essere cambiato diminuendo il momento e agendo su eta.

**BATCH VS ONLINE LEARNING**

L’apprendimento batch o anche detto apprendimento off-line, fa si che ci sia un accumulo dei contributi di gradiente dato da tutti gli esempi presenti nel training set che vengono elaborati e in seguito alla fine ovvero dopo una epoca si vanno ad aggiornare i pesi della rete.



Il gradiente sul training set non è altro che la somma dei gradienti per ogni tupla del training set presentata alla rete.

Nell’ On-Line learning i pesi vengono aggiornati immediatamente dopo aver considerato ogni tupla del training set, ovvero subito dopo aver elaborato una tupla l’aggiornamento che essa comporta viene propagato alla rete. Questo metodo è meno corretto, ma da una buona approssimazione e il vantaggio che comporta è quello di una minore richiesta di memoria.

Text

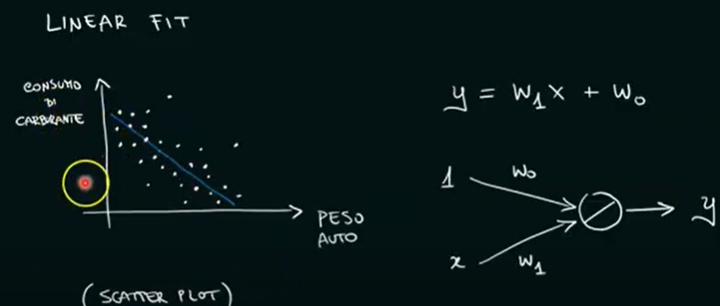
Description automatically generated

Nell’apprendimento On-Line l’ordine con cui vengono ad essere forniti le tuple cambia i risultati ottenuti dalla rete.

Un via di mezzo come approccio di apprendimento è il Mini-Batch in cui i pesi vengono aggiornati dopo un certo numero n di tuple (dati del training set) che vengono ad essere fornite alla rete dove n chiaramente sarà < del totale di tuple del training set. [ex. Supponendo di avere un training set di 100 dati è possibile dividerlo in 10 mini-batch da 10 tuple ciascuno ed effettuare l’aggiornamento della rete alla fine di ogni mini-batch].

LINEAR FIT

Vediamo un primo esempio pratico di applicazione delle reti neurali. Immaginiamo di avere un data set con due features una per il consumo di carburante e una altra per il peso dell’auto, queste sono espresse attraverso numeri reali poiché una rete neurale come già visto necessità di input sotto forma di numeri reali. Queste features possono essere rappresentate nel piano tramite punti, ora l’idea è quella di trovare una retta che vada a fittare i dati (ovvero effettuare la regressione lineare) di modo che la distanza al quadrato da tutti i punti sia minimizzato. Per fare questo fit dei dati vogliamo trovare la retta di regressione dei dati che se la vediamo in forma analitica ovvero Y = w1x + w0 possiamo vedere alcune somiglianza con il processo di apprendimento di un neurone della rete neurale come messo in evidenza nell’immagine:



Per cui se consideriamo un singolo neurone addestrato tramite la delta rule esse sarebbe in grado di mappare il peso di ogni auto ad un corrispettivo consumo di carburante seguendo come indice di apprendimento la retta di regressione dei dati.

**PERCETTRONE DI MC CULLOCH & PITTS**

**problema della classificazione**

Quando un neurone viene utilizzato per effettuare una classificazione binario ovvero predire se un input appartiene come output a una classe piuttosto che un'altra dove le classi sono rappresentate come 0 e 1(oppure -1 e +1) allora esso prende il nome di PERCETTRONE. Il problema della classificazione ha l’obiettivo di trovare un piano capace di dividere i punti del piano rappresentanti i dati in input in due categorie distinti separando(in termini di distanze) gli uni dagli altri.

A picture containing diagram

Description automatically generated

Come è possibile vedere in questo esempio la rete ha 3 input di cui il primo rappresenta il Bias della rete (con il quale si andrà ad identificare l’intercetta della retta di separazione dei dati), gli altri due inputi saranno le caratteristiche dei dati ( che rappresenteranno i valori lungo le dimensioni del piano ) e i pesi a questi ultimi due input fanno a definire il coefficiente angolare della retta di separazione.



Ora se rappresentiamo con w segnato il vettore dei pesi e con p segnato il vettore delle coordinate possiamo dare vita tramite a degli assiomi di algebra lineare a un equazione di decision boundary

Text

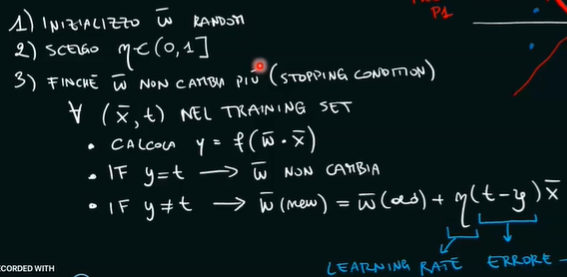
Description automatically generated

La prima cosa da tenere in considerazione adesso è che la delta rule per l’addestramento del percettrone non va bene perché qui il percettrone è non lineare.

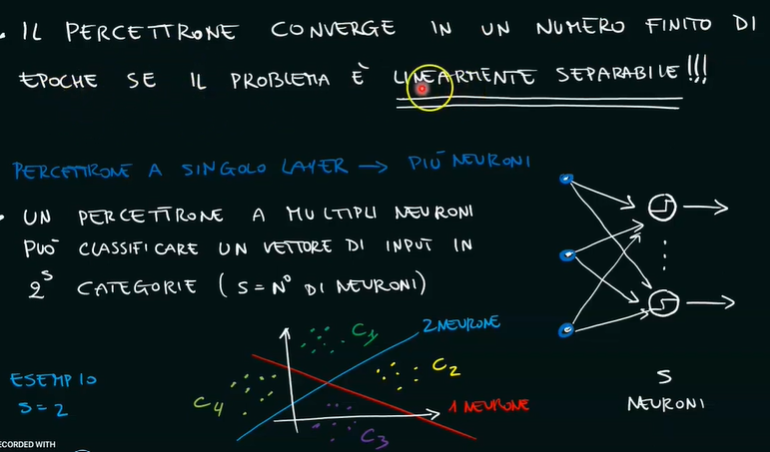
Ora l’idea di base è quella di fare in modo che il prodotto interno dato dai vettori dei pesi w e delle coordinate x sul quale risultata agisce la funzione segno dia come risultato il target ovvero la classificazione che ci aspettiamo ovvero:



Vediamo adesso l’algoritmo di addestramento del percettrone :



La differenza tra la delta rule e questo algoritmo e che mentre nella delta rule il valore può assumere un qualsiasi valore reale, qui invece è vincolato ai soli risultati della funzione segno ovvero -1, 0 e +1.



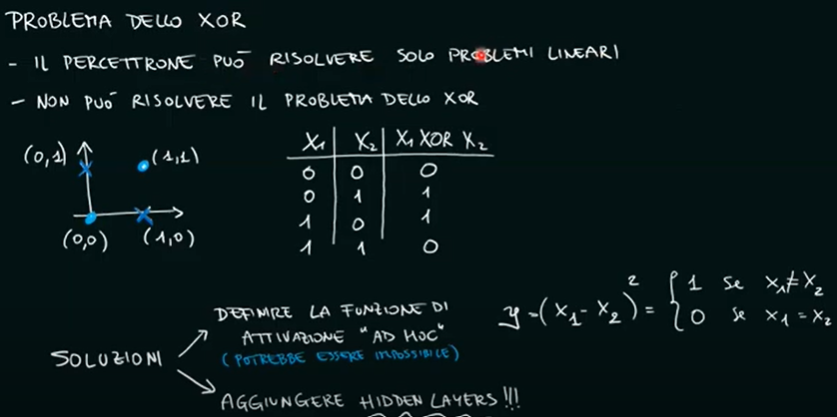
Ogni percettrone è come se andasse a tracciare una retta nel piano ed è capace di andare a dividere gli input ricevuti in due categorie, per cui in una rete neurale ove ci sono più percettroni in un layer nascosto questi sono in grado di dividere l’input in un numero di classi pari a 2s ove s è il # dei percettroni del layer nascosto.

**PROBLEMA DELLO XOR, PROBLEMI NON LINEARMENTE SEPARABILI**

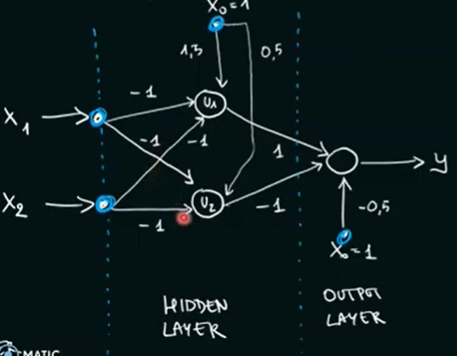
**E NASCITA DEL**

**MLP MULTI-LAYER-PERCEPTRON**

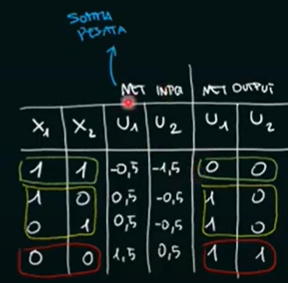
Un problema è lineare è un problema i cui input possono essere mappati in due classi differenti di output e questo equivale nella rappresentazione su grafico nel piano a separare i dati rappresentati come punti in due semi piani individuati dal semi piano che interseca il semipiano dei punti tagliandolo in due semi piani:



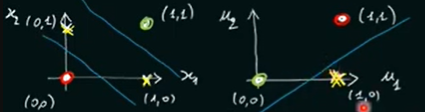
Per risolvere il problema dello xor abbiamo 2 strade percorribili o definire una funzione di attivazione ad hoc che sia capace di gestire il problema (non usando quindi la funzione segno) oppure aggiunger più layer nascosti alla rete.



L’idea dietro l’aggiunta di più livelli è quello di portare lo spazio di rappresentazione del problema attuale in un nuovo spazio diverso da quello iniziale in cui sia possibile separare linearmente il problema.



Questo processo quindi porta la rete a ragionare su di un nuovo spazio del problema ovvero su di una differente rappresentazione al fine di trovare un soluzione possibile come è possibile distinguer dei due grafici presentati a sx troviamo lo spazio no linearmente separabile mentre a destra troviamo la rappresentazione data dal hidden layer la quale è separabile linearmente:



Gli hidden layer permettono di risolvere i problemi di classificazione più complessi di quello binario.

Text

Description automatically generated

Diagram

Description automatically generated

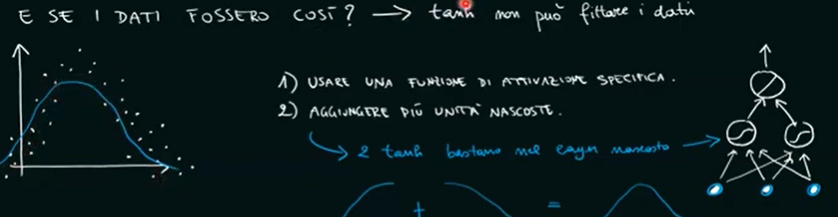
**RETI MLP PER IL FITTING LINEARE**

Nei problemi di fitting non lineare per poter fare in modo che i dati siano equidistanti il più possibile ad una curva non lineare si aggiungono hidden layer alla rete neurale. In questo caso le funzioni di attivazione del layer nascosto saranno funzioni di attivazione non lineare come la funzione della tangente iperbolica o quella del sigmoide.

Diagram, schematic

Description automatically generated

Supponiamo di avere dei dati disposti come nel seguente grafico i quali seguano l’andamento di una funzione a campana, in tal caso la funzione iperbolica della tangente no sarebbe capace di svolgere il suo compito in modo corretto, per cui le alternative potrebbero essere o utilizzare una funzione definita appositamente per i dati come visto per il problema dello xor oppure siccome il nostro obbiettivo è quello di generalizzare il processo di apprendimento della rete senza legarlo a specifiche caratteristiche dei dati. Per fare ciò andiamo ad aggiungere un numero maggiore di layer hidden nella rete.

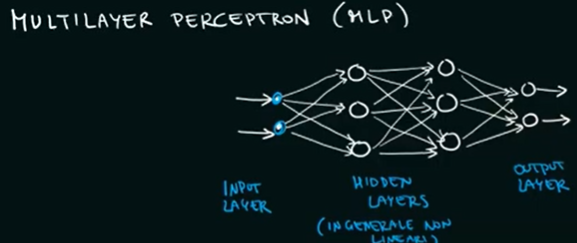


Aumentare il numero nei layer o il numero di neuroni nascosti nello stesso hidden layer può essere controproducente e invece di portare ad un miglioramento potrebbe deteriorare le prestazioni della rete, questo perché stiamo andando ad aumentare la complessità del nostro modello di apprendimento e più la complessità cresce più cresce l’errore che si può commettere ne consegue che diminuisce l’accurancy.

Un’altra nota importante è che si può dimostrare che qualsiasi funzione può essere espressa come una combinazione lineare di più funzioni di tangente iperbolica. Essa, infatti, è considerata come una UNIVERSAL BASIS FUNCTION. Ci sono tuttavia anche altre funzioni per le quali è dimostrabile la stessa tesi e queste sono le funzioni sigmoide e quelle radiali [queste ultime danno vita a delle particolare reti neurali chiamate reti RBF].

**MULTILAYER PERCEPTRON**

Altro non è che una rete neurale a più strati con funzioni di attivazioni non lineari. Per il training di una rete così fatta MLP non è possibile usare la delta rule(quindi il gradient descend), poiché non conosciamo l’output per ogni strato intermedio, ma solo ‘output finale. In questo caso si ricorre al concetto della delta rule, ma in maniera differente.



Text

Description automatically generated

L’idea del training è quella di dare in pasto i dati, facendo propagare l’input in tutta la rete e ad ogni livello sarà applicata la delta rule generalizzata in modo che si calcoli l’output finale nell’output layer e su di questo si applichi la funzione di errore che agirà come la delta rule e una volta calcolato questo errore questa informazione sarà retro-propagata nella reta per andare a modificare i vari pesi della rete e si ridarà nuovamente dell’input.

Text

Description automatically generated

L’algoritmo è chiamo error back propagation.