Classificazione

→ Generative

→ Discriminative → modello cerca di distinguere tra diverse classi basandosi sui dati in input, senza modellare distribuzione dati

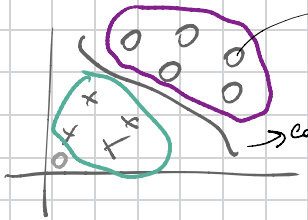
↳ IMPARA una f che mappa input alle etichette di classe, ottimizzando un confine decisionale che separa le classi

OBIETTIVO → Imparare conf. decisionale che separa meglio le classi

FOCUS → $\max(P(Y|X))$

↳ classe → caratteristiche

VISTA GEOMETRICA



→ Dato → Punto in uno spazio multidimensionale, qui dimensione → caratteristiche

→ Confine di decisione → Modello definisce superficie che separa le regioni dello spazio corrispondenti a diverse classi

$x^{(i)} \in \mathbb{R}^d \qquad y = \{0,1\}$

Ingredienti → Classificazione

- Modello
- Loss function (Misura errori)
- Algoritmo di Learning
- Evaluation

Reti Neurali

Introduzione al Contesto: Un Modello (Molto) Semplice (e Impreciso) del Nostro Cervello

L'apprendimento automatico (machine learning) nasce dall'ambizione di replicare il funzionamento del nostro cervello. I primi modelli di machine learning si ispirano infatti al funzionamento dell'unità computazionale di base del cervello: il **neurone**. Questo contesto introduce un modello semplificato e idealizzato di come i neuroni operano e di come, connettendosi tra loro, formano reti neurali che emulano alcune capacità del cervello umano.

I Neuroni

Un neurone può essere visto come una semplice macchina computazionale che riceve una serie di input, li elabora e produce un output. Le sue componenti principali sono:

1. **Dendriti**: Strutture che raccolgono segnali in ingresso da altri neuroni o fonti esterne.
2. **Soma**: Il corpo centrale del neurone, che elabora gli input ricevuti.
3. **Assone**: Il canale di output, che trasmette segnali ad altri neuroni.

I neuroni comunicano attraverso segnali elettrici, che possono essere "alti" (quando il neurone "si attiva" o "spara") o "bassi" (quando è inattivo). Questo comportamento binario (0 o 1) permette di considerare il neurone come un'unità computazionale elementare, che trasforma un insieme di input binari in un output binario.

Reti Neurali

La complessità del cervello non deriva dai singoli neuroni, che sono relativamente semplici, ma dalla loro connessione in una **rete neurale**. Queste reti sono collegate agli organi sensoriali (come occhi e orecchie), che forniscono gli input. In base ai segnali ricevuti, alcuni neuroni si attivano, mentre altri rimangono inattivi.

I neuroni si organizzano in **strati**, con comunicazioni prevalentemente tra strati adiacenti. Gruppi di neuroni si specializzano per attivarsi in risposta a stimoli specifici: ad esempio, alcuni rispondono a linee orizzontali, altri a linee verticali. I neuroni situati in strati più alti della gerarchia si attivano in presenza di concetti più astratti, come volti umani o categorie di oggetti, suggerendo una rappresentazione gerarchica delle informazioni nel cervello.

Questo modello, pur semplificato e non del tutto accurato, getta le basi per comprendere come le reti neurali artificiali, ispirate ai neuroni biologici, siano diventate un pilastro dell'apprendimento automatico moderno.

Se vuoi approfondire un aspetto specifico, come il funzionamento delle reti neurali artificiali o il legame con la classificazione, fammi sapere!

1 modello

Modello McCulloch-Pitt 1943

Primo tentativo **Minima neurone**

Unità computazionale che riceve in ingresso n input binari

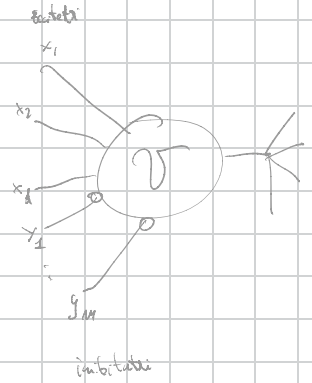
Modello restituisce output binario

Ogni neurone è associato ad un parametro ϑ (threshold)

Output può essere connesso ad altri neuroni (anche più di uno)

Input può provenire da altri neuroni o può essere vettore binario o una congiunzione

Due tipi di input — Eccitatorio / Inibitorio



La funzione di un input eccitatorio è di **aumentare l'eccitazione totale del neurone ricevente**. Secondo la fonte, l'eccitazione totale viene calcolata sommando i valori degli input eccitatori attivi: $x = x_1 + \cdots +$ x<0xE2><0x82><0x99>. Se questa somma (x) è maggiore o uguale a una soglia (θ) associata al neurone, e non c'è alcun input inibitorio attivo, allora l'output del neurone sarà 1 (il neurone "spara"). Quindi, un input eccitatorio attivo contribuisce a far sì che il neurone produca un output elevato

D'altra parte, un **input inibitorio** ha l'effetto opposto. Se il numero di input eccitatori attivi è di almeno uno ($m \geq 1$) e **almeno uno degli input inibitori è attivo (uguale a 1)**, allora il neurone viene **inibito** e l'output è 0 (il neurone non "spara"). Questo significa che la presenza di un solo input inibitorio attivo è sufficiente a bloccare l'attivazione del neurone, indipendentemente dal numero di input eccitatori attivi (purché ce ne sia almeno uno) e dal loro valore totale.

Con una scelta appropriata della soglia θ
Neurone McCulloch-Pitts può implementare AND

se inibitore acceso => output = 0 Aggiunetti a

if $m \geq 1$ AND $\exists\, g_j \mid g_j = 1$ then

Output 0

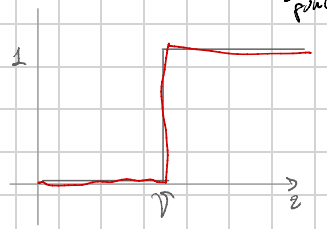
else if $\sum_{i=1}^{z} x_i \geq \vartheta$ then

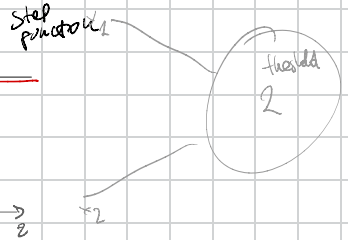
Output 1

else

Output 0

Vedai non binari in binari



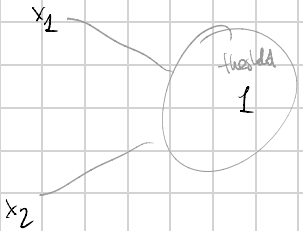


AND

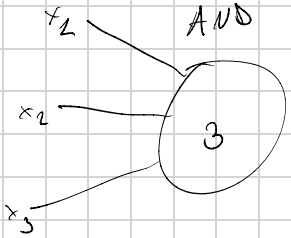
$x_1 + x_2 \geq \vartheta$

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1

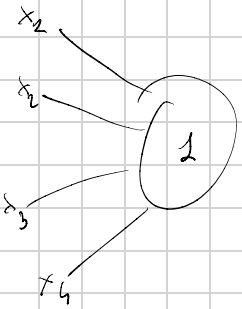
OR



AND

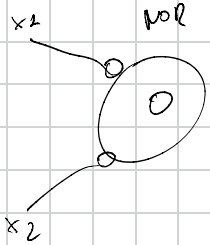


Oppure complesse unità logiche

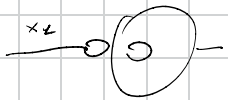


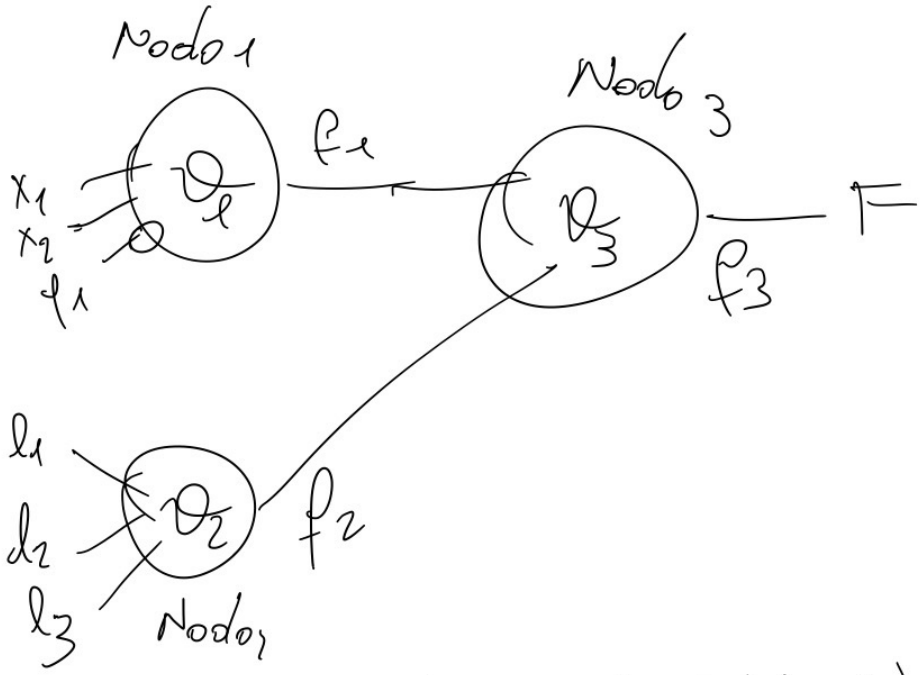
Con inibizione!

NOR



NOT





$$f_3\left(f_1(x_1 \dots x_f),\ f_2(l_1, \dots, l_f)\right) \geq \vartheta_3$$

$$[\Sigma x_f \geq \vartheta_1] + [\Sigma l_f \geq \vartheta_2] \geq \vartheta_3$$

z predicato

$$[z] = \begin{cases} 1 & \text{SE } z \text{ TRUE} \\ 0 & \text{SE } z \text{ FALSE} \end{cases}$$

McCulloch-Pitts modello parametrico ⟶ comportamento e funzione che implementa dipendono da un insieme di parametri che posso essere modificati

Le fonti affermano che il neurone di McCulloch-Pitts è un **modello parametrico**. Questo significa che il suo comportamento e la funzione che implementa sono determinati da un insieme di **parametri** che possono essere scelti o impostati.
Nel contesto del neurone di McCulloch-Pitts, le fonti identificano esplicitamente i seguenti parametri:

- **La natura di ciascuna connessione (eccitatoria o inibitoria)**. Come discusso in precedenza, le connessioni eccitatorie aumentano l'eccitazione del neurone ricevente, mentre le connessioni inibitorie tendono a bloccarne l'attivazione. La scelta del tipo di connessione per ciascun input è quindi un parametro del modello.
- **Il valore della soglia** θ. La soglia determina il livello di eccitazione totale (derivante dalla somma degli input eccitatori) necessario affinché il neurone produca un output di 1 (spari), in assenza di inibizione. La scelta di questa soglia è un altro parametro che influenza il comportamento del neurone.

La fonte sottolinea che, con una scelta appropriata di questi parametri (la natura delle connessioni e il valore della soglia θ), il neurone di McCulloch-Pitts può implementare semplici porte logiche come AND e OR. L'uso di connessioni inibitorie permette anche di implementare unità logiche più complesse.

Quindi, nel contesto del neurone di McCulloch-Pitts, l'essere un modello parametrico significa che la sua funzionalità non è fissa, ma può essere configurata e adattata per eseguire diverse operazioni logiche attraverso la **selezione dei suoi parametri**. McCulloch e Pitts hanno anche dimostrato che impilando diverse unità (neuroni) con scelte appropriate dei parametri si potevano implementare funzioni più complesse, creando così le prime forme di "reti neurali artificiali".
È importante notare che una delle limitazioni del neurone di McCulloch-Pitts, come evidenziato nella fonte, è che questi **parametri sono fissi e devono essere scelti manualmente**. Ciò significa che per adattare il modello a diversi problemi, era necessario conoscere la soluzione logica del problema e impostare di conseguenza i parametri. Questo contrastava con l'ideale di poter trovare automaticamente i parametri appropriati attraverso un procedimento di apprendimento basato su esempi, una limitazione che il successivo modello del percettrone di Rosenblatt ha cercato di superare introducendo i pesi modificabili tramite un algoritmo di apprendimento. Nel percettrone, i pesi delle connessioni diventano i parametri del modello, e il loro valore (positivo o negativo) generalizza la distinzione tra connessioni eccitatorie e inibitorie del modello di McCulloch-Pitts.

Aspetto cruciale : impilando diverse di queste unità si possono implementare funzioni più complesse

Limiti Modello

Pensato per ricevere input binari

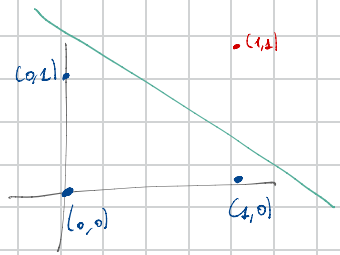
Pensato solo per funzioni logiche / input binari

Parametri fissati a mano → No procedure di Learning

* Non esistono pesi

Tutte le funzioni sono linearmente separabili → funzione AND separiamo con classi dove possiamo mettere su asse binario

0	0	0
0	1	0
1	0	0
1	1	1



LINEARMENTE SEPARABILI!

Il problema dell'**XOR** evidenzia ulteriormente la necessità e il vantaggio di connettere più neuroni. La fonte "" spiega che, sebbene un singolo percettrone (un modello successivo ispirato al neurone di McCulloch-Pitts) non possa risolvere il problema dell'XOR perché i dati non sono linearmente separabili, è stato dimostrato che **una rete a due strati di percettroni può farlo**. Questa rete a due strati trasforma i dati in uno spazio in cui diventano linearmente separabili, consentendo all'ultimo percettrone di classificarli correttamente. Sebbene l'esempio dell'XOR si riferisca a percettroni, il principio di base - che **combinando più unità computazionali semplici in una rete a più strati è possibile risolvere problemi più complessi** - è un'evoluzione diretta dell'idea di McCulloch e Pitts di "stacking different units".

Percettrone di Rosenblatt

Le fonti presentano il **Percettrone di Rosenblatt** come un modello significativo nell'evoluzione delle reti neurali artificiali e dell'apprendimento automatico, costruito sulle fondamenta del precedente modello del neurone di McCulloch-Pitts.

Nel contesto più ampio dell'**apprendimento automatico**, il Percettrone rappresenta uno dei primi tentativi di creare un modello capace di **apprendere dai dati**. A differenza del neurone di McCulloch-Pitts, i cui parametri (tipo di connessione e soglia) dovevano essere impostati manualmente, il Percettrone introduce un **algoritmo di apprendimento** in grado di trovare automaticamente i pesi appropriati del modello a partire da un insieme di esempi. Questo è un passo cruciale verso la creazione di sistemi di apprendimento automatico capaci di adattarsi a diversi problemi senza la necessità di una conoscenza esplicita della loro soluzione logica.

Nel contesto delle **reti neurali**, il Percettrone è visto come un modello simile al neurone di McCulloch-Pitts ma che ne supera alcune limitazioni. Le principali differenze e innovazioni introdotte dal Percettrone sono:

- **Pesi sulle connessioni**: Il Percettrone assegna **pesi** a ciascuna delle connessioni in ingresso. Questi pesi vengono moltiplicati per i valori degli input. La possibilità di avere **pesi negativi** elimina la necessità di distinguere esplicitamente tra connessioni eccitatorie e inibitorie presenti nel modello di McCulloch-Pitts. Un peso positivo avrà un effetto simile a una connessione eccitatoria, mentre un peso negativo avrà un effetto simile a una connessione inibitoria.
- **Input flessibili**: Mentre il neurone di McCulloch-Pitts opera con input binari, il Percettrone può gestire **input binari, interi o reali**. Questo lo rende più adatto a lavorare con caratteristiche più generali degli elementi di input. θ ciao $3x+2$
- **Soglia come peso**: Il Percettrone include un peso aggiuntivo (w_0 o w'), che funge da **soglia** (indicata come θ nel modello di McCulloch-Pitts). Introducendo un input "dummy" $x_0 = 1$, l'operazione di soglia può essere integrata nella somma pesata.
- **Algoritmo di apprendimento**: La principale innovazione di Rosenblatt è stata la fornitura di un **algoritmo di addestramento** per il Percettrone. Questo algoritmo è un processo iterativo in cui i pesi del modello vengono inizializzati casualmente e poi **aggiustati per ridurre il numero di errori** commessi sui dati di addestramento. L'aggiornamento dei pesi avviene in base alla differenza tra l'output previsto dal Percettrone e l'etichetta corretta dell'esempio, regolato da un **tasso di apprendimento** η. L'algoritmo itera su tutti gli esempi di addestramento fino a quando non vengono più commessi errori. È stato dimostrato che questo algoritmo **converge a una soluzione se i punti di input sono linearmente separabili**.

Dal punto di vista geometrico, le fonti spiegano che un Percettrone può essere visto come un **classificatore binario** che prende come input vettori di caratteristiche e produce un output binario. In uno spazio a n dimensioni, un Percettrone trova un **iperpiano** (una linea in 2D, un piano in 3D, ecc.) che **separa lo spazio in due regioni**, corrispondenti alle due classi. L'equazione $w_1x_1 + w_2x_2 + w_0 > 0$ definisce una di queste regioni, dove i punti che la soddisfano vengono classificati come positivi, mentre l'ineguaglianza opposta definisce la regione dei negativi. La linea o l'iperpiano definito da $w_1x_1 + w_2x_2 + w_0 = 0$ è chiamato **confine decisionale**.

Tuttavia, le fonti sottolineano anche le **limitazioni del Percettrone singolo**. In particolare, un Percettrone è un **classificatore lineare** e può risolvere solo problemi in cui i dati di addestramento sono **linearmente separabili**. L'esempio del **problema XOR** viene citato come un caso classico di problema non linearmente separabile che un singolo Percettrone non può risolvere. Minsky e Papert (1969) hanno dimostrato formalmente questa limitazione.

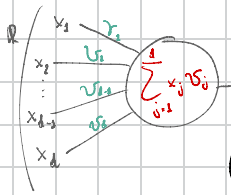
Nonostante questa limitazione, le fonti evidenziano che **reti a più strati di Percettroni** possono superare il problema dell'XOR. Un esempio di rete a due strati viene fornito, mostrando come la prima "layer" di Percettroni possa **trasformare lo spazio delle caratteristiche** in modo che i dati diventino linearmente separabili per il Percettrone finale. Questo dimostra il potenziale dello "stacking" di unità (concetto già introdotto con il neurone di McCulloch-Pitts) per implementare funzioni più complesse.

Infine, le fonti menzionano che, sebbene le reti XOR possano risolvere il problema XOR, l'algoritmo di apprendimento del Percettrone descritto non può essere applicato direttamente

a queste reti più complesse. Questa osservazione, insieme alle limitazioni del Percettrone singolo, contribuì al primo "**inverno dell'IA**" (un periodo di ridotto interesse e finanziamento nell'IA), che terminò solo con la formalizzazione dell'algoritmo di **backpropagation**.

In sintesi, il Percettrone di Rosenblatt rappresenta un avanzamento significativo rispetto al neurone di McCulloch-Pitts, introducendo l'apprendimento automatico attraverso un algoritmo per la regolazione dei pesi. Sebbene potente per problemi linearmente separabili e fondamentale per la concettualizzazione delle reti neurali multistrato, la sua incapacità di risolvere problemi non linearmente separabili come l'XOR ha evidenziato la necessità di architetture e algoritmi di apprendimento più sofisticati.

Perceptrone



v_0 aggiungo nella posizione 0 ↳ la posizione come input

Compute combinazione lineare del resto input

stima modello

step function

$\hat{y} = \{0, 1\}$

Compute $f(x_1, \dots, x_d) = \begin{cases} 1 & \text{se } \sum_{i=1}^{d} v_i x_i > v_0 \\ 0 & \text{ALTRIMENTI} \end{cases}$

Se introduciamo variabile dummy $x_0 = 1$

$\Rightarrow f(x_1, \dots, x_d) = \begin{cases} 1 & \text{se } \sum_{i=0} v_i x_i > 0 \\ 0 & \text{altrimenti} \end{cases}$

possiamo anche scrivere

$f(x) = [w^t x > 0]$

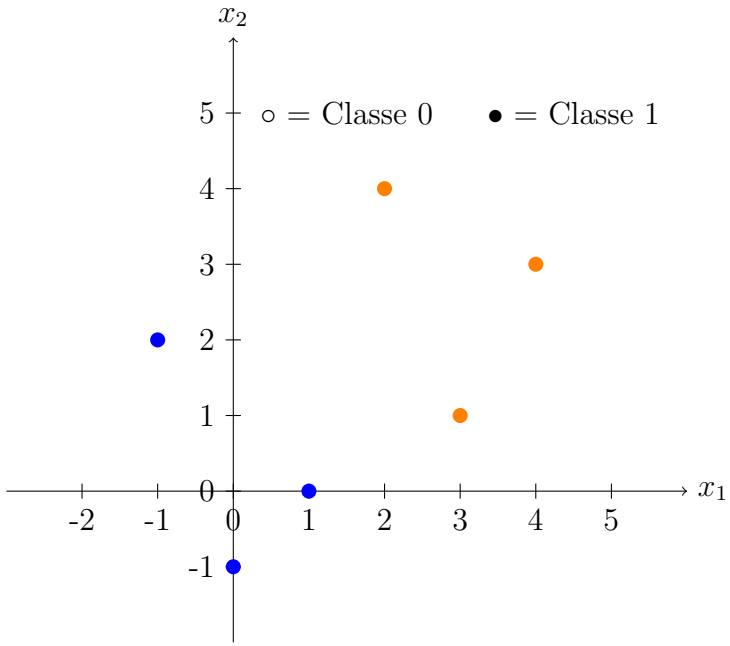
$x_0 = 1, x_1^{(1)}, \dots, ?$

$[v^T \vec{x} > 0] = \boxed{v} \boxed{x}$

1 Interpretazione Geometrica

I percettroni prendono come input vettori di caratteristiche e restituiscono valori binari. Pertanto, possono essere visti come **classificatori binari**, e questo è infatti il primo utilizzo dimostrato dei percettroni.

Consideriamo un insieme di dati con due caratteristiche reali distribuite come mostrato di seguito:



Ogni esempio (o punto dati) $x = (x_1, x_2)$ è rappresentato nel grafico come un punto bidimensionale nello spazio cartesiano. I punti appartengono a due classi: "0" (blu) e "1" (arancione).

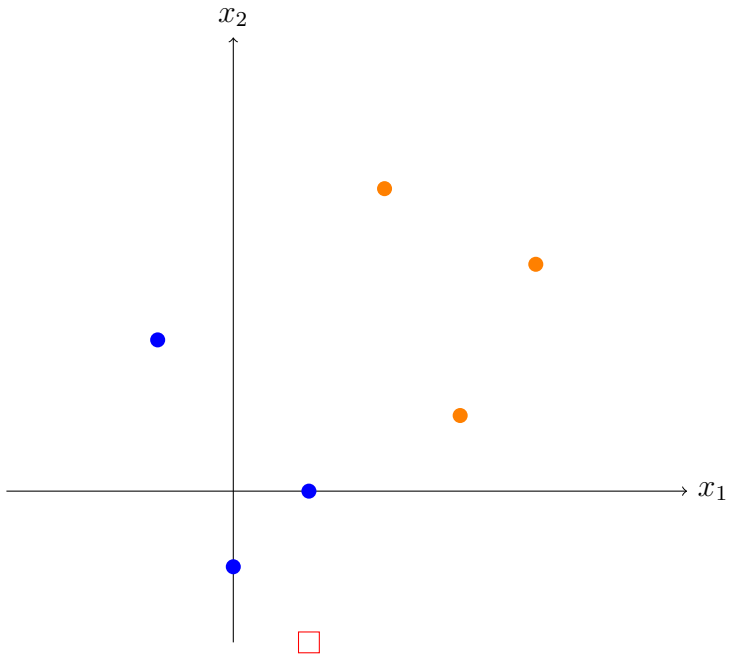
Un percettrone può risolvere questo problema di classificazione binaria trovando valori appropriati per i tre parametri (numero di dimensioni + 1 parametro per la soglia):

$$w_0, w_1, w_2$$

Se addestriamo (parleremo più avanti dell'algoritmo di apprendimento) un percettrone su questi dati, potremmo ottenere la seguente soluzione:

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -5.74 \\ 1.86 \\ 1.99 \end{pmatrix}$$

Consideriamo un nuovo punto con caratteristiche $(1, -2)$, rappresentato nel grafico come una croce rossa:



Per questo nuovo input, il percettrone calcolerà il seguente valore:

$$f((1,-2)) = \begin{cases} 1 & \text{se } -5.74 + 1.86 \cdot 1 + 1.99 \cdot (-2) > 0 \\ 0 & \text{altrimenti} \end{cases} = \begin{cases} 1 & \text{se } -7.86 > 0 \\ 0 & \text{altrimenti} \end{cases} = 0$$

Questo risultato è ragionevole, poiché la croce rossa è più vicina ai punti

blu che a quelli arancioni. Idealmente, possiamo eseguire questo calcolo per un insieme casuale di punti campionati dallo spazio per capire come il percettrone classifica l'intero spazio rappresentato nel diagramma.

In pratica, possiamo osservare che la classe prevista sarà positiva per i punti bidimensionali che soddisfano la seguente disuguaglianza:

$$w_1x_1 + w_2x_2 + w_0 > 0$$

Mentre il percettrone classificherà come negativi i punti che soddisfano:

$$w_1x_1 + w_2x_2 + w_0 < 0$$

Il classificatore sarà massimamente incerto quando:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

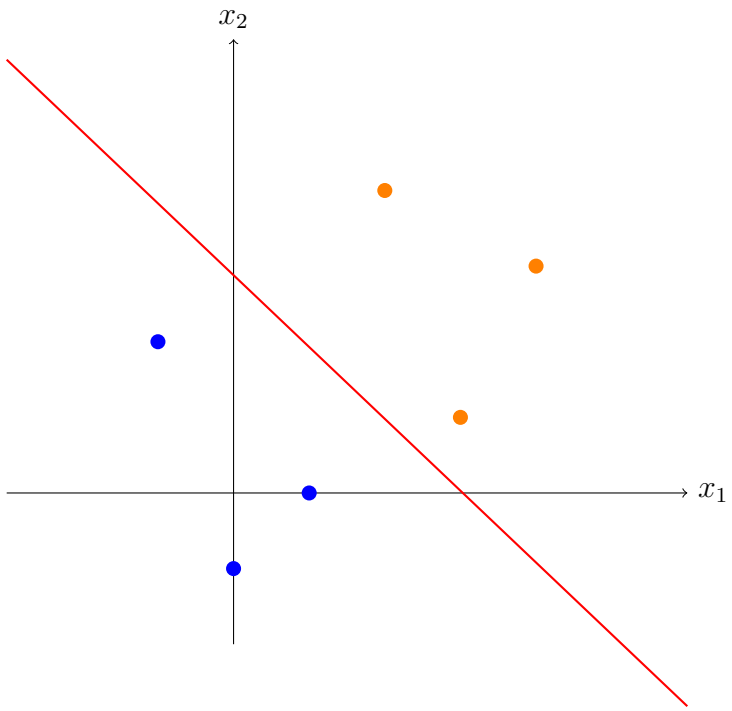
O equivalentemente:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

Sostituendo $\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -5.74 \\ 1.86 \\ 1.99 \end{pmatrix}$, otteniamo:

$$x_2 = -0.95x_1 + 2.88$$

Questa è l'equazione di una retta nello spazio bidimensionale. Tracciando questa linea sui dati, otteniamo:

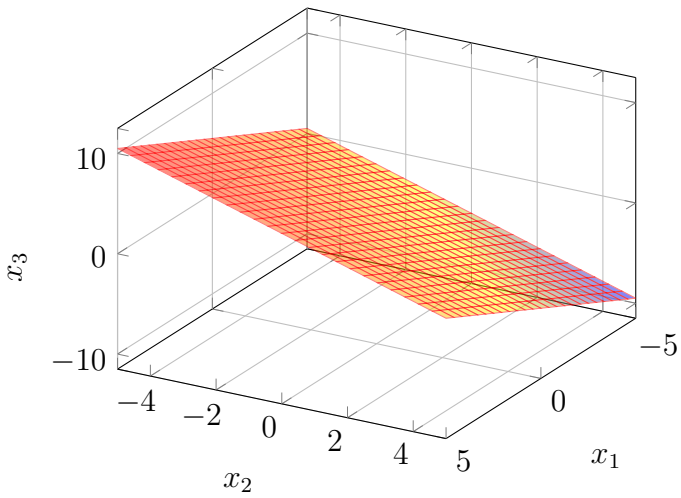


In pratica, il percettrone divide lo spazio in due aree:

- Tutti i punti sopra la linea soddisfano la disuguaglianza $w_1x_1 + w_2x_2 + w_0 > 0$ e saranno classificati come esempi positivi.
- Tutti i punti sotto la linea soddisfano la disuguaglianza $w_1x_1 + w_2x_2 + w_0 < 0$ e saranno classificati come esempi negativi.
- Per i punti che soddisfano l'equazione $w_1x_1 + w_2x_2 + w_0 = 0$, il modello è massimamente incerto. In questo caso, i punti potrebbero essere classificati casualmente come positivi o negativi.

La linea tracciata sopra è chiamata **confine di decisione** e indica come il classificatore ha diviso lo spazio in aree.

Sebbene abbiamo visto un esempio in 2D, questo ragionamento si estende a più dimensioni. In generale, il confine di decisione è un iperpiano nello spazio n-dimensionale. Ad esempio, in 3D, potremmo avere un confine di decisione come questo:


10
0
−10
x_3
−4
−2
0
2
4
x_2
5
0
−5
x_1

Algoritmo di Apprendimento

Innovazione principale Rosenblatt → fornire procedura di addestramento per il perceptrone.

↳ processo iterativo in cui i parametri del modello (pesi) vengono inizialmente inizializzati casualmente

Apprendimento → processo cruciale per settare i parametri del modello

↳ consente a un modello la capacità di migliorare le proprie prestazioni su un compito specifico attraverso l'esperienza con i dati

Parametri fissati → NON MANUALMENTE!

↳ Modello deve essere capace di apprendere → determinati attraverso una procedura

Come faccio a fissare i parametri?

$\sum x_j v_j > 0$ $\quad \bar{y} = 0$ $\quad \hat{y} = 1$ Come posso settare i pesi?

↑ pesi

$x \in \mathbb{R} \Rightarrow x \in \{0,1\}$ Decrementiamo i parametri

$y \in \{0,1\}$

$y = 1$ $\quad \hat{y} = 0$

Incrementiamo i parametri

Errore scarto a inizio ← compito ← tutti pesi

incrementare

$$v_j \leftarrow v_j + (y - \hat{y})\, x_j$$

quantità per addestrare in $\{0,1\}$

Procedura

Inizializzare i pesi random in $[0,1]$ → Conosciamo valore pesi x aggiustare

Considera $x^{(i)}$ e computa $\hat{y}^{(i)}$ $\forall i$

Per le predizioni errate aggiusta i pesi

$y = 0$ $\quad \hat{y} = 1$ Decrementa

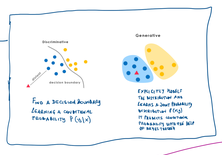
$y = 1$ $\quad \hat{y} = 0$ incrementa

$$v_j^{t+1} \leftarrow v_j^{t} + (y - \hat{y})\, x_j$$

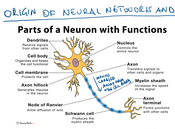
Ritorna al passo 2 fino a convergenza

Pipeline in ML

Generative vs. Discriminative Models



Origin of Neural Networks and Deep Learning



SUM

1958 — Perceptron (Rosenblatt)
Resembles a neuron
- Adjust parameters
- Weights fixed ±1
- Meth on to learn: adjust parameters

1960 — Adaline (Widrow and Hoff)
Adaptive linear neuron
Physical device
Adjust parameters with weights sum of the input

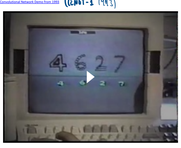
1969 — Perceptrons (Minsky and Papert)
Mathematical proof of convergence and limitations (e.g. XOR function + convexity → MLP)

AI winter

1970-1986 — Back propagation (Linnainmaa 1970) (Werbos 1974) (Rumelhart, Hinton, Williams 1986)
Computational methods useful to learn MLP

1997 — LSTM (Hochreiter, Schmidhuber)
Temporal data, language sequences

1998 — CNN (LeCun) (LeCun Bottou Bengio Haffner)
For OCR
Convolutional Network Demo from 1989 (LeCun + 1993)



2006 — Deep learning (Hinton, Osindero, Teh)
- Training of DBNs (Deep Belief Nets) with unsupervised learning
- Pre-training layer-wise greedily for the weights of multiple hidden layers, followed by supervised learning
- Backpropagation + LeCun Optimal Brain Damage

2009 — ImageNet (Deng, Fei-Fei)
Large scale visual recognition challenge. 1000 classes. Accuracy by Deng, Jiao, Satheesh, Berg

2015 — AlexNet (Krizhevsky, Sutskever, Hinton)
First convolutional net to reach a human level accuracy
Trained on ImageNet
Training on GPUs
Rectified linear units (ReLU)
Dropout
Data augmentation
Stochastic gradient descent with momentum
Batch size 128
Cross entropy loss
Ensemble of 7 models
Error rate 15.4% (next best 26.2%)
Linear, Relu, ...

Deep learning "golden" era

2017 — NASNet (Zoph, Le, Vasudevan)

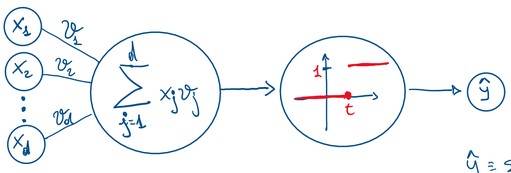
2018 — BERT (Devlin)

Any function can approximate any function (universal theorem)
Any standard function of order 3 to have a good approximation
Today
- Bigger, big
- Bigger networks
- Better patterns

Perceptron (Libro: Neural Network - A Systematic Introduction, Cap 3)

Proposed by Rosenblatt as machine for binary classification $\Rightarrow x \in \mathbb{R}^d, y \in \{0,1\}$

Main idea
- One weight ϑ_j $\forall$ input component x_j.
- Multiply weights with respect to the inputs and sum all of them (linear combination)
- If result is larger than a threshold t, than return 1 otherwise return $\emptyset$.



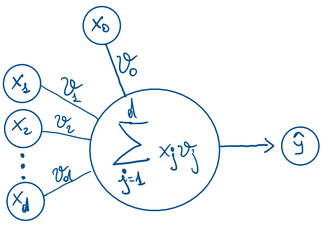
$\hat{y}$ = stima

y = etichetta del training

$$\hat{y} = \begin{cases} 1 & \sum_{j=1}^{d} x_j \vartheta_j > t \\ 0 & \text{otherwise} \end{cases}$$

↑ parametrized sign function

How to set threshold?
- Add bias $x_0 = +1$ and ϑ_0 in order to learn also the threshold



We have a model. How to train it? How to adjust parameters?

Perceptron Training (Libro: Neural Network - A Systematic Introduction, Cap 4)

Rosenblatt main innovation was on the training procedure for perceptron

Algorithm for learning perceptron

1) Initialize weights randomly with small values $\in [0,1]$ and set random threshold $\in \mathbb{R}$

$\vartheta_j \leftarrow \text{random}([0,1])$

$t \leftarrow \text{random}(\mathbb{R})$

2) Take samples $x^{(i)}$ and predict $\hat{y}^{(i)}$ by using perceptron model

3) For erroneous predictions update weights as follows:
- If the prediction $\hat{y}^{(i)} = 0$ and $y^{(i)} = 1$ then "increase weights"
- If the prediction $\hat{y}^{(i)} = 1$ and $y^{(i)} = 0$ then "decrease weights"

4) Repeat from 2 until "no errors are made"

How to write this with math?

$$\underbrace{\vartheta_j}_{\text{new } \vartheta_j \text{ at time } t+1} \leftarrow \overbrace{\vartheta_j}^{\text{weight value at time } t} + \underbrace{\left(y^{(i)} - \hat{y}^{(i)}\right)}_{\text{these can assume values}} \underbrace{x_j^{(i)}}_{\text{we can assume normalization in } [0,1]}$$

or we can substitute with a small $\eta \in [0,1]$

$y^{(i)}$	$\hat{y}^{(i)}$
0	0
0	1
1	0
1	1

Is the above procedure converging?
Who is t?
Who is ϑ?
Which is the model?
Who is P?
How to evaluate errors during training?

Cost function (Loss function) $\equiv$ count # of errors $\hat{y}^{(i)} \neq y^{(i)}$ with respect to the training set. (case of perceptron)

↑ We wish to minimize

output sum of pesate degli input

$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n = \sum w_j x_j$

Adaline algorithm variation

Approximate supervised → output continuo

Weights can be adjusted with a weighted sum of the input $\hat{y} = \sum_j \vartheta_j x_j$

$\uparrow^{c}$

$$v_0 + x_1 v_1 + x_2 v_2 = 0$$

decision boundary

$\hookrightarrow$ eq. retta

iperpiani

al variare del vasto input

$$x_2 = \overset{-0{,}55}{\left(\frac{v_1}{v_2}\right)} x_1 - \frac{v_0}{v_2}$$

$$y = mx + q$$