Lezione 5 25/10/2023

Le ipotesi sono gli iperpiani separatori, la scelta delle ipotesi è fatta spostando il piano in base agli errori.

Si può quindi creare un piano dove i punti sono gli esempi, che avranno valori +1 o -1 in base a da che parte dell'ipotesi sono.

La regola del percettrone

Ora non guarerò più solo se sbaglio sull'esempio che ho ricevuto, ma farò delle scelte intelligenti su come muovere il piano. Rileggiamo quindi il percettrone in un modo nuovo:

$w'_i := w_i + \Delta w_i = w_i + \eta(t-y)x_i$

- $t = c(x)$ è il valore target
- y è l' output del percettrone
- η è una piccola costante (e.g. 0.1) detta "tasso di apprendimento"

Quindi prima se y=0 ma doveva essere 1 (target = 1) facevo +x. Ora invece sommo l'"errore", ovvero la distanza tra il mio valore y e il target. Posso anche dosare questo errore con la costante del tasso di apprendimento.

Discesa lungo il gradiente

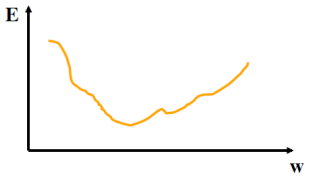
Si consideri una unità lineare, con stati continui e output
$y = w_0 + w_1x_1 + \ldots + w_nx_n$

Qui non c'è lo "scatto", non c'è soglia.

Strategia di apprendimento: minimizzare una opportuna funzione dei pesi w_i, ad esempio l'errore quadratico
$E[w_0,\ldots,w_n] = \frac{1}{2}\Sigma_{d\in D}(t_d-y_d)^2$
ove D è il training set

Qui possiamo calcolare l'errore di un training set D dati i pesi W.

Tecnica di minimizzazione: discesa lungo il gradiente (rispetto ai pesi)



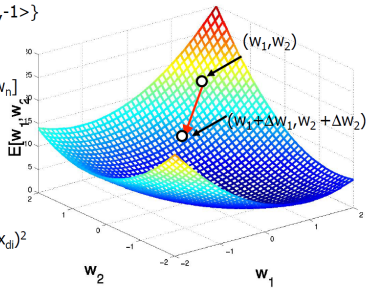
D={<(1,1),1>,<(-1,-1),1>,
<(1,-1),-1>,<(-1,1),-1>}

Gradient:

$\nabla E[w]=[\partial E/\partial w_0, \dots \partial E/\partial w_n]$

$w' = w + \Delta w$
$= w - \eta \nabla E[w]$

$\Delta w_i = -\eta \partial E/\partial w_i$
$= -\eta \partial/\partial w_i \, 1/2 \Sigma_d (t_d - y_d)^2$
$= -\eta \partial/\partial w_i \, 1/2 \Sigma_d (t_d - \Sigma_i w_i x_{di})^2$
$= -\eta \Sigma_d (t_d - y_d)(-x_i)$



Abbiamo tante ipotesi (il piano che separa) date dai vettori w e da ciascun errore.

Il punto più basso di questo grafico sarà quindi l'**ipotesi ottimale**.

Però calcolare l'errore su tutti i dati del dataset è molto oneroso.

L'idea è quindi quella di scendere sul gradiente dell'errore.

Anzichè calcolare tutti i possibili errori dei vettori che sarebbero tanti visto che ci sono due dimensioni da esplorare, parto da un'ipotesi corrente, e miglioro (ovvero scendo) (il delta).

Passo quindi dal vettore w al vettore w' sottraendo l'errore moltiplicato per la costante di apprendimento.

Non conoscendo l'errore, possiamo approssimarlo come scritto sotto. La sommatoria di d è il gradiente. Questo è il calcolo di un errore che approssima quale sarà il gradiente vero. Quello che era il percettrone sarà quindi più o meno l'approssimazione del gradiente dell'errore.

Questa è una formulazione generica, ma è per noi il passaggio verso l'approccio moderno, perchè vogliamo in qualche modo poter approssimare il gradiente di modo che la nuova ipotesi sia migliore di quella corrente.

Se l'aggiornamento dell'ipotesi (e quindi dei pesi) dipende dall'insieme di esempi D, decidendo un peso che è la costante di apprendimento, abbiamo che ogni istanza di apprendimento può essere usata per migliorare i pesi.

Passaggio per passaggio stiamo quindi introducendo elementi nuovi che abbandonano l'idea del percettrone: il peso, il calcolo dell'errore, e l'aggiornamento dei pesi.

DG(*esempi*, η)

Un esempio è una coppia $\langle(x_1,\ldots x_n), t\rangle$ ove $(x_1,\ldots,x_n)$ è il vettore di input, t è l'output corrispondente, η è il tasso di apprendimento (e.g. 0.1)

- Si inizializza ogni w_i con un piccolo valore casuale
- Fino al raggiungimento della condizione di terminazione:
  - Inizializza ogni Δw_i a zero
  - Per ogni $\langle(x_1,\ldots x_n),t\rangle$ in *esempi*
    - Invia l'input $(x_1,\ldots,x_n)$ all'unità lineare e calcola l'output y
    - Aggiorna la variazione dei pesi
      - $\Delta w_i = \Delta w_i + \eta\,(t-y)\,x_i$
  - Aggiorna i pesi
    - $w_i := w_i + \Delta w_i$

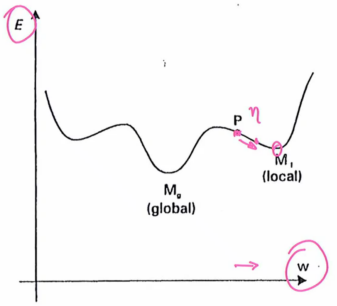
- Modalità Batch (computazionalmente costosa):
  $w = w - \eta \nabla E_D[w]$ calcolato rispetto all'**intero** insieme D
  $$E_D[w]=1/2\Sigma_d(t_d-y_d)^2$$
- Modalità Incrementale:
  $w = w - \eta \nabla E_d[w]$ calcolato rispetto a **singoli** esempi d
  $$E_d[w]=1/2\,(t_d-y_d)^2$$
- La discesa lungo il gradiente incrementale può approssimare la discesa lungo il gradiente Batch arbitrariamente se η è abbastanza piccolo

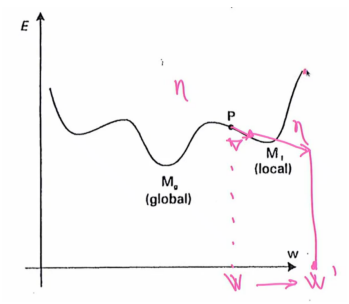
Il percettrone aggiornava i propri pesi un esempio alla volta, quando c'era un errore. Qui invece aggiorniamo con un insieme di vettori D (esempi) alla volta. Questo metodo è chiamato **Batch**. è un metodo lento.

Più simile al percettrone è invece la **modalità incrementale**, che prende un vettore alla volta e migliora l'ipotesi. Questo approccio è più rischioso, di conseguenza sarà necessario una costante di apprendimento più piccola.

Quindi con la funzione che calcola gli errori per tutti i possibili esempi riesce a trovare il minimo errore possibile nel gradiente con tutti gli esempi, ma dato che ciò non è possibile noi usiamo questi metodi per approssimare l'errore minimo.

L'approssimazione ha però rischi, come quello di entrare in un minimo locale, perchè non si ha una visione globale del problema.



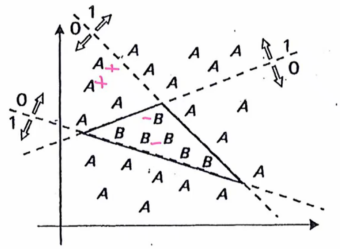


Oppure se ho un eta troppo grande potrei spostarmi troppo e trovarmi in una situazione dove in realtà l'errore è più grande.

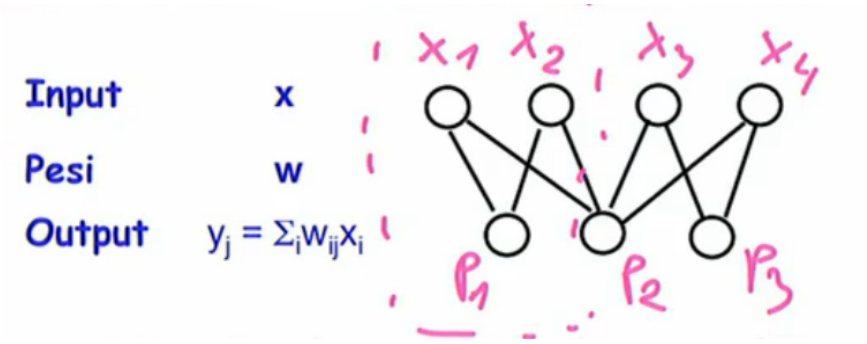
Ora proviamo a risolvere il problema di quando i punti non sono separabili.

Se i punti non sono separabili da una linea, può essere che possano essere separati da più linee. Ciascuna linea è un iperpiano separatore.

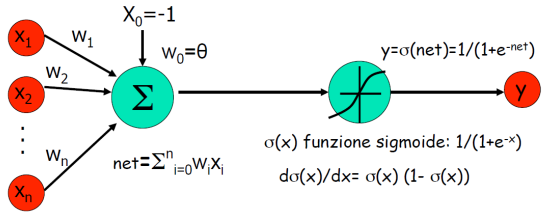
Quindi viene confrontato ciascun iperpiano, e solo uno deve ritornare in output 1, l'output è quindi un OR dei piani.



Addestramento rete a due strati



Se prima si usava l'OR, ora vogliamo che questa diventi una valutazione fatta da un neurone. Io posso quindi calcolare l'errore finale di ciascun percettrone, ma come faccio a cambiare i pesi dei nodi precedenti?



Innanzitutto vogliamo poter calcolare l'errore analiticamente e senza approssimarlo. Usiamo quindi una derivata, che ritrova l'espressione in basso a destra.

è analitica perchè per ogni x posso calcolare quanto cambia la pendenza della funzione al variare dei pesi.

Partendo da questa funzione che è derivabile vogliamo poi arrivare a derivare anche l'errore, ma non ancora qui.

Possiamo quindi avere più strati, e ogni neurone calcola l'uscita con una sigmoide. Il segnale complessivo che entra è il prodotto interno del vettore x e del vettore w, rappresentato analiticamente come sommatoria per j sulle connessioni entranti del peso per il segnale che precede il neurone cioè x_j.

Stato di un neurone: $x \in [0,1]$

Funzione di transizione: $x_k = \sigma(\sum_j w_{jk} x_j)$ con $\sigma(x)$ funzione sigmoide

Questo calcolo deve però essere fatto per tutti gli strati. è quindi una concatenazione di funzioni, x in ingresso diventa output dei primi neuroni che diventano input del secondo strato, etc.

Però alla fine io vedrò soltanto l'output che rappresenta l'errore, ma i pesi sono contenuti negli strati interni in diversi neuroni.

Quindi se io voglio apprendere a più strati devo scoprire qual è la f complessiva, e in base alla f calcolare l'errore rispetto all'obiettivo almeno approssimativamente.

Grazie alla derivabilità della funzione sigmoide, l'aggiornamento dei pesi interni è possibile andare a ricavare i gradienti interni. Questo si farà facendo una composizione di derivate. Useremo l'algoritmo di back propagation.

Questo metodo però è sempre limitato dal problema dei limiti locali. C'è anche rischio di overfitting sul dataset dato.

Per migliorare si può tarare il tasso di apprendimento, si può usare un range di stati (non solo 1 -1), si possono variare il numero di strati e neuroni, etc.