

La funzione Loss

Quella vista nella lezione di learning. Passa quantifica la differenza tra le predizioni di un modello e gli output reali. Quindi quindi l'errore medio di generalizzazione.

I valori ottimali sono quelli che minimizzano F .

Ci sono 2 tipi di funzioni loss in base al problema

• regression loss
output \equiv 1 o più val.

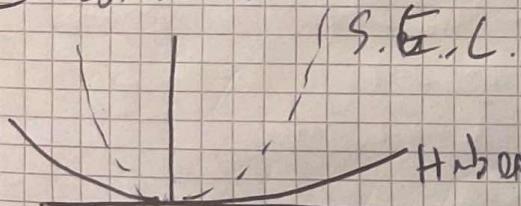
• classification loss
output: dichiarazione prob.

Regression Loss

Squared error loss: $L(y, \hat{y}) = (y - \hat{y})^2$

Huber loss: $L(y, \hat{y}) = \begin{cases} (y - \hat{y})^2 & \text{se } |y - \hat{y}| \leq \delta \\ 2\delta(|y - \hat{y}| - \frac{1}{2}\delta) & \text{altrimenti} \end{cases}$

$\delta \equiv$ costante



Huber $\delta = 1$

$$\text{MSE} : \text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Un'altra sinossi è l'RMSE ovvero la radice del MSE.

Nella pratica si usa l'MSE perché il calcolo della sua derivata è già semplice ed efficiente per l'algoritmo di risalita del gradiente.

A causa del quadrato di potenza terribile e per ridurre di molto il valore di MSE

Per evitare questo problema è comune sostituire l'errore medio quadrato Huber con

Nel caso l'output abbia già valori quindi si può utilizzare il calcolo finendo lo stesso ma con $\|y - \hat{y}\|$ al posto di $|y - \hat{y}|$

Classification Loss

Abbiamo n classi.

Il training set è formato da coppie (\vec{x}, \vec{p}) dove \vec{x} è l'input e \vec{p} è la dist. a prob.

$p_c = \text{prob. } x \in \text{classe } c$

E se la rete è progettata per dare in output una dist. di prob. \vec{q} dove \vec{q} deve essere softmax tra \vec{p} e \vec{q} allora le quantificano la distanza

Entropia

Ci sia un alfabeto di n simboli.
Si vuole trasmettere un messaggio con d in simboli.
In ogni punto del messaggio la prob di apparire
di ciascun simbolo è p_i .
L'entropia necessaria per codificare
il messaggio è $H(\vec{p}) = -\sum_{i=1}^n p_i \log p_i$.

$-\log p_i$ è il numero di bit necessari per codificare il simbolo i .

Uno schema ottimale via $H(\vec{p})$ bit.

Supponiamo l'output della rete sia \vec{q} .
Per ogni simbolo i servono $-\log q_i$ bit.

Siamo in uno schema di codifica sub-ottimale
e serve questo numero di bit (medio).

$$C(\vec{p} \parallel \vec{q}) = -\sum_{i=1}^n p_i \log q_i$$

Questo opposto è detto Entropia incodificabile
e fa sì che il quanto \vec{q} (output) è più (più T)

Questo numero dice quanto tempo servirà.

$$C(\vec{p} \parallel \vec{q}) \geq H(\vec{p})$$

Vogliamo che C sia più vicina possibile a H .

Vogliamo minimizzare la divergenza Kullback-Leibler
che è la differenza tra C e H .

Minimizza il numero medio di bit in cui deve apparire
ogni simbolo.

$$KL(\vec{p} \parallel \vec{q}) = C(\vec{p} \parallel \vec{q}) - H(\vec{p}) = \sum_{i=1}^n p_i \log \frac{p_i}{q_i}$$

Minimizzare $KL \Leftrightarrow$ Minimizzare C . \rightarrow Nella pratica
Finito è C_{\min} KL

Training di uno Rete neurale

Caricato nel lavoro i dati che minimizzano il costo medio in un training set.

L'obiettivo è garantire un costo medio basso.
Data la gran numero di parametri il rischio di overfitting è alto.

Gradiente

$$x = x_1 \dots x_m$$

Il gradiente di y rispetto a x è il vettore le cui componenti sono le derivate parziali prime di y rispetto ad ogni x_i .

$$\nabla_x y = (\frac{\partial y}{\partial x_1} \dots \frac{\partial y}{\partial x_m})$$

La matrice Jacobiana detta in gergo rete di output delle funzioni avrà le colonne y e \vec{y}

La matrice Jacobiana di \vec{y} rispetto a \vec{x} è la matrice formata dalle derivate parziali prime di ogni componente di \vec{y} rispetto a ogni componente di \vec{x}

$$J_{\vec{x}}(\vec{y}) = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial y_1}{\partial x_m} & \dots & \frac{\partial y_m}{\partial x_m} \end{pmatrix}$$

Metodo di Bisez del gradiente

È una tecnica per trovare il minimo di una funzione anche se già variabile.

La funzione da minimizzare è la loss

Permette da una posizione iniziale il metodo iterativamente cerca la direzione di massima crescita del valore della funzione e aggiorna i valori procedendo in quella direzione.

La direzione di massima crescita è opposta al gradiente o della Jacobiana se l'output è un vettore

Algoritmo

$f: \mathbb{R}^m \rightarrow \mathbb{R}$ funzione da minimizzare

\vec{x}_t vettore d'iniziativo di valori calcolato all'istante t

1) $t=0$: Calcolo \vec{x}_0 lombard

2) Calcolo $y_t = f(\vec{x}_t)$ e gradiente $\nabla_{\vec{x}_t} y_t$

3) $\vec{x}_{t+1} = \vec{x}_t - \eta \nabla_{\vec{x}_t} y_t$

4) $t++$, ripeti da 2

Il metodo è fatto fino a convergenza.

E' analogo per la Jacobiana.

I minimi locali (ma non assoluti) possono dare problemi

Learning rate η

Determina la velocità con cui si verifica che il metodo converge

Valori bassi \Rightarrow troppe iterazioni

Valori alti \Rightarrow causano oscillazioni troppo alte entendo di arrivare a convergenza

Un metodo per trovare un buon valore di η consiste nel partire da un valore alto e ad ogni passo moltiplicare η per β ($0 < \beta < 1$) fino a quando non trovi un valore idoneo

Inizializzazione dei pesi delle reti

Occhio routine da che valori iniziare dei pesi.
Invece di fare completamente random facciamo in modo
che i pesi siano i pesi iniziali dei vari neuroni.

Ci sono 2 approcci:

- Scegli tra -1 e 1 un solo vettore uniforme
- Scegli in maniera random secondo una Normale

Stochastic gradient descent

Variante corrente: Ogni iterazione lavora su un piccolo
campione di dati del training nel senso di sotto a sotto

Più veloce del metodo classico e sbatto a Th grad
Il gradiente deve essere calcolato in qualche modo.

Un algoritmo è la backpropagation che usando
il grafo computazionale calcola il gradiente in
maniera veloce.

Il grafo computazionale è un grafo ocidis diretto

DAG che contiene il flusso di dati di una rete neurale.

A ogni nodo è associato un operando e
oggi solitamente un operatore

Operando può essere: un valore, un vettore, una matrice

L'operazione può essere un operatore algebrico, una
funzione d'attivazione o una Loss.

Un arco collega A e B se l'output prodotto da A
è passato al nodo B

Se ad esempio A è un nodo e B è un altro nodo
l'arco associa l'operazione ai valori input ricevuti.

- Apprendendo l'operazione si valori input ricevuti
- Apprendendo il risultato dell'operazione del nodo

Una rete di grafo organizzando permette di creare la regolarità di operazioni da eseguire in un solo momento.

Back propagation

Allora se il grafo in senso ha elementi, ovvero nell'ultimo strato, si esegue operazione.

Si parte dal gradiente della funzione Loss L rispetto a \vec{y} (output).

A ritroso calcoliamo i gradienti di L rispetto agli altri nodi del grafo.

Facciamo ora finché non i ~~other nodes~~ ovvero il gradiente di L rispetto all'insieme dei dati W.

Per questi calcoli si usa la formula della catena.

Regole della catena

Se $y = g(x)$ $z = p(y) = p(g(x))$ segue

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

La regola si estende a funzioni a più variabili.

$$\frac{\partial z}{\partial x} = \sum_{i=1}^m \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

La regola si estende a funzioni di vettori usando gradienti e Jacobiane.

Se $\vec{y} = g(\vec{x})$ $z = p(\vec{y}) = p(g(\vec{x}))$ segue

$$\nabla_{\vec{x}} z = J_{\vec{x}}(\vec{y}) \nabla_{\vec{y}} z$$

Se $\vec{y} = (\vec{y}_1, \dots, \vec{y}_m)$ $\vec{y}_i = g_i(\vec{x}) \dots \vec{y}_m = g_m(\vec{x})$ segue

$$\nabla_{\vec{x}} z = \sum_{i=1}^m J_{\vec{x}}(\vec{y}_i) \nabla_{\vec{y}_i} z$$

Ad ogni passo il gradiente di L rispetto ad un operando O di un nodo N è calcolato prendendo in considerazione i gradienti di L rispetto agli operatori nei nodi vicini.

Il gradiente di L rispetto ad O è calcolato usando le formulazioni della regola della catena.

La formulazione visto di sopra dal momento è necessaria nel caso del tipo di operando (numero o vettore).

L'ultimo passo dell'algoritmo consiste nel calcolo di ∇_{WL} . W è una matrice, non sono uscite la regole della catena.

Vediamo W come un vettore di vettori $W_1 \dots W_m$
 $W_i \equiv$ i-esima colonna.

Abbiamo quindi da calcolare $\nabla_{W_i} L$ e di conseguenza anche ∇_{WL} .

Continuiamo con algoritmo

Rete neurale R , training set T

- 1) Apreggia W matrice per inizializzarla random
- 2) Addestra R con W in T
- 3) Calcola W sotto gli output predetti e gli output reali L con L media.
- 4) Fai back propagation e calcola ∇_{WL}
- 5) Usando il metodo di discesa del gradiente aggiorna la matrice $W = W - \eta \nabla_{WL}$
- 6) Ripeti da 2 a 5 fino a che la loss L non decresce più in modo significativo oppure dopo un numero limitato di volte.
(η si dice "epoch")

Con i valori ottenuti ad ogni iterazione si calcola la Loss.

Obiettivo: far diminuire ad ogni epoca il valore della Loss sia sul Training set sia sul Test set.

Lo scopo finale è minimizzare la Loss media.

Un problema comune è quello di costruire un modello molto buono sul training set ma che non generalizza e non funziona su nuovi input. Siamo in Overfitting.

Risolve overfitting \rightarrow Modello di regolarizzazione

Si osserva che i valori hanno valori assoluti > costante per molti modelli più generalizzabili.

Si forza il metodo del gradiente a aggiungere un termine di perdita alla Loss L0.

In questo modo i pesi sono tutti <= valore dei costi.

Regolarizzazione L1 e L2

$$(1) L = L_0 + \alpha \sum_{i=1}^k |W_k|$$

LASSO

$$(2) L = L_0 + \alpha \sum_{i=1}^k W_k^2$$

RIDGE

α è un iper-parametro tra 0 e 1 = parametro di regolarizzazione

L2 forza i pesi vicini a 0 mentre con L1 almeno un peso avrà una additiva > 0

L1 è preferibile se si vuole fare feature selection ponendo a 0 le feature meno importanti.

Variante

In base alla regularization la regolarizzazione che lavora sommando il termine alla fine con

Dropout

Scop: Evitare i classi a diventare soffici.

Tecnica simile al bootstrapping.

Prefettibile se la testa reale è grande.

Ad ogni epoca si seleziona un numero ~~casuale~~ di nodi random e si cancellano.

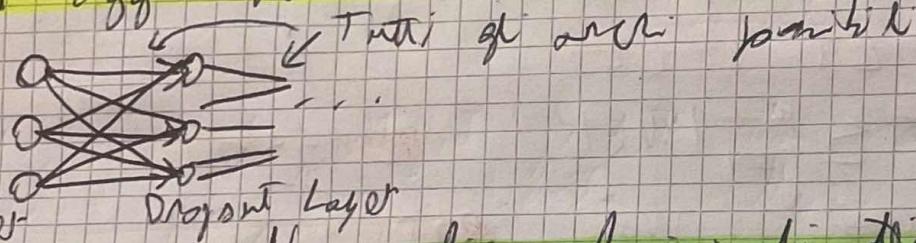
Sulla rete ribalta si effettua l'itiazione.

La frequenza di nodi cancellati ad ogni passo è la ~~stessa~~ dropout rate.

Alla fine del training set i vari nodi filtrati cioè moltificati per il dropout rate.

Perché? Training più veloce e si abbetta la rete a lavorare su dati ancora più diversi.

Al layer si applica il dropout layer prima un layer seguente totalmente connesso al layer.

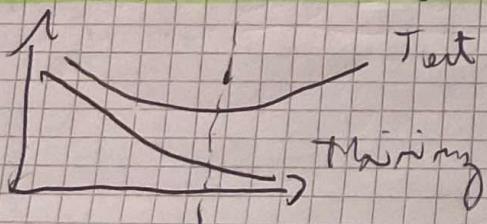


Non applica funzioni di attivazione o altro. Filtra solamente il layer corrispondente (ovvero il precedente).

Il filtro si fa distinguendo alcuni nodi del dropout layer.

Early Stopping

Nella realtà si osserva che non riuscire a minimizzare la loss L, ma decresce leggermente e a crescere nulla Test set



La tecnica per evitare questo problema consiste in stoppare appena la loss nel Test inizia a crescere.

Con l'early stopping si può sbloccare in entro a
overfitting nel training set.

Per evitare ciò si usa un terzo set: validation.
Si effettua l'early stopping. Mentre training e validation
set. Si stoppa l'ottimizzazione del training appena la loss
rispetto al validation set inizia a crescere

Aumento del training set

L'aumento delle tute neural aumenta
se ci sono già dati nel training set.

Training set piccolo \Rightarrow Aggiungere dati artificiali.
I dati artificiali applicando trasformazioni ai dati del
training set o aggiungendo rumore.

Aggiungendo (senza esagerare) rumore molto articolato
per il modello e adattare meglio la tuta.

Esempio: tuta usata per classificare immagini.
Aggiungere immagini finte solo le immagini bidimensionali,
aggiungendo un po' di rumore...

Tipologie di tute neurali

Feed - Forward Networks, FFNs

Le informazioni si muovono in una sola direzione.
Ciò non può avvenire cicli.

Convolutional Neural Networks, CNNs

Contiene 1 o più layer convolutionali.
I pesi degli input sono gli stessi per tutti i nodi
del layer convolutionale stesso.

In genere si ottengono layer convolutionali a layer
pooled con un numero di nodi che minimizza
il moto progressivo.

Le CNN sono viste esclusivamente nelle classificazioni delle immagini. Minimo i fototecnici utilizzati.

I nodi di un convolutional layer si pensa immaginare come dei filtri.

Il filtro ~~è~~ deve lavorare su un quadrato $p \times p$.
A sua volta gusta nel layer precedente.
 p : dim filtro. Nelle immagini: $p \times p$ pixels.

Considerato il pixel x_{ij} il filtro è applicato sul quadrato dove x_{ij} è in alto a sin.

Output del nodo $x_{ij} \Rightarrow z_{ij} = \sum_{k=0}^p \sum_{l=0}^p w_{kl} x_{i+k, j+l}$

E' possibile usare anche altri schemi: x_{ij} pixel centrale

Attivazione Obiettivo convolutional layer: output stessa dim input

Obiettivo Pooling Layer: output ~~stessa~~ dim minore dell'input

Stride e padding

Per calcolare gli output probabili dai nodi del convolutional layer basta osservare:

- Scorrere il filtro
- Applicare ad ogni posizione

Stride: indica di quante posizioni il filtro si sposta per arrivare da una posizione alla successiva.

Soltanto $\text{Stride} = 1 \Rightarrow$ Output stesso dim input
Se $\text{Stride} > 1 \Rightarrow$ output più piccolo

La matrice di output potrebbe rimettere problema da collocare ~~a~~ secondo sulla dim dell'input che potrebbe non permettere una completa convolution.

Soluzione: zero-padding, ovvero aggiungere righe e colonne 0 alla matrice

Pooling layer

Prende come input l'output di un convolutional layer, produce un output più piccolo.

Ribassone dimensione: dovuta a una funzione di pooling come la funzione max. Essa aggiunge una regine di valori a un solo valore.

Una funzione di pooling agisce su un quadrato di $k \times k$ di valori input.

Pur ottenere l'output completo di una convoluzione con filtro col parametri stesse

$\frac{1}{2} \text{ etto} \rightarrow$ output più piccolo
 $\frac{1}{4} \text{ etto} \rightarrow$ output più piccolo

Valori eccessivamente alti di $f \Rightarrow$ perdita informazione

CNN si può applicare anche nelle immagini a colori: 3 canali R G B

Il filtro non sarà più un quadrato $f \times f$ ma una struttura $f \times f \times 3$

Un solo filo di un layer convoluzionale avrà $f \times f \times 3$ per

Esempio rete neurale

FC con rientro immagini: VGG16

Recurrent Neural Networks (RNN)

Modello nel text mining

Passo contrario cioè

L'output di un layer può diventare l'input per un layer successivo (o lo stesso layer)

E' equivalente a una rete in cui tutti i passi fanno 1 o più volte

I layer riguardano i vari step come memoria delle stesse

per ricordare valori avvistati in precedenza

Sono abili quindi a rappresentare temporali di valori.

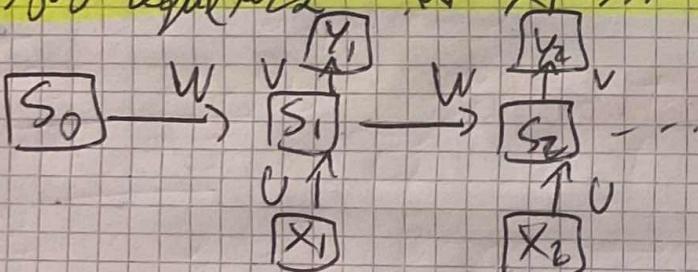
Ese: testo. Si può stabilire un rapporto binario nel tempo.

Questo perché le RNN sono abili alla predizione di valori successivi a partire da una sequenza di eventi avvistati.

Input: $(\vec{x}_1 \dots \vec{x}_n)$ Output: $(\vec{y}_1 \dots \vec{y}_n)$

I pesi sono rappresentati da 3 matrici U, V e W

\vec{s}_t stato nascosto funzione di memoria
tutte info memorizzate prima e t mello
regressione $\vec{x}_1 \dots \vec{x}_t$



Per calcolare s_t si sommano W e U più y
e moltiplicare per V

\vec{s}_0 è un vettore di 0.

Se si ottiene applicando una funzione di attivazione
non lineare σ all'input al passo t e \vec{s}_{t-1}

$$\vec{s}_t = \sigma(U\vec{x}_t + W\vec{s}_{t-1} + b) \quad b = \text{vettore bias}$$

L'output al tempo t è ottenuto applicando una
forza di attrazione considerando lo stato
attuale.

$$\vec{F}_t = g (\vec{V}_{S_t} + \vec{C}) \quad \vec{C} = \text{vettore di forza}$$

In certe applicazioni è necessario un solo output
della fine del processo.

In questo caso si pensa di ottenere prodotti ad
ogni step su uno "step" totalmente connesso
per generare l'output finale.

Gestire la lunghezza variabile (e: le parole)

- tokens-pulling: si pisa una lunghezza max.
Se una parola è meno di n si "completa"
con caratteri fissi. Se è più di n si tronca.

- bucketing: si raggruppa le parole nella base
della loro lunghezza e si entra in una
RNN diversa per ogni parola volta a lunghezza.

Risultato si vede un approccio ibrido.

Si usano diversi bucket che gestiscono singolarmente
certe lunghezze.

Ogni parola si associa ad un bucket in modo da
gestire separatamente quella lunghezza. O leggermente
più lunghe.

Se necessario ad un bucket scelto si fa padding.

Limiti

La memoria non è "a lungo termine".

Determinare il tempo che parla vicino, non basta.

Un verbo potrebbe creare paradossi lontani dal soggetto
in una frase.

Per il momento questo è tipico in problemi
di imparabilità di valori (soluzioni o esplosione).

Long-Short-Term Memory LSTM

Modificano le RNN per ricordare relazioni a lungo distanza.

Proprietà

- Forget: eliminare dalla memoria info non più necessarie.
- Save: capire di dove prendere info nella memoria a lungo termine.
- Focus: capacità di focalizzarsi solo su argomenti della memoria che sono rilevanti.
Questo in base alla situazione specifica in cui sono focalizzarsi nella memoria.

Per avere questo tipo di proprietà non accadeva nulla memoria a lungo termine e una memoria concreta.

- Long-term: ha info già memoriate.
- Concrete: info di immediata rilevanza.

Struttura

Gli stati rappresentano i 2 tipi di memoria.

- State nascosto: S_t in t indica stato concreto.
- Cell state: C_t in t indica memoria a lungo termine.
 $\xrightarrow{h_t}$: Cambia temporaneamente a S_t .

Salvo

Vettori usati per far passare in modo selettivo alcune info di uno stato, il resto è zero.

- Input gate: I_t determina cosa di F_t dev'essere mossa.
- Forget gate: F_t indica quali aspetti della long term memory.
- Output gate: O_t indica cosa spostare dalla long alla concreta.

Ajornamento del cell state (memoria a lungo termine)

- 1) Calcola update terprimes dell' stato & $\vec{s}_t = \vec{r}_t$
- 2) Calcola input gate e forget gate
- 3) Ajornare memoria lungo termine

$$\vec{c}_t = \vec{c}_{t-1} \circ \vec{p}_t + \vec{r}_t \circ \vec{i}_t$$
 - \circ = prodotto & Hadamard, molto più facile che vettori
o matrici

Ajornamento stato nascosto (memoria corta)

- 1) Calcola output gate (h_t è da fare su input di stato precedente s_{t-1})
- 2) Ajornare memoria cortante

$$\vec{s}_t = \tanh(\vec{c}_t \circ \vec{o}_t)$$

Calcolo dell'output

Come una qualcosa RNN

$$\vec{y}_t = g(V \vec{s}_t + \vec{b})$$

$$V \equiv \text{wei}$$

$$\vec{b} \equiv \text{bias}$$

Catene di Markov

Ti tipo (Q , $I = \{p(\pi_1 = s)\}_{s \in Q}$, A)

- $Q = \{1, 2, \dots, k\}$ insieme finito di stati
 - π_1, π_2 denotano gli stati osservati a ogni istante
 - I : insieme prob. iniziali
 - A : insieme prob. di transizione destra da sinistra per ogni U, V in Q
- $$a_{UV} = P(\pi_t = V | \pi_{t-1} = U)$$

In questo caso la prob. del 1° stato, ovvero catena deve essere la prob. di trovare in π_1 il quale sarà da π_{t-1} .

Si dice che la catena non ha memoria.

s_1, \dots, s_t : sequenza stati osservati

$$\begin{aligned} P(\pi_t = s_t | \pi_1 = s_1, \dots, \pi_{t-1} = s_{t-1}) &= \\ &= P(\pi_t = s_t | \pi_{t-1} = s_{t-1}) = a_{s_{t-1}, s_t} \end{aligned}$$

Sono s, r, r, r, sn, sn

$$P(s, r, r, r, sn, sn) =$$

$$= P(s) \times P(r|s) \times P(r|r) \times P(r,r) \times P(sn|r) \times P(sn)$$

$$P(\pi_1, \dots, \pi_m) = P(\pi_1) \cdot \prod_{i=1}^{m-1} P(\pi_{i+1} | \pi_i)$$

Matrice Stocistica

	s_1	\dots	s_k
s_1			
s_2			
\vdots			
s_k			

$$\sum_{j=1}^k P(\pi_t = s_j | \pi_{t-1} = s_i) = 1$$

Per ogni s_i la prob. di trovare in s_k (V.K.) al massimo dunque 1

Proprietà Stocistica

Catene di Markov Istruttive

Dato ogni stato c'è già fissato quale stato sarà stato.

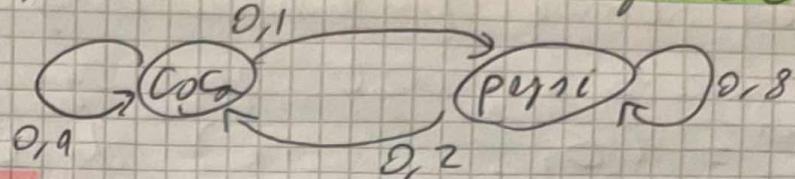
Ajutando

Perché stato è probabile da cui viene

Se $p_{ii} > 1$ → è assorbito

Un altro Markov Ajutante ha 0 stati assorbiti

$$P = \begin{bmatrix} 0,9 & 0,1 \\ 0,2 & 0,8 \end{bmatrix}$$



Prob. eventi successivi

Per una persona che compra jeans a t quel 't' b
prob che a $t+2$ compri Coca Cola

$$\begin{aligned} P(\pi_{t+2} = \text{Coca} \mid \pi_t = \text{Pepsi}) &= P(\pi_{t+2} = \text{Coca}, \pi_{t+1} = \text{Coca} \mid \pi_t = \text{Pepsi}) \\ &+ P(\pi_{t+2} = \text{Coca}, \pi_{t+1} = \text{Pepsi} \mid \pi_t = \text{Pepsi}) = \\ &= 0,2 \cdot 0,9 + 0,8 \cdot 0,2 = \underline{0,35} \end{aligned}$$

$$P^2 = \begin{bmatrix} 0,9 & 0,1 \\ 0,2 & 0,8 \end{bmatrix} \begin{bmatrix} 0,9 & 0,1 \\ 0,2 & 0,8 \end{bmatrix} = \begin{bmatrix} 0,83 & 0,17 \\ 0,35 & 0,65 \end{bmatrix}$$

Basta moltiplicare per un altro per avere le prob
di pross. stati (oppure $t+2$)

Per domani $t+3$? P^3

Mettiamo tutte e quattro le prob iniziali

0,6 Coca

0,4 jeans

Per cui le persone che comprano Coca negli ultimi 3 anni?

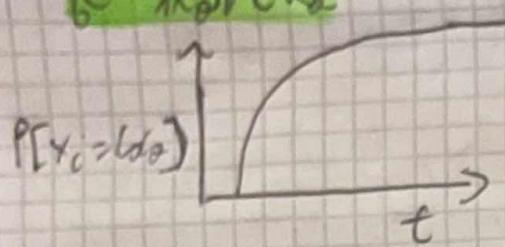
$$D_3 = D_0 * P^3$$

$$D_0 = (0,6, 0,4) \rightarrow \text{iniziali}$$

$$D_i = \text{dist. alle temp i}$$

Dist. iniziale

Anzalduo t minimo da D dove avviorare
la markov



Si dimostra che: $\lim_{n \rightarrow +\infty} P(\Pi_n = j | \Pi_1 = c) = \lim_{n \rightarrow +\infty} A_{cj}^n = G_j$
dove G_j è il prob. corrispondente al limite

Per lezioni i calcoli per ogni j ottengono
 $\vec{G} = (G_1, \dots, G_k)$ che è la dist. stazionaria

Se \vec{G} ha valori non nulli $\rightarrow G = \text{dist. stazionaria}$

Se \vec{G} ha tutti valori non nulli $\rightarrow G = \text{chica dist. staz.}$

Hidden Markov Model HMM

Gli stati sono "nasconduti"

Ogni stato emette un certo simbolo con una prob.

L'osservazione vede solo la sequenza di simboli

$$H_1 \rightarrow H_2 \dots \rightarrow H_L$$

$$\downarrow \\ X_1 \rightarrow X_2 \dots \rightarrow X_L$$

dai osservabili

~~gli stati nasconduti~~

HMM è una quintupla $(Q, I = \{p(\Pi_i = s)\}, A, \Sigma, E)$

• $Q = \{1, 2, \dots, K\}$ insieme finito di stati

• Π_1, Π_2, \dots : stati osservati a ogni istante

• T : prob iniziali

• A : prob A_{UV} $\forall U, V$ in Q

• $\Sigma = \{S_1, \dots, S_m\}$ alfabeto di simboli

• X_1, X_2, \dots : simboli emessi a ogni istante

• E : matrice emissioni dim: $K \times m$

E contiene più oggetti e ogni vettore b può essere un altro b.

$$c_{ob} = P(X_t = b | \pi_t = 1)$$

Problemi principali in HMM

Evaluation: Data HMM M e seq. X calcolare $P(X)$

Decoding: Data HMM M e seq. X , trovare seq. π di stati che maximizza $P(\pi | X)$

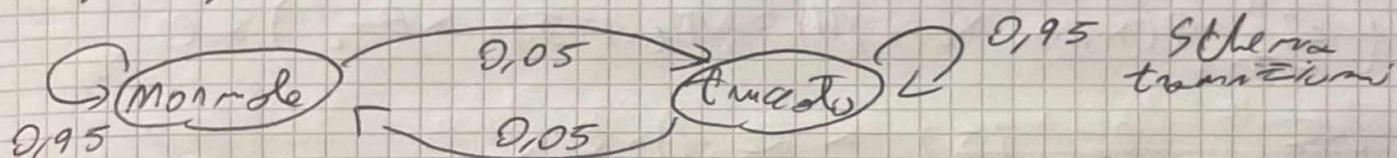
Learning: Data HMM M con prob. b di transizione non specificate e seq. X trovare parametri $Z = (e_{ij}, \alpha_{ij})$ del modello che maximizzano $P(X | Z)$

Esempio Chorister dinamico

Data monade: $P(i) = 1/6$

Dato Chorister $P(1) = P(2) = \dots = P(5) = 1/10 \quad P(6) = 1/2$

Chorister può contenere solo



Data una seq. di lanci, quale è la prob. che sia stata generata dal modello?

Quando il chorister ha fatto il loto quale monade è stata?

Stimare matrice di transizione e matrice di emissione

1) Data M e X $P(X) = ?$ Evaluation

$P(X_1, \dots, X_n, \pi_1, \dots, \pi_m) = \prod_{i=1}^m P(\pi_i, \dots, \pi_m) \times P(X_1, \dots, X_n | \pi_1, \dots, \pi_m) =$ $\prod_{i=1}^m P(\pi_i) \times \prod_{i=1}^{m-1} P(\pi_{i+1} | \pi_i) \times P(X_i | \pi_1, \dots, \pi_m) =$

$= P(\pi_1) \times \prod_{i=1}^{m-1} P(\pi_{i+1} | \pi_i) \times P(X_i | \pi_1, \dots, \pi_m) =$

$= P(\pi_1) \times \prod_{i=1}^{m-1} P(\pi_{i+1} | \pi_i) P(X_i | \pi_i) = P(\pi_1) \prod_{i=1}^{m-1} \delta_{\pi_i, \pi_{i+1}} c_{\pi_i, X_i}$

Per trovare $P(X)$ si devono sommare tutte le probabilità di tutti i simboli x che sono generate da π

$$P(X) = \sum_i P(X, \pi) \quad \pi \text{ regge } (t_n)$$

Definisco $p_t(i) = P(X_1 \dots X_t, \pi_1 \dots \pi_t = i)$ Prob. corrente

E' la prob. di ottenere una seq. di simboli e di generare un ~~certo~~ insieme di simboli dallo stato i .

Si può calcolare in maniera induttiva

$$p_t(i) = P(\pi_t = i) e_{i|x_t}$$

Passo induttivo

$$\begin{aligned} p_t(i) &= P(X_1 \dots X_t, \pi_1 \dots \pi_t = i) = \\ &= \left(\sum_{j=1}^k P(X_1 \dots X_{t-1}, \pi_1 \dots \pi_{t-1} = j) \times \alpha_{ji} \right) \times e_{i|x_t} = \\ &= \left(\sum_{j=1}^k p_{t-1}(j) \times \alpha_{ji} \right) \times e_{i|x_t} \end{aligned}$$

Il passo induttivo deriva dal fatto che se i è in t posso scrivere i con qualsiasi j (in $t-1$) dopo aver scritto i simboli $X_1 \dots X_{t-1}$

Per generare da i a j si fa una transizione, mettendo dunque x_t

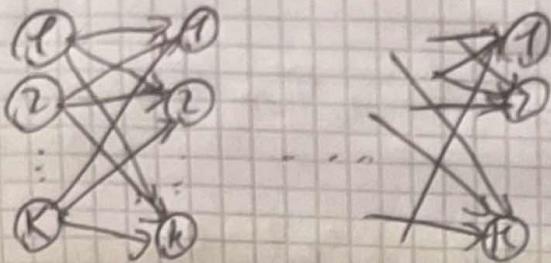
Grazie a queste proprietà posso ottenere $P(X)$

$$\begin{aligned} P(X) &= \sum_i P(X, \pi) = \sum_{i=1}^k P(X_1 \dots X_m, \pi_1 \dots \pi_m = i) = \\ &= \sum_{i=1}^k p_m(i) \end{aligned}$$

Grazie a questo prob. si ottiene un algoritmo (ogni fin) che calcola $P(X)$ in $\Theta(k^2 m)$

$k^2 \equiv$ stati
 $m \equiv lung(X)$

\otimes Anteprima l'algoritmo forward equivale a percorrere tutti i simboli emessi in grafo detto Lattice



Omission (intervalli)

$$x_1 \ x_2 \ \dots \ x_M$$

Decoding

Dato HMM M e seq X trovare $\Pi = (\pi_1^*, \dots, \pi_n^*)$ che maximizza la prob corrispondente di orecchie di n simboli emessi dagli n stati

$$P(X_1, \dots, X_n, \Pi_1^*, \dots, \Pi_n^*) = \max_{\Pi_1, \dots, \Pi_n} P(X_1, \dots, X_n, \Pi_1, \dots, \Pi_n)$$

Scrivere dove

$$P(X_1, \dots, X_n, \Pi_1, \dots, \Pi_n) = P(\Pi_1) \prod_{i=1}^{n-1} \alpha_{\Pi_i, \Pi_{i+1}} e_{\Pi_i} x_i$$

Usciamo l'algoritmo di Viterbi

Invece di tenere i contributi provenienti da ogni stato di volta in volta rendiamo il contributo migliore (val max)

Nel lattice andiamo a trovare un percorso lungo m

$$V_{tE}(c) = \max_{\Pi_1, \dots, \Pi_{tE-1}} P(X_1, \dots, X_t, \Pi_1, \dots, \Pi_{tE-1}, \Pi_t = c)$$

Così base: $V_t(c) = P(\Pi_t = c) \times e_c x_t$

Poco induttivo: $V_t(c) = \max_{\Pi_1, \dots, \Pi_{tE-1}} P(X_1, \dots, X_t, \Pi_1, \dots, \Pi_{tE-1}, \Pi_t = c) =$

$$= \left[\max_{j=1 \dots K} \max_{\Pi_1, \dots, \Pi_{tE-1}} P(X_1, \dots, X_{t-1}, \Pi_1, \dots, \Pi_{tE-1} = j) \times \alpha_{ji} e_j x_t \right] \times e_c x_t =$$

$$= \left[\max_{j=1 \dots K} V_{t-1}(j) \times \alpha_{ji} \right] e_j x_t$$

Prob finale

$$P(x_1 \dots x_n, \pi_1^*, \dots, \pi_n^*) = \max_{\pi_1 \dots \pi_n} P(x_1 \dots x_n, \pi_1 \dots \pi_n)$$

$$= \max_{c=1 \dots k} \max_{\pi_1 \dots \pi_{n-1}} P(x_1 \dots x_n, \pi_1 \dots \pi_{n-1}, \pi_n = c) = \max_{c=1 \dots k} \pi_n(c)$$

Possiamo costruire un algoritmo che calcoli la prob finale in $\Theta(k^2 n)$

Nell'algoritmo è utile una struttura dati detta ph, che tiene traccia ad ogni t degli stati da cui provengono gli aggiornamenti ogni c .

$\text{ph}_t(c) \equiv$ stato migliore da cui partire al $t-1$ per arrivare in c in t

Tanto questa struttura possa contenere il percorso idoneo (quello che minimizza) la prob di arrivare in reg. di simboli.

Posterior decoding

Caso particolare di decoding

Data X si vuole stabilire a posteriori lo stato s più probabile in t

$$P(\pi_t = j | X) = \arg \max_{c=1 \dots k} P(\pi_t = c | X)$$

$$P(\pi_t = i | X) = P(\pi_t = i, X) / P(X)$$

~~$$P(\pi_t = i, X) = P(x_1 \dots x_t, \pi_1 \dots \pi_{t-1}, \pi_t = i, \pi_{t+1} \dots \pi_n)$$~~

$$= P(x_1 \dots x_t, \pi_1 \dots \pi_{t-1}, \pi_t = i, \pi_{t+1} \dots \pi_n, x_{t+1} \dots x_n) =$$

$$= P(x_1 \dots x_t, \pi_1 \dots \pi_{t-1}, \pi_t = i) P(\pi_{t+1} \dots \pi_n, x_{t+1} \dots x_n | \pi_t = i)$$

forward $f_t(i)$

backward $b_t(i)$

$$P(\pi_t = i \mid X) = f_t(i) \cdot b_t(i) / P(X)$$

Prob Backward

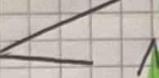
Sig. ricavare backword in induzione

$$\begin{aligned} b_t(i) &= P(\pi_{t+1}, \dots, \pi_m, x_{t+1}, \dots, x_n \mid \pi_t = i) = \\ &= \sum_{j=1} b_{t+1}(j) \times c_{jx_{t+1}} \times \alpha_{ij} \end{aligned}$$

Anche l'algoritmo per calcolare la backward è
confronto $\Theta(K^2 M)$

Learning

2 situazioni

 Graziano sapeva cosa era Bob
Non sapeva

- Trovare i λ che maximizzano $P(X \mid Z)$

$$Z = (c_{ij}, \alpha_{ij})$$



X, π reg stati nota

$m =$ simboli distinti $k =$ stati distinti in HMM

$A_{st} =$ # volte s segue t in π

$E_{sb} =$ # volte che s in π emette il simbolo b nella stessa posizione nella seq. πX

I valori sono quelli delle nostre tute che

$$\alpha_{st} = \frac{A_{st}}{\sum_{i=1}^k A_{si}}$$

$$c_{sb} = \frac{E_{sb}}{\sum_{j=1}^m E_{sj}}$$

Se abbiamo pochi dati i valori dei parametri trovati non saranno ottimali

- Q** X, Π 1a stati non nate
- ⑥ Iniziamo le valori fatti in Ase e Esb
Aggiorniamo via via i valori in base alla composizione
eppure ad ogni passo si fissa un adattamento
- Algoritmo: Expectation - Maximization EM

- 1) Stima di Ase e Esb usando i valori composti
Ase e Esb fatti matrici di transizione ed emisione
- 2) Aggiornano i parametri in base a Ase e Esb
- 3) Ripeti 1 e 2 fino a convergenza

Definiamo $S_t(i, j) =$

$$= P(\Pi_t = i, \Pi_{t+1} = j | X) =$$

$$= \frac{f(i) \times \alpha_{ij} \times c_j x_{t+1} \times b_{t+1}(j)}{\sum_{i=1}^K \sum_{j=1}^K p_t(i) \times \alpha_{ij} \times c_j x_{t+1} \times b_{t+1}(j)}$$

$\delta_t(i)$ Prob a posteriori di trovarsi nel tempo t
nello stato i $\delta_t(i) = P(\Pi_t = i | X) = \sum_{j=1}^K S_t(i, j)$

α_{ij} = Prob di passare da i a j nelle volte
Volte in cui do i passano a qualsiasi stato
 $\alpha_{ij} = \frac{\sum_{t=1}^{M-1} S_t(i, j)}{\sum_{t=1}^{M-1} \delta_t(i)}$

c_{ib} : Prob di entrare b nello stato i nello
Volte in cui sono in i (entro)

$$c_{ib} = \frac{\sum_{t=1, x_t=b}^M \delta_t(i)}{\sum_{t=1}^{M-1} \delta_t(i)}$$

Poniamo di avere l'algoritmo di Baum-Welch

• Inizializzazione:

• inizializza matrice di transizione ed emissione T_{ab}
(o in base a
urz conoscente
magena)

• Iterazione

• Calcola le prob forward α

• Calcola le prob backward β

• Aggiorna i param. Z calcolando le stime A_{ij}, E_{ib}

• Calcola $P(X|Z)$ ovvero la prob di ottenere la seq X dato i parametri Z

• Ripeti fino a convergenza ($P(X|Z)$ non cambia più
il valore in maniera significativa)

Complessità: $O(k^2 n)$

Non trova i migliori parametri (i 2ndi)

Potrebbe convergere a un minimo locale