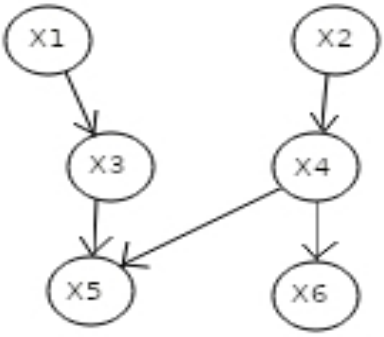
Work Project 3 – Reti di Bayes e Reti Neurali

Esercizio 1

1.a) L'idea delle reti bayesiane è il concetto di causalità tra le variabili: l'essere umano è portato naturalmente a registrare evidenze tra informazioni in un verso che segue un *rapporto causa-effetto*. Ad esempio, un meteorologo, in base ai dati che ha raccolto durante la sua attività lavorativa, è in grado di stabilire direttamente qual è la probabilità di pioggia dato un certo livello di umidità, ma in generale non è in grado di dire il contrario, cioè la probabilità che l'umidità sia maggiore di una certa soglia data la pioggia: rispondere a questa domanda gli richiederebbe un certo tipo di inferenza. Questo perché è difficile, se non impossibile, determinare una tabella delle probabilità congiunte, ed inoltre è una struttura che ha una crescita esponenziale con il numero di variabili. Mediante le reti bayesiane, si ottiene invece una rappresentazione lineare con il numero di variabili:



X1 e X2 sono delle evidenze, ovvero delle "cause indipendenti", degli eventi considerati puramente aleatori nell'ambito dell'astrazione che si sta facendo del dominio. Ad esempio, X1 potrebbe essere un terremoto, oppure un livello di umidità misurato. Poi, X3 è effetto di X1, X4 è effetto di X2, X5 è effetto di X3 ed X4, X6 è effetto di X4. Vale la proprietà che ogni nodo è dipendente solo dai suoi diretti ascendenti: questo implica, ad esempio, che X5 ed X6 sono indipendenti, nonostante il fatto che X4 sia causa di entrambi.

Dall'introduzione fatta, si capisce che un essere umano è in grado di costruire una rete del genere, perché tipicamente, mediante delle statistiche, possiede tutte le informazioni necessarie per assegnare i valori delle distribuzioni di probabilità ai nodi:
P(X1), P(X2), P(X3|X1), P(X4|X2), P(X5|X3,X4), P(X6|X4)
In generale: P(Xi|Ascendenti(Xi))
A questo punto vorremmo porci delle domande, ad esempio:
P(X1=true,X3=true,X5=false), che si indica anche:
P(x1,x3,¬x5)
In generale, rispondere ad una domanda di questo tipo richiederebbe la marginalizzazione di un insieme di righe della tabella delle probabilità congiunte. In effetti, dalla rete di Bayes è possibile ricostruire le righe che servono ed effettuare dunque un'inferenza per enumerazione, sfruttando la regola del prodotto ed implementando un algoritmo ricorsivo, *depth first*: si ottiene un albero di ricerca binaria (perché si considerano i casi true e false delle variabili non coinvolte nella domanda) di profondità generica n, quindi la complessità temporale è 2^n.
Un'inferenza più efficiente è quella per *simulazione stocastica*: l'idea è basata sull'approccio frequentista alla probabilità. Piuttosto che sommare lungo tutto lo spazio degli eventi, si fa "cadere una moneta" che serve solo negli eventi relativi all'inferenza che si sta considerando, ed in base alle probabilità si simula l'accadere degli eventi, nel senso che per un certo evento si ottiene un campione true con una probabilità che dipende dall'evento e che si ottiene dalla rete di Bayes. Dunque, tornando alla domanda di esempio posta prima, un possibile campione è: <X1=false, X3=true, X5=true>. A questo punto, iterando il procedimento N volte, e contando il numero di volte

che si ottiene l'evento di interesse (inteso come vettore) in rapporto al numero N di "lanci", si ottiene una stima della probabilità:

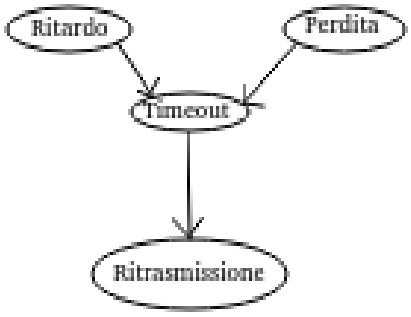
$P_{stimata}(x1, \dots, xn) = N_{PS}(x1, \dots, xn) = S_{PS}(x1, \dots, xn) / N$

Si ha che:

$$\lim_{N \to \infty} N_{PS}(x1, \dots, xn) = P(x1, \dots, xn)$$

Ovvero, aumentando il numero di "lanci", la stima della probabilità tende al valore esatto della probabilità.

1.b) Supponiamo di modellare con una rete di Bayes i servizi offerti dal livello trasporto in rete, riguardo alla ritrasmissione dei pacchetti in seguito a timeout; gli eventi che possono scatenare un timeout sono il ritardo in rete e la perdita di pacchetti, nell'ipotesi in cui l'altro terminale risponda sempre e che quindi non si abbiano timeout dovuti a "silenzi volontari". Dunque, le cause di un timeout sono: ritardo, perdita di pacchetti; la causa di una ritrasmissione è il timeout. Bisogna tenere in conto il fatto che la connessione potrebbe "cadere" e quindi la ritrasmissione in seguito a timeout non è un evento certo, ma segue una logica secondo la quale dopo un certo numero di timeout consecutivi, si può considerare la connessione caduta e non ritrasmettere.



Indichiamo Ritardo=Delay=D, Perdita=Loss=L, Timeout=T, Ritrasmissione=R. Assegniamo le seguenti probabilità:
P(D) = 0.4
P(L) = 0.05

D	L	P(T|D,L)
T	T	1
T	F	0.8
F	T	1
F	F	0.01

Notare che in caso di perdita di pacchetti, il timeout è un evento certo. Per quanto riguarda la ritrasmissione, supponiamo che quattro timeout consecutivi siano l'allarme di connessione caduta: in tal caso il terminale ritrasmetterà tre volte su quattro. E' chiaro che non sempre capita che ci siano molti timeout consecutivi, però per semplicità assegniamo la probabilità condizionata secondo questo criterio:
P(R|T) = ¾ = 0.75

Ora ci chiediamo:
P(r, t, d, ⌐l) = ?

P(r, t, d, ⌐l) = P(t, d, ⌐l) P(r | t, d, ⌐l) =
= P(d, ⌐l) P(t | d, ⌐l) P(r | t, d, ⌐l) =
= P(d) P(⌐l) P(t | d, ⌐l) P(r | t)

dove: P(d, ⌐l) = P(d) P(⌐l) perché d ed l sono indipendenti,
P(r | t, d, ⌐l) = P(r | t) per l'indipendenza condizionale garantita dalla rete di Bayes, o dalla *Markov blanket*, se vogliamo; negli altri passaggi si è usata la regola del prodotto.

Dunque:
P(r, t, d, ⌐l) = 0.4 * 0.95 * 0.8 * 0.75 = 0.228

Abbiamo mostrato come ottenere una riga della tabella delle probabilità congiunte; per fare un'inferenza su un numero inferiore di variabili, è sufficiente calcolare più righe e fare una marginalizzazione. Vediamo un caso più raffinato:
P(L | r) = ?

Questa volta si vuole in uscita una distribuzione di probabilità, non un valore scalare.
P(L | r) = α P(L, r)

Dato che sappiamo che:
P(R,T,D,L) = P(D) P(L) P(T | D, L) P(R | T)

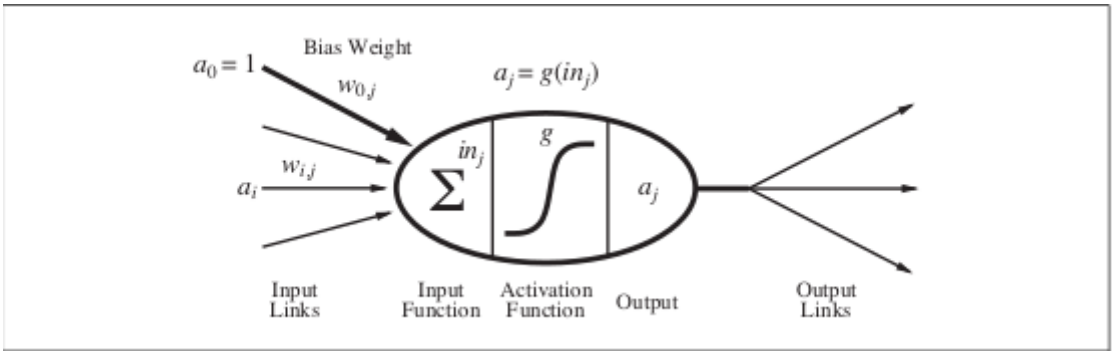
Possiamo scrivere:
$$P(R,T,L)=\sum_{d\in D} P(R,T,d,L)P(d)$$

E poi:
$$P(L,R)=\sum_{t\in T} P(L,R,t)P(t)$$

Notare che P(T) non è nota quindi neanche P(t) ovviamente, e qui si "apprezza" il fatto che l'inferenza per enumerazione è pesante, perché fare una marginalizzazione richiede altre marginalizzazioni e spesso si ripetono calcoli ridondanti. Comunque, continuando di questo passo, è possibile arrivare alla distribuzione di probabilità d'interesse. La simulazione stocastica è un approccio più efficiente su un calcolatore. Un approccio più *smart* al quale ricorrere "a mano" consisterebbe nell'applicare la legge di Bayes, ma nel caso specifico ci si ritroverebbe comunque P(R | L) non nota.

Esercizio 2

2.a) Un modello diffuso di neurone è quello che prevede il calcolo di una somma pesata di ingressi, per poi usare il risultato come input di una *activation function* che può essere una funzione a gradino oppure una funzione logistica, ed infine in generale ha N archi uscenti in cui l'uscita della activation function è replicata:



Il fatto di avere $a_0 = 1$ lo si può interpretare vedendo a_0 come un ingresso fittizio relativo all'intercetta dell'asse delle ordinate, se si pensa ad un *linear model fitting*, ed in tal caso il peso $w_{0,j}$ è proprio l'intercetta. Si ha, in uscita:

$$a_j = g(\sum_{i=0}^{n} \text{in}_j) = g(\sum_{i=0}^{n} w_{i,j} a_i)$$

dove $w_{i,j}$ è il peso associato all'ingresso a_i del neurone j, g è l'activation function, a_j è l'uscita del neurone. Un neurone da solo può realizzare delle funzioni lineari, ma chiaramente quello che è interessante è capire come collegare tra loro più neuroni per realizzare complesse funzioni non lineari. Distinguiamo architetture *feed-forward* e *recurrent*, e poi *single-layer* e *multi-layer*. In un'architettura feed-forward gli archi sono orientati tutti verso la stessa direzione, cioè verso l'uscita, quindi non ci sono cicli. In un'architettura recurrent, per negazione, ci sono cicli e quindi si implementa un concetto di stato. In un'architettura single-layer, le uscite dei neuroni sono le uscite della rete neurale, quindi con tale architettura è possibile implementare solo funzioni lineari. In un'architettura multi-layer, le uscite dei neuroni possono essere poste in ingresso ad altri neuroni, e così via, realizzando a tutti gli effetti degli "strati". Con architetture di questo tipo, è possibile realizzare complesse funzioni non lineari, ad esempio per il riconoscimento di pattern in segnali multimediali. Per implementare funzioni di questo tipo in una rete MLP (Multi-Layer *Perceptron*), bisogna, a valle della scelta del numero di livelli e del numero di neuroni per livello, assegnare i pesi agli archi. Quando si parla di reti neurali, si parla di *supervised learning*, e quindi si deve avere un meccanismo che permette di assegnare dei pesi agli archi dato un insieme di esempi classificati. Questo meccanismo si chiama *error back propagation*: l'idea è di aggiustare iterativamente i pesi sugli archi finché l'errore non scende al di sotto di una certa soglia; l'errore si può calcolare perché si conosce il risultato che ci si aspetta da ogni esempio. Formalmente:
Per il nodo di uscita generico:

$$Err_i = y_i - a_i$$

$$\Delta_i = Err_i \cdot g'(\text{in}_i)$$

dove con g' si intende la derivata della activation function.
Aggiustamento pesi:

$$W_{j,i} = W_{j,i} + \alpha \cdot a_j \cdot \Delta_i$$

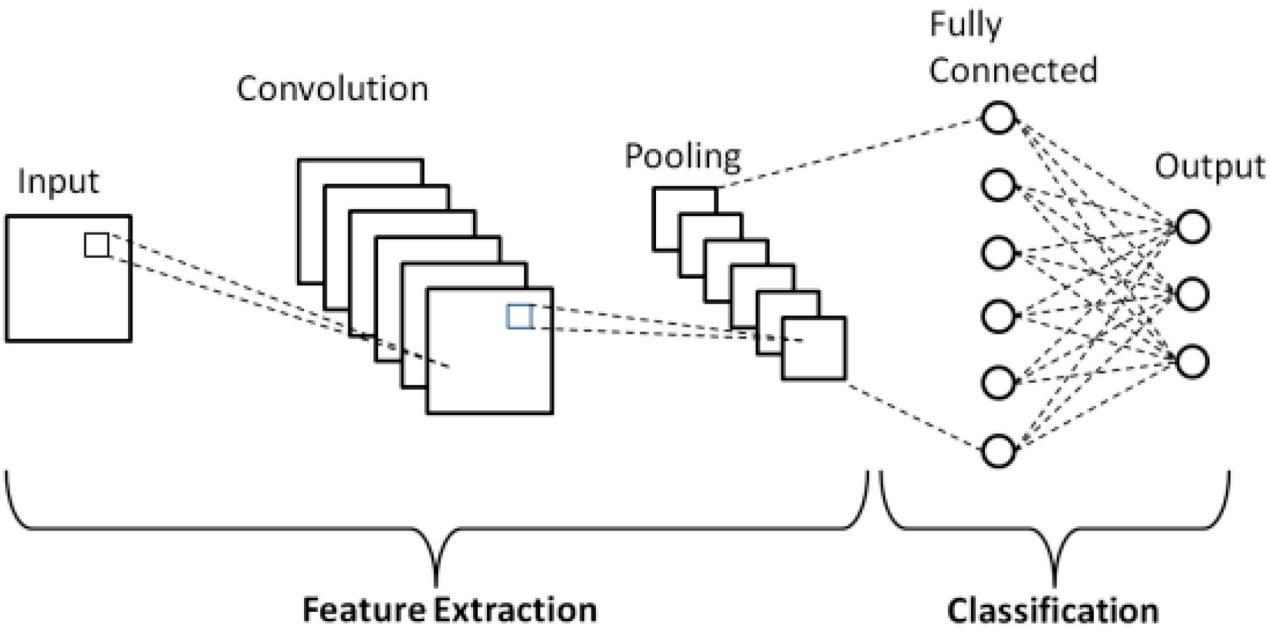
Back propagation:

$$\Delta_j = (\sum_i W_{j,i} \Delta_i) \cdot g'(\text{in}_j)$$

e così via per altri aggiustamenti dei pesi ed ulteriori propagazioni all'indietro. α è detto coefficiente di apprendimento ed indica con "quanta velocità" si aggiustano i pesi. Notare che ogni neurone che

non è di uscita calcola il proprio Δ come somma pesata dei Δ dei neuroni del layer successivo, ed i pesi sono proprio gli archi uscenti. Con una opportuna scelta di α e con un certo numero di iterazioni, si riesce a convergere alla funzione target; ogni iterazione è detta epoca.

2.b) Affrontiamo questo problema partendo da un approccio ingenuo ed elencandone i problemi. Un approccio ingenuo può essere fornire in ingresso alla rete neurale direttamente le immagini classificate, utilizzando nel primo layer un neurone per pixel: il primo problema con cui ci scontreremmo sarebbe quello delle diverse dimensioni delle immagini; inoltre, per immagini a colori, ci vorrebbero tre canali. Supponendo di risolvere questi due problemi, non avremmo, comunque, risolto tutti i problemi: gli alberi possono differire per illuminazione, per dimensione, forma, la quantità ed il colore delle foglie varia con le stagioni, possono essere ripresi da diversi angoli, ad esempio dal basso o dall'alto, ci sono delle componenti di rumore associate allo specifico sensore che ha scattato la foto, e così via. Quindi, è facile convincersi che, a meno di non avere un *training set* che tenga in conto di tutti i possibili "rumori", la rete neurale produrrà sempre risultati che mettono in evidenza l'*overfitting* con il training set, perché esso difficilmente sarà sufficientemente generale. Inoltre, in realtà, avendo scelto un neurone per pixel nel primo layer, abbiamo sbagliato fin dal principio, perché la rete neurale dovrebbe essere allenata a riconoscere alberi indipendentemente dalla posizione degli stessi all'interno delle immagini. Dunque, per quanto detto, questo problema richiede un *pre-processing* delle immagini, ovvero bisogna sfruttare la *conoscenza del dominio*. Bisogna fare delle operazioni di: denoising, segmentazione, thresholding in modo da ottenere nell'immagine delle strutture sottili, dopodiché un'opportuna combinazione di operazioni morfologiche, scelte conoscendo la struttura geometrica che dovrebbe presentare un albero, e di un filtraggio locale con degli LBP (*Local Binary Pattern*), permette di ottenere in uscita un vettore di *features*. L'estrazione di features è un tipo di "riduzione delle dimensioni" tale da rappresentare in modo efficiente parti interessanti di un'immagine nella forma di un compatto vettore di features. Il vettore in uscita può quindi variare abbastanza a seconda del contenuto dell'immagine, ma una buona scelta delle features da parte dell'esperto di dominio può determinare l'esistenza di una funzione non lineare, presente in zone abbastanza definite in un piano ad N dimensioni (dove N è il numero di features) , tale da determinare la classificazione. Il fatto che la funzione si trovi in zone abbastanza definite del piano è importante: se così non fosse, il vettore di features non sarebbe quello giusto, e si rischierebbe un overfitting, ancora una volta. Dunque, a valle della scelta di un buon vettore di features, quello che ci resta da fare è far apprendere alla rete neurale, che supponiamo di scegliere feed-forward e multi-layer, la funzione che lega il vettore di features alla classificazione. Per fare ciò, si prende il training set con le immagini classificate, si calcola il vettore di features per ogni immagine e si dà "in pasto" all'algoritmo di error back propagation l'insieme di esempi, in cui ciascun esempio ha come input un'istanza del vettore di features e come output uno scalare, che è la classificazione. Uno potrebbe chiedersi: ma perché non far apprendere alla rete neurale anche l'insieme, o una rilevante parte, delle operazioni di pre-processing, in modo da implementare la feature extraction con il deep learning? Esiste un'architettura di rete neurale in grado di lavorare direttamente sulle immagini? La risposta è sì: modificando il pattern di connettività tra i neuroni in modo da ispirarsi alla corteccia visiva animale, si ottiene un'architettura di rete neurale detta *rete neurale convoluzionale*, la quale si progetta appunto in modo da minimizzare la quantità di pre-processing.
Una rappresentazione semplificata:


Fully
Connected
Convolution
Pooling
Input
Output
Feature Extraction
Classification