DISPENSA DI INTELLIGENZA ARTIFICIALE

ANDREA CASINI

Appunti del corso di Intelligenza Artificiale

Laurea Specialistica in Informatica
Università Ca' Foscari di Venezia

8 luglio 2014 – versione 0.9

INDICE

ELENCO DELLE FIGURE

Parte I

RETI NEURALI

1

MODELLO NEURALE

Le reti neurali artificiali sono nate per riprodurre attività tipiche del cervello umano come la percezione di immagini, il riconoscimento di forme, la comprensione del linguaggio, il coordinamento senso-motorio, ecc. A tale scopo si sono studiate le caratteristiche del cervello umano.

Nel sistema nervoso esistono miliardi di neuroni (cellule nervose). Nel modello biologico un **neurone** è caratterizzato da:

- **corpo cellulare (soma)**: l'unità di calcolo del neurone (5/10 micron);
- **assone**: meccanismo di output di un neurone;
- **dendriti**: ricevono segnali in input da altri assoni tramite le **sinapsi**

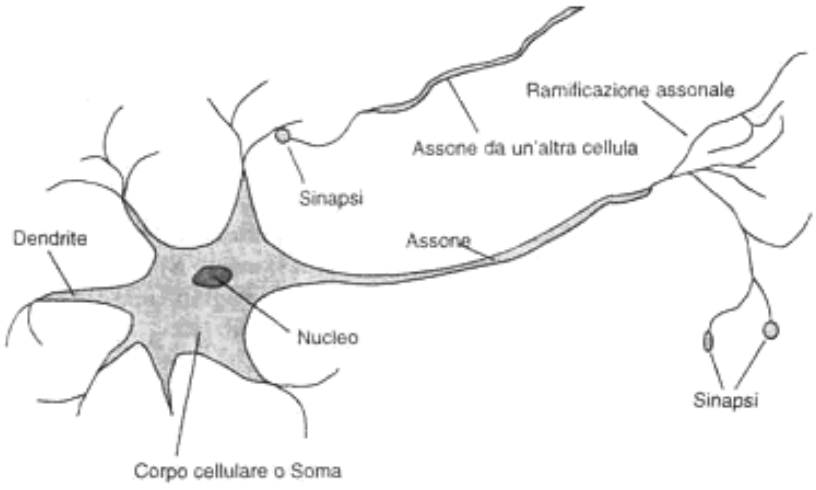


Figura 1: Neuroni biologici

All'estremità l'assone si ramifica formando terminali attraverso i quali i segnali elettrici vengono trasmessi ad altre cellule (ad esempio ai dendriti di altri neuroni). Tra un terminale di un assone e la cellula ricevente esiste uno spazio. I segnali superano questo spazio per mezzo di sostanze chimiche dette *neurotrasmettitori*. Il punto di connessione tra terminale e dendrite è detto **sinapsi**.

La trasmissione di un segnale nella corteccia cerebrale è un processo complesso. In maniera molto semplificata, tale processo si compone delle seguenti fasi:

1. Il corpo cellulare esegue una somma pesata dei segnali in ingresso;
2. Se il risultato supera un certo valore soglia allora si produce un **potenziale d'azione**; una scarica di impulsi elettrici, che viene inviata all'assone. In questo caso, si dice che la cellula "spari" altrimenti non fa nulla;
3. Quando il segnale elettrico raggiunge la sinapsi, viene rilasciato chimicamente un "neuro trasmettitore" che sarà combinato con i "recettori" nella membrana post-sinaptica (vedi Figura 2);
4. I recettori post-sinaptici provocano una diffusione del segnale elettrico nel neurone post-sinaptico.

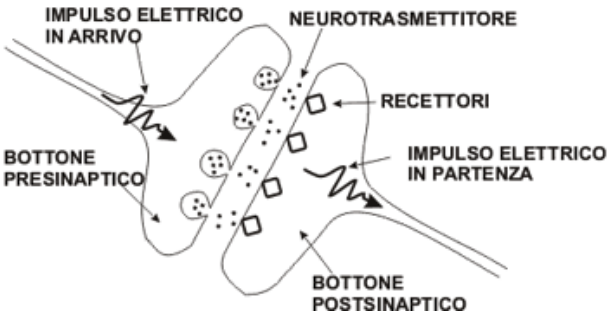


Figura 2: La sinapsi

L'apprendimento si attua modificando la cosiddetta **efficacia sinaptica**, ovvero l'ammontare di corrente che entra nel neurone post-sinaptico rispetto al potenziale d'azione del neurone pre-sinaptico. In questo contesto le sinapsi possono essere: **eccitatorie**, nel senso che favoriscono la generazione di potenziale d'azione nel neurone post-sinaptico, oppure **inibitorie** che, invece, ne limitano la generazione. Il cervello umano è un calcolatore complesso, non lineare e parallelo. Pur essendo costituito da elementi di elaborazione molto semplici (i neuroni), è in grado di eseguire computazioni complesse, come il riconoscimento, la percezione e il controllo del movimento, molte volte più velocemente del più veloce degli attuali calcolatori.

Nel cervello non esiste un controllo centralizzato, nel senso che le varie zone del cervello funzionano insieme, influenzandosi reciprocamente e contribuendo alla realizzazione di uno specifico compito.

Infine, il cervello è *fault tolerant*, cioè se un neurone o una delle sue connessioni sono danneggiati, il cervello continua a funzionare, anche se con prestazioni leggermente degradate. In particolare, le prestazioni del processo cerebrale degradano gradualmente man mano che si distruggono sempre più neuroni (*graceful degradation*). A tale scopo si sono studiate le caratteristiche del cervello umano. Quindi, per riprodurre artificialmente il cervello umano occorre realizzare una rete di elementi molto semplici che sia una struttura **distribuita**, massicciamente **parallela**, capace di **apprendere** e quindi di **generalizzare** (cioè produrre uscite in corrispondenza di ingressi non incontrati durante l'addestramento).

Il metodo più usato per addestrare una rete neurale consiste nel presentare in ingresso alla rete un insieme di esempi (training set). La risposta fornita dalla rete per ogni esempio viene confrontata con la risposta desiderata, si valuta la differenza (errore) fra le due e, in base a tale differenza, si aggiustano i pesi. Questo processo viene ripetuto sull'intero training set finché le uscite della rete producono un errore al di sotto di una soglia prestabilita.

Anche se di recente introduzione, le reti neurali trovano valida applicazione in settori quali predizione, classificazione, riconoscimento e controllo, portando spesso contributi significativi alla soluzione di problemi difficilmente trattabili con metodologie classiche.

1.1 MODELLO DI MCCULLOCH E PITTS

Si introduce ora un modello artificiale del neurone biologico. Il neurone artificiale ha molti ingressi ed una sola uscita. Ogni ingresso ha associato un peso, che determina la conducibilità del canale di ingresso. L'attivazione del neurone è una funzione della somma pesata degli ingressi.

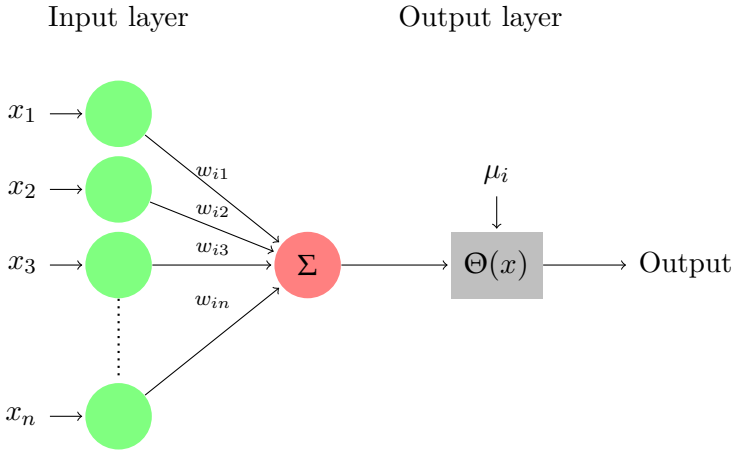


Figura 3: Neurone artificiale nel modello M&P

Un insieme di neuroni artificiali forma una *rete artificiale*. Una rete neurale è un grafo diretto pesato $N = (V, E, w)$ dove $V = \{0, 1, \ldots, n\}$ è l'insieme delle unità (o neuroni), $E \subseteq V \times V$ è l'insieme di connessioni e $w : E \to \mathbb{R}$ è una funzione che assegna un peso con valore reale $w(i, j)$ per ogni connessione $(i, j) \in E$.

I pesi rappresentano l'efficacia sinaptica: pesi positivi amplificano il segnale, mentre quelli negativi lo inibiscono. Come notazione sarà utilizzata w_{ij} anziché $w(i, j)$.

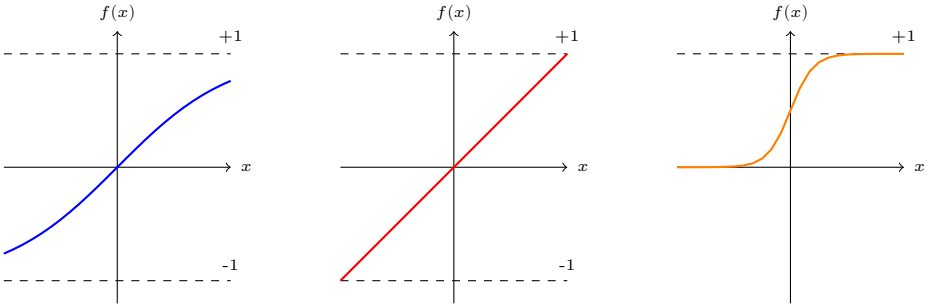
Un neurone si attiva se la somma pesata $\sum_j w_{ij} n_j$ degli input supera un certo valore soglia μ_i. In termini matematici:

$$n_i(t+1) = \Theta\left(\sum_j w_{ij} n_j(t) - \mu_i\right) \tag{1}$$

Si tratta di un **modello semplificato** dove $\Theta(x)$ è la funzione di transizione o di attivazione ed in questo caso è discreta.

$$n_i(t+1) = \begin{cases} 1 & \text{if } \sum_j w_{ij} n_j \geq \mu_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Anziché utilizzare una funzione discreta si possono adottare funzioni continue in modo tale da rendere il sistema più realistico. In questa caso l'output corrisponde alla frequenza delle volte in cui un neurone tramette. Esistono diverse funzioni di transizione continue; le più usate sono la sigmoidea e la tangente iperbolica.



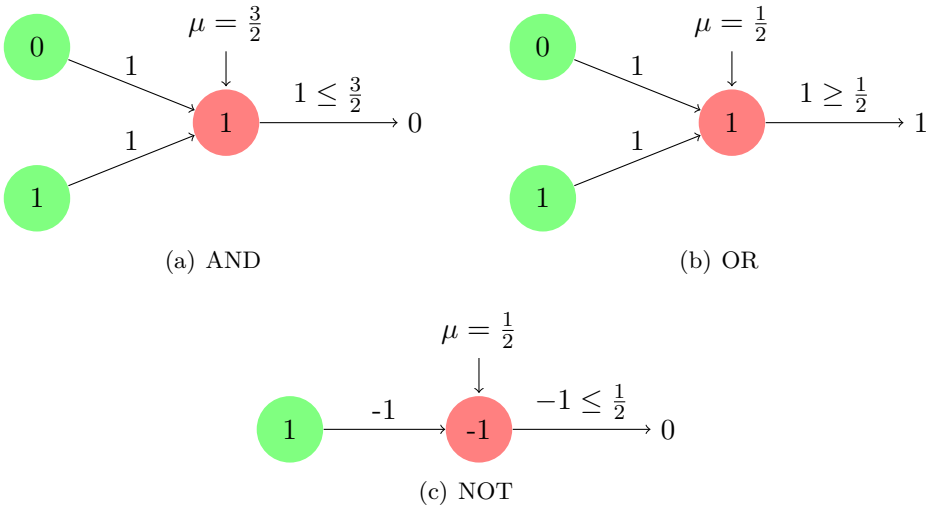
(a) Tangente iperbolica (b) Lineare (c) Sigmoidea

Figura 4: Le funzioni di attivazione più utilizzate.

Lo scopo delle funzioni di attivazione è ridurre la varianza in output di un neurone mappando il risultato della somma pesata entro un intervallo; generalmente compreso tra $[-1, 1]$ oppure $[0, 1]$.

1.1.1 *Proprietà del modello M&P*

Combinando opportunamente i neuroni di M&P si possono costruire reti in grado di realizzare qualsiasi operazione del calcolo proposizionale.



(a) AND (b) OR

(c) NOT

Figura 5: Operazioni logiche elementari nel modello M&P.

È importante notare che in questo modello **non** avviene ancora nessun tipo apprendimento. I pesi, infatti, rimangono fissi; l'apprendimento si realizza attraverso la variazione/aggiornamento dei pesi sinaptici.

1.2 TIPI DI ARCHITETTURE DELLA RETE

Il modo con cui è strutturata la rete dipende dall'algoritmo di apprendimento che si ha intenzione di usare. In generale si identificano tre classi di reti:

- **reti feedforward ad uno strato**: In questa forma semplice di rete a strati, abbiamo i nodi di input (input layer) e uno strato di neuroni (output layer). Il segnale nella rete si propaga in avanti in modo aciclico, partendo dal layer di input e terminando in quello di output;

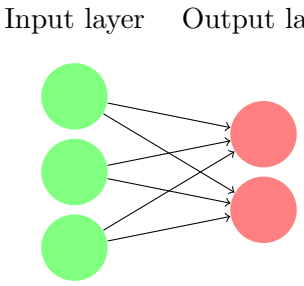


Figura 6: Rete forward ad uno strato

- **reti feedforward a più strati:** questa classe di reti feedforward si distingue dalla precedente dal fatto che tra lo strato di input e quello di output abbiamo uno o più strati di neuroni nascosti (**hidden layers**). Questo tipo di architettura fornisce alla rete una prospettiva globale in quanto aumentano le interazioni tra neuroni;

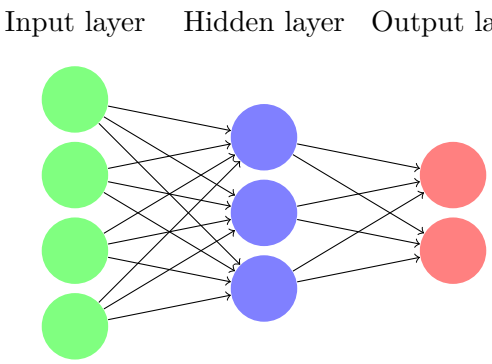


Figura 7: Rete forward a più strati

- **reti feedback o ricorrenti**: una rete ricorrente si distingue dalle precedenti nel fatto che è ciclica. La presenza di cicli ha un impatto profondo sulle capacità di apprendimento della rete e sulle sue performance, in particolare rendono il sistema **dinamico**.

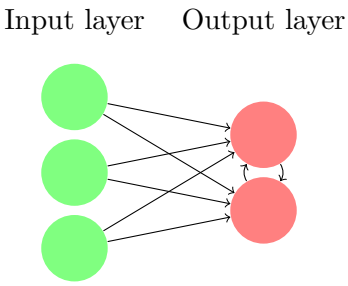


Figura 8: Rete feedback.

2

PROBLEMA DI CLASSIFICAZIONE

In un problema tipico di classificazione abbiamo:

- alcune **caratteristiche** $f_1, f_2, \ldots, f_n$;
- alcune **classi** $c_1, c_2, \ldots, c_m$.

Il problema consiste nel classificare un oggetto in base alla caratteristiche. Un oggetto può essere rappresentato in uno spazio n-dimensionale e pertanto è possibile trattare problemi di classificazione attraverso metodi geometrici.

Questi oggetti sono comunemente chiamati **pattern**. Una rete neurale riconosce pattern a seguito di una *fase di addestramento*, nella quale alla rete vengono presentati ripetutamente un insieme di pattern di addestramento con specificato per ognuno la categoria a cui appartengono. Quando sarà presentato un pattern mai visto prima ma appartenente ad una stessa classe di pattern che la rete ha appreso, essa sarà in grado di classificarlo grazie alle informazioni estratte dai dati di addestramento.

Supponendo di operare nel contesto bidimensionale ogni pattern è rappresentato da un punto. Questo spazio è suddiviso in **regioni di decisione**, ognuna delle quali associata ad una classe. I confini di queste regioni sono determinate dalla rete attraverso il processo di addestramento.

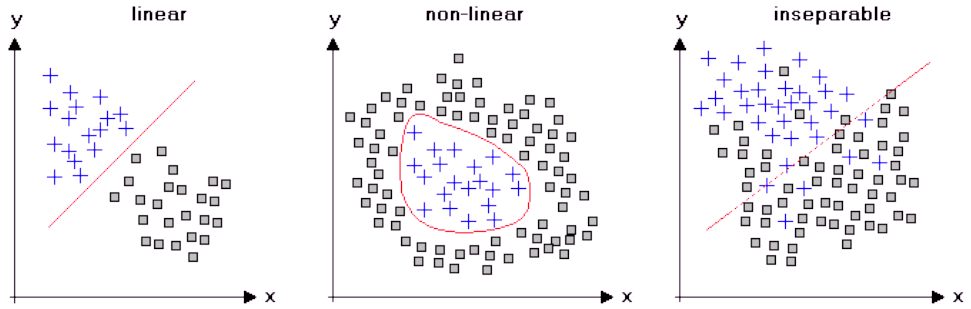


Figura 9: Rappresentazione geometrica di alcuni problemi di classificazione.

2.1 PERCETTRONE A SINGOLO STRATO

Una rete neurale può essere utilizzata per risolvere un problema di classificazione. La prima idea fu di utilizzare una rete neurale ad uno strato con connessione feedforward: il **percettrone** (Rosenblatt 1958).

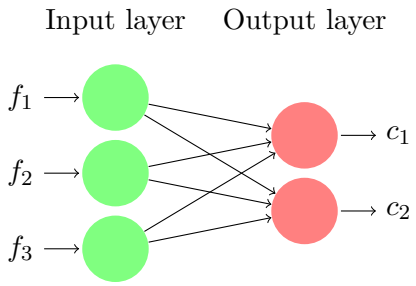


Figura 10: Rete forward ad uno strato: le tre unità in input rappresentano tre caratteristiche, mentre quelle output rappresentatano le due classi che la rete dovrà imparare o predire.

È possibile sbarazzarsi dei valori soglia associati ai neuroni aggiungendo una nuova unità extra permanentemente bloccata a -1. In questo modo i valori soglia diventano pesi e possono essere opportunamente aggiustati durante l'apprendimento.

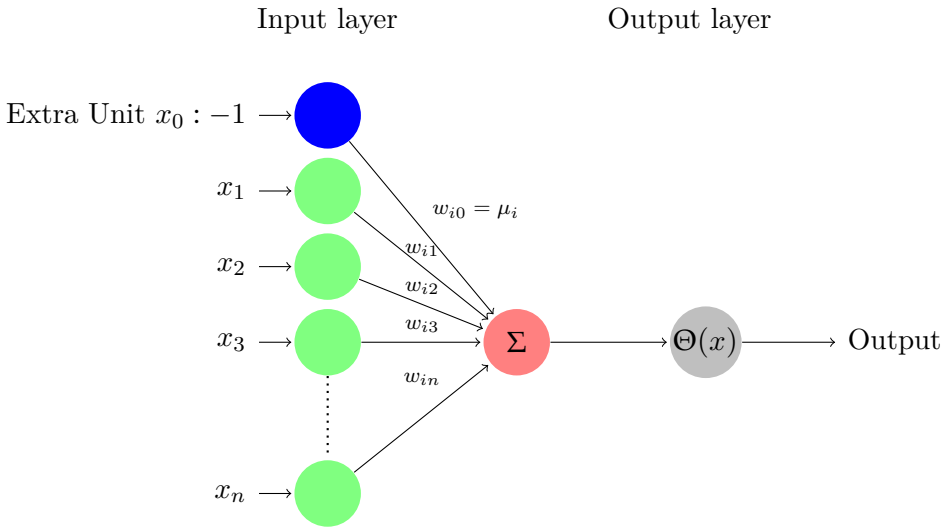


Figura 11: Valori soglia come pesi.

Il percettrone è un idea semplice ed elegante: imita il neurone umano ed è in grado di imparare da esempi che gli vengono presentati. Tuttavia soffre di alcune limitazioni. Il percettrone è, infatti, in grado di risolvere solo problemi *linearmente separabili*, ovvero problemi in cui le regioni di decisione si possono separare con un iperpiano.

Ad esempio, non è possibile implementare l'operazione XOR attraverso un percettrone in quanto non linearmente separabile. Infatti, come si vede nel seguente schema sono necessarie due rette per poter separare le due classi.

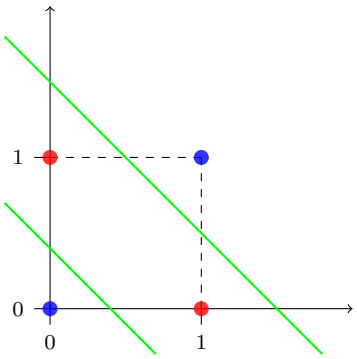


Figura 12: Non separabilità lineare dell'operatore XOR

La convergenza del percettrone è garantita dal seguente teorema:

Teorema 1 (Teorema di Convergenza del Percettrone)**.** *Se il problema di classificazione è linearmente separabile allora la fase di apprendimento converge (termina) ad un'appropriata impostazione dei pesi in un numero finito di passi.*

Nella pratica non è dato sapere se un problema è o meno linearmente separabile. La convergenza si può comunque ottenere aggiustando i parametri del percettrone come ad esempio il numero di iterazioni o il fattore di apprendimento. In questo caso, tuttavia, la convergenza è artificiale.

2.2 PERCETTRONE A PIÙ STRATI

Come si è visto nei paragrafi precedenti il percettrone a singolo strato è limitato ai problemi linearmente separabili. Aggiungendo strati nascosti è possibile, tuttavia, rappresentare tutte le operazioni booleane incluso lo XOR. I percettroni multi-strato sono noti da molto tempo, ma gli algoritmi per l'apprendimento sono stati ideati solo recentemente (1986).

La rete di un percettrone multi-strato consiste in un insieme di ingressi (input layer), uno o più strati nascosti di neuroni (hidden layers) e un insieme di neuroni di uscita (output layer). Il segnale di input si propaga attraverso la rete in avanti da layer a layer. Una rete di questo tipo ha tre caratteristiche distintive:

- ogni neurone include una **funzione di attivazione non lineare differenziabile** (ad es: funzione sigmoidale);
- la rete contiene uno o più strati nascosti (**hidden layers**) che non fanno parte né dell'input né dell'output della rete;
- la rete ha un **alta connettività**.

In questa rete troviamo due tipi di segnali:

1. **segnali di funzione:** un segnale di funzione è un segnale di input o stimolo che entra nel layer di input, si propaga in avanti attraverso gli strati nascosti per emergere al layer di uscita come segnale di output;
2. **segnale di errore:** un segnale di errore ha origine nel layer di uscita e si propaga all'indietro attraverso la rete.

Ogni neurone nascosto o d'uscita di un percettrone multistrato esegue due computazioni:

1. la **computazione del segnale di funzione** espressa come funzione non lineare continua di un segnale di input e dei pesi sinaptici associati al neurone;
2. la **computazione di un vettore gradiente** necessario per aggiornamento dei pesi sinaptici.

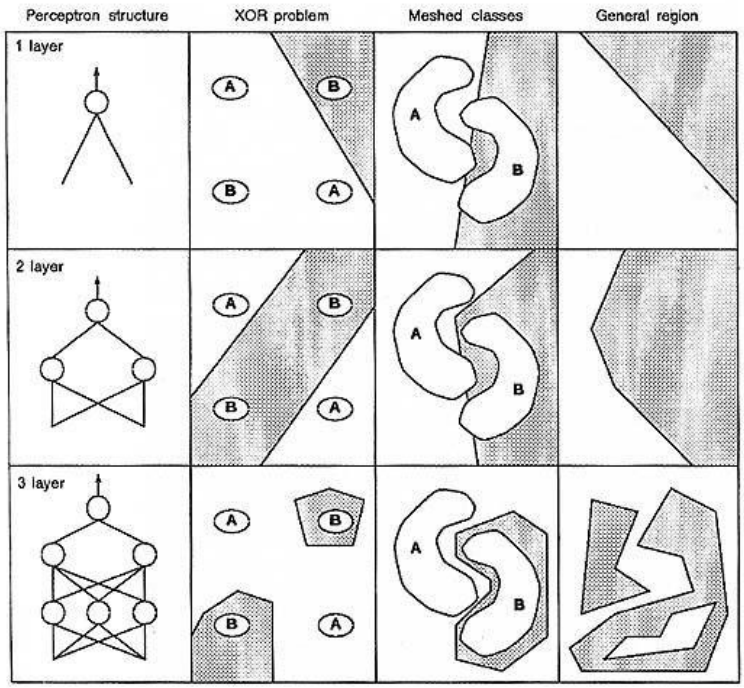


Figura 13: Confronto tra diverse architetture di rete. Si osservi che ad ogni nuovo strato intermedio della rete, si ha una migliore classificazione delle forme che identificano le regioni delle due classi.

2.3 PREVISIONE DI SERIE TEMPORALI

Oltre ai problemi di classificazione, una rete neurale è in grado di affrontare anche altri tipi di problemi come ad esempio la previsione di una serie temporale **(time series prediction)**. Si tratta di un problema *continuo* a differenza della classificazione che è, invece, *discreto*.

Si consideri una sequenza P di vettori o scalari che dipendono dal tempo t.

$$P = \{x(t_0), x(t_1), \cdots, x(t_i - 1), x(t_i), x(t_i + 1), \cdots\}$$

t è un valore reale e $x(t)$ è un segnale continuo.

Per ottenere una serie $\{x[t]\}$ da un segnale continuo è necessario *campionare* dal segnale in punti discreti.

Partendo da un tempo t e campionando a ritroso, otteniamo la serie temporale $\{x[t], x[t-1], \cdots\}$. Lo scopo dell'analisi è di stimare x a un tempo futuro.

$$\hat{x}[t+s] = f(x[t], x[t-1], \cdots)$$

Dove s è noto come *orizzonte della previsione*. Si tratta di un problema di approssimazione di una funzione e può essere risolto utilizzando una rete neurale a due strati. Il risultato è garantito dal seguente teorema:

Teorema 2 (Teorema dell'approssimazione universale)**.** *Una rete neurale multistrato feed-forward con un singolo strato nascosto, un percettrone, che contiene un numero finito di neuroni nascosti è un approssimatore universale di funzioni continue su sottoinsiemi compatti* $\mathbb{R}^n$.

3

ALGORITMO DI BACK-PROPAGATION

Il problema incontrato nell'addestramento delle reti multistrato è il seguente: volendo adottare un meccanismo di aggiornamento dei pesi simile a quello nelle reti a singolo strato (in cui l'errore è calcolato come differenza tra l'uscita desiderata e l'uscita effettiva di ciascun neurone) si riesce ad aggiornare solo i pesi relativi ai neuroni di uscita, ma non quelli relativi ai neuroni degli strati nascosti. Infatti, mentre per lo strato di uscita si conosce l'uscita desiderata (tale uscita viene data come secondo elemento delle coppie che costituiscono gli esempi del training set), niente si sa dell'uscita desiderata dai neuroni nascosti.

Questo problema è stato risolto dopo molti anni di disinteresse per le reti neurali (in quanto non si riusciva ad addestrarle) solo nel 1986, quando fu introdotto l'algoritmo di backpropagation. Tale algoritmo prevede di calcolare l'errore commesso da un neurone dell'ultimo strato nascosto propagando all'indietro l'errore calcolato sui neuroni di uscita collegati a tale neurone. Lo stesso procedimento è poi ripetuto per tutti i neuroni del penultimo strato nascosto, e così via.

L'algoritmo di backpropagation prevede che, per ogni esempio del training set, i segnali viaggino dall'ingresso verso l'uscita al fine di calcolare la risposta della rete. Dopo di che c'è una seconda fase durante la quale i segnali di errore vengono propagati all'indietro, sulle stesse connessioni su cui nella prima fase hanno viaggiato gli ingressi, ma in senso contrario, dall'uscita verso l'ingresso. Durante questa seconda fase vengono modificati i pesi.

I pesi sono inizializzati con valori casuali. Come funzione di uscita non lineare dei neuroni della rete si adotta in genere la funzione sigmoidea (l'algoritmo richiede che la funzione sia derivabile). L'algoritmo è basato sul metodo della **discesa del gradiente**.

3.1 METODO DI DISCESA DEL GRADIENTE

Il gradiente di una funzione f, denotato con ∇f è il vettore che ha come componenti le derivate parziali della funzione.

$$\nabla f(\bar{x}) = \left(\frac{\partial f(\bar{x})}{\partial x_1}, \frac{\partial f(\bar{x})}{\partial x_2}, \ldots, \frac{\partial f(\bar{x})}{\partial x_n} \right) \tag{3}$$

La discesa del gradiente è una tecnica di ottimizzazione di tipo locale. Consiste nel valutare, inizialmente in un punto scelto a caso nello spazio multidimensionale (primo punto), sia la funzione stessa sia il suo gradiente.

Il gradiente indica la direzione in cui la funzione tende a un minimo/massimo, ovvero dove il gradiente tende a 0. Se si è interessati alla ricerca di un **minimo locale** si considerano i passi proporzionali al *negativo* del gradiente della funzione nel punto corrente. Se, invece, si prendono i passi proporzionali al *positivo* del gradiente si raggiungerà un **massimo locale**; la procedura in questo caso è nota come *ascesa del gradiente*.

A titolo di esempio, il gradiente di $f(x, y) = x^2 + xy$ è:

$$\nabla f(x, y) = \left\{ \frac{df}{dx}, \frac{df}{dy} \right\} = \{2x + y, x\}$$

3.2 APPRENDIMENTO SUPERVISIONATO

L'algoritmo di backpropagation è un algoritmo di *apprendimento supervisionato*, che prevede di presentare alla rete per ogni esempio di addestramento la corrispondente uscita desiderata.

Di solito i pesi vengono inizializzati con valori casuali all'inizio dell'addestramento. Poi si cominciano a presentare, uno alla volta, gli esempi costituenti l'insieme di addestramento (training set). Per ogni esempio presentato si calcola l'errore commesso dalla rete, cioè la differenza tra l'uscita desiderata e l'uscita effettiva della rete. L'errore è usato per aggiustare i pesi.

Il processo viene di solito ripetuto ripresentando alla rete, in ordine casuale, tutti gli esempi del training set finchè l'errore commesso su tutto il training set (oppure l'errore medio sul training set) risulta inferiore ad una soglia prestabilita.

Dopo l'addestramento la rete viene testata controllandone il comportamento su un insieme di dati, detto test set, costituito da esempi non utilizzati durante la fase di training. La fase di test ha quindi lo scopo di valutare la capacità di generalizzazione della rete neurale. Si dice che la rete ha imparato, cioè è in grado di fornire risposte anche per ingressi che non le sono mai stati presentati durante la fase di addestramento.

Ovviamente le prestazioni di una rete neurale dipendono fortemente dall'insieme di esempi scelti per l'addestramento. Tali esempi devono quindi essere rappresentativi della realtà che la rete deve apprendere e in cui verrà utilizzata. L'addestramento è in effetti un processo ad hoc dipendente dallo specifico problema trattato.

Tecnicamente, è necessario un insieme di addestramento L della forma:

$$L = \{(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_p, \bar{y}_p)\} \tag{4}$$

dove

$\bar{x}_\mu$ con $\mu = 1, \ldots, p$ è il vettore input
$\bar{y}_\mu$ con $\mu = 1, \ldots, p$ è il vettore dell'output desiderato

L'apprendimento consiste nel trovare una configurazione di pesi tale che l'output della rete sia il più vicino possibile all'output desiderato per tutti gli esempi presenti nel training set.

Formalmente, si tratta di trovare una configurazione di pesi tra tutte le possibili combinazioni $\overline{W}$ che minimizzi la seguente *funzione di errore*. Essa dipende dal vettore μ-esimo di output out_μ restituito dalla rete, dato il vettore μ-esimo di ingresso $\overline{x}_\mu$ e dal vettore μ-esimo di output $\overline{y}_\mu$ del training set. In termini matematici:

$$\begin{aligned}\min_{\bar{w}\in\overline{W}} E(\bar{w}) &= \frac{1}{2}\sum_{\mu=1}^{p} \| \underbrace{\bar{y}_\mu}_{\text{desiderato}} - \underbrace{out_\mu(\bar{w})}_{\text{reale}} \|_2^2 \\ &= \frac{1}{2}\sum_{\mu=1}^{p}\sum_{k=1}^{n} \left(y_k^\mu - out_k^\mu(\bar{w})\right)^2\end{aligned}$$

dove l'indice k rappresenta il valore corrispondente al k-esimo neurone di output e il termine $1/2$ è aggiunto per cancellare l'esponente durante la derivazione.

3.3 PROPAGAZIONE DELL'ERRORE

La funzione E è altamente non lineare pertanto il problema di ottimizzazione non è facile da risolvere. Per minimizzare la funzione si utilizza il metodo della discesa del gradiente. Per calcolare le derivate parziali $\partial E/\partial w_{ij}$ è necessario l'algoritmo di back propagation. L'algoritmo può essere diviso in due passi:

- *Forward pass:* l'input dato alla rete è propagato al livello successivo e così via ai livelli successivi (il flusso di informazioni si sposta in avanti, cioè forward). Si calcola dunque $E(\bar{w})$, l'errore commesso.
- *Backward pass:* L'errore fatto dalla rete è propagato all'indietro (backward) e i pesi sono aggiornati in maniera appropriata.

Il seguente schema descrive il funzionamento dell'algoritmo di retroazione.

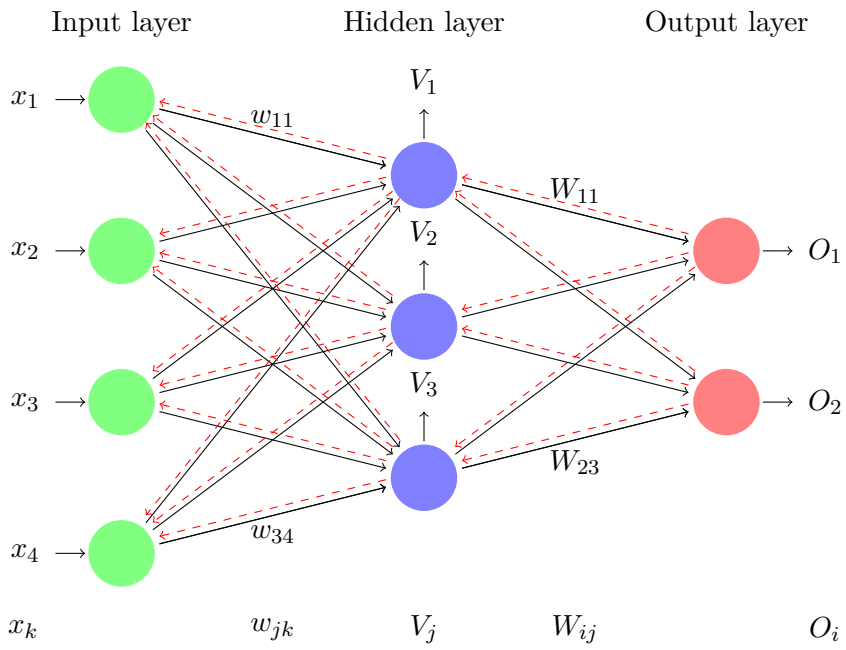


Figura 14: Schema back-propagation: le linee nere indicano il segnale propagato in avanti, mentre quelle rosse indicano l'errore propagato all'indietro.

Considereremo un metodo semplice che aggiorna i pesi di pattern in pattern fino al raggiungimento di un'epoca. **Un'epoca** è un ciclo di presentazione di tutti gli esempi del training set. L'aggiustamento dei pesi viene fatto in base all'errore calcolato per ogni pattern presentato alla rete.

Si definiscono ora, in termini matematici, gli input e gli output prodotti dai neuroni nascosti e di output. Dato un pattern μ, l'unità nascosta j riceve un input netto dato da:

$$h_j^\mu = \sum_k w_{jk} x_k^\mu \tag{5}$$

e produce come output:

$$V_j^\mu = g(h_j^\mu) = g\left(\sum_k w_{jk} x_k^\mu\right) \tag{6}$$

Dove g è la funzione di attivazione. In modo analogo il segnale entrante nell'unità output i è definito come:

$$h_i^\mu = \sum_j W_{ij} \cdot V_j^\mu \tag{7}$$

E l'output prodotto O_i è:

$$O_i = g(h_i^\mu) = g\left(\sum_j W_{ij} V_j^\mu\right) \tag{8}$$

I pesi sono aggiornati nella direzione opposta rispetto al gradiente; siamo infatti interessati a trovare un minimo per la funzione E. Quindi i pesi all'iterazione $t+1$ sono così calcolati:

$$w^{t+1} = w^t - \eta \cdot \nabla(E(w^t))$$

Il parametro η noto come **fattore di apprendimento** ha una forte influenza sul comportamento dell'algoritmo (velocità di convergenza, oscillazioni, ecc..).

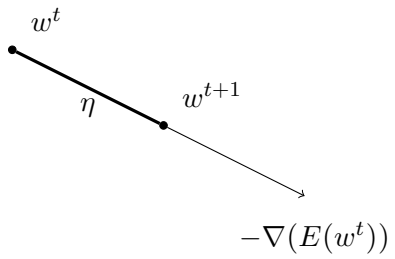


Figura 15: Direzione di aggiornamento dei pesi.

3.3.1 *Aggiornamento pesi strato nascosto-output*

Calcoliamo ora la variazione di peso sinaptico tra il neurone di output i e il neurone nascosto j per un certo pattern μ del training set. Da notare come nella Figura 14 utilizziamo la W maiuscola per indicare i pesi tra il layer nascosto e quello ouput.

$$\begin{aligned}
\Delta W_{ij} &= -\eta \cdot \frac{\partial E}{\partial W_{ij}} \\
&= -\eta \cdot \frac{\partial}{\partial W_{ij}} \left[\frac{1}{2} \sum_{\mu} \sum_{k} (y_k^\mu - O_k^\mu)^2 \right] \\
&= -\eta \cdot \sum_{\mu} \sum_{k} (y_k^\mu - O_k^\mu) \cdot \left(-\frac{\partial O_k^\mu}{\partial W_{ij}} \right) = \eta \cdot \sum_{\mu} \sum_{k} (y_k^\mu - O_k^\mu) \frac{\partial O_k^\mu}{\partial W_{ij}}
\end{aligned}$$

Per $k \neq i$, O_k non dipende da W_{ij}, quindi la derivata per O_k è 0; ad esempio vedi Figura 14 il peso W_{11} non influenza O_2.

$$\begin{aligned}
&= \eta \cdot \sum_{\mu} (y_i^\mu - O_i^\mu) \frac{\partial O_i^\mu}{\partial W_{ij}} \\
&= \eta \cdot \sum_{\mu} (y_i^\mu - O_i^\mu) \cdot g'(h_i^\mu) \cdot \frac{\partial h_i^\mu}{\partial W_{ij}}
\end{aligned}$$

Calcoliamo la derivata di $\partial h_i^\mu / \partial W_{ij}$:

$$\begin{aligned}
\frac{\partial h_i^\mu}{\partial W_{ij}} &= \frac{\partial}{\partial W_{ij}} \sum_{l} V_l^\mu \cdot W_{il} \\
&\text{Come prima per } l \neq j \text{ la derivata di } h_i^\mu \text{ è } 0 \\
&= \frac{\partial}{\partial W_{ij}} V_j^\mu \cdot W_{ij} = V_j^\mu
\end{aligned}$$

Quindi:

$$\Delta W_{ij} = \eta \cdot \sum_{\mu} (y_i^\mu - O_i^\mu) \cdot g'(h_i^\mu) \cdot V_j^\mu = \eta \cdot \sum_{\mu} \delta_i^\mu V_j^\mu$$

Dove:

$$\delta_i^\mu = (y_i^\mu - O_i^\mu) g'(h_i^\mu)$$

è l'errore del neurone i-esimo ed è noto come **gradiente locale**.

3.3.2 *Aggiornamento pesi strato input-nascosto*

Si passa ora al calcolo della variazione di peso sinaptico tra il neurone nascosto j e il neurone di input k per un certo pattern μ del training set.

$$\begin{aligned}\Delta w_{jk} &= -\eta \cdot \frac{\partial E}{\partial w_{jk}} \\ &= \eta \cdot \sum_{\mu} \sum_{i} (y_i^\mu - O_i^\mu) \frac{\partial O_i^\mu}{\partial w_{jk}} \\ &= \eta \cdot \sum_{\mu} \sum_{i} (y_i^\mu - O_i^\mu) \cdot g'(h_i^\mu) \frac{\partial h_i^\mu}{\partial w_{jk}}\end{aligned}$$

In questo caso la seconda sommatoria rimane in quanto l'output dipende indirettamente dall'input; ad esempio il segnale entrante in O_1 dipende indirettamente da w_{jk}. Calcoliamo ora $\partial h_i^\mu / \partial w_{jk}$:

$$\begin{aligned}\frac{\partial h_i^\mu}{\partial w_{jk}} &= \sum_{l} W_{il} \frac{\partial V_l^\mu}{\partial w_{jk}} \\ &\quad \text{Come prima per } l \neq j,\ V_l \text{ non dipende da } w_{jk}. \\ &= W_{ij} \frac{\partial V_j^\mu}{\partial w_{jk}} \\ &= W_{ij} \frac{\partial g(h_j^\mu)}{\partial w_{jk}} \\ &= W_{ij} g'(h_j^\mu) \frac{\partial h_j^\mu}{\partial w_{jk}}\end{aligned}$$

La derivata di $\partial h_j^\mu / \partial w_{jk}$ è:

$$\begin{aligned}\frac{\partial h_j^\mu}{\partial w_{jk}} &= \frac{\partial}{\partial w_{jk}} \sum_{m} w_{jm} x_m^\mu \\ &\quad \text{Come al solito per } m \neq k \text{ la derivata è 0.} \\ &= \frac{\partial}{\partial w_{jk}} w_{jk} x_k^\mu \\ &= x_k^\mu\end{aligned}$$

Quindi abbiamo:

$$\begin{aligned}\Delta w_{jk} &= \eta \cdot \sum_{\mu} \sum_{i} (y_i^{\mu} - O_i^{\mu}) \cdot g'(h_i^{\mu}) W_{ij} g'(h_j^{\mu}) x_k^{\mu} \\ &= \eta \cdot \sum_{\mu} \sum_{i} \delta_i^{\mu} W_{ij} g'(h_j^{\mu}) x_k^{\mu} \\ &= \eta \cdot \sum_{\mu} \hat{\delta}_j^{\mu} x_k^{\mu}\end{aligned}$$

Dove:

$$\hat{\delta}_j^{\mu} = g'(h_j^{\mu}) \sum_{i} \delta_i^{\mu} W_{ij}$$

è il gradiente locale del neurone nascosto j e la sommatoria rappresenta l'errore medio sullo strato di output causato dal neurone j dello strato nascosto.

3.3.3 *In generale*

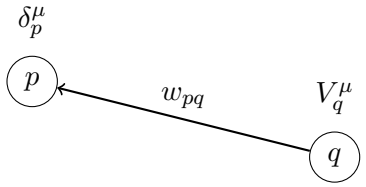


Figura 16: Variazione di peso sinaptico.

In generale per l'aggiornamento dei pesi non sono necessarie informazioni globali, ma solo locali. La variazione di peso sinaptico tra un neurone p e q relativo ad un pattern μ del training set è dato da:

$$\Delta w_{pq} = \eta \, \delta_p^\mu \, V_q^\mu \tag{9}$$

Dove:

$$\delta_p^\mu = \begin{cases} (y_p^\mu - O_p^\mu) \, g'(h_p^\mu), & \text{se } p \text{ è un neurone di ouput} \\ g'(h_p^\mu) \sum_i \delta_i^\mu \, W_{ip} & \text{altrimenti} \end{cases}$$

Questo modello prende il nome di apprendimento **on-line** in quanto la rete apprende un esempio alla volta. Esiste però un altro modello detto **off-line** o batch che considera tutti gli esempi dell'insieme di addestramento alla volta, anche se questo è meno coerente al modello biologico. Per l'apprendimento off-line abbiamo la seguente regola di apprendimento:

$$\Delta w_{pq} = \eta \sum_\mu \delta_p^\mu \, V_q^\mu \tag{10}$$

3.4 I PASSI DELL'ALGORITMO

Si consideri una rete con M strati e $m = 0, \ldots, M$. Si denota inoltre

- V_i^m: output della i-esima unità del livello m;
- w_m^{ij}: peso della connessione tra il j-esimo neurone dello strato $m-1$ e l'i-esimo neurone dello strato m.

L'algoritmo di retroazione si compone dei seguenti passi:

① Inizializzazione dei pesi a (piccoli) valori casuali;

② Scelta di un pattern $\overline{x}^\mu$ da inserire allo strato input ($m = 0$):

$$V_k^0 = x_k^\mu \qquad \forall k$$

③ Propagazione del segnale in avanti:

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij} V_j^{m-1}\right)$$

④ Calcolo degli errori δ dello strato output;

$$\delta_i^M = (y_i^M - V_i^M)\, g'(h_i^M)$$

⑤ Calcolo degli errori δ per tutti gli strati precedenti;

$$\delta_i^{m-1} = g'\left(h_i^{m-1}\right) \sum_j \delta_j^m W_{ji}$$

⑥ Aggiornamento dei pesi;

$$w_{ij}^{NEW} = w_{ij}^{OLD} + \Delta w_{ij} \qquad \text{dove } \Delta w_{ij} = \eta \delta_i^m V_j^{m-1}$$

⑦ Torna al passo 2 fino alla convergenza.

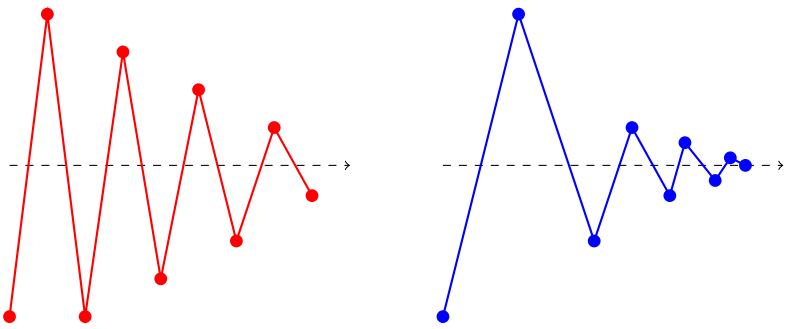
L'algoritmo si potrebbe fermare quando $\|w^{t+1} - w^t\| < \epsilon$ oppure dopo un certo numero di epoche se la condizione precedente non è rispettata.

3.5 IL MOMENTO

La scelta di η influenza molto il comportamento dell'algoritmo, infatti se scegliamo valori troppo piccoli, la convergenza sarà lenta, mentre se si scelgono valori troppo grandi si rischia di avere una rete instabile con comportamento oscillatorio. Un metodo semplice per incrementare il fattore di apprendimento senza il rischio di rendere la rete instabile consiste nel modificare la regola di aggiornamento inserendo un nuovo termine: *il momento*, una costante di proporzionalità alla precedente variazione dei pesi.

$$\Delta w_{pq}(t+1) = \underbrace{\alpha \Delta w_{pq}(t)}_{\text{momento}} - \eta \frac{\partial E}{\partial w_{pq}} \tag{11}$$

Dove α è un numero positivo a scelta preferibilmente compreso in $[0, 1)$. In questo modo, il cambiamento dei pesi per l'esempio $t+1$ dipende dal cambiamento apportato ai pesi per l'esempio t. L'utilizzo del momento ha l'effetto di incrementare o decrementare l'ampiezza di aggiornamento in modo tale da **accellerare nelle discese** o **stabilizzare** l'algoritmo in caso di oscillazioni. Questo fatto ha, inoltre, il vantaggio di **prevenire** (non con certezza) che il processo di apprendimento incappi in minimi locali della funzione d'errore.



(a) Assenza di momento $\alpha = 0$ (b) Con momento $\alpha = 0.5$

Figura 17: Momento e discesa del gradiente su una semplice superficie.

3.6 PROBLEMA DEL MINIMO LOCALE

L'algoritmo di back-propagation non sempre è in grado di trovare il *minimo globale*. Il problema risiede nell'esistenza di "buoni" o "cattivi" punti di minimo. Si consideri il seguente grafico:

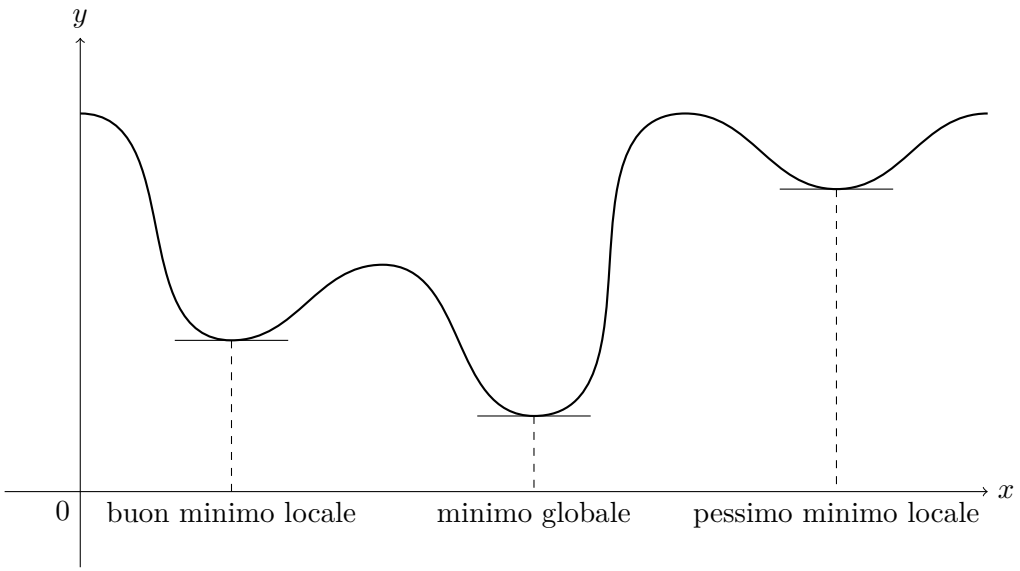


Figura 18: Rappresentazione del problema del minimo locale.

Per evitare il problema è importante scegliere una configurazione iniziale adeguata dei pesi. Se essi sono troppo elevati la derivata della funzione g sarà vicina a zero e di conseguenza Δw avrà valori molto piccoli. In questo modo il rischio di incappare in un minimo locale aumenta. Nella pratica si utilizza la seguente euristica per impostare i pesi iniziali:

$$w_{ij} \simeq \frac{1}{\sqrt{k_i}}$$

dove k_i è il numero di unità entranti nell'unità i.

4

EURISTICHE PER MIGLIORARE L'APPRENDIMENTO

Per garantire la convergenza di un algoritmo di apprendimento non esistono criteri ben definiti. Ci sono una serie di questioni pratiche e teoriche da affrontare quando si vuole addestrare una rete neurale. Ad esempio: di quanti strati una rete dovrà essere composta, oppure, quante unità devono esserci per ogni strato. Per rispondere a queste domande si introduce il concetto di **generalizzazione**.

4.1 GENERALIZZAZIONE

Definizione 1 (Generalizzazione)**.** *Per generalizzazione si intende la capacità di una rete di classificare correttamente esempi mai incontrati all'interno dell'insieme di addestramento (esempi di test).*

Si assume che gli esempi di test siano tratti dalla stessa popolazione usata per generare il training set. Si introduce dunque un nuovo insieme noto come insieme dei test (*test set*).

Il problema di apprendimento può essere visto come un problema di "approssimazione di una curva". In questo caso la rete è considerata semplicemente come una mappa input/output non lineare, quindi una buona generalizzazione è vista come una buona interpolazione dei dati di input.

Una rete progettata per generalizzare bene produce mappature input/output corrette anche se l'input è lievemente differente dagli esempi usati in fase di training; se però la rete viene addestrata con troppi esempi, si rischia di memorizzare il training set; la rete cioè apprende il *rumore* presente nel training set. Questo fenomeno è detto **overfitting** o **overtraining**. Una rete "sovra-addestrata" è troppo rigida e pertanto perde la capacità di generalizzazione.

In generale una rete troppo piccola impara poco, mentre una rete troppo grande impara molto, ma non generalizza abbastanza. In entrambi i casi si rischia di non riuscire a risolvere il problema. È, dunque, necessario trovare un **giusto compromesso**.

Dal momento che stiamo considerando l'apprendimento come un'approssimazione di una curva, il compromesso precedente può essere visto sotto il seguente profilo: troppi pochi parametri e.g. $y = ax + b$ non permettono di risolvere il problema, mentre con troppi parametri e.g. $y = ax^3 + bx^2 + cx + d$ si rischia l'overfitting.

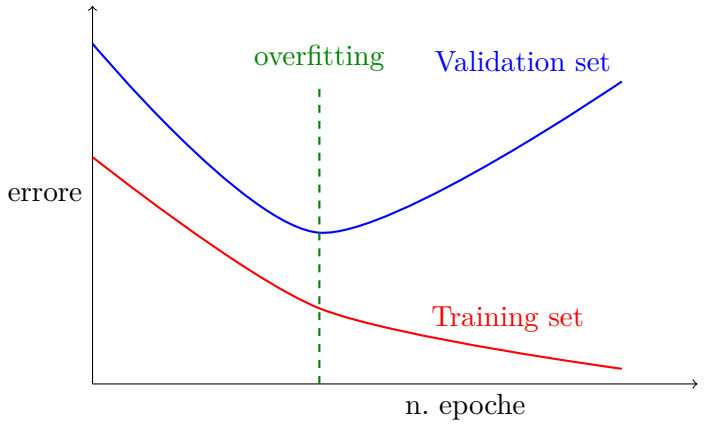


Figura 19: La curva blu mostra l'andamento dell'errore nel classificare i dati di training, mentre la curva rossa mostra l'errore nel classificare i dati di test o validazione. Se l'errore di validazione aumenta mentre l'errore sui dati di training diminuisce, ciò indica che siamo in presenza di un possibile caso di overfitting.

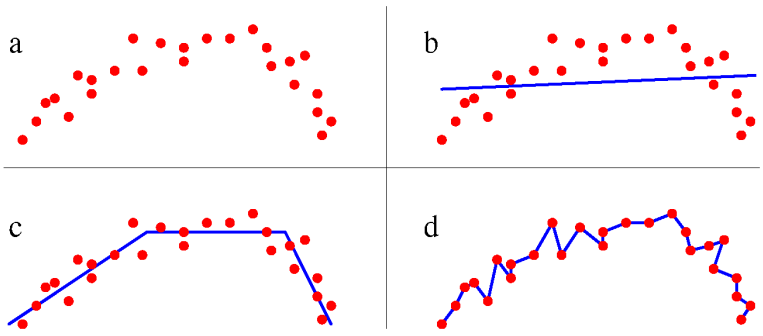


Figura 20: (a) dati del training set, (b) sotto approssimazione, (c) una buona stima sui dati, (d) overfitting: la curva di apprendimento è perfettamente disposta sul training set.

La capacità di generalizzazione è influenzata da tre fattori principali: le dimensioni del training set, l'architettura della rete neurale e la complessità del problema.

Alla luce del fatto che non si ha alcun controllo sulla complessità del problema, è possibile affrontare il problema della generalizzazione sotto due punti di vista differenti:

1. l'architettura della rete è prefissata e lo scopo è determinare una dimensione del training set ottimale per una buona generalizzazione;
2. la dimensione del training set è prefissata e lo scopo è di determinare la migliore architettura di rete per una buona generalizzazione.

4.2 CROSS-VALIDATION

Si introduce ora uno strumento per risolvere il problema della selezione dell'architettura della rete neurale; più precisamente questo strumento consente, dato un insieme di possibili modelli, di scegliere quello migliore secondo certi criteri. Inizialmente il set di dati viene ripartito in modo random in un training set e un test set. Il training set è ulteriormente diviso in due insiemi disgiunti:

- **Estimation subset:** usato per il training del modello;
- **Validation subset:** usato per validare il modello.

La motivazione è validare il modello su un set di dati diverso da quello usato per l'addestramento. In questo modo è possibile usare il training set per stimare le prestazioni dei vari modelli candidati e scegliere quindi il migliore. C'è comunque la possibilità che il modello più performante in realtà sia incappato in un overfitting del validation set. Per evitare questo si ricorre al test set per verificare la capacità di generalizzazione della rete.

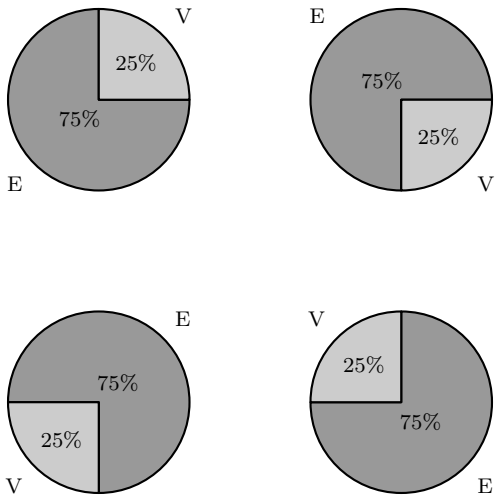


Figura 21: Esempio di cross-validation: partizionamenti del training set in estimation subset E e validation subset V.

4.2.1 *Variante del cross-validation*

Quando la dimensione del set di dati è piccola, si può ricorrere al *multifold cross-validation* dividendo il set di N esempi in K sottoinsiemi, $K > 1$. Il modello viene poi addestrato su ogni sottoinsieme ad eccezione di uno; quest'ultimo formerà il validation set. Questa procedura viene ripetuta K volte utilizzando per il validation set a turno uno dei K sottoinsiemi. Le prestazioni del modello vengono misurate facendo la media degli errori quadrati sul validation set per ognuna delle K ripetizioni.
Lo svantaggio di questa tecnica è che richiede molta computazione dal momento che il modello deve essere addestrato K volte. Quando il numero di esempi è ancora più piccolo si può ricorrere ad una forma estrema di questa tecnica detta **leave-one-out** in cui $K = N$.

4.3 METODO DI TRAINING "EARLY-STOPPING"

Con l'obiettivo di una buona generalizzazione è molto difficile decidere quando è il momento di bloccare il training. C'è il rischio di overfitting dei dati se non si ferma l'addestramento al punto giusto.
È possibile evitare il fenomeno dell'overfitting ricorrendo al metodo precedente; l'estimation set viene usato per addestrare la rete e il validation set serve a testarla dopo ogni epoca. Più precisamente il processo procede nel seguente modo:

- dopo un periodo di addestramento sull'estimation set, si calcola l'errore di validazione per ogni esempio del validation set;
- quando la fase di validazione è completa, si riprende la fase di addestramento per un altro periodo.

Se si osserva la sola curva dell'errore dell'estimation set (vedi Figura 22), questo si riduce all'aumentare delle epoche, e verrebbe naturale pensare di eseguire il training anche oltre il punto di minimo della curva del validation set. In realtà ciò che la rete apprende dopo quel punto è il rumore contenuto nei dati del training/estimation set. Questa euristica suggerisce quindi di fermare l'addestramento in corrispondenza del minimo della curva relativa al validation set.

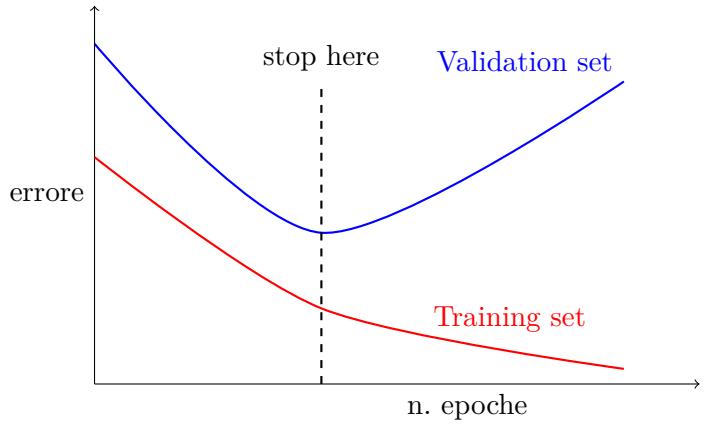


Figura 22: Andamento di errore e metodo di early stopping.

4.4 TECNICHE DI PRUNING

Le capacità funzionali e di generalizzazione di una determinata rete sono fortemente influenzate dalla sua dimensione (ovvero il numero di neuroni nascosti). Con una rete troppo piccola si rischia di non riuscire a risolvere il problema, mentre con una troppo grande si rischia di apprendere il rumore deteriorando la capacità di generalizzare. È necessario quindi un **compromesso**.

Le tecniche di network pruning hanno lo scopo di minimizzare le dimensioni della rete mantenendo buone prestazioni. È possibile raggiungere questo obiettivo in due modi:

1. **pruning**: si parte da una rete sufficientemente grande per poi ridurla eliminando connessioni sinaptiche o neuroni interi.
2. **growing**: si parte da una rete piccola per poi espanderla;

La prima tecnica di solito è più veloce in termini di numero di epoche (convergenza più veloce), ma richiede maggior peso computazionale dal momento che ci sono più unità. Ad ogni modo questo metodo è il più adottato.

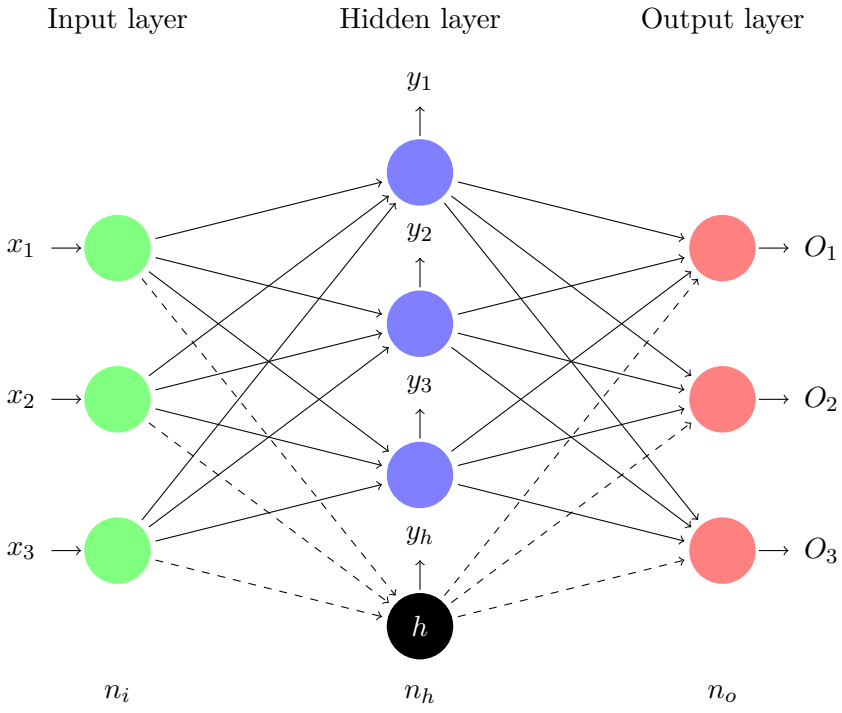


Figura 23: Rimozione del neurone h in una rete neurale.

L'approccio pruning consiste innanzitutto nel rimuovere un neurone h e successivamente nell'aggiustare opportunamente i *pesi entranti nelle unità servite dal neurone* h in modo tale da preservare il comportamento di input/output dell'intera rete. In termini matematici si tratta di mantenere la seguente relazione:

$$\underbrace{\sum_{j=1}^{n_h} w_{ij} y_j^\mu}_{\text{prima}} = \underbrace{\sum_{\substack{j=1 \\ j\neq h}}^{n_h} (w_{ij} + \delta_{ij})\, y_j^\mu}_{\text{dopo}} \qquad \forall i = 1, \ldots, n_o, \quad \forall \mu = 1, \ldots, P$$

dove δ_{ij} sono i fattori di aggiustamento che dovranno essere calcolati e y_j^μ indica l'output dell'unità j corrispondente a un certo pattern μ. Attraverso procedimenti algebrici l'equazione precedente può essere riscritta nel seguente modo:

$$\sum_{j=1}^{n_h} w_{ij} y_j^\mu = \sum_{\substack{j=1 \\ j\neq h}}^{n_h} w_{ij} y_j^\mu + \sum_{\substack{j=1 \\ j\neq h}}^{n_h} \delta_{ij} y_j^\mu$$

E quindi rimane:

$$\sum_{\substack{j=1 \\ j\neq h}}^{n_h} \delta_{ij} y_j^\mu = w_{ih} y_h^\mu$$

che è un tipico sistema lineare di equazioni con incognite $\{\delta_{ij}\}$. È possibile porre tale sistema in forma compatta:

$$A\bar{x} = b$$

dove A è la matrice di dimensione $P_{n_o} \times n_o(n_h - 1)$ e le cui colonne sono i vettori output delle unità servite da h. Dal momemto che il sistema è sovradeterminato (ci sono più equazioni che incognite) si può utilizzare il metodo dei minimi quadrati per approssimare una soluzione:

$$\min_{\bar{x}} \|A\bar{x} - b\| \tag{12}$$

In questo modo si riesce a preservare il comportamento della rete avendo rimosso il neurone h. Uno dei problemi fondamentali dell'algoritmo pruning è come meglio selezionare le unità che saranno rimosse. Idealmente, la scelta più appropriata sarebbe eliminare tutte le unità nascoste che hanno il più piccolo *residuo finale* calcolato dal sistema corrispondente.

Questo garantisce che la rimozione dell'unità avrà un impatto minimo sul comportamento della rete. Tuttavia, il calcolo del residuo minimo finale ha un costo computazionale piuttosto elevato (a causa del numero di soluzioni del sistema), pertanto nella pratica si cerca una soluzione ottimale utilizzando metodi di riduzione dei residui: si parte da una soluzione iniziale x_0 e si produce una sequenza di punti $\{x_k\}$ tale che i residui si riducono:

$$\|A\bar{x} - b\| = r_k$$

dove $r_0 \geq r_1 \geq \cdots \geq r_k \geq r_{k+1}$. Un'approssimazione accettabile alla soluzione ottimale globale sarebbe scegliere l'unità h da eliminare in modo tale che il *residuo iniziale* sia il più piccolo. Dal momento che x_0 di solito è il vettore nullo allora:

$$x_0 = 0 \implies r_0 = \|b\|$$

Si tratta quindi di scegliere l'unità con $\|b\|$ minimo. Naturalmente questo criterio non garantisce la soluzione ottimale globale in quanto non è detto che partendo dal residuo iniziale minimo si giunga al residuo finale minimo. Tuttavia, nella pratica, questo criterio si è dimostrato efficace e semplice da implementare.

Riassumendo i passi fondamentali dell'algoritmo di pruning sono i seguenti:

① Data una rete sovradimensionata;

② Ripetere:

   (a) Trovare l'unità h con $\|b\|$ minimo;

   (b) Risolvere il sistema corrispondente;

   (c) Rimuovere l'unità h.

   Fino a quando $Performance(pruned) - Performance(original) < \epsilon$

③ Rifiuta la rete ridotta.

5

RETI DI HOPFIELD

Si passa ora allo studio delle reti neurali viste come sistemi dinamici non lineari, con particolare attenzione al problema della loro stabilità o neurodinamica (vedi Appendice B). Un modo implicito per introdurre in una rete neurale la variabile tempo è mediante l'utilizzo di reti con feedback o ricorrenti.

5.1 MODELLO DI HOPFIELD: CASO DISCRETO

Le reti ricorrenti con unità non lineari sono generalmente difficili da analizzare. Possono comportarsi in modi differenti: convergere a uno stato stabile, oscillare o seguire *traiettorie caotiche* il cui andamento non è prevedibile.

Tuttavia, il fisico americano *John Hopfield* (e altri gruppi) si accorsero che se le connessioni sono **simmetriche** esiste una *funzione di energia globale*.

Le reti di Hopfield sono dunque:

- **reti ricorrenti ad uno strato** in cui ogni neurone è connesso a tutti gli altri (sono assenti connessioni con se stesso), quindi $w_{ii} = 0$;
- **simmetriche**: perché hanno la matrice dei pesi sinaptici simmetrica, ovvero $W = W^T$;
- **non lineari**: ogni neurone ha una funzione di attivazione non lineare invertibile.

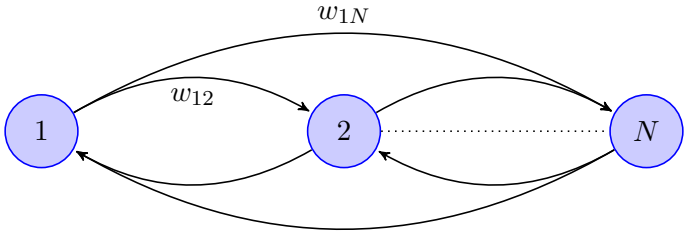


Figura 24: Una rete ricorrente.

Per quanto riguarda l'aggiornamento di un neurone si possono scegliere tre strade diverse:

1. **aggiornamento asincrono**: in cui si aggiorna un neurone alla volta;
2. **aggiornamento sincrono**: tutti i neuroni vengono aggiornati allo stesso istante;
3. **aggiornamento continuo**: in cui tutti i neuroni si aggiornano continuamente.

In questa sezione saranno trattate le reti di Hopfield nel caso in cui il tempo scorra in maniera discreta e i neuroni si aggiornino in modo asincrono.

Nel caso discreto si adotta lo stesso modello di McCulloch e Pitts con l'aggiunta di un fattore esterno che influenza l'input al neurone i.

$$H_i = \underbrace{\sum_{j \neq i} w_{ij} V_j}_{\text{modello M\&P}} + \underbrace{I_i}_{\text{input esterno}} \tag{13}$$

E l'output di ogni neurone è così definito:

$$V_i = \begin{cases} +1, & \text{se } H_i \geq 0 \\ -1, & \text{se } H_i < 0 \end{cases} \tag{14}$$

L'aggiornamento dei neuroni è un processo casuale. Per selezionare il neurone da aggiornare si può procedere in due modi:

1. ad ogni istante temporale si sceglie a caso l'unità i da aggiornare (utile nelle simulazioni);
2. Ogni unità si aggiorna indipendentemente con probabilità costante ad ogni istante.

A differenza delle reti feed-forward quelle di Hopfield sono sistemi dinamici. La rete parte da uno stato iniziale

$$V(0) = (V_1(0), \ldots, V_n(0))^T$$

e si evolve lungo una traiettoria fino a raggiungere un punto fisso in cui $V(t+1) = V(t)$ (convergenza).

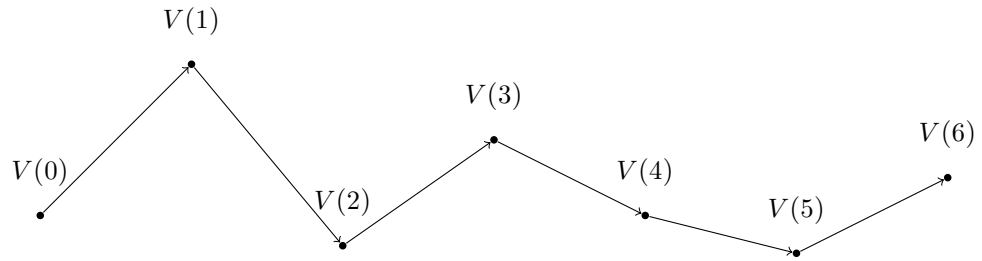


Figura 25: Traiettoria di una rete di Hopfield

5.1.1 *La funzione energia*

Si procede ora allo studio del comportamento di una rete di Hopfield; fino ad ora infatti non è stato dimostrato se la rete converge oppure produce cicli. Il sistema è governato da una funzione di energia E. L'energia globale è la somma di una serie di contributi locali. Ogni contributo dipende da **una** connessione w_{ij} e dagli stati binari dei **due** neuroni V_iV_j. La funzione energia è dunque così definita:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} w_{ij}V_iV_j - \sum_{i=1}^{n} I_iV_i \tag{15}$$

Dove I è il fattore esterno (termine bias) e coinvolge lo stato di un'unità individualmente. Il fattore $1/2$ è aggiunto perché i termini identici $w_{ij}x_ix_j$ e $w_{ji}x_jx_i$ sono presenti nella doppia sommatoria. Questa funzione permette ad ogni unità di calcolare **localmente** come il suo cambio di stato influenza l'energia globale. In termini matematici la differenza di energia è definita nel seguente modo:

$$\textit{Energy gap } \Delta E = E(t+1) - E(t) \tag{16}$$

5.1.2 *Teorema di Hopfield nel caso discreto*

Si enuncia ora una condizione sufficiente alla convergenza del sistema.

Teorema 3 (Terorema di Hopfield - Caso discreto)**.** *Se la matrice dei pesi di una rete di Hopfield è:*

1. *simmetrica:* $W = W^T$*;*
2. *gli elementi lungo la diagonale di* W *sono nulli:* $diag(W) = 0$*.*

Allora la funzione di energia:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}V_iV_j - \sum_{i=1}^{n} I_iV_i$$

è una funzione di Lyapunov per il sistema. E quindi:

$$\Delta E = E(t+1) - E(t) < 0$$

dove $\Delta E = 0$ *solo quando il sistema ha raggiunto un punto stazionario.*

Dimostrazione: Assumendo che i neuroni si aggiornano in modo asincrono, supponiamo che il neurone h cambi il proprio stato. La differenza di energia è data da:

$$\begin{aligned}
\Delta E &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}V_i(t+1)V_j(t+1) - \sum_{i=1}^{n} I_iV_i(t+1) \\
&\quad + \frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}V_i(t)V_j(t) + \sum_{i=1}^{n} I_iV_i(t) \\
&= \underbrace{-\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_{ij}\cdot[V_i(t+1)V_j(t+1) - V_i(t)V_j(t)]}_{A} \\
&\quad \underbrace{-\sum_{i=1}^{n} I_i\cdot[V_i(t+1) - V_i(t)]}_{B}
\end{aligned}$$

Si consideri il secondo termine B:

$$B = -\sum_{i=1}^{n} I_i \Delta V_i$$

ora dal momento che h è l'unico stato che è cambiato abbiamo che

$$B = -I_h \Delta V_h$$

in quanto tutti gli altri Δ sono uguali a 0.

Si passa ora al primo termine A. Si espande la sommatoria isolando il neurone h:

$$A = \underbrace{-\frac{1}{2}\sum_{i\neq h}\sum_{j\neq i} w_{ij}\cdot[V_i(t+1)V_j(t+1) - V_i(t)V_j(t)]}_{A_1}$$
$$\underbrace{-\frac{1}{2}\sum_{j\neq h} w_{hj}\cdot[V_h(t+1)V_j(t+1) - V_h(t)V_j(t)]}_{A_2}$$

Dal momento che solo il neurone h cambia stato allora $V_i(t) = V_i(t+1)$ per $i \neq h$ quindi A_1 può essere riscritto nel seguente modo:

$$A_1 = -\frac{1}{2}\sum_{i\neq h}\sum_{j\neq i} w_{ij}\cdot[\underbrace{V_i(t+1)}_{=V_i(t)}V_j(t+1) - V_i(t)V_j(t)]$$
$$= -\frac{1}{2}\sum_{i\neq h}\sum_{j\neq i} w_{ij}\cdot V_i(t)\underbrace{[V_j(t+1) - V_j(t)]}_{=\Delta V_j}$$

A questo punto $\Delta V_j \neq 0$ solo quando $j = h$ quindi $\Delta V_j = \Delta V_h$

$$= -\frac{1}{2}\sum_{i\neq h} w_{ih} V_i(t)\Delta V_h$$

Per quanto riguarda A_2 il procedimento è lo stesso:

$$A_2 = -\frac{1}{2}\sum_{j\neq h} w_{hj}\cdot[V_h(t+1)V_j(t+1) - V_h(t)V_j(t)]$$
$$= -\frac{1}{2}\sum_{i\neq h} w_{hi} V_i(t)\Delta V_h$$

Siccome la matrice dei pesi è simmetrica $W = W^T$ allora $w_{ih} = w_{hi}$ e quindi:

$$A = A_1 + A_2 = -\Delta V_h \sum_{i \neq h} w_{ih} V_i(t)$$

In questo modo si ottiene:

$$\begin{aligned}
\Delta E &= -\Delta V_h \sum_{i \neq h} w_{ih} V_i(t) - I_h \Delta V_h \\
&= -\Delta V_h \underbrace{\left[\sum_{i \neq h} w_{ih} V_i(t) + I_h \right]}_{\text{input di } h \text{ al tempo t}} \\
&= -\Delta V_h \cdot H_h
\end{aligned}$$

A questo punto è necessario dimostrare che il prodotto $\Delta V_h \cdot H_h$ è sempre positivo in modo tale da provare la tesi, ovvero che $\Delta E \leq 0$. Per la regola di apprendimento (14):

1. se $V_h > 0 \Leftrightarrow H_h \geq 0$
2. se $V_h < 0 \Leftrightarrow H_h < 0$

Dunque il prodotto è sempre positivo C.V.D. □

5.2 REGOLA DI HEBB

Nei computer se si vuole accedere ad una certa informazione si utilizza il suo preciso indirizzo di memoria. Questo tipo di memoria è definita *byte-addressable memory*. Nel cervello umano, invece, la memoria è indirizzata in base al contenuto, *content-addressable memory*. Ad esempio pensare alla parola "volpe" potrebbe automaticamente attivare memorie relative ad altri animali simili, alla caccia, oppure al concetto di furbizia.

In questo contesto, è stata introdotta la *regola di Hebb* per descrivere questo meccanismo di accesso alle informazioni. L'apprendimento Hebbiano si basa sul semplice principio che se due neuroni si attivano contemporaneamente, la loro interconnessione deve essere rafforzata. In dettaglio il postulato di Hebb afferma che:

Quando un assone di un neurone A è abbastanza vicino da eccitare un neurone B e questo, in modo ripetitivo e persistente, gli invia un potenziale di azione, inizia un processo di crescita in uno o entrambi i neuroni tali da incrementare l'efficienza di A.

In termini matematici la memoria è rappresentata da un insieme di p patterns x^μ, con $\mu = 1, \ldots, p$. Quando viene presentato un nuovo pattern x, la rete risponde producendo il patern salvato in memoria che più assomiglia ad x.

Hebb suggerisce variazioni sull'efficacia sinaptica proporzionali alla correlazione nell'attivazione tra un neurone pre e post sinaptico.

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} x_i^\mu x_j^\mu \tag{17}$$

dove N è il numero di unità binarie con output $s_1, \ldots, s_N$
Si consideri il seguente esempio con due pattern che la rete ha imparato:

$$x^1 = (-1, -1, -1, +1)$$
$$x^2 = (+1, +1, +1, +1)$$

I pesi sono cosi calcolati:

$$w_{ij} = \frac{1}{4}\sum_{\mu=1}^{2} x_i^\mu x_j^\mu$$

Ad esempio:

$$w_{12} = \frac{1}{4}(x_1^1 \cdot x_2^1 + x_1^2 \cdot x_2^2) = \frac{1}{4}(-1 \cdot -1 + 1 \cdot 1) = \frac{1}{4} \cdot 2$$

E quindi la matrice dei pesi ha la seguente forma:

$$W = \frac{1}{4} \cdot \begin{bmatrix} 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Si ricorda che l'ouput dell'unità i è dato da:

$$s_i = Sgn\left(\sum_j w_{ij} s_j\right)$$

Ad esempio dato l'input $(-1, -1, -1, +1)$ la rete produce il seguente output:

$$\begin{aligned} s_1 &= Sgn\left(\frac{1}{4} \cdot \begin{bmatrix} 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \cdot (-1, -1, -1, +1)^T\right) \\ &= Sgn(-1.5, -1.5, -1.5, 0.5) = (-1, -1, -1, +1) \end{aligned}$$

Ed è una soluzione stabile in quanto corrisponde ad uno dei pattern memorizzati dalla rete, mentre il seguente esempio è una soluzione spuria:

$$input^2 = (-1, -1, -1, -1) \longrightarrow output^2 = (-1, -1, -1, -1) \qquad \text{(spuria)}$$

5.3 MODELLO DI HOPFIELD: CASO CONTINUO

In questo modello i neuroni non sono più dispositivi binari con stati (0,1) o (-1,1), ma assumono un numero reale che identifica la quantità di corrente elettrica. Lo scopo di Hopfield era infatti quello di fornire un'implementazione fisica del suo modello e che imiti il più possibile il funzionamento di un cervello biologico.

In questo caso, l'output di un neurone i è il seguente:

$$V_i = g_\beta(\mu_i) = g_\beta\left(\sum_j w_{ij}V_j + I_i\right) \qquad \text{oppure } \mu_i = g_\beta^{-1}(V_i) \tag{18}$$

dove g_β è la funzione di attivazione crescente, continua e non lineare. Le funzioni più utilizzate come g sono la tangente iperbolica e la sigmoidea:

$$\tanh_\beta \mu = \frac{e^{\beta\mu} - e^{-\beta\mu}}{e^{\beta}\mu + e^{-\beta\mu}} \quad \in]-1,1[\qquad\qquad g_\beta(\mu) = \frac{1}{1+e^{-\beta\mu}} \quad \in]0,1[$$

Il parametro β indica la "stickiness" della funzione. Ad esempio con $\beta \to \infty$ la funzione diventa sempre più ripida fino a diventare lineare.

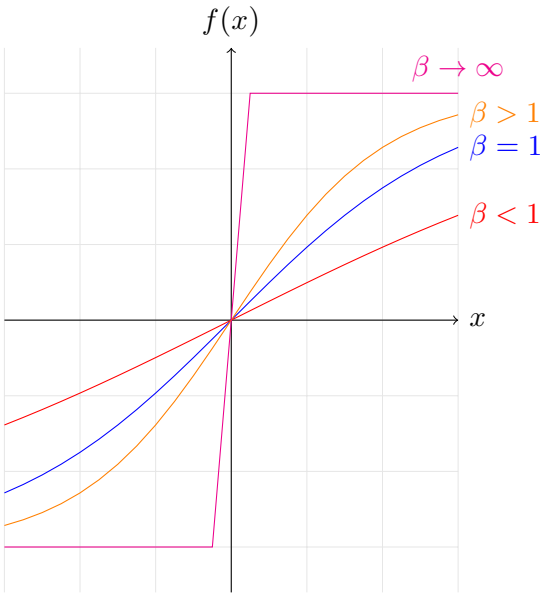
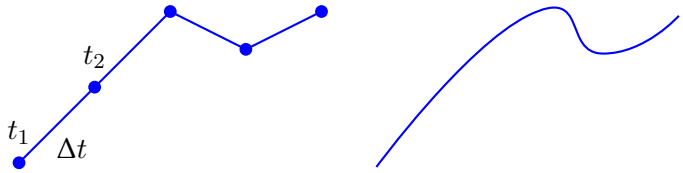


Figura 26: La funzione tangente iperbolica $f(x) = \tanh(\beta\mu)$

L'utilizzo di valori continui permette la modalità di *aggiornamento continuo*. In questo modo il tempo scorre in maniera continua e i neuroni si aggiornano continuamente.



(a) Caso discreto: Δt costante (b) Caso continuo: $\Delta t \to 0$

Figura 27: Traiettoria di una rete di Hopfield nel caso continuo e discreto.

Per descrivere completamente il comportamento del neurone si deve aggiungere alla (18) l'equazione che definisce il valore di attivazione del neurone stesso (vedi Appendice B); in questo caso, esso non è più la semplice somma algebrica degli ingressi (caso discreto) ma è una soluzione della seguente equazione differenziale:

$$V_i + \tau_i \frac{dV_i}{dt} = g_\beta \left(\sum_j w_{ij} V_j + I_i \right)$$

$$\tau_i \frac{dV_i}{dt} = -V_i + g_\beta(\mu_i) = -V_i + g_\beta \left(\sum_j w_{ji} V_j + I_i \right)$$

dove τ_i è una costante ($\tau_i > 0$) e rappresenta la resistenza elettrica. Il sistema raggiunge la stabilità quando il vettore velocità del sistema si azzera $\Delta t = dV_i/dt = 0 \, \forall i$, ovvero quando l'uscita di ogni neurone V_i è uguale a $g_\beta(\mu_i)$:

$$V_i = g_\beta(\mu_i)$$

In maniera del tutto equivalente è possibile esprimere l'equazione di stato non incentrando l'attenzione sulla variazione V_i dello stato nel tempo, quanto sulla variazione μ_i dell'input netto nel tempo, pervenendo alla seguente equazione:

$$\mu_i + \tau_i \frac{d\mu_i}{dt} = \sum_j w_{ij} V_j + I_i = \sum_j w_{ij} g_\beta(\mu_j) + I_i \tag{19}$$

$$\tau_i \frac{d\mu_i}{dt} = -\mu_i + \sum_j w_{ij} g_\beta(\mu_j) + I_i \tag{20}$$

La funzione energia nel modello continuo è simile a quella nel caso discreto ed è così definita:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} V_i V_j + \sum_i \int_0^{V_i} g_\beta^{-1}(V)\, dV - \sum_i I_i V_i \tag{21}$$

Si dimostra ora il teorema di Hopfield nel caso continuo:

Teorema 4 (Teorema di Hopfield - Caso Continuo)**.** *Se $W = W^T$ e $diag(W) = 0$ allora la funzione di energia E è Lyapunoviana per il sistema e quindi $dE/dt \leq 0$, dove l'uguaglianza si ottiene quando il sistema ha raggiunto un punto stazionario.*

Dimostrazione: È necessario dimostrare che $dE/dt \leq 0$ con uguaglianza quando un punto fisso è stato raggiunto. Si ricorda che solo V_i dipende dal tempo.

Nel calcolo della derivata dE/dt, al primo termine è applicata la regola del prodotto mentre per il secondo la regola della catena e il teorema fondamentale del calcolo integrale (vedi Appendice A). In questo modo si ottiene:

$$\frac{dE}{dt} = -\frac{1}{2}\sum_i \sum_j w_{ij} \frac{dV_i}{dt} V_j - \frac{1}{2}\sum_i \sum_j w_{ij} V_i \frac{dV_j}{dt} + \sum_i \underbrace{g_\beta^{-1}(V_i)}_{=\mu_i} \frac{dV_i}{dt} - \sum_i I_i \frac{dV_i}{dt}$$

Assumendo che W è simmetrica le prime due sommatorie sono uguali e quindi:

$$\begin{aligned}
\frac{dE}{dt} &= -\sum_i \sum_j w_{ij} \frac{dV_i}{dt} V_j + \sum_i \mu_i \frac{dV_i}{dt} - \sum_i I_i \frac{dV_i}{dt} \\
&= -\sum_i \frac{dV_i}{dt} \underbrace{\left(\sum_j w_{ij} V_j - \mu_i + I_i \right)}_{= \tau_i \frac{d\mu_i}{dt}} \\
&= -\sum_i \tau_i \frac{dV_i}{dt} \frac{d\mu_i}{dt}
\end{aligned}$$

dove

$$\frac{dV_i}{dt} = \frac{d}{dt}\left(g_\beta(\mu_i)\right) = \frac{d\mu_i}{dt} \cdot g'_\beta(\mu_i)$$

e quindi:

$$\frac{dE}{dt} = -\sum_i \tau_i g'_\beta(\mu_i) \left(\frac{d\mu_i}{dt}\right)^2 \leq 0$$

L'ultima disuguaglianza è vera perché g_β è monotona crescente quindi la sua derivata $g'_\beta \geq 0$, $\tau_i > 0$ e il quadrato di un numero è sempre positivo. Dunque vale la seguente doppia implicazione:

$$\frac{dE}{dt} = 0 \Leftrightarrow \frac{d\mu_i}{dt} = 0$$

cioè il valore dell'energia nel tempo rimane fisso se e solo se la rete ha raggiunto un punto di equilibrio, ovvero μ_i. □

5.4 CORRISPONDENZA TRA I DUE MODELLI

Esiste una relazione stretta tra il modello continuo e quello discreto. Si noti che:

$$V_i = g_\beta(\mu_i) = g_1(\beta\mu_i) = g(\beta\mu_i)$$

Da cui si ricava che:

$$\mu_i = \frac{1}{\beta} g^{-1}(V_i)$$

Il secondo termine della funzione energia diventa quindi:

$$\sum_i \int_0^{V_i} g_\beta^{-1}(V)\, dV = \frac{1}{\beta} \cdot \sum_i \int_0^{V_i} g^{-1}(V)\, dV$$

Ora per $\beta \to \infty$ il termine diventa trascurabile e la funzione E risulta uguale a quella nel modello discreto.

$$\begin{aligned} E &= -\frac{1}{2} \sum_i \sum_j w_{ij} V_i V_j + \underbrace{\frac{1}{\beta} \cdot \sum_i \int_0^{V_i} g^{-1}(V)\, dV}_{=0 \text{ per } \beta \to \infty} - \sum_i I_i V_i \\ &= -\frac{1}{2} \sum_i \sum_j w_{ij} V_i V_j - \sum_{i=1}^{n} I_i V_i \end{aligned}$$

Intuitivamente per $\beta \to \infty$ la funzione diventa sempre più rigida fino a diventare lineare (vedi Figura 26). Di conseguenza la funzione energia del caso continuo ora coincide con quella del modello discreto.

6

OTTIMIZZAZIONE CON LE RETI NEURALI

In questo capitolo saranno introdotti alcuni famosi problemi NP-Difficili: il problema del commesso viaggiatore (TSP) e il problema della cricca massima (MCP). Dal momento che questi non sono risolvibili in termini polinomiali sarà fornita una formulazione alternativa del problema in modo tale da poter utilizzare tecniche di ottimizzazione per approssimare una soluzione.

6.1 PROBLEMA DEL COMMESSO VIAGGIATORE

Il problema del commesso viaggiatore (in inglese "travelling salesman problem") nasce dalla sua più tipica rappresentazione: data una rete di città, connesse tramite delle strade, trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città una e una sola volta per poi tornare alla città di partenza.

Il problema è di considerevole importanza pratica, al di là delle ovvie applicazioni nella logistica e nei trasporti.

6.1.1 *Formulazione del problema*

Espresso nei termini della teoria dei grafi il TSP è così formulato: *dato un grafo completo pesato* $G = (V, E, w)$*, trovare il cammino di costo minore visitando tutti i nodi* V *una sola volta e tornando al nodo di partenza.*

La rete di città può essere rappresentata come un grafo in cui le città sono i nodi V, le strade gli archi E e le distanze i pesi sugli archi w. La differenza sostanziale rispetto al Ciclo Hamiltoniano si trova nel fatto che quest'ultimo è formulato su di un grafo privo di pesi. Il problema è NP-completo: per poter trovare il percorso minimo è necessario elencare tutti gli $n!$ percorsi.

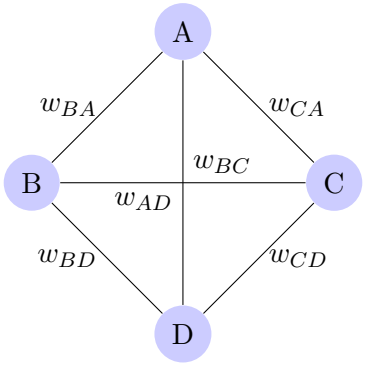


Figura 28: Grafo completo pesato.

6.1.2 *Soluzione del TSP con le reti di Hopfield*

È possibile affrontare questo problema utilizzando le reti di Hopfield. Per fare ciò è necessario *trovare una funzione di energia E adeguata in modo tale che il minimo corrisponda alla soluzione del problema che si sta affrontando.*

Per prima cosa è necessario rappresentare il TSP attraverso una matrice di permutazione per identificare un percorso. Ad esempio dato il grafo in Figura 28, il percorso $B \to A \to C \to D$ può essere rappresentato con la seguente matrice:

Città \ Fermata	1	2	3	4
A	0	**1**	0	0
B	**1**	0	0	0
C	0	0	**1**	0
D	0	0	0	**1**

Tabella 1: Matrice di permutazione del percorso BACD

Dunque per n città saranno necessari $n \times n$ neuroni nella rete di Hopfield in cui ogni neurone identifica una città e una fermata.

La soluzione del problema sarà dunque una matrice V di dimensione $n \times n$. Si introducono ora alcune notazioni:

- X indica la città;
- i una fermata in cui è visitata una città;
- $V_{X,i}$ è l'output dell'unità X, i;
- $T_{Xi,Yj}$ sono i pesi delle connessioni;
- $V_{X,i} = 1$ se la città X è visitata alla fermata i;
- $d_{X,Y}$ distanza tra la città X e Y.

Si definisce ora la funzione obiettivo da minimizzare che rappresenta il costo totale del cammino:

$$E_1 = \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{X,Y} V_{X,i} (V_{Y,i-1} + V_{Y,i+1})$$

dove D è una costante e gli indici sono intesi modulo n. La matrice V senza vincoli di alcun genere può descrivere zero o più percorsi di lunghezza arbitraria (quindi può anche non toccare tutte le città). Se la matrice V descrivesse solo percorsi unici Hamiltoniani, allora minimizzando la funzione costo E_1 otterremmo la soluzione. Tuttavia il problema del commesso viaggiatore presenta una serie di **vincoli** che devono essere rispettati dalla matrice V.

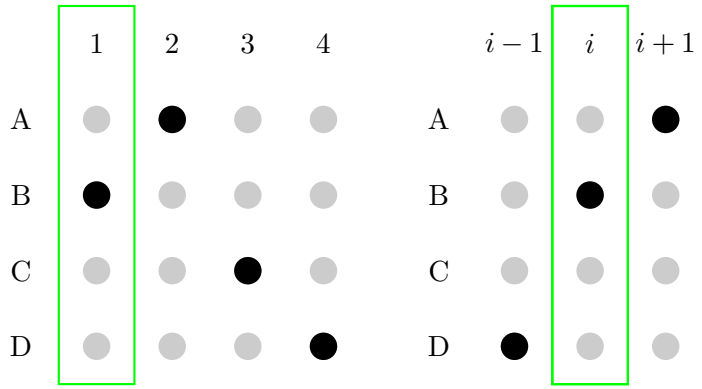


Figura 29: Interpretazione grafica di E_1

Di seguito si riportano i vincoli per il TSP:

1. **Vincolo sulle righe**: Ogni città deve essere visitata una sola volta. Ovvero le righe della matrice V devono avere soltanto un elemento impostato ad 1, il resto a 0;

   $$E_2 = \frac{A}{2} \sum_X \sum_i \sum_{j \neq i} V_{X,i} V_{X,j}$$

   che vale 0 (il minimo) se ogni città è visitata al massimo una volta.

2. **Vincolo sulle colonne**: Ogni fermata deve contenere una città, ovvero ogni colonna della matrice V abbia un elemento impostato ad 1 ed il resto a 0;

   $$E_3 = \frac{B}{2} \sum_i \sum_X \sum_{Y \neq X} V_{X,i} V_{Y,i}$$

   che vale 0 se ogni step del tour contiene al massimo una città.

3. **Vincolo sulle entrate**: in totale devono essere attraversate n città.

   $$E_4 = \frac{C}{2} \left(\sum_X \sum_i V_{X,i} - n \right)^2$$

   che vale 0 se il numero di città attraversate è esattamente n.

Ci si trova dunque con quattro funzioni energia da minimizzare, tuttavia per poter utilizzare una rete di Hopfield è necessaria un'unica funzione. Per questo motivo si esprime la funzione costo totale come combinazione lineare delle funzioni energia E_i:

$$E = E_1 + E_2 + E_3 + E_4 \tag{22}$$

dove il peso da attribuire a ciascun termine è determinato dalle costanti positive A, B, C, D. La funzione energia risultante dovrà essere della forma tipica di una rete di Hopfield ovvero nella forma dell'Equazione (15):

$$E = -\frac{1}{2} \sum_{X,Y} \sum_{i,j} T_{X_i,Y_j} V_{X_i} V_{Y_j} - \sum_{X_i} I_{X_i} V_{X_i}$$

Per ricondursi alla funzione energia tipica di Hopfield si utilizza un "trucco": si introduce il seguente termine:

$$\delta_{i,j} = \begin{cases} 1, & \text{se } i = j \\ 0, & \text{se } i \neq j \end{cases}$$

Le funzioni energia si possono ora riscrivere nel seguente modo:

$$\begin{aligned} E_1 &= \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{X,Y} V_{X,i} (V_{Y,i+1} + V_{Y,i-1}) \\ &= \frac{D}{2} \sum_{X,Y} \sum_{i,j} d_{X,Y} V_{X,i} V_{Y,j} \cdot (\delta_{j,i-1} + \delta_{j,i+1}) \end{aligned}$$

$$\begin{aligned} E_2 &= \frac{A}{2} \sum_X \sum_i \sum_{j \neq i} V_{X,i} V_{X,j} \\ &= \frac{A}{2} \sum_{X,Y} \sum_{i,j} V_{X,i} V_{Y,j} \delta_{X,Y} (1 - \delta_{i,j}) \end{aligned}$$

$$\begin{aligned} E_3 &= \frac{B}{2} \sum_i \sum_X \sum_{Y \neq X} V_{X,i} V_{Y,i} \\ &= \frac{B}{2} \sum_{X,Y} \sum_{i,j} V_{X,i} V_{Y,j} \delta_{i,j} (1 - \delta_{X,Y}) \end{aligned}$$

$$\begin{aligned} E_4 &= \frac{C}{2} \left(\sum_X \sum_i V_{X,i} - n \right)^2 \\ &= \frac{C}{2} \left(\left(\sum_X \sum_i V_{X,i} \right)^2 - 2n \cdot \sum_X \sum_i V_{X,i} + n^2 \right) \end{aligned}$$

dove

$$\begin{aligned} \left(\sum_X \sum_i V_{X,i} \right)^2 &= \left(\sum_X \sum_i V_{X,i} \right) \cdot \left(\sum_X \sum_i V_{X,i} \right) \\ &= \sum_{X,Y} \sum_{i,j} V_{X,i} V_{Y,j} \end{aligned}$$

In E_4 il termine costante n^2 si può tralasciare, in quanto non modifica il punto in cui si ottiene il minimo, ma il valore della funzione E_4 in corrispondenza del minimo. Mettendo il tutto assieme si ottiene la **funzione di energia totale**:

$$\begin{aligned} E = & \frac{A}{2} \sum_{X,Y} \sum_{i,j} V_{X,i} V_{Y,j} \delta_{X,Y} (1 - \delta_{i,j}) \\ & + \frac{B}{2} \sum_{X,Y} \sum_{i,j} V_{X,i} V_{Y,j} \delta_{i,j} (1 - \delta_{X,Y}) \\ & + \frac{C}{2} \left(\sum_{X,Y} \sum_{i,j} V_{X,i} V_{Y,j} - \sum_{X,i} V_{X,i} \right) \\ & + \frac{D}{2} \sum_{X,Y} \sum_{i,j} d_{X,Y} V_{X,i} V_{Y,j} \cdot (\delta_{j,i-1} + \delta_{j,i+1}) \\ & = -\frac{1}{2} \sum_{X,Y} \sum_{i,j} T_{X_i,Y_j} V_{X_i} V_{Y_j} - \sum_{X_i} I_{X_i} V_{X_i} \end{aligned}$$

Dove $T_{Xi,Yj}$ sono i pesi che la rete deve scoprire:

$$\begin{aligned} T_{Xi,Yj} = & -A\delta_{XY}(1 - \delta_{ij}) && \text{(peso inibitorio in ogni riga)} \\ & - B\delta_{ij}(1 - \delta_{XY}) && \text{(peso inibitorio in ogni colonna)} \\ & - C && \text{(Inibizione globale)} \\ & - Dd_{XY}(\delta_{j,i-1} + \delta_{j,i+1}) && \text{(Termine dei dati)} \end{aligned}$$

e il vettore di corrente esterna I ha come Xi-esima componente:

$$I_{Xi} = C_n \qquad \text{(Corrente esterna eccitatoria)}$$

6.1.3 *Un'altra formulazione*

La determinazione dei parametri A, B, C, D è particolarmente difficile. Pertanto un altro modo per esprimere i vincoli del TSP (cioè la matrice di permutazione V) è il seguente:

$$E_2 = \frac{A}{2} \sum_X \left(\sum_i V_{X,i} - 1 \right)^2 \qquad \text{(Vincolo sulle righe)}$$

$$E_3 = \frac{B}{2} \sum_i \left(\sum_X V_{X,i} - 1 \right)^2 \qquad \text{(Vincolo sulle colonne)}$$

Quindi la funzione energia diventa:

$$E = \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{X,Y} V_{X,i} (V_{Y,i+1} + V_{Y,i-1}) + E_2 + E_3$$

Dal momento che E_4 non è usato abbiamo tre parametri anziché quattro.

6.1.4 *Problema delle n regine*

Una famosa variante del TSP è il *problema delle n regine*. Si immagini una scacchiera $n \times n$ e n regine; una regina può muoversi lungo le righe, le colonne e le diagonali. Il problema consiste nel posizionare questi n pezzi in modo tale che nessuno di essi possa attaccarsi l'uno contro l'altro.

Si può costruire una rete di Hopfield $n \times n$ in cui il neurone (i, j) è attivo se e solo se una regina occupa la posizione (i, j). Ci sono inoltre quattro vincoli:

1. solo una regina in ciascuna riga;
2. solo una regina in ciascuna colonna;
3. solo una regina in ciascuna diagonale;
4. solo n regine sulla scacchiera.

Il problema è analogo al TSP con l'aggiunta del vincolo sulle diagonali. La matrice dei pesi è quindi data da:

$$
\begin{aligned}
-T_{ij,kl} = & -A\delta_{ik}(1-\delta_{ij}) && \text{(peso inibitorio in ogni riga)} \\
& + B\delta_{jl}(1-\delta_{ik}) && \text{(peso inibitorio in ogni colonna)} \\
& + C && \text{(Inibizione globale)} \\
& + D(\delta_{i+j,k-l} + \delta_{i-j,k-l})(1-\delta_{ik}) && \text{(peso inibitorio sulle diagonale)}
\end{aligned}
$$

6.2 PROBLEMA DELLA CRICCA MASSIMA

Sia $G = (V, E)$ un grafo non orientato con V l'insieme dei vertici ed E quello degli archi, tale per cui $V = 1, \ldots, n$ e $E \subseteq V \times V$. Si definisce cricca (o clique) $C \subseteq V$, un sottoinsieme di vertici di G mutuamente adiacenti (che formano un grafo completo), ovvero tali che $\forall i, j \in C$ con $i \neq j, (i, j) \in E$.

Definizione 2 (Clique massimale)**.** *Si definisce* ***clique massimale*** *di G una clique di G che non è contenuta in nessun'altra clique di G.*

Definizione 3 (Clique massima)**.** *Si definisce* ***clique massima*** *di G una clique massimale di G di cardinalità massima.*

Il problema consiste nel trovare una cricca massimale (MCP). Ad esempio si consideri il seguente grafo:

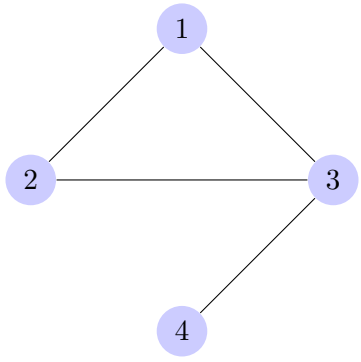


Figura 30: Un grafo non orientato.

Alcune clique nel grafo sono:

$C_1 = \{1, 2\}$ (C_1 non è massimale perché $C_1 \subseteq C_2$)

$C_2 = \{1, 2, 3\}$ (Massimale e massima)

$C_3 = \{3, 4\}$ (Massimale)

Trovare una clique massimale è un problema facile, mentre trovare quella massima è NP-difficile, così come trovare la dimensione di tale clique. In questa sezione si affronta il problema sotto quest'ultimo punto di vista.

6.2.1 *Formulazione continua di MCP*

Per affrontare il problema con le reti neurali è necessario trasformare MCP da problema discreto a problema continuo. Nell'esempio del TSP con il modello di Hopfield, non è detto che ci sia il percorso inverso (potremo ottenere ad esempio una matrice che non ha significato); in questo nuovo problema MCP, la bidirezionalità è d'obbligo.

Si utilizza quindi un nuovo approccio, ma prima sono necessarie alcune notazioni:

- Se $C \subseteq V$, x^C indica il **vettore caratteristico** definito come:

$$x_i^C = \begin{cases} \dfrac{1}{|C|}, & \text{se } i \in C \\ 0, & \text{altrimenti} \end{cases}$$

  dove $|C|$ indica la cardinalità dell'insieme C e $i \in \{1, \ldots, |V|\}$

- S_n è il **simplesso standard** in $\mathbb{R}^n$:

$$S_n = \left\{ x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = 1 \text{ e } x_i \geq 0, \forall i \right\}$$

  Per qualunque vettore caratteristico vale la relazione $x^C \in S_n$ con $n \geq |C|$

- $A = (a_{ij})$ è la matrice di adiacenza di G:

$$a_{ij} = \begin{cases} 1, & \text{se } (i,j) \in E \\ 0, & \text{altrimenti} \end{cases}$$

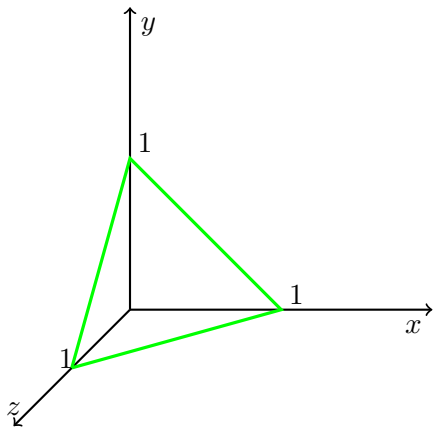


Figura 31: Rappresentazione grafica del simplesso standard

Si consideri la seguente funzione quadratica continua in n variabili:

$$f_G(x) = x^T A x = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j \qquad \text{(Lagrangiano del grafo)}$$

$$\text{oppure } f_G(\bar{x}) = \sum_{(i,j)\in E} x_i x_j$$

dove x^T è il vettore trasposto e A è la matrice di adiacenza. Ad esempio se si considera il grafo in Figura 30 allora:

$$f_G(x) = x_1x_2 + x_1x_3 + x_2x_3 + x_3x_4$$

È stata dunque definita una funzione continua che rappresenta il problema: il Lagrangiano del grafo. A questo punto l'approccio tipico per risolvere MCP è quello di costruire un sistema dinamico che converga ai massimi di tale funzione. Questi punti corrisponderanno alle soluzioni nello spazio discreto del problema originale. A tale scopo si introduce il teorema di Motzkin-Strauss.

Teorema 5 (Teorema di Motzkin-Strauss)**.** *Sia x^* un massimo globale di f_G in $x \in S_n$ allora la cardinalità della clique massima è legata a $f_G(x)$ dalla seguente formula:*

$$\omega(G) = \frac{1}{1 - f(x^*)}$$

Inoltre un sottoinsieme di vertici C è una clique massima se e solo se il suo vettore caratteristico $x^C \in S_n$ è un massimo globale per f_G in S_n.

Il teorema di Motzkin-Strauss fornisce una connessione tra la cardinalità della clique massima $\omega(G)$ di un grafo G con n vertici e il massimo del suo Lagrangiano definito nel simplesso standard di $\mathbb{R}^n$. In particolare è stato mostrato che una clique C è massima se e solo se il suo vettore caratteristico x^C è un massimizzatore globale della funzione f_G su S_n.

Tuttavia non tutti i massimizzatori di f_G sono nella forma di vettori caratteristici e pertanto non possono essere utilizzati direttamente per ricavare informazioni sulle clique massime. Tali massimizzatori sono definiti *soluzioni spurie*. Ad esempio, si consideri il seguente grafo:

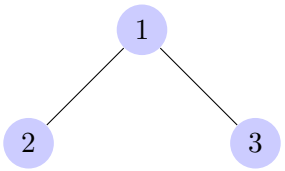


Figura 32: Grafo esempio.

Il grafo presenta due massimi globali:

$$C_1 = \{1, 2\} \qquad x^{C_1} = (1/2, 1/2, 0)$$
$$C_2 = \{1, 3\} \qquad x^{C_2} = (1/2, 0, 1/2)$$

Tuttavia come si evince dal seguente grafico, sono massimi globali anche tutti i punti del segmento $x'x''$ ovvero tutti i punti in $(1/2, \alpha/2, (1-\alpha)/2)\, \forall \alpha \in [0, 1]$.

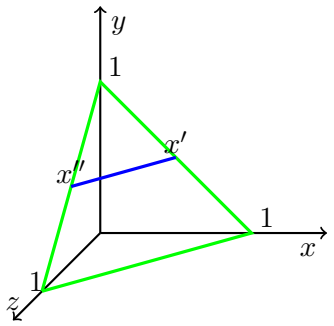


Figura 33: Soluzioni spurie in MCP

Dunque x' e x'' **non** sono vettori caratteristici e non possono essere utilizzati per la soluzione del MCP.

Quello che è stato fatto finora è prendere il problema P di clique massima nel discreto e trasformarlo in un problema P' di ottimizzazione quadratica nel continuo. A questo punto è necessario mappare la soluzione del problema P' in una soluzione del problema originale P. Questo è possibile se la soluzione ottenuta in P' è un vettore caratteristico.

Il problema delle soluzioni spurie è stato risolto da Immanuel Bomze (1995) proponendo una versione regolarizzata di $f_G(x)$. La soluzione consiste nel sommare $1/2$ alla diagonale principale della matrice di adiacenza A.

$$A' = A + \frac{1}{2}I$$

Da cui si ottiene:

$$\hat{f}_G(x) = x^T A' x = x^T \left(A + \frac{1}{2}I \right) x$$

Dove I è la matrice identità, ovvero una matrice quadrata dello stesso lato di A con tutti gli elementi in diagonale principale ad 1 e tutti gli elementi al di fuori di essa a 0.

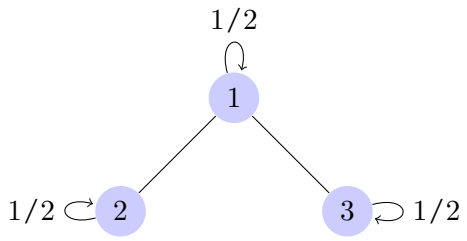


Figura 34: Soluzione problema delle soluzioni spurie (Bomze).

Riassumendo il problema della cricca massima dato un grafo $G = (V, E)$ è definito nel seguente modo:

$$\max(\hat{f}_G(x)) \text{ tale che } x \in S_n$$

Sia $C \subseteq V$ e sia x^C il suo vettore caratteristico allora:

1. C è una clique massima di G sse x^C è un massimo globale di $\hat{f} \in S_n$;
2. C è una clique massimale di G sse x^C è un massimo locale di $\hat{f} \in S_n$;
3. ogni massimo locale è un vettore caratteristico ed è locale stretto.

Il risultato precedente garantisce che *tutti* i massimizzatori di $\hat{f}$ su S_n sono stretti, e sono sono vettori caratteristici di clique massima/massimale in un grafo. In particolare esiste una corrispondenza uno-a-uno da una parte tra le clique massimali e massimizzatori locali di $\hat{f}$ su S_n, dall'altra tra clique massima e massimizzatore globale.

Parte II

TEORIA DEI GIOCHI

7

TEORIA DEI GIOCHI

La teoria dei giochi analizza le decisioni individuali in situazioni in cui vi sono iterazioni tra agenti[1] diversi, tali per cui le decisioni di un agente possono influire sui risultati conseguibili da parte dei rivali.

Queste situazioni conflittuali si possono vedere come dei *giochi*, caratterizzati da tre componenti:

1. **giocatori**: gli agenti che partecipano al conflitto;
2. **strategie**: le possibili azioni o decisioni che ciascun agente può prendere;
3. **utilità o payoff**: il beneficio che un giocatore ottiene nel adottare una certa strategia in un dato contesto di gioco[2].

7.1 GIOCHI FINITI IN FORMA NORMALE

Un gioco finito in forma normale consiste in un insieme di *giocatori*

$$I = \{1, \dots, n\} (n \geq 2) \tag{23}$$

ciascuno dei quali ha associato un insieme di strategie pure

$$S_i = \{1, \dots, m_i\} \qquad m_i \geq 2 \tag{24}$$

Una **strategia pura** fornisce una definizione completa del modo in cui un giocatore gioca una partita. In particolare, essa determina quale mossa farà il giocatore in qualsiasi situazione che potrebbe affrontare. L'insieme delle strategie pure s_i giocate dagli individui in un dato istante forma un *profilo strategico* puro.

$$\mathbf{s} = (s_1, s_2, \dots, s_n) \tag{25}$$

Nota: i simboli in grassetto indicano un vettore.

1 un agente è un entità che partecipa al processo decisionale.
2 con contesto di gioco si intende l'insieme di strategie giocate da ciascun agente.

L'insieme dei profili strategici puri forma lo *spazio delle strategie* pure.

$$S = S_1 \times \cdots \times S_n \tag{26}$$

In seguito alla giocata di un profilo strategico $\mathbf{s} \in S$ ciascun individuo $i \in I$ ottiene un *payoff*. Con il termine payoff si intende una quantificazione del beneficio che un individuo ha in seguito ad una giocata; in economia il payoff può denotare il profitto di un impresa o l'utilità di un consumatore. Ovviamente lo scopo di ogni giocatore è di massimizzare il payoff.

Si rappresenta il payoff dell'i-esimo giocatore come una funzione $\pi_i : S \to \mathbb{R}$. La *funzione di payoff combinata* $\pi : S \to \mathbb{R}_n$ assegna a ogni profilo strategico puro $\mathbf{s}$ il vettore di payoffs $\pi(\mathbf{s}) = (\pi_1(\mathbf{s}), \pi_2(\mathbf{s}), \ldots, \pi_n(\mathbf{s}))$.

Nel caso speciale di giochi a due giocatori, è utile rappresentare le due funzioni di payoff in forma matriciale assegnando al primo giocatore la matrice di payoff

$$A = (a_{hk}) \qquad \text{dove } a_{hk} = \pi_1(h, k), \forall h, k \in S_1$$

e al secondo giocatore la matrice

$$B = (b_{hk}) \qquad \text{dove } b_{hk} = \pi_2(h, k), \forall h, k \in S_2$$

Ogni riga di ciascuna delle due matrici corrisponde quindi ad una strategia pura giocata dal giocatore 1, mentre ogni colonna corrisponde ad una strategia pura giocata dal giocatore 2.

È possibile a questo punto rappresentare un gioco in forma normale come una tripletta $G = (I, S, \pi)$ dove I è l'insieme di individui, S è lo spazio delle strategie pure e π è la funzione di payoff combinata.

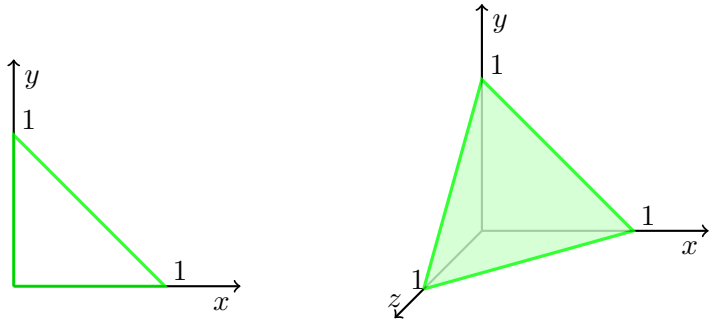
Si suppone ora che ciascun giocatore i decida la strategia pura da usare in base ad una distribuzione di probabilità sull'insieme di strategie pure S_i. È possibile rappresentare questa distribuzione con un vettore m_i-dimensionale $\mathbf{x}_i$ nello spazio $\mathbb{R}^{m_i}$. La componente x_{ih} rappresenterà la probabilità che la strategia pura h sia usata dal giocatore i.

Si parla in questo caso di **strategia mista** per un giocatore; una distribuzione di probabilità sull'insieme delle strategie pure che costui ha a disposizione.

Trattandosi di distribuzioni di probabilità si ottiene che ciascuna componente x_{ih} con $h = 1, 2, ...$ è positiva e la loro somma è uguale a 1, quindi il vettore appartiene al simplesso unitario Δ_i m_i-dimensionale.

$$\Delta_i = \left\{ x_i \in \mathbb{R}_+^{m_i} : \forall h = 1, \ldots, m_i : x_{ih} > 0 \text{ and } \sum_{h=1}^{m_i} x_{ih} = 1 \right\} \tag{27}$$

$$\Delta_i = \{ x_i \in \mathbb{R}_+^{m_i} : \mathbf{e}'\mathbf{x_i} = 1 \} \qquad \text{(Notazione vettoriale)}$$



(a) (b)

Figura 35: A sinistra il simplesso a due dimensioni. A destra il simplesso a tre dimensioni. Da notare che le strategie pure corrispondono ai lati del simplesso standard.

L'insieme delle componenti di $\mathbf{x_i} \in \Delta_i$ a cui sono state assegnate probabilità non nulle forma il *supporto* di $\mathbf{x_i}$

$$\sigma(\mathbf{x_i}) = \{ h \in S_i : x_{ih} > 0 \}$$

Una strategia pura h può essere vista come una strategia mista estrema x_i che assegna cioè probabilità uno alla componente x_{ih} e zero a tutte le altre, ovvero $\mathbf{x_i} = \mathbf{e}_i^h$ dove $\mathbf{e}_i^h$ è un vettore unità; un vettore con tutte le componenti uguali a 0 eccetto quella in posizione h che è uguale a 1. Ogni strategia nel simplesso unitario può essere espressa come combinazione lineare di strategie miste estreme, ovvero di vettori unità $\mathbf{e}_i^h$.

$$\mathbf{x_i} = \sum_{i=0}^{m_i} x_{ih} \mathbf{e}_i^h$$

Si definisce *profilo strategico misto* il vettore $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ dove ogni i-esima componente è una strategia mista del giocatore $i \in I$; *spazio delle strategie miste* $\Theta = \Delta_1 \times \Delta_2 \times \cdots \times \Delta_n$, il prodotto cartesiano dei profili strategici misti.

Nell'approccio standard, si assume che le scelte dei giocatori siano indipendenti. Pertanto, per la regola del prodotto, la probabilità che un profilo strategico puro $\mathbf{s} = (s_1, \ldots, s_n)$ sia usato, quando viene giocato un profilo strategico misto $\mathbf{x}$ è data da:

$$\mathbf{x}(\mathbf{s}) = \prod_{i=1}^{n} x_{is_i} \tag{28}$$

Quindi il valore atteso del payoff per il giocatore i in seguito alla giocata di un profilo strategico $\mathbf{x}$ è dato da:

$$u_i(\mathbf{x}) = \sum_{\mathbf{s} \in S} \mathbf{x}(\mathbf{s}) \pi_i(\mathbf{s}) \tag{29}$$

Nel caso speciale di giochi a due giocatori, è possibile, come già fatto notare per i giochi a strategie pure, rappresentare la funzione di payoff con una coppia di matrici (A, B) dove $A(B)$ è la matrice di payoff del giocatore $1(2)$. Quindi per un qualunque profilo strategico misto $(x_1, x_2) \in \Theta$ si ottiene:

$$u_1(x_1, x_2) = \sum_{h=1}^{m_1} \sum_{k=1}^{m_2} x_{1h} a_{hk} x_{2k} = \mathbf{x}_1^T A \mathbf{x}_2$$
$$u_2(x_1, x_2) = \sum_{h=1}^{m_1} \sum_{k=1}^{m_2} x_{1h} b_{hk} x_{2k} = \mathbf{x}_1^T B \mathbf{x}_2$$

7.2 GIOCHI SIMMETRICI A DUE GIOCATORI

Si definisce un gioco $G = (I, \Theta, u)$ simmetrico a due giocatori se $I = \{1, 2\}$, $\Delta_1 = \Delta_2$ e $u_2(\mathbf{x}_1, \mathbf{x}_2) = u_1(\mathbf{x}_2, \mathbf{x}_1) \in \Theta$. Dal momento che $u_2(\mathbf{x}_1, \mathbf{x}_2) = u_1(\mathbf{x}_2, \mathbf{x}_1)$ si considera d'ora in avanti un unica funzione di pay-off $u(\mathbf{x}_1, \mathbf{x}_2)$ dove $(\mathbf{x}_1, \mathbf{x}_2) \in \Theta$ e allo stesso modo, siccome $\Delta_1 = \Delta_2$ si considera un unico simplesso Δ valido per entrambi i giocatori. Si dice che una strategia $\mathbf{x} \in \Delta$ è:

- *debolmente dominata* da una strategia $\mathbf{y} \in \Delta$ se

  $$u(\mathbf{y}, \mathbf{z}) \geq u(\mathbf{x}, \mathbf{z}), \forall \mathbf{z} \in \Delta$$

- *fortemente dominata* (o strettamente dominata) da una strategia $\mathbf{y} \in \Delta$ se

  $$u(\mathbf{y}, \mathbf{z}) > u(\mathbf{x}, \mathbf{z}), \forall \mathbf{z} \in \Delta$$

- *non dominata* se non esiste alcuna strategia $\mathbf{y} \in \Delta$ che la domina debolmente.

Si analizza ora un altro concetto importante, quello di **best reply**. Data una strategia $\mathbf{y} \in \Delta$ possiamo creare un insieme di strategie che se giocate contro $\mathbf{y}$ ottengono un payoff maggiore o uguale. Formalmente si definisce l'insieme di best replies a $\mathbf{y} \in \Delta$ nel seguente modo:

$$\beta^*(\mathbf{y}) = \{\mathbf{x} \in \Delta : u(\mathbf{x}, \mathbf{y}) \geq u(\mathbf{z}, \mathbf{y}), \forall \mathbf{z} \in \Delta\} \tag{30}$$

Non è detto che la miglior risposta sia unica. Infatti, tranne casi estremi nei quali l'unica miglior risposta è una strategia pura, il numero di best replies è infinito. Ad esempio si consideri la seguente matrice di payoff:

$$A = \begin{bmatrix} 1 & 6 & 0 \\ 2 & 6 & 4 \\ 5 & 4 & 3 \end{bmatrix} \implies \begin{cases} \beta^*(1) = \{3\} \\ \beta^*(2) = \{1, 2\} \\ \beta^*(3) = \{2\} \end{cases}$$

Considerando che le colonne indicano le strategie del giocatore 2, si fissa la colonna relativa alla strategia j; si cerca in tale colonna l'elemento massimo. La best reply del giocatore 1 sarà dunque la strategia i (riga) alla quale corrisponde l'elemento massimo.

Assumendo di poter conoscere le strategie degli altri giocatori è possibile giungere ad una situazione in cui ogni giocatore ottiene il massimo profitto e pertanto nessuno di essi ha motivo di cambiare strategia in quanto porterebbe a una perdita di beneficio. Questa situazione è nota come **equilibrio di Nash**.

Teorema 6 (Equilibrio di Nash nei giochi simmetrici a due giocatori)**.** *Un profilo strategico* $(\mathbf{x}, \mathbf{y}) \in \Theta$ *è un equilibrio di Nash sse:*

$$\mathbf{x} \in \beta^*(\mathbf{y}) \ e \ \mathbf{y} \in \beta^*(\mathbf{x})$$

Un profilo strategico $(\mathbf{x}, \mathbf{y}) \in \Theta$ *è un equilibrio di Nash* ***stretto*** *sse*

$$\beta^*(\mathbf{y}) = \{\mathbf{x}\} \ e \ \beta^*(\mathbf{x}) = \{\mathbf{y}\}.$$

L'insieme degli equilibri di Nash si rappresenta con Θ^{NE}*.*

È stato dimostrato (Nash 1951) che ogni gioco finito in forma normale ammette un equilibrio di Nash.

7.3 ALCUNI ESEMPI

Un primo esempio tipico della teoria dei giochi è un gioco sulla coordinazione: due giocatori devono coordinare le proprie strategie in modo tale da massimizzare il payoff. La seguente matrice rappresenta il payoff che i due giocatori ottengono rispettivamente adottando la strategia A oppure la strategia B.

		Giocatore 2	
		Strategia A	Strategia B
Giocatore 1	Strategia A	4,4	1,3
	Strategia B	3,1	2,2

Tabella 2: Matrice di payoff nel gioco di coordinazione.

Il gioco ammette due equilibri di Nash: nel primo i due giocatori si sincronizzano adottando entrambi la strategia A che fornisce il payoff massimo; nel secondo, invece, i giocatori adottano entrambi la strategia B. In quest'ultimo caso, sebbene i giocatori non ottengono il beneficio massimo non hanno alcun guadagno nel cambiare la propria strategia (da 2 a 1).

Un secondo esempio è rappresentato da un guidatore che guida lungo una strada e vede sopraggiungere un altro veicolo nella direzione opposta. A questo punto i due guidatori dovranno scegliere se sterzare a destra o a sinistra per evitare l'incidente.

È possibile rappresentare questo "gioco" con la seguente matrice di payoff dove il beneficio 0 indica il verificarsi dell'incidente mentre 10 nessun incidente.

		Guidatore 2	
		Sinistra	Destra
Guidatore 1	Sinistra	10,10	0,0
	Destra	0,0	10,10

Tabella 3: Matrice di payoff nel gioco della guida.

In questo caso esistono due equilibri di Nash basati su strategie pure: ovvero quando entrambi girano a destra o a sinistra.

Se si ammettono anche le strategie miste (dove una strategia pura è scelta a caso in base ad una distribuzione di probabilità) allora il gioco ammette **tre** equilibri di Nash. Infatit, assegnando le seguenti distribuzioni di probabilità si ottiene lo stesso comportamento dei due equilibri di Nash basati su strategie pure:

- (0%, 100%) per il giocatore 1 e (0%, 100%) per il giocatore 2;
- (100%, 0%) per il giocatore 1 e (100%, 0%) per il giocatore 2;

Infine, il terzo equilibrio di Nash basato su strategie miste si ottiene assegnando a ciascun giocatore le probabilità (50%, 50%).

L'ultimo esempio è meglio conosciuto come il *"dilemma del prigioniero"*. Il dilemma può essere descritto come segue. Due criminali vengono accusati di aver commesso un reato. Gli investigatori li arrestano entrambi e li chiudono in due celle diverse, impedendo loro di comunicare. Ad ognuno di loro vengono date due scelte: confessare l'accaduto e tradire il compagno, oppure non confessare.

Se entrambi i prigionieri non confessano otterranno una condanna più pesante rispetto al caso in cui uno dei due confessi. Il payoff è rappresentato dalla seguente matrice dove un payoff maggiore corrisponde a una sentenza più leggera.

		Prigioniero 2	
		Confessa	Non Confessa
Prigioniero 1	Confessa	4,4	0,5
	Non Confessa	5,0	3,3

Tabella 4: Matrice di payoff nel dilemma del prigioniero.

In questo gioco la ricompensa massima per ogni giocatore (in questo caso 5) si ottiene quando i giocatori effettuano scelte differenti. Il prigioniero può migliorare la sua situazione decidendo di cambiare idea e confessare oppure non confessare, sapendo che la migliore decisione dell'avversario è di non confessare (il payoff è infatti (5,3)). Il dilemma del prigioniero ammette dunque un equilibrio di Nash: entrambi i prigionieri decidono di non confessare.

Questo esempio è un interessante caso di studio in quanto l'equilibrio di Nash trovato è globalmente inferiore rispetto a "(confessa, confessa)". Tuttavia ogni giocatore può migliorare la sua situazione rompendo la mutua cooperazione, a prescindere da come decida di agire l'altro giocatore.

8

DINAMICHE DI REPLICAZIONE

Le dinamiche di replicazione sono una classe di sistemi dinamici studiati nel contesto della teoria dei giochi evoluzionistici, una disciplina nata da J. Maynard Smith con l'obiettivo di modellare l'evoluzione del comportamento animale utilizzando i principi e i mezzi della teoria dei giochi.

Si consideri una *popolazione grande*, idealmente infinita, appartenente alla stessa specie che compete per un particolare insieme di *risorse limitate*, come cibo, territorio, etc... Si suppone inoltre che ciascun individuo sia pre-programmato ad una particolare strategia pura. È possibile modellare questo tipo di conflitto come un gioco in cui iterativamente vengono estratti a caso due giocatori dalla popolazione e fatti competere.

La vittoria del gioco contribuisce alla sopravvivenza della specie, in quanto il payoff in questo contesto rappresenta il successo riproduttivo. La riproduzione avviene in modo asessuato per cui, a meno di mutazioni, ogni nuovo nascituro sarà un clone del genitore, ovvero, nel nostro caso, programmato alla sua stessa strategia pura. Nell'evolversi della dinamica avremo che, per il **principio di selezione naturale**, gli individui più forti, che hanno cioè adottato una strategia migliore, tenderanno a dominare gli individui più deboli, con la conseguente estinzione di questi ultimi.

8.1 TEORIA DEI GIOCHI EVOLUZIONISTICI

Si consideri una grande popolazione di individui programmati a strategie pure $i \in \{1, \ldots, n\}$.

Sia $\mathbf{x}(t) \in \Delta$ il vettore che rappresenta lo stato della popolazione al tempo t dove $x_i(t)$ è la percentuale di popolazione programmata alla strategia pura i. Sia $A = a_{ij}$ la matrice $n \times n$ di payoff dove a_{ij} rappresenta il payoff che si ottiene giocando la strategia i contro la strategia j di un altro individuo. Se la popolazione si trova allo stato x il payoff atteso di un individuo che gioca la strategia i è dato da:

$$\pi_i(x) = \sum_{j=1}^{n} a_{ij} x_j = (Ax)_i \tag{31}$$

mentre il payoff medio sull'intera popolazione è:

$$\pi(x) = \sum_{i=1}^{n} x_i \pi_i(x) = (x^T Ax) \tag{32}$$

Nella teoria dei giochi evoluzionistici si fa l'assunzione che la popolazione giochi iterativamente generazione dopo generazione e che l'azione della selezione naturale porti alla sopravvivenza delle strategie più forti, od in questo caso di quelle con payoff maggiore.

Dinamiche di questo tipo possono essere descritte da un insieme di equazioni differenziali rispetto al tempo del tipo:

$$\dot{x}_i = g_i(\mathbf{x}) x_i \qquad \forall i$$

dove $g_i(\mathbf{x})$ indica il fattore di replicazione delle strategie pure i quando la popolazione si trova nello stato $\mathbf{x}$.

Un punto $\mathbf{x}$ è detto **stazionario** se $\dot{x}_i = 0, \forall i$. Un punto stazionario è **asintoticamente stabile** se qualunque traiettoria, che inizia sufficientemente vicino ad $\mathbf{x}$, converge a $\mathbf{x}$ quando $t \to \infty$.

Nel caso in cui il tempo sia discreto, l'equazione di replicazione assume la seguente forma:

$$x_i(t+1) = \frac{x_i(t)\pi_i(t)}{\sum_{j=1}^{n} x_j(t)\pi_j(t)} \tag{33}$$

mentre nel caso in cui il tempo scorra in maniera continua:

$$\frac{d}{dt}x_i(t) = x_i(t)\left(\underbrace{\pi_i(t)}_{\text{payoff strategia pura } i} - \underbrace{\sum_{j=1}^{n} x_j(t)\pi_j(t)}_{\text{payoff medio della popolazione}}\right) \tag{34}$$

I punti stazionari di questa dinamica si ottengono se e solo se tutte le strategie nel supporto di $\mathbf{x}$ ottengono lo stesso payoff.

8.2 EQUILIBRI EVOLUTIONARY STABLE

Si suppone ora che un piccolo gruppo di mutanti (o invasori) appaia in una grande popolazione di individui programmata a giocare la stessa strategia $\mathbf{x} \in \Delta$ (*strategia incombente*).

Si suppone che gli invasori siano programmati a giocare una strategia $\mathbf{y} \in \Delta$ (*strategia mutante*). Sia ϵ la frazione di mutanti con $\epsilon \in (0, 1)$. Coppie di individui sono scelti casualmente a giocare in base ad una distribuzione di probabilità uniforme.

Se un individuo viene scelto allora la probabilità che questo si scontri con un mutante è ϵ mentre la probabilità che l'avversario non lo sia è $1 - \epsilon$.

Il payoff che un individuo ottiene in questa popolazione è lo stesso che otterrebbe se si scontrasse con un individuo programmato a giocare la strategia:

$$\mathbf{w} = \epsilon\mathbf{y} + (1-\epsilon)\mathbf{x} \in \Delta$$

Quindi un individuo della popolazione originale otterrebbe un payoff pari a $u(\mathbf{x}, \mathbf{w})$, mentre un mutante otterrebbe un payoff pari a $u(\mathbf{y}, \mathbf{w})$. Da un punto di vista biologico ci si aspetta che l'evoluzione forzi una selezione contro gli

individui mutanti solo se la strategia mutante ottiene un payoff inferiore rispetto a quella incombente. In termini matematici:

$$\underbrace{u(\mathbf{x},\mathbf{w})}_{\text{incombente}} > \underbrace{u(\mathbf{y},\mathbf{w})}_{\text{mutante}} \tag{35}$$

In questo contesto si introduce il seguente teorema:

Teorema 7 (Evolutionary Stable Strategy)**.** *Una strategia* $\mathbf{x} \in \Delta$ *è detta* evolutionary stable *(ESS) se per ogni* $\mathbf{y} \in \Delta - \{\mathbf{x}\}$ *esiste un* $\delta \in (0,1)$*, tale per cui per ogni* $\epsilon \in (0,\delta)$ *la disuguaglianza* (35) *è vera. Si definisce* Δ^{ESS} *l'insieme delle strategie ESS. Formalmente è possibile definire l'insieme delle strategie ESS come:*

$$\Delta^{ESS} = \left\{\mathbf{x} \in \Delta^{NE} : u(\mathbf{y},\mathbf{y}) < u(\mathbf{x},\mathbf{y}), \forall \mathbf{y} \in \beta^*(\mathbf{x}), \mathbf{y} \neq \mathbf{x}\right\}$$

O in modo del tutto equivalente $x \in \Delta^{ESS}$ *sse*

1. $u(\mathbf{y},\mathbf{x}) \leq u(\mathbf{x},\mathbf{x}) \quad \forall y \in \Delta$ *(Equilibrio di Nash);*
2. $u(\mathbf{y},\mathbf{x}) = u(\mathbf{x},\mathbf{x}) \implies u(\mathbf{y},\mathbf{y}) < u(\mathbf{x},\mathbf{y}) \quad \forall \mathbf{y} \in \Delta - \{\mathbf{x}\}$ *(Condizione di stabilità).*

Ciò significa che una strategia evolutionary stable non può essere invasa da un altra strategia. Si derivano ora alcuni importanti risultati riguardanti gli stati asintoticamente stabili nelle dinamiche di replicazione.

Proposizione 1. *Se* $\mathbf{x} \in \Delta$ *è asintoticamente stabile, allora* $\mathbf{x} \in \Delta^{NE}$

Questo risultato garantisce che se con le dinamiche di replicazione si giunge in uno stato $\mathbf{x}$ e questo rimane stabile anche se sottoposto a piccole perturbazioni, allora $\mathbf{x}$ è un equilibrio di Nash.

Tuttavia il fatto che il contrario non valga significa che esistono giochi privi di stati asintoticamente stabili, per i quali quindi è più problematica la ricerca di un equilibrio di Nash.

Proposizione 2. *Se* $\mathbf{x} \in \Delta^{ESS}$ *allora* $\mathbf{x}$ *è asintoticamente stabile.*

Gli equilibri ESS sono quindi particolarmente interessanti perché godono della stabilità asintotica, ma anche in questo caso, non è possibile affermare se esista o meno uno corrispondenza uno ad uno tra equilibri ESS e stati asintoticamente stabili. In altre parole, a differenza degli equilibri di Nash, l'esistenza di equilibri ESS non è garantita.

Si consideri come esempio il gioco "Carta, Forbice, Sasso". La matrice di payoff è data da:

		Giocatore 2		
		Sasso	Forbice	Carta
Giocatore 1	Sasso	0,0	1,-1	-1, 1
	Forbice	-1,1	0,0	1,-1
	Carta	1,-1	-1,1	0,0

Tabella 5: Matrice di payoff del gioco Carta, Forbice, Sasso

Si nota che il gioco Carta, Forbici e Sasso è un gioco a somma zero: in qualunque stato del gioco, la somma delle utilità dei giocatori è zero. Il gioco Carta, Forbici e Sasso è anche finito: l'insieme N dei giocatori ha cardinalità finita, così come gli insiemi di strategie $S_1, \ldots, S_n$.

Il gioco non ammette equilibri di Nash basati su strategie pure. Tuttavia, se si definisce $\mathbf{x} = (1/3, 1/3, 1/3)^T$ come distribuzione di probabilità di una strategia mista, allora $(\mathbf{x}, \mathbf{x})$ è un equilibrio di Nash misto. Infatti è possibile verificare che se il giocatore 2 usa la distribuzione di probabilità $\mathbf{x}$, il giocatore 1 non ha alcun incentivo nel giocare una strategia che sia diversa da $\mathbf{x}$.

È possibile dimostrare che il gioco non ammette equilibri ESS. Si consideri una strategia "mutante" $\mathbf{y} = (1, 0, 0)^T$. Si nota che $u(\mathbf{y}, \mathbf{y}) = 0$ e $u(\mathbf{x}, \mathbf{y}) = 0 + 1 - 1 = 0$ per cui $u(\mathbf{y}, \mathbf{y}) = u(\mathbf{x}, \mathbf{y})$ e quindi $\Delta^{ESS} = \emptyset$ in quanto non vale la seconda condizione nel teorema 7.

8.2.1 *Giochi doppiamente simmetrici ed equilibri ESS*

Un gioco $G = (I, \Theta, u)$ a due giocatori è detto doppiamente simmetrico se oltre ad essere simmetrico $u(\mathbf{x}, \mathbf{y}) = u(\mathbf{y}, \mathbf{x}) \quad \forall (\mathbf{x}, \mathbf{y}) \in \Theta$.

Losrt e Akin (1983) mostrarono che il teorema fondamentale di selezione naturale[1] si applica a tutti i giochi doppiamente simmetrici. Essi mostrarono che, con le dinamiche di replicazione, se $A = A^T$ il fitness (payoff) medio della popolazione $f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x}$ cresce lungo tutti i cammini di soluzione non stazionari; formalmente mostrarono che:

$$\frac{d}{dt} f(\mathbf{x}(t)) \geq 0 \qquad \forall t \geq 0 \qquad \text{(Caso continuo)}$$

$$f(\mathbf{x}(t+1)) \geq f(\mathbf{x}(t)) \qquad \text{(Caso discreto)}$$

con l'uguaglianza sse $\mathbf{x}$ è un punto stazionario.
Come conseguenza di questi risultati si ottiene la seguente caratterizzazione degli stati asintoticamente stabili per i giochi doppiamente simmetrici.

Proposizione 3. *Per un qualunque gioco doppiamente simmetrico le seguenti affermazioni sono equivalenti:*

1. $\mathbf{x} \in \Delta^{ESS}$;
2. $\mathbf{x} \in \Delta$ *è un massimo locale di* $u(\mathbf{x}, \mathbf{x})$ *in* Δ;
3. $\mathbf{x} \in \Delta$ *è asintoticamente stabile nelle dinamiche di replicazione.*

1 Il teorema fondamentale di selezione naturale afferma che se c'è selezione naturale, il payoff medio di una popolazione tende ad aumentare

9

CLUSTERING

Il clustering può essere considerato il più importante problema di **apprendimento non supervisionato** in quanto trova innumerevoli applicazioni in svariati campi del sapere. L'obiettivo che si pone è organizzare dati non classificati in gruppi, i cui membri sono simili per un qualche criterio. Un **cluster** è quindi una collezione di oggetti che sono simili tra di loro, e dissimili dagli oggetti appartenenti ad altri cluster. Formalmente:

Definizione 4 (Problema del clustering)**.** *Dati n oggetti e una matrice di similarità $n \times n$ lo scopo è partizionare gli inputs in gruppi massimalmente omogenei (i.e **clusters**).*

Il criterio di similarità che deve essere fornito per poter fare un clustering dei dati può essere visto come una funzione ϕ che dati due oggetti ritorna la loro similarità. Questa misura è *simmetrica* se per una qualunque coppia di oggetti (a, b) abbiamo che $\phi(a, b) = \phi(b, a)$ altrimenti è *asimmetrica*.

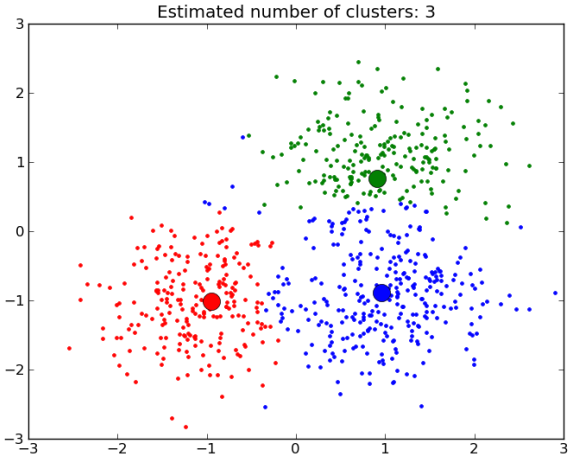


Figura 36: Esempio di clustering

In questo esempio è possibile identificare tre cluster nei quali possono essere suddivisi i dati e il criterio di similarità usato è la *distanza*, quindi due oggetti fanno parte di uno stesso cluster solo se sono sufficientemente vicini tra di loro.

Questo tipo di clustering è detto *distance-based*, ovvero basato sulla distanza. In generale, a seconda del tipo di input il problema del partizionamento si divide in:

1. **feature-based:** gli oggetti sono rappresentati come vettori di caratteristiche. Nella letteratura l'algoritmo più noto è il *k-means*;
2. **pairwise:** le proprietà degli oggetti sono meglio descritte in termini di *similarità/dissimilarità* tra di essi. In questo caso l'input è determinato da una matrice di affinità che indica la similarità tra tutte le coppie di oggetti. Si tratta quindi di un approccio più generale rispetto al primo.

Non sempre è facile ed intuitivo trovare un raggruppamento per i dati, come invece lo era per l'esempio visto, in quanto è difficile stabilire cosa costituisce un "buon clustering" e cosa no. In generale si può affermare che un cluster deve soddisfare i seguenti criteri:

- **criterio interno**: tutti gli oggetti all'*interno* di un cluster devono essere il più possibile simili tra loro;
- **criterio esterno**: tutti gli oggetti all'*esterno* di un cluster devono essre il più possibile dissimili rispetto a quelli contenuti al suo interno.

Nelle sezioni seguenti saranno presentate alcune metodologie di clustering e ci si concentrerà, infine, sull'utilizzo della teoria dei giochi per risolvere il problema.

9.1 K-MEANS

K-Means è probabilmente il più famoso algoritmo di partizionamento feature-based. Dato un insieme di osservazioni $\{x^{(1)}, \ldots, x^{(n)}\}$ dove x_i è un vettore m-dimensionale, lo scopo dell'algortimo è partizionare l'insieme in k insiemi $k \leq n$ il più possibile coesi tra loro (problema del clustering).

L'algoritmo seleziona a caso k prototipi, rappresentanti il proprio gruppo di appartenenza, idealmente il centro del cluster. Formalmente si ha un insieme di vettori:

$$\mu_j, \text{ dove } j = 1, \ldots, k$$

L'obiettivo ora consiste nel raggruppare le osservazioni e reimpostare i vettori μ_j in modo tale che la somma del quadrato delle distanze da un punto al proprio centroide μ_j sia minima. In termini matematici si tratta di minimizzare la seguente funzione:

$$\min_{\{\mu_1,\ldots,\mu_k\}} \sum_{i=1}^{n} \sum_{j=i}^{k} r_{ij} \|x_i - \mu_j\|^2 \tag{36}$$

dove r_{ij} è un operatore binario $r_{ij} \in \{0, 1\}$. In fase di inizializzazione, si scelgono k punti casuali come centri e in seguito si esegue una procedura iterativa per minimizzare (36). Ogni iterazione si compone di **due** fasi: nella prima fase (*Expectation-Step*) si minimizza rispetto a r_{ij} mantenendo μ_j fisso, mentre nella seconda fase (*Maximization-Step*) si minimizza rispetto a μ_j mantenendo r_{ij} fisso. Formalmente, nella fase E, il calcolo di r_{ij} è dato da:

$$r_{ij} = \begin{cases} 1, & \text{se } j = argmin_j \|x_i - \mu_j\|^2 \\ 0, & \text{altrimenti} \end{cases}$$

Intuitivamente si assegna l'osservazione x_i al gruppo con μ_j più vicino.

Nella seconda fase μ_k è dato da:

$$\mu_j = \frac{\sum_i r_{ij} x_i}{\sum_i r_{ij}}$$

Ovvero, si cerca di "spostare" al centro del cluster il prototipo μ_j come la media di tutti i punti x_i assegnati al cluster k. Queste due fasi si ripetono fino a quando non c'è più alcuna variazione negli assegnamenti o quando un numero massimo di iterazioni è raggiunto. Infatti, trattandosi di un problema NP-Difficile non è dato sapere a priori se l'algoritmo converga oppure no.

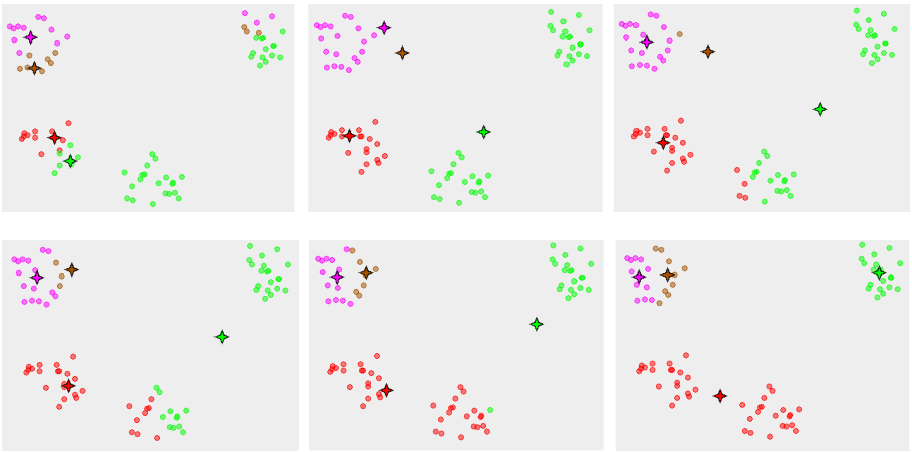


Figura 37: Esempio sulla convergenza (dopo 5 iterazioni) a un minimo locale dell'algoritmo k-means

9.2 NORMALIZED CUT

A differenza del k-means l'algoritmo presentato in questa sezione è basato sulla teoria dei grafi e trova ampio utilizzo nella *segmentazione delle immagini*, un problema analogo al clustering. L'algoritmo *normalized cut* "taglia" l'immagine utilizzando tecniche della **teoria spettrale** dei grafi. Un buon taglio divide pixel che sono dissimili tra loro. Per trovare un buon partizionamento l'algoritmo segue i seguenti passi:

1. Si costruisce un grafo di similarità $G = (V, E, w)$ in cui ogni nodo rappresenta un pixel dell'immagine e il peso su ciascun arco rappresenta una misura di similarità (intensità, colore ecc..) tra ciascun pixel.

2. Si calola il Laplaciano normalizzato definito come:

$$L = D^{-1/2}(D - A)D^{-1/2} \tag{37}$$

dove A è la matrice di adiacenza pesata e D è la matrice dei gradi così definita:

$$d_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

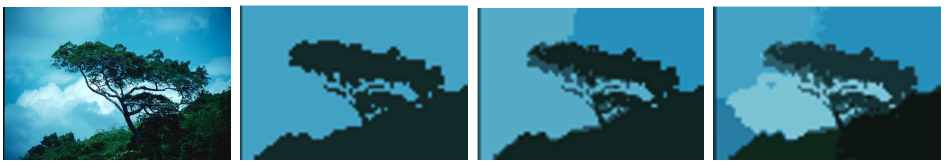
Si calcolano gli autovettori del Laplaciano normalizzato:

$$Lx = \lambda x \tag{39}$$

3. Utilizzando il segno nel secondo autovettore più piccolo (il primo è un vettore nullo) si segmenta l'immagine in due parti.

4. La procedura si ripete ricorsivamente fino a quando si ottiene il numero desideerato di segmenti.

L'algoritmo nella pratica si è dimostrato abbastanza efficace tuttavia il calcolo degli autovettori è un problema computazionalmente costoso: $O(n^3)$ dove n è il numero di pixel. Tuttavia è possibile velocizzare l'algoritmo sfruttando il fatto che la matrice è sparsa (i valori sono quasi tutti pari a zero) ed abbassare la complessità a $O(n\sqrt{n})$



(a) *original* (b) $k = 2$ (c) $k = 3$ (d) $k = 4$

Figura 38: Esempio di segmentazione ottenuta con NCUT.

9.3 INSIEMI DOMINANTI

Il concetto di insieme dominante nasce dallo studio sulla formulazione continua del problema della cricca massima (vedi Sezione 6.2). L'insieme dominante, infatti, non è altro che una clique di un grafo nel caso in cui gli archi siano pesati, ovvero insiemi di vertici che hanno alta omogeneità interna e alta disomogeneità verso l'esterno.

In questo contesto i dati sono, dunque, rappresentati da un grafo non orientato $G = (V, E, w)$ dove $V = \{1, \ldots, n\}$ è l'insieme dei vertici, $E \subseteq V \times V$ l'insieme degli archi e $w : E \to \mathbb{R}_+$ è la funzione dei pesi positiva. I vertici rappresentato i datapoints, gli archi le relazioni tra essi e i pesi rispecchiano la similarità tra coppie di vertici connessi.

Come di consueto, il grafo è rappresentato con la sua matrice di adiacenza pesata A, che è la matrice simmetrica $n \times n$ dove $A = a_{ij}$ è definito nel seguente modo:

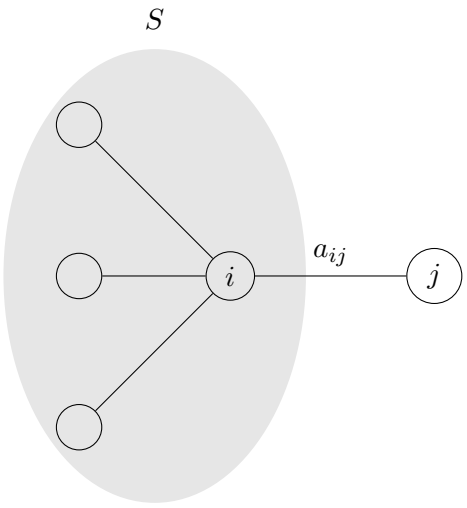
$$a_{ij} = \begin{cases} w(i,j), & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Si consideri $S \subseteq V$ un insieme non vuoto di vertici $i \in S$. Si introduce il concetto di **grado pesato medio** di i rispetto ad S definito come:

$$awdeg_S(i) = \frac{1}{|S|} \sum_{j \in S} a_{ij}$$

E nel caso in cui $j \notin S$:

$$\phi_S(i,j) = a_{ij} - awdeg_S(i)$$



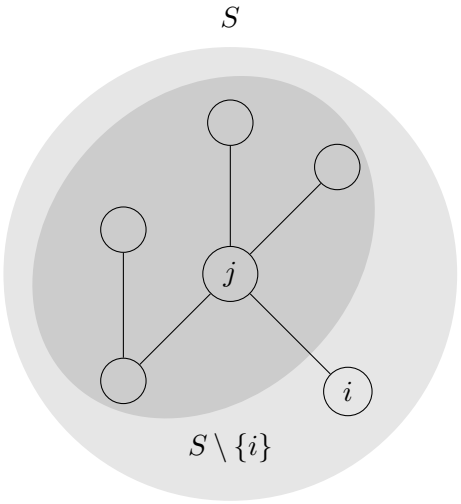
Intuitivamente $\phi_S(i,j)$ misura la similarità tra il nodo j e il nodo i rispetto alla similarità media tra il nodo i e i suoi vicini in S

Il peso di i rispetto ad S è quindi definito come:

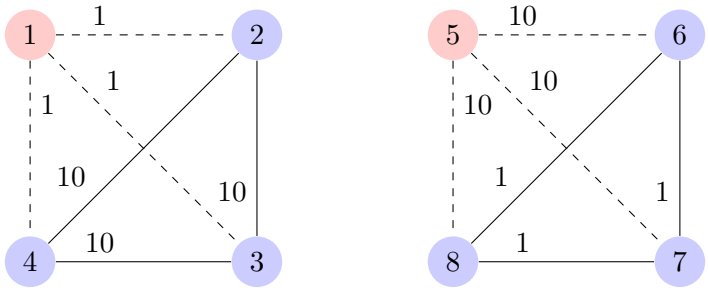
$$w_S(i) = \begin{cases} 1, & \text{if } |S| = 1 \\ \sum_{j \in S} \phi_{S\setminus\{i\}}(i,j) \cdot w_{S\setminus\{i\}}(j) & \text{otherwise} \end{cases}$$

Si tratta di una definizione ricorsiva e permette di assegnare un peso (similarità relativa) ad ogni vertice. E il peso totale di S è:

$$W(S) = \sum_{i \in S} w_S(i)$$



Intuitivamente $w_S(i)$ ci dà la misura di similarità complessiva tra il nodo i e i vertici $S \setminus \{i\}$ rispetto alla similarità complessiva di ciascun vertice in $S \setminus \{i\}$. Si considerino i seguenti grafi.



(a) $W_{\{1,2,3,4\}(1)} < 0$ (b) $W_{\{5,6,7,8\}(5)} > 0$

Figura 39: Rappresentazione della similarità: i nodi $\{2,3,4\}$ sono altamente simili tra loro. Se si tenta di aggiungere il vertice 1 che è dissimile dalla similarità interna dell'insieme $\{2,3,4\}$ allora la similarità complessiva del nuovo insieme $\{1,2,3,4\}$ diminuisce i.e. $W_{\{1,2,3,4\}}(1) < 0$. Nel secondo caso il vertice 5 è altamente similare ai nodi $\{6,7,8\}$ di conseguenza aggiungendolo all'insieme la similarità complessiva aumenta i.e. $W_{\{5,6,7,8\}}(5) > 0$.

È stata dunque definita una misura che descrive cosa comporta (in termini di similarità) l'aggiunta o la rimozione di un nodo. Questo porta alla definizione di *insiemi dominanti*.

Definizione 5 (Insieme Dominante)**.** *Un sottoinsieme di vertici non vuoto $S \subseteq V$ tale che $W(T) > 0$ per ogni insieme non vuoto $T \subseteq S$ è detto* ***dominante*** *se:*

1. $W_S(i) > 0, \forall i \in S$ *(omogeneità interna)*
2. $W_{S\cup\{i\}}(i) < 0, \forall i \notin S$ *(disomogeneità esterna)*

L'insieme dominante è dunque un insieme di vertici massimalmente coesi tra loro e questa definizione corrisponde con quella di cluster.

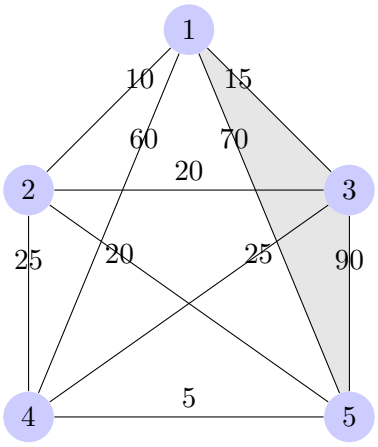


Figura 40: L'insieme $\{1, 3, 5\}$ è dominante.

Quando la matrice di affinità è binaria 0/1 allora l'insieme dominante coincide con la cricca massimale (stretta) in un grafo (vedi Sezione 6.2).

9.3.1 *Insiemi dominanti e ottimi locali*

In questa sezione sarà mostrato come la teoria di Nash possa essere utilizzata per risolvere in modo approssimato la ricerca di insieme dominanti.

Dato un grafo pesato $G = (V, E, w)$ e la sua matrice di adiacenza pesata A, si consideri il seguente programma quadratico standard:

$$\begin{aligned} &\text{maximize } f(x) = x^T Ax \\ &\text{subject to } x \in \Delta \end{aligned}$$

dove Δ è il simplesso standard su R^n. Si introduce il seguente teorema:

Teorema 8. *Se S è un sottoinsieme dominante di vertici, allora il suo vettore caratteristico pesato x^S definito come:*

$$x_i^S = \begin{cases} \frac{W_S(i)}{W(S)}, & \text{se } i \in S \\ 0 & \text{altrimenti} \end{cases}$$

è un massimizzatore locale stretto di f in Δ.
Al contrario se x^* è un massimizzatore locale stretto di f in Δ allora il suo supporto:

$$\sigma = \sigma(x^*) = \{i \in V : x_i^* \neq 0\}$$

è un insieme dominante a condizione che $W_{\sigma\cup\{i\}} \neq 0$ per ogni $i \notin \sigma$.

Dove la condizione $W_{\sigma\cup\{i\}} \neq 0$ è un tecnicismo dovuto alla presenza di soluzioni spurie, ovvero soluzioni che non ammettono vettore caratteristico pesato. Si tratta di una generalizzazione del teorema di Motzkin Strauss.

Ora che è stata fornita una caratterizzazione di insieme dominante è possibile utilizzare una qualunque tecnica di ottimizzazione quadratica per risolvere il problema (ad esempio la discesa del gradiente). Tuttavia le dinamiche di replicazione (vedi Sezione 8.1) si sono rivelate particolarmente adatte per affrontare questo problema. È stato infatti dimostrato il seguente teorema:

Teorema 9 (Torsello, Rota Bulò, Pelillo 2006)**.** *Strategie evolutionary stable (ESS) di un problema di clustering con matrice di affinità A sono in corrispondenza uno-a-uno con gli insiemi dominanti.*

Quindi sia A la matrice di adiacenza del grafo di similarità. Si pone:

$$W = A(= W^T \geq 0)$$

Allora un sistema di replicazione con matrice di payoff W partendo da uno stato iniziale arbitrario convergerà, per il principio di selezione naturale, a un massimizzatore della funzione $f(x) = x^T Ax$ nel simplesso standard. Questo corrisponderà a un insieme dominante in un grafo, ovvero ad un cluster di vertici. Quindi la ricerca di dominant sets di un grafo pesato e non orientato G, corrisponde a trovare equilibri di Nash asintoticamente stabili e le dinamiche di replicazione sono un ottimo strumento a disposizione per perseguire questo scopo.

La teoria dei giochi evoluzionistici opera in uno scenario in cui coppie di individui sono scelti ripetutamente a caso da una grande popolazione per competere in un gioco a due giocatori. A differenza della teoria dei giochi classica, i giocatori non si comportano razionalmente ma sono pre-programmati a un certo pattern comportamentale o una strategia mista. Col passare del tempo il principio di selezione naturale inciderà sulla distribuzione dei comportamenti.

Ad esempio si supponga di volere separare lo sfondo in un'immagine dagli elementi in primo piano. Nel gioco evoluzionistico ogni giocatore è pre-programmato a selezionare con una certa probabilità un elemento (un pixel) dall'immagine. In questo contesto la selezione naturale porterà giocatori che adottano strategie migliori (payoff più alto) ad espandersi, mentre quelli che adottano strategie peggiori ad estinguirsi. Nel caso descritto, ci si aspetta che la selezione naturale porti all'estinzione i giocatori che selezionano lo sfondo, per poi convergere a una popolazione che seleziona solo gli elementi in primo piano.

Concludendo le caratteristiche principali che rendono questo approccio preferibile rispetto ad altri sono le seguenti:

1. Non richiede alcuna rappresentazione dei dati ovvero che gli elementi debbano essere rappresentato come punti in uno spazio vettoriale;
2. assenza di assunzioni circa la struttura della matrice di affinità: è stato dimostrato che l'approccio funziona anche nel caso di funzioni di similarità asimmetriche o negative;
3. non è necessaria alcuna conoscenza a priori circa il numero di clusters;
4. permette l'estrazione di cluster sovrapposti.

9.4 PROBLEMA DI ETICHETTATURA

In un tipico labelling (consistent) problem si hanno:

- m oggetti $B = \{b_1, \ldots, b_n\}$;
- n etichette (labels) $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$.

Lo scopo è assegnare a ciascun oggetto in B un'etichetta in Λ.

I legami tra oggetti ed etichette sono espressi attraverso una matrice 4-dimensionale $n^2 \times m^2$ di coefficienti di compatibilità reali $R = \{r_{ij}(\lambda, \mu)\}$: la componente $r_{ij}(\lambda, \mu)$ misura la forza di compatibilità tra le due ipotesi "λ è sull'oggetto b_i" e "μ è sull'oggetto b_j". Alti valori significano alta compatibilità mentre bassi valori indicano incompatibilità.

Sia $p_i(\lambda)$ il grado di confidenza dell'ipotesi "l'etichetta λ è sull'oggetto b_i". Allora la distribuzione di probabilità delle etichette su un oggetto b_i è un vettore m-dimensionale:

$$\bar{p}_i = (p_1(\lambda_1), \ldots, p_n(\lambda_m))^T$$

con $p_i(\lambda) \geq 0$ e $\sum_\lambda p_i(\lambda) = 1$. Mettendo insieme i $\bar{p}_i$ si ottiene un *assegnamento pesato delle etichette* per gli oggetti in B denotato come $\bar{p}$, una matrice $n \times m$. Lo spazio degli assegnamenti pesati delle etichette è un insieme lineare convesso in $\mathbb{R}^{nm}$:

$$\mathcal{K} = \underbrace{\Delta \times \cdots \times \Delta}_{m \text{ times}}$$

dove Δ è il simplesso standard in $\mathbb{R}^n$. Ogni vertice in $\mathcal{K}$ rappresenta un assegnamento non ambiguo ovvero che assegna esattamente un'etichetta a ciascun oggetto. L'insiemi di questi labelling sarà denotato da $\mathcal{K}^*$.

Si consideri un labelling $\bar{p} \in \mathcal{K}$. La **consistenza** di un oggetto misura il grado di confidenza tra l'ipotesi "b_i è etichettato con λ" ed il contesto. Questo concetto può essere quantificato attraverso la seguente funzione di supporto lineare:

$$q_i(\lambda; \bar{p}) = \sum_j \sum_\mu r_{ij}(\lambda, \mu) p_j(\mu)$$

Sia $p \in \mathcal{K}^*$ e sia $\lambda(i)$ l'etichetta assegnata a b_i da $\bar{p}$ (i.e. $p_i(\lambda(i) = 1)$). Allora si dice che $\bar{p}$ è consistente se e solo se l'etichetta assegnata a ciascun oggetto riceve il più alto supporto da quell'oggetto. Questo corrisponde ad avere:

$$q_i(\lambda; \bar{p}) \leq q_i(\lambda(i); \bar{p})$$

per ogni i e λ. Applicando lo stesso ragionamento l'assegnamento pesato $\bar{p} \in \mathcal{K}$ è *consistente* se:

$$\sum_{\lambda} v_i(\lambda) q_i(\lambda; \bar{p}) \leq \sum_{\lambda} p_i(\lambda) q_i(\lambda; \bar{p})$$

per ogni $i = 1, \ldots, n$ e $\bar{v} \in \mathcal{K}$. Inoltre se la disuguaglianza è stretta per ogni $\bar{v} \neq \bar{p}$, allora $\bar{p}$ è detto *strettamente consistente*.

Hummel e Zucker hanno dimostrato che se la matrice di compatibilità è simmetrica $r_{ij}(\lambda, \mu) = r_{ji}(\mu, \lambda)$ allora una condizione sufficiente affinché $\bar{p}$ sia consistente corrisponde a un minimo locale della seguente funzione energia che misura la (in)consistenza tra labelling:

$$A(\bar{p}) = \sum_{i,\lambda} \sum_{j,\mu} r_{ij}(\lambda, \mu) p_i(\lambda) p_j(\mu)$$

Un processo **relaxation labelling** prende in input un assegnamento di etichette iniziale $\bar{p}^{(0)} \in \mathcal{K}$ e lo aggiusta iterativamente tenendo conto del modello di compatibilità utilizzando la seguente regola di aggiornamento:

$$p_i^{(t+1)}(\lambda) = \frac{p_i^{(t)} \lambda q_i^{(t)} \lambda}{\sum_{\mu} p_i^{(t)} \mu q_i^{(t)} \mu} \tag{40}$$

Purchè la matrice di compatibilità sia non-negativa. Il processo evolve fino a quando non viene raggiunto un punto fisso ovvero quando $\bar{p}^{(t+1)} = \bar{p}(t)$. Nella pratica è solito fermare il processo quando la distanza tra due etichettamenti successivi è sotto una certa soglia oppure dopo un certo numero di iterazioni. Queste tecniche sono particolarmente utilizzate nel campo della visione artificiale.

9.4.1 *Relaxation labelling e teoria dei giochi*

È possibile formulare un processo relaxation labelling in termini di teoria dei giochi. In questa formulazione:

- i giocatori sono gli oggetti;
- le etichette sono le strategie pure;
- gli assegnamenti pesati corrispondono alle strategia miste;
- la matrice di compatibilità è la matrice di payoff.

Pertanto un equilibrio di Nash corrisponde a un labelling consistente e un equilibrio di Nash stretto a un labelling consistente stretto. Inoltre la regola di aggiornamento (40) corrisponde alla dinamica di replicazione utilizzata nella teoria dei giochi evoluzionistici (vedi Sezione 8.1).

Si tratta di un **polimatrix game**: un gioco non-cooperativo succinto in cui per ogni coppia di giocatori i, j esiste una matrice di payoff che indica una componente del guadagno di un giocatore i. Il payoff totale per tale giocatore è la somma delle componenti. Formalmente:

- Ci sono n giocatori con m strategie;
- Per ogni coppia i, j di giocatori c'è una matrice di payoff A^{ij} di dimensione $m \times m$;
- Il payoff del giocatore i per una certa combinazione di strategie $s_1, \ldots, s_n$ è dato da:

$$\mu_i(s_1, \ldots, s_n) = \sum_{j \neq i} A^{ij}_{s_i s_j}$$

Il numero di payoff per rappresentare questo gioco è $O(n^2, m^2)$. Il problema di trovare un equilibrio di Nash è PPAD-completo.

9.5 TRASDUZIONE DI GRAFI

La trasduzione di grafi è una popolare classe di tecniche di apprendimento semi-supervisionato che mirano a stimare la funzione di classificazione definita su un grafo in cui alcuni nodi (datapoints) sono classificati e altri no. L'idea generale consiste nel propogare in maniera consistente l'informazione appresa nei nodi etichettati a quelli non etichettati.

Formalmente dato un insieme di datapoints $\mathcal{D} = \{\mathcal{D}_\ell, \mathcal{D}_u\}$ dove:

- $\mathcal{D}_\ell = \{(x_1, y_1), \ldots, (x_\ell, y_\ell\}$ sono gli oggetti etichettati;
- $\mathcal{D}_u = \{x_{\ell+1}, \ldots, x_n\}$ sono gli oggetti non etichettati.

Le relazioni tra questi oggetti sono date da un grafo pesato non orientato $G = (V, E)$ dove i vertici rappresentano gli oggetti e gli archi misurano la similarità tra le coppie di nodi. Nel caso speciale in cui il grafo non sia pesato (la matrice di adiacenza è binaria) la presenza di un arco tra due nodi i, j indica una similarità perfetta. In questo caso un cluster è dunque una componente connessa del grafo.

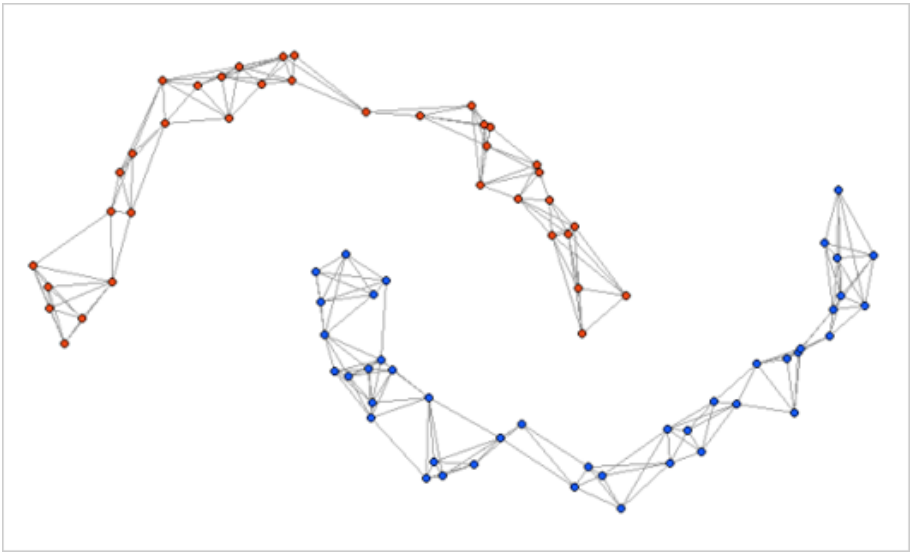


Figura 41: Apprendimento trasduttivo su un grafo non pesato.

Sia $A = a_{ij}$ la matrice di adiacenza binaria allora il problema della trasduzione su grafi non pesati può essere formalizzato come un *problema di soddisfacimento di vincoli* (CSP) definito su un insieme di variabili $V = \{v_1, \ldots, v_n\}$ i cui valori appartengono ai seguenti domini:

$$D_{v_i} = \begin{cases} \{y_i\}, & \text{per ogni } 1 \leq i \leq \ell \\ Y, & \text{per ogni } \ell + 1 \leq i \leq n \end{cases}$$

e soggetto ai seguenti vincoli binari:

$$\forall i, j : \text{se } a_{ij} = 1, \text{allora } v_i = v_j$$

Ad esempio in un problema a due classi i vincoli sono così definiti:

$$R_{ij} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Una soluzione ad un CSP è un assegnamento di valori alle variabili che soddisfi tutti i vincoli e corrisponde ad un'etichettatura consistente per i nodi non classificati.

È possibile risolvere un CSP con la teoria dei giochi. Si consideri il seguente gioco chiamato "*graph transduction game*" (GTG). Si assuma che ogni giocatore $i \in \mathcal{I}$ corrisponda a un particolare punto nel dataset $\mathcal{D} = \{d_1, \ldots, d_n\}$ e che possa scegliere una strategia nell'insieme $S_i = \{1, \ldots, c\}$ in cui ciascuna strategia rappresenta l'ipotesi di appartenenza ad una certa classe e c è il numero totale di classi.

Per definizione del problema i giocatori sono divisi in due gruppi disgiunti:

1. **giocatori etichettati**: consapevoli della loro classe di appartenenza e denotati dall'insieme $\mathcal{I}_\ell = \{\mathcal{I}_{\ell|1}, \ldots, \mathcal{I}_{\ell|c}\}$ in cui ogni sottoinsieme $\mathcal{I}_{\ell|k}$ indica i giocatori che giocano sempre le loro k strategie pure.
2. **giocatori non etichettati**: $\mathcal{I}_\mu$.

Ovviamente solo i giocatori non etichettati andranno a competere nel GTG in quanto i giocatori etichettati sono vincolati a giocare una strategia definita.

Si suppone che in questo gioco ogni arco nel grafo rappresenti un gioco a due giocatori e quindi il payoff di ciascun giocatore è la somma dei payoff guadagnati ad ogni giocata con uno dei suoi vicini. Formalmente dato un profilo strategico misto $x = (x_1, \ldots, x_n)$ la funzione di payoff di un giocatore $i \in \mathcal{I}$ è:

$$\mu_i(x) = \sum_{j=1}^{n} x_i^T A_{ij} x_j$$

dove A_{ij} è la *matrice di payoff parziale* tra i giocatori i e j. Il gioco è quindi governato dalla seguente funzione payoff:

$$\mu_i(x) = \sum_{j \in \mathcal{I}_\mu} x_i^T A_{ij} x_j + \sum_{k=1}^{c} \sum_{j \in \mathcal{I}_{\mathcal{D}|k}} x_i^T (A_{ij})_k$$

Rimane ora da definire la matrice di payoff parziale. Data la matrice pesata $W = (w_{ij})$ allora la matrice di payoff parziale tra due giocatori è data da $A_{ij} = w_{ij} \times I_c$ dove I_c è la matrice identità di dimensione c. Si noti che se la matrice di payoff parziale è rappresentata a blocchi $A = (A_{ij})$, allora la matrice A è data dal prodotto di Kronecker $A = I_c \otimes W$. Ad esempio in un problema a tre classi la matrice di payoff parziale A_{ij} è nella seguente forma:

$$A_{ij} = \begin{pmatrix} w_{ij} & 0 & 0 \\ 0 & w_{ij} & 0 \\ 0 & 0 & w_{ij} \end{pmatrix}$$

Si noti infine che nel caso di matrici di similarità binarie, allora le matrici di payoff parziale coincidono con le matrici di compatibilità definite nel CSP. Inoltre, se sono ammesse solo strategie pure allora il GTG si riduce al CSP. In questo caso in un equilibrio di Nash puro i giocatori nel vicinato giocano la stessa strategia pura con lo scopo di ottenere il payoff massimo contro i vicini. Tale equilibrio di Nash corrisponde alla soluzione del CSP se ne esiste una altrimenti il CSP è insoddisfabile e quindi il GTG non ammette equilibri di Nash.

Parte III

APPENDICE

A

RICHIAMI DI MATEMATICA

A.1 REGOLE DI DERIVAZIONE

Siano $f(x)$ e $g(x)$ funzioni reali di variabile reale x derivabili, e sia D l'operazione di derivazione rispetto a x:

$$\mathrm{D}[f(x)] = f'(x) \qquad \mathrm{D}[g(x)] = g'(x)$$

- **Regola della somma**:

$$\mathrm{D}[\alpha f(x) + \beta g(x)] = \alpha f'(x) + \beta g'(x) \qquad \alpha, \beta \in \mathbb{R}$$

- **Regola del prodotto**:

$$\mathrm{D}[f(x) \cdot g(x)] = f'(x) \cdot g(x) + f(x) \cdot g'(x)$$

- **Regola del quoziente**:

$$\mathrm{D}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x) \cdot g(x) - f(x) \cdot g'(x)}{g(x)^2}$$

- **Regola della funzione reciproca**:

$$\mathrm{D}\left[\frac{1}{f(x)}\right] = -\frac{f'(x)}{f(x)^2}$$

- **Regola della funzione inversa**:

$$\mathrm{D}[f^{-1}(y)] = \frac{1}{f'(x)}$$

con:

$y = f(x) \qquad x = f^{-1}(y)$

- **Regola della catena**

$$\mathrm{D}\left[f\left(g(x)\right)\right] = f'\left(g(x)\right) \cdot g'(x)$$

A.2 INTEGRALI

Teorema 10 (Teorema Fondamentale del Calcolo Integrale (prima parte))**.** *Sia* $f\colon [a,b] \to \mathbb{R}$ *una funzione integrabile. Si definisce funzione integrale di* f *la funzione* F *tale che:*

$$F(x) = \int_a^x f(t)dt \qquad a \leq x \leq b$$

Se f *è limitata, allora* F *è una funzione continua in* $[a,b]$*.*
Se inoltre f *è una funzione continua in* (a,b)*, allora* F *è funzione differenziabile in tutti i punti in cui* f *è continua e si ha:*

$$F'(x) = f(x)$$

cioè F *risulta essere una primitiva di* f*.*

B

SISTEMI DINAMICI

Un modello matematico per descrivere le dinamiche di un sistema non lineare è lo spazio degli stati. Con questo modello si pensa in termini di variabili di stato i cui valori ad un certo istante temporale sono considerati sufficienti a predire la futura evoluzione del sistema.

Sia $\bar{x}(t) = x_1(t), \ldots, x_N(t)$ il **vettore di stato**, un vettore contenente le variabili di stato di un sistema dinamico non lineare, in cui la variabile indipendente è il tempo t e N è l'ordine del sistema. È possibile allora descrivere un largo numero di sistemi dinamici non lineari mediante un sistema di equazioni differenziali di primo ordine chiamate equazioni dello spazio degli stati ed aventi la seguente forma:

$$\frac{d}{dt}\bar{x}(t) = F(\bar{x}(t)) \tag{41}$$

dove $F(\bar{x})$ è una funzione vettore che se applicata ad un vettore $\bar{x}$ ritorna un vettore che ha come j-esima componente $F_j(x_j)$ con F_j una qualche funzione; si può pensare a quest'ultimo come come ad un **vettore di velocità**. Un sistema in cui la funzione vettore F non dipende esplicitamente dal tempo è detto **autonomo**. In questo capitolo saranno trattati solo sistemi dinamici autonomi.

L'Equazione (41) può essere vista come un descrittore del movimento di un punto nello spazio degli stati N-dimensionale; questo punto non è altro che lo stato del sistema osservato ad un certo istante t. Con lo scorrere del tempo il punto t descrive una curva nello spazio degli stati detta **traiettoria** o orbita del sistema.

La velocità istantanea della traiettoria è rappresentata da un vettore tangente; possiamo quindi derivare un vettore di velocità per ogni punto della traiettoria. La famiglia delle traiettorie, per differenti condizioni iniziali, è chiamato *ritratto degli stati* del sistema e comprende tutti quei punti per cui $F(\bar{x})$ è definita; nel caso di sistemi autonomi per ogni punto dello spazio esiste una sola traiettoria che vi passa.

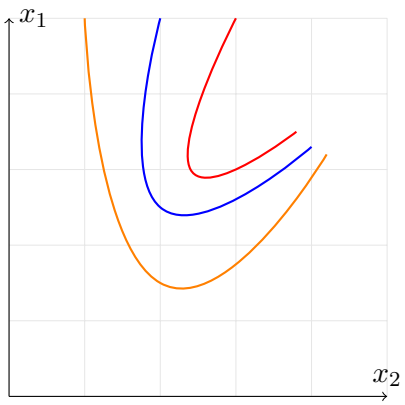


Figura 42: Un ritratto degli stati bidimensionale di un sistema dinamico.

B.1 CONDIZIONE DI LIPSCHITZ

Affinchè l'equazione dello spazio degli stati abbia soluzione e questa sia unica è necessario imporre alcune restrizioni sul vettore $F(\bar{x})$. Condizione sufficiente affinchè vi sia soluzione è che $F(x)$ sia continua in tutti i suoi argomenti; tuttavia questa condizione non garantisce l'unicità della soluzione. Serve una restrizione più forte conosciuta come condizione di Lipschitz. Una funzione:

$$\mathbf{f} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$$

si dice "lipschitziana" su Ω se:

$$\exists K \geq 0$$

tale che

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\| \leq K \|\mathbf{x} - \mathbf{y}\| \qquad \forall \mathbf{x}, \mathbf{y} \in \Omega$$

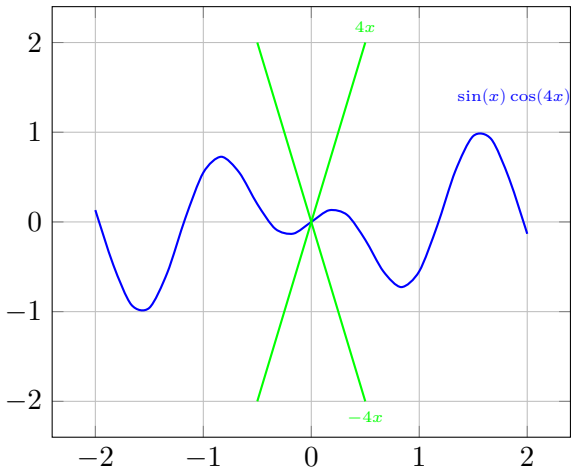


Figura 43: Interpretazione grafica della Condizione di Lipschitz: la funzione $f = \sin(x)\cos(4x)$ è lipschitziana con $K = 4$. Ciò significa che se si considera un qualunque punto lungo la funzione e si tracciano le rette di coefficienti angolari 4 e -4, allora f non sarà mai confinata tra le due rette.

B.2 STATI DI EQUILIBRIO

Intuitivamente un punto/stato di equilibrio stabile è un punto che non risente delle piccole perturbazioni: se ci si sposta poco da un punto stabile il sistema continuerà a rimanere anche in futuro nelle vicinanze di quel punto. L'esempio più semplice è quello di una pallina disposta esattamente nel fondo di una valle, se la spostiamo di poco dal fondo può rotolare in basso ed oscillare ma non aumenta la sua distanza dal punto di equilibrio.

Viceversa un punto di equilibrio instabile è tale per cui basta una perturbazione arbitrariamente piccola dall'equilibrio per far allontanare significativamente il sistema dalla posizione iniziale. Un esempio è una pallina disposta sulla cima di una collina.

Un vettore costante $\bar{x} \in M$ è detto stato di equilibrio (stazionario) se è soddisfatta la seguente condizione:

$$\mathbf{F}(\bar{x}) = 0$$

dove 0 è il vettore nullo. Il vettore velocità si annulla nel punto di equilibrio e quindi $\mathbf{x}(t) = x$ è una soluzione dell'equazione dello spazio degli stati. Lo stato di equilibrio è anche detto punto singolare e la traiettoria degenera nel punto stesso.

Si enunciano ora una serie di definizioni sulla stabilità degli stati di equilibrio.

Definizione 6. *Lo stato di equilibrio x è detto* ***uniformemente stabile*** *se per ogni $\bar{x}$ positivo, esiste un ϵ positivo tale che la condizione:*

$$|\mathbf{x}(0) - \bar{x}| < \delta$$

implica

$$|\mathbf{x}(t) - \bar{x}| < \epsilon$$

per ogni $t > 0$.

Questa definizione afferma che una traiettoria del sistema può essere fatta stare in un piccolo intorno dello stato di equilibrio $\bar{x}$ se il suo stato iniziale è vicino a $\bar{x}$.

Definizione 7. *Lo stato di equilibrio x è detto* ***convergente*** *se esiste un δ positivo tale che la condizione:*

$$|\mathbf{x}(0) - \bar{x}| < \delta$$

implica che:

$$\lim_{t \to \infty} \mathbf{x}(t) = \bar{x}$$

Questa seconda definizione afferma che se lo stato iniziale di una traiettoria è adeguatamente vicino allo stato di equilibrio x, allora la traiettoria descritta dal vettore di stato $x(t)$ convergerà a x all'aumentare del tempo.

Definizione 8. *Lo stato di equilibrio $\bar{x}$ è detto* ***asintoticamente stabile*** *se è stabile e convergente.*

Definizione 9. *Lo stato di equilibrio x è detto* ***globalmente asintoticamente stabile*** *se è stabile e se tutte le traiettorie del sistema convergono a x per t che tende ad infinito.*

B.3 TEOREMA DI LYAPUNOV

Si enunciano ora i due teoremi di Lyapunov che forniscono condizioni sufficienti per l'esistenza di un punto stazionario.

Teorema 11 (Primo teorema di Lyapunov)**.** *Lo stato di equilibrio $\bar{x}$ è stabile se in un piccolo intorno di $\bar{x}$ esiste una funzione definita positiva $V(x)$ tale che la sua derivata rispetto al tempo è minore o uguale a 0 in quella regione.*

Teorema 12 (Secondo teorema di Lyapunov)**.** *Lo stato di equilibrio $\bar{x}$ è asintoticamente stabile se in un piccolo intorno di $\bar{x}$ esiste una funzione definita positiva $V(x)$ tale che la sua derivata rispetto al tempo sia strettamente minore di 0 in quella regione.*

Una funzione scalare $V(x)$ che soddisfa queste proprietà è detta **funzione di Lyapunov** per lo stato di equilibrio $\bar{x}$. La funzione $V(x)$ è definita positiva nello spazio degli stati L, se $\forall x \in L$, soddisfa le seguenti proprietà:

1. la funzione $V(x)$ ha derivate parziali rispetto agli elementi di $\mathbf{x}$ continue;
2. $V(\bar{x}) = 0$;
3. $V(x) > 0$ se $\mathbf{x} \neq \bar{x}$.

Supposta $V(x)$ una funzione di Lyapunov, lo stato di equilibrio $\bar{x}$ è stabile se:

$$\frac{d}{dt} \cdot V(\mathbf{x}) \leq 0, \qquad \text{se } |x - \bar{x}| \leq \epsilon$$

dove ϵ è un numero positivo. Lo stato di equilibrio è asintoticamente stabile se:

$$\frac{d}{dt} \cdot V(\mathbf{x}) < 0, \qquad \text{se } |x - \bar{x}| \leq \epsilon$$

Il criterio è una generalizzazione del fatto, ben noto nella fisica, che un sistema meccanico, se lasciato libero di evolvere, tende a portarsi in una configurazione dove la sua energia potenziale è minima. La funzione di Lyapunov può quindi essere interpretata come una funzione di energia potenziale generalizzata. Il criterio dice che uno stato di equilibrio è stabile se:

1. è minimo per una certa funzione di energia generalizzata (cioè se esiste una funzione di Lyapunov definita positiva);
2. se il sistema tende a portarsi verso la configurazione di minimo della funzione di Lyapunov (cioè se la derivata della funzione di Lyapunov è semidefinita negativa).

Il criterio è una **condizione sufficiente ma non necessaria**. Non è detto in generale che l'origine non sia stabile se non esiste una funzione di Lyapunov definita in un intorno dell'origine. Inoltre non esiste un algoritmo per trovare la funzione di Lyapunov relativa a un sistema, ma si deve cercare per tentativi, basandosi sul tipo di funzione di stato e su considerazioni puramente fisiche.

BIBLIOGRAFIA

[1] Gulli Daniel Andrea Casavola. Identificazione di modelli attraverso reti neurali.

[2] Samuel Rota Bulò. Appunti di reti neurali. http://www.dsi.unive.it/~srotabul/files/AppuntiRetiNeurali.pdf.

[3] Samuel Rota Bulò. Clustering con affinità asimmetriche: un approccio basato sulla teoria dei giochi. Master's thesis, Università Ca' Foscari di Venezia, 2005.

[4] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. An iterative pruning algorithm for feedforward neural networks. *Neural Networks, IEEE Transactions on*, 8(3):519–531, 1997.

[5] Aykut Erdem and Marcello Pelillo. Graph transduction as a noncooperative game. *Neural Computation*, 24(3):700–723, 2012.

[6] John Hertz. *Introduction to the theory of neural computation*, volume 1. Basic Books, 1991.

[7] Massimiliano Pavan and Marcello Pelillo. Dominant sets and pairwise clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(1):167–172, 2007.

[8] Marcello Pelillo. Replicator equations, maximal cliques, and graph isomorphism. *Neural Computation*, 11(8):1933–1955, 1999.

[9] Marcello Pelillo. Replicator dynamics in combinatorial optimization replicator dynamics in combinatorial optimization. In *Encyclopedia of optimization*, pages 3279–3291. Springer, 2009.

[10] Marcello Pelillo. Slides del corso intelligenza artificiale, 2013. http://www.dsi.unive.it/~pelillo/Didattica/.

[11] Marcello Pelillo, Kaleem Siddiqi, and Steven W Zucker. Matching hierarchical structures using association graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(11):1105–1120, 1999.

[12] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8): 888–905, 2000.

[13] Peter Norvig Stuart Russel. *Artificial Intelligence, A Modern Approach.* Pearson, third edition, 2001.

[14] Mauro Tempesta. Appunti di Mauro Tempesta, 2012.

[15] Andrea Torsello, Samuel Rota Bulo, and Marcello Pelillo. Grouping with asymmetric affinities: A game-theoretic perspective. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 292–299. IEEE, 2006.