Rota Bulò Samuel 796408

Appunti
di
reti neurali

CONTENUTO

1 Introduzione

1.1 Cosa sono le reti neurali

Sono sistemi computazionali ispirati a processi biologici con le seguenti caratteristiche:

- formati da milioni di unità computazionali (**neuroni**) capaci di eseguire una somma pesata;
- elevato numero di connessioni pesate (**sinapsi**) tra le unità;
- altamente paralleli e non lineari;
- adattivi e addestrabili e l'apprendimento avviene attraverso la modifica dei pesi delle connessioni;
- tolleranti agli errori in quanto la memorizzazione avviene in modo diffuso;
- non c'è distinzione tra memoria e area di calcolo;
- capacità di **generalizzazione**: producono output ragionevoli con input mai incontrati prima durante l'apprendimento.

1.2 La mente umana

Il **sistema nervoso** umano è un sistema a 3 stadi:

1. **rete neurale**: riceve continue informazioni, le percepisce e fa appropriate decisioni;
2. **recettori**: convertono gli stimoli esterni in impulsi elettrici che vengono inviati alla rete neurale;
3. **attuatori**: convertono gli impulsi elettrici generati dalla rete neurale in risposte al sistema esterno.

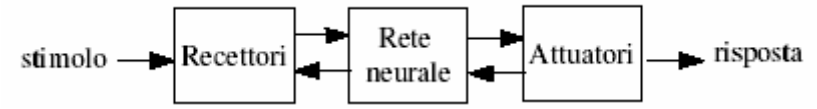

Figura 1: diagramma a blocchi del sistema nervoso.

- La corteccia celebrale è formata da circa 10 bilioni di neuroni e 60 trilioni di sinapsi.
- ogni neurone ha velocità dell'ordine dei millisecondi, contro i nanosecondi dei chip in silicio;
- l'efficienza energetica è di circa 10^{-16} J per operazione per secondo contro i 10^{-6} del migliore conputer in uso oggi.

Un **neurone** è caratterizzato da:

- **corpo cellulare**: l'unità di calcolo (soma);
- **assone**: linea di trasmissione in uscita;
- **dendriti**: le zone recettive.

Il neurone ha il compito di fare una somma pesata degli ingressi e se il risultato supera un certo valore soglia produce un potenziale d'azione che viene inviato all'assone, altrimenti non fa nulla.

Il **potenziale d'azione** consiste in una scarica di impulsi elettrici e il motivo di ciò è che l'assone è molto lungo e fine, caratterizzato da alta resistenza elettrica e un voltaggio applicato ad un estremità decade in modo esponenziale con la distanza; inviando diversi impulsi elettrici questo problema di trasmissione viene evitato.

Le sinapsi sono unità funzionali dalla struttura elementare che gestisce le iterazioni tra neuroni. La sinapsi più comune è la sinapsi chimica che svolge le seguenti operazioni:

- un **processo presinaptico** libera una sostanza trasmettitrice che si diffonde lungo la giunzione sinaptica tra 2 neuroni;
- un **processo postsinaptico** azionato dalla sostanza trasmettitrice rigenera un segnale elettrico.

Quindi una sinapsi converte un segnale elettrico in chimico della presinapsi e poi da chimico a elettrico nella postsinapsi.

Ci sono 2 tipi di sinapsi:

1. **eccitatorie**: favoriscono la generazione di elettricità nel neurone postsinaptico;
2. **inibitorie**: inibiscono la generazione di elettricità nel neurone postsinaptico.

Pare che l'apprendimento avvenga attraverso variazioni nell'efficacia sinaptica di ogni sianpsi.

2 Neurone di McCulloch-Pitts

2.1 Modello neurone

Un **neurone** è l'unità di calcolo fondamentale della rete neurale ed è formato da 3 elementi di base nel modello neurale:

1. un insieme di **sinapsi** o **connessioni** ciascuna delle quali è caratterizzata da un peso (efficacia sinaptica); a differenza del modello umano, il modello artificiale può avere pesi sia negativi che positivi;
2. un **sommatore** che somma i segnali in input pesati dalle rispettive sinapsi, producendo in output una combinazione lineare degli input;
3. una **funzione di attivazione** per limitare l'ampiezza dell'output di un neurone. Tipicamente per comodità l'ampiezza degli output appartengono all'intervallo [0,1] oppure [-1,1].

Il modello neuronale include anche un **valore soglia** che ha l'effetto, a seconda della sua positività o negatività, di aumentare o diminuire l'input netto alla funzione di attivazione.

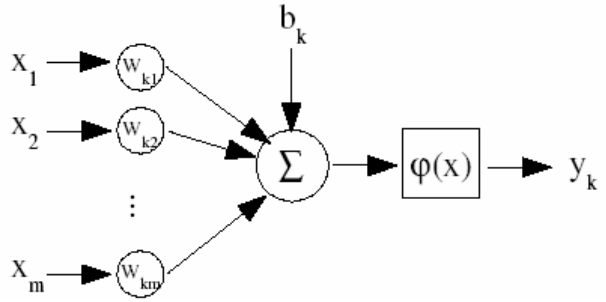

Figura 2: modello non lineare di un neurone.

In termini matematici descriviamo un neurone k con le seguenti equazioni:

$$u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j$$

$$y_k = \varphi(u_k + b_k)$$

dove:

- x_i sono i pesi sinaptici del neurone k;
- u_k è la combinazione lineare degli input nel neurone k;
- b_k è il valore soglia del neurone k;
- $\varphi(x)$ è la funzione di attivazione;
- y_k è l'output generato dal neurone k.

Possiamo riformulare le 2 equazioni inglobando il valore soglia in u_k ottenendo così un modello equivalente al precedente:

$$v_k = \sum_{j=0}^{m} w_{kj} \cdot x_j$$

$$y_k = \varphi(v_k)$$

dove l'input $x_0=1$ e $w_{k0}=b_k$. Definiamo v_k **potenziale di attivazione**.

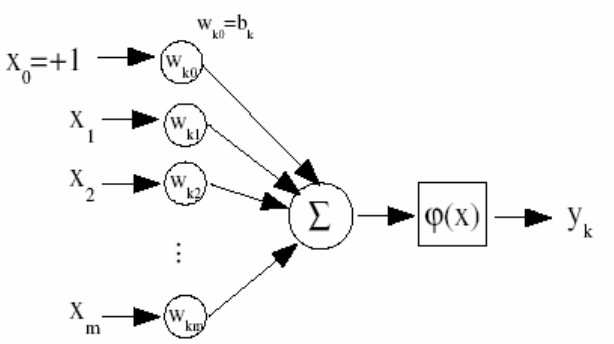

Figura 3: altro modello non lineare di un neurone.

2.2 Tipi di funzione di attivazione

Identifichiamo 3 tipi di funzione di attivazione base:

2.2.1 Threshold function (o Heaviside function)

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$

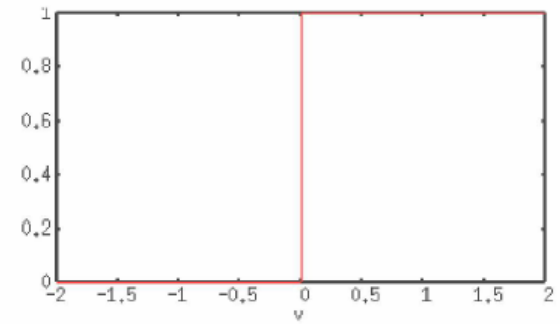

Figura 4: Threshold function

Questa funzione è usata nel modello di McCulloch-Pitts.

2.2.2 Piecewise-linear function

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq \frac{1}{2} \\ v & \text{se } -\frac{1}{2} < v < \frac{1}{2} \\ 0 & \text{se } v \leq -\frac{1}{2} \end{cases}$$

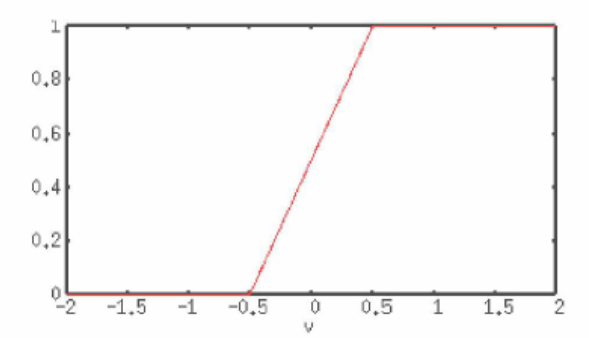

Figura 5: Piecewise-linear function

2.2.3 Sigmoid function

E' la funzione più usata nella costruzione di reti neurali artificiali. E' una funzione strettamente crescente che esibisce un bilanciamento tra un comportamento lineare e non lineare.
Esempi:

$$\varphi(v) = \frac{1}{1 + e^{-a \cdot v}}$$

dove a è un parametro che indica la pendenza della funzione.

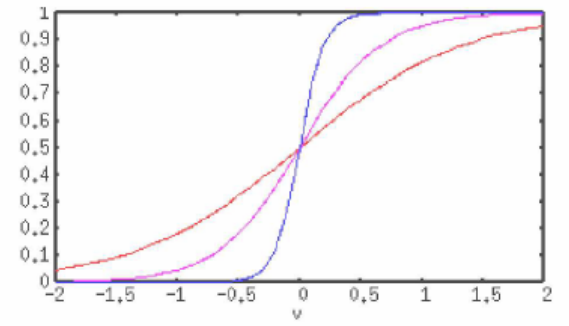

Figura 6: funzione logistica. In rosso a=1.5; in viola a=3; in blu a=10.

Questa funzione approssima la threshold function se a $\rightarrow +\infty$.
A volte è desiderabile avere una funzione di attivazione nell'intervallo [-1,1]. In questo caso possiamo utilizzare la tangente iperbolica.

$$\varphi(v) = \tanh(a \cdot v)$$

dove a è un parametro di pendenza.

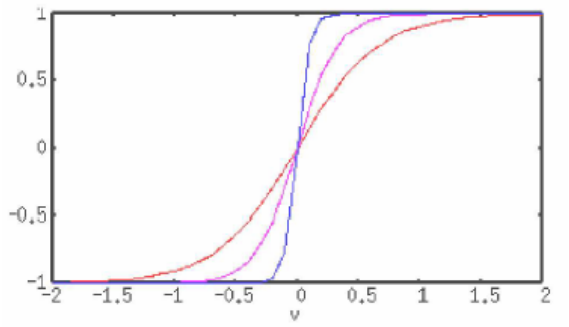

Figura 7: Tangente iperbolica. In rosso a=1.5; in viola a=3; in blu a=10.

3 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 identifichiamo 3 classi di reti.

3.1 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. Non ci sono connessioni che tornano indietro e nemmeno connessioni trasversali nel layer di output.

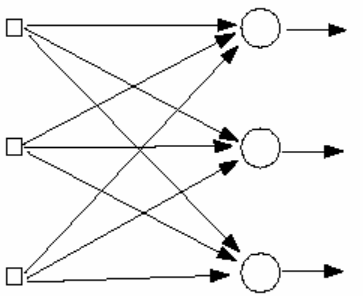

Figura 8: rete forward ad uno strato.

3.2 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**). Ogni strato ha connessioni entranti dal precedente strato e uscenti in quello successivo, quindi la propagazione del segnale avviene in avanti senza cicli e senza connessioni trasversali.
Questo tipo di architettura fornisce alla rete una prospettiva globale in quanto aumentano le interazioni tra neuroni.

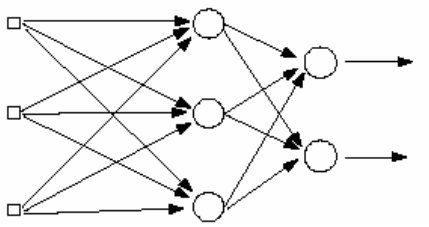

Figura 9: rete feedforward a 2 strati.

3.3 Reti ricorrenti o feedback

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.

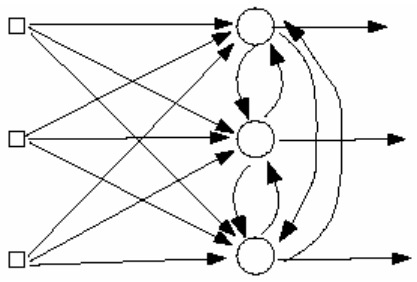

Figura 10: rete ricorrente o feedback.

4 Processi di apprendimento

L'apprendimento è un processo col quale parametri liberi di una rete neurale sono adattati, attraverso un processo di stimolazione, all'ambiente in cui essa è inserita. Il tipo di apprendimento è determinato dal modo in cui questi adattamenti avvengono.

Un **algoritmo di apprendimento** è un insieme di regole ben definite che risolvono un problema di

apprendimento.

Un algoritmo di apprendimento può essere di due tipi:

1. **con supervisione**: c'è un insegnante che conosce l'ambiente che fornisce mappature corrette di input/output. La rete dovrà aggiustare i propri parametri liberi in modo da emulare l'insegnante in modo statisticamente ottimale.
2. **senza supervisione**: la rete apprende autonomamente.

Vediamo alcune tipologie di apprendimento.

4.1 Apprendimento con correzione di errore

Ogni neurone k riceve in ingresso un segnale di stimolo **x**(n) e genera una risposta $y_k(n)$, con n un tempo discreto. Indichiamo inoltre con $d_k(n)$ la risposta desiderata. Di conseguenza si genera un **segnale di errore** $e_k(n)$.

$$e_k(n) = d_k(n) - y_k(n)$$

Il segnale di errore $e_k(n)$ attua un meccanismo di controllo con l'obiettivo di applicare una sequenza di aggiustamenti ai pesi sinaptici del neurone k al fine di avvicinare la risposta ottenuta a quella desiderata.

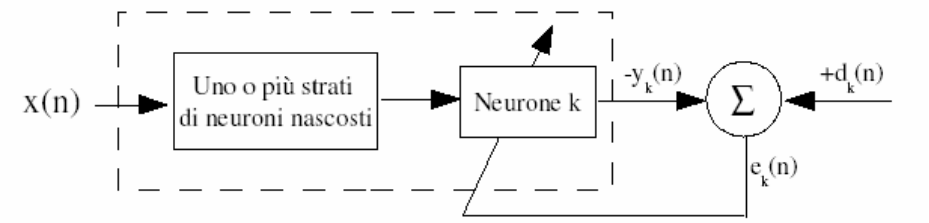

Figura 11: esempio di apprendimento con correzione di errore con rete multistrato feedforward.

Questo processo avviene un passo alla volta minimizzando una **funzione costo**:

$$E(n) = \frac{1}{2} \cdot e_k^2(n)$$

Per fare ciò si ricorre al **metodo del gradiente** o metodo di Widrow-Hoff. Siano $w_{kj}(n)$ i pesi sinaptici del neurone k eccitati dall'elemento $x_j(n)$ del segnale di stimolo, allora l'aggiustamento applicato a $w_{kj}(n)$ è:

$$\Delta w_{kj} = \eta \cdot e_k(n) \cdot x_j(n)$$

dove ηuna costante positiva detta **tasso di apprendimento**.
I nuovi pesi saranno allora:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

L'apprendimento con correzione di errore è un esempio di sistema ricorrente la cui stabilità dipende dai parametri che costituiscono il ciclo e in particolare η. La scelta di questo parametro è importante perché influenza la convergenza e la stabilità del processo di apprendimento.

4.2 Apprendimento basato sulla memoria

Nell'apprendimento basato sulla memoria, tutti (o molti) delle passate esperienze vengono archiviate in una larga memoria di coppie input-output correttamente classificate $\{(\mathbf{x}_i, d_i)\}_{i=1}^{n}$, dove $\mathbf{x}_i$ è il vettore di input e d_i è la risposta desiderata che senza perdita di generalità abbiamo ridotto ad uno scalare.
Quando verrà richiesta la classificazione di un esempio mai incontrato prima $\mathbf{x}_{test}$ il sistema risponde trovando ed analizzando gli esempi memorizzati in un intorno di $\mathbf{x}_{test}$.

Tutti i metodi di apprendimento basati su memoria comportano 2 ingredienti base:

- criterio usato per definire l'intorno di un vettore test $\mathbf{x}_{test}$;
- metodo di apprendimento applicato sugli esempi nell'intorno di $\mathbf{x}_{test}$.

Un esempio di questo metodo è il **metodo nearest neighbor** in cui l'esempio più vicino all'esempio test $\mathbf{x}'_N$ è quello che ha distanza euclidea minima.

$$\mathbf{x}'_N \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$$
$$\min_i [d(\mathbf{x}_i, \mathbf{x}_{test})] = d(\mathbf{x}'_N, \mathbf{x}_{test})$$

dove d è la distanza euclidea. La classe che viene assegnata ad $\mathbf{x}_{test}$ è la stessa di $\mathbf{x}'_N$.

Una variante di questo metodo è il **k-nearest neighbor** in cui:
- l'intorno dell'esempio test non è più uno solo, ma l'insieme dei k esempi memorizzati più vicini;
- la classe assegnata è quella con la frequenza maggiore nell'intorno dell'esempio test.

4.3 Apprendimento di Hebbian

Postulato di Hebb sull'apprendimento: 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.

Da ciò possiamo ricavare due regole:
1. se 2 neuroni connessi da una sinapsi si attivano simultaneamente, allora il peso della sinapsi viene progressivamente incrementato;
2. se 2 neuroni connessi si attivano in modo asincrono, allora il peso della sinapsi viene progressivamente diminuito o eliminato.

Una sinapsi di questo tipo è detta **sinapsi di Hebbian**.

Se la correlazione dei segnali porta ad un incremento dell'efficacia sinaptica chiameremo questa modifica **hebbiana**, se porta ad una riduzione la chiameremo **anti-hebbiana**.

4.4 Apprendimento competitivo

Con l'apprendimento competitivo i neuroni di uscita di una rete neurale competono tra di loro per divenire attivi. Solo un neurone può essere attivo in un certo momento n.

Ci sono 3 elementi base per un metodo di apprendimento competitivo:
1. un insieme di neuroni uguali a meno di pesi sinaptici generati a caso che rispondono in maniera differente ad un dato insieme di inputs;
2. un limite alla "forza" di ciascun neurone;
3. un meccanismo che permette ai neuroni di competere per il diritto a rispondere ad un dato sottoinsieme di inputs cosicché un solo neurone o uno solo per gruppo è attivo in un certo momento. Il neurone che vince è chiamato **winner-takes-all**.

In questo modo i neuroni tendono a specializzarsi su un insieme di input simili divenendo riconoscitori di caratteristiche per differenti classi di inputs.

Nella forma più semplice la rete neurale ha un solo strato di neuroni di uscita, completamente connessi ai nodi di input (**connessioni forward eccitatorie**). La rete può includere connessioni tra i neuroni che portano ad inibizioni laterali (**connessioni feedback inibitorie**).

Il neurone k che vince la competizione è quello con input netto v_k più alto per un dato input **x**. Il segnale di output y_k del neurone vittorioso è settato a 1, mentre quello degli altri viene settato a 0.

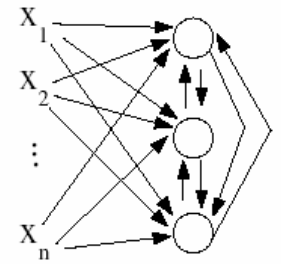

Figura 12: struttura di una semplice rete di apprendimento competitivo.

$$y_k = \begin{cases} 1 & \text{se } v_k > v_j \quad \forall j,\ j \neq k \\ 0 & \text{altrimenti} \end{cases}$$

dove v_k è la combinazione lineare di tutti gli input forward e feedback.
Siano $w_{kj} \geq 0$ i pesi sinaptici tra input j e neurone k. Supponiamo che ogni neurone abbia un ammontare fisso di peso sinaptico distribuito tra i nodi di input:

$$\sum_j w_{kj} = 1 \quad \forall k$$

Il neurone che vince la competizione apprende spostando i pesi sinaptici dagli input inattivi agli input attivi. Per fare ciò si utilizza la regola standard di apprendimento competitivo:

$$\Delta w_{kj} = \begin{cases} \eta \cdot (x_j - w_{kj}) & \text{se il neurone k vince} \\ 0 & \text{altrimenti} \end{cases}$$

5 Attività di apprendimento

5.1 Associazione di pattern

Una **memoria associativa** è una memoria ispirata al cervello distribuita che impara per associazioni.

Le associazioni possono essere di due forme:

1. **autoassociazioni**: una rete neurale memorizza un insieme di pattern (vettori) presentandoli ripetutamente alla rete. Successivamente viene presentato alla rete un pattern (tra quelli memorizzati nella rete) parziale o distorto e l'obiettivo è che la rete richiami la versione del pattern completa/originale che è stata precedentemente memorizzata. L'apprendimento in questo caso è generalmente non supervisionato;
2. **eteroassociazioni**: ogni pattern è associato ad un altro pattern. Sia $\mathbf{x}_k$ il pattern chiave a cui è associato $\mathbf{y}_k$ ovvero il pattern memorizzato. Il pattern chiave $\mathbf{x}_k$ opera come uno stimolo per richiamare il pattern $\mathbf{y}_k$. Nelle autoassociazioni avevamo che $\mathbf{x}_k = \mathbf{y}_k$ mentre in questo caso abbiamo che $\mathbf{x}_k \neq \mathbf{y}_k$.

La memoria associativa si articola in 2 fasi:

1. **fase di memorizzazione**: in cui la rete viene addestrata per fare in modo che i pattern vengano associati;
2. **fase di richiamo**: in cui si richiama dalla rete un pattern memorizzato a seguito della presentazione di una versione parziale o distorta di un pattern chiave.

5.2 Riconoscimento di pattern

Il **riconoscimento di pattern** è il processo in cui un pattern/segnale viene assegnato ad una classe (categoria). Una rete neurale riconosce pattern a seguito di una sessione 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 verrà presentato un pattern mai visto prima ma appartenete ad una stesso categoria di pattern che ha appreso, la rete sarà in grado di classificarla grazie alle informazioni estratte dai dati di addestramento.

Ogni pattern rappresenta nello **spazio decisionale** multidimensionale un punto. Questo spazio è suddiviso in regioni, ognuna delle quali associata ad una classe. I confini di queste regioni sono determinate dalla rete attraverso il processo di addestramento.

Il riconoscimento di pattern con reti neurali può assumere 2 forme:

1. il sistema viene splittato in 2 parti: una rete non supervisionata per l'estrazione delle caratteristiche (**features**) e una rete supervisionata per la classificazione. Un pattern **x** è rappresentato da un punto dimensionale dello **spazio dei dati**. L'estrazione di caratteristiche è una trasformazione che mappa **x** in uno punto **y** dello spazio q-dimensionale intermediario detto **spazio delle caratteristiche** con q<m. La classificazione è una trasformazione che mappa dal punto **y** in una classe dello spazio r-dimensionale detto **spazio decisionale** con r il numero totale di classi;
2. il sistema consiste di una rete feedforward a più strati singola addestrata con un algoritmo di apprendimento supervisionato. L'attività di estrazione delle caratteristiche è un processo interno alla rete svolta dagli strati nascosti.

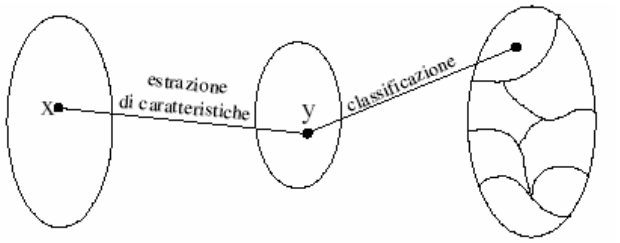

Figura 13: approccio classico alla classificazione di pattern .

5.3 Approssimazione di funzioni

Si consideri una mappatura input-output non lineare descritta da:

$$\mathbf{d} = f(\mathbf{x})$$

dove **x** è l'input e **d** è l'output e sia la funzione f la nostra incognita. Ci vengono però forniti un insieme di esempi:

$$\{(\mathbf{x}_i, \mathbf{d}_i)\}_{i=1}^{n}$$

Vogliamo creare una rete neurale che approssimi la funzione f in modo che la funzione F che descrive la mappa input-output realizzata dalla rete è vicina ad f in senso euclediano per tutti gli input:

$$\|F(\mathbf{x}) - f(\mathbf{x})\| < \varepsilon, \quad \forall \mathbf{x}$$

dove ε è un numero positivo piccolo. Maggiori sono le dimensioni e i parametri liberi della rete minore sarà ε.
Il problema dell'approssimazione di funzioni è un candidato perfetto per l'apprendimento supervisionato con $\mathbf{x}_i$ il vettore di input e $\mathbf{d}_i$ la risposta desiderata.

6 Single Layer Perceptron

Il **percettrone** è la forma più semplice di rete neurale usata per classificare pattern **linearmente separabili**, ovvero pattern che stanno ai lati opposti di un iperpiano. Esso consiste di un singolo neurone con pesi sinptici e soglia modificabili.

Di seguito viene proposto l'algoritmo di apprendimento conosciuto come algoritmo di convergenza del percettrone. Rosenblatt dimostrò che se i patterns scelti per addestrare la rete appartengono a 2 classi linearmente separate, allora l'algoritmo di apprendimento converge e lo spazio decisionale viene diviso in 2 da un iperpiano. Questo teorema è chiamato **teorema di convergenza del percettrone**.

6.1 Teorema di convergenza del percettrone

Affinché il percettrone funzioni correttamente è necessario che lo spazio decisionale sia formato da 2 classi C_1 e C_2, separate linearmente da un iperpiano.
Siano X_1 e X_2 gli insiemi formati dagli input di addestramento che appartengono rispettivamente alle classi C_1 e C_2, e la cui unione forma l'intero insieme di addestramento X.
Assumiamo senza perdita di generalità che la funzione di attivazione φ=sgn (funzione segno), in modo che:

$$y(n) > 0 \quad \text{se } \mathbf{x}(n) \in X_1$$
$$y(n) \leq 0 \quad \text{se } \mathbf{x}(n) \in X_2$$

ma dal momento che y(n) e v(n) hanno lo stesso segno possiamo mettere v(n) al posto di y(n). Quindi posto **w** un vettore di pesi che classifica correttamente gli $\mathbf{x} \in X$, sarà corretta la seguente relazione:

$$v(n) = \mathbf{w}^T \cdot \mathbf{x} > 0 \quad \forall \mathbf{x} \in X_1$$
$$v(n) = \mathbf{w}^T \cdot \mathbf{x} \leq 0 \quad \forall \mathbf{x} \in X_2$$

L'algoritmo di apprendimento può essere formulato nel seguente modo:

1. Se l'n-esimo elemento di X, **x**(n), è classificato correttamente dal vettore dei pesi **w**(n), allora non vengono fatte correzioni al vettore **w**(n) quindi **w**(n+1)=**w**(n);
2. Altrimenti il vettore **w**(n) viene aggiornato nel seguente modo:
   $$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n) \cdot \mathbf{x}(n) \quad \forall \mathbf{x} \in X_2$$
   $$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n) \cdot \mathbf{x}(n) \quad \forall \mathbf{x} \in X_1$$
   dove il fattore di apprendimento η(n) controlla le modifiche applicate ai pesi all'n-esima iterazione.

Se η(n)=η>0 con η una costante indipendente dal numero di iterazioni n, allora abbiamo il metodo **fixed increment adaptation** per il percettrone.
Nel seguito verrà dimostrata la convergenza dell'apprendimento con metodo fixed increment adaptation rule con η=1.

Sia **w**(0)=**0**. Supponiamo $\mathbf{w}^T(n) \cdot \mathbf{x}(n) < 0$ per n= 1, 2, ... e $\mathbf{x}(n) \in X_1$, ovvero il percettrone classifica in modo errato i vettori di input **x**(1), **x**(2), ...
La regola di aggiornamento dei pesi per questo caso è la seguente:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad \forall \mathbf{x} \in X_1$$

Dal momento che **w**(0)=**0** abbiamo che:

$$\mathbf{w}(n+1) = \sum_{i=1}^{n} \mathbf{x}(i)$$

Sia $\mathbf{w}_0$ una soluzione che classifica correttamente gli **x**(i) e definiamo come:

$$\alpha = \min_{\mathbf{x}(n)\in X_1} \mathbf{w}_0^T \cdot \mathbf{x}(n)$$

dove $\alpha>0$ dal momento che la classificazione è corretta con $\mathbf{w}_0$ vettore dei pesi e $\mathbf{x}(n)\in X_1$.
Possiamo dedurre che:

$$\mathbf{w}_0^T \cdot \mathbf{w}(n+1) = \mathbf{w}_0^T \cdot \sum_{i=1}^{n} \mathbf{x}(i) \geq \alpha \cdot n$$

Utilizziamo a questo punto la disuguaglianza di Cauchy-Scwarz e scriviamo:

$$\|\mathbf{w}_0\|^2 \cdot \|\mathbf{w}(n+1)\|^2 \geq \left[\mathbf{w}_0^T \cdot \mathbf{w}(n+1)\right]^2 \geq n^2 \cdot \alpha^2$$

quindi

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \cdot \alpha^2}{\|\mathbf{w}_0\|^2}$$

Facciamo ora queste altre considerazioni. Riscriviamo la regola di aggiornamento applicando la norma quadrata euclediana ad entrambi i lati ottenendo:

$$\|\mathbf{w}(n+1)\|^2 = \|\mathbf{w}(n)\|^2 + \|\mathbf{x}(n)\|^2 + 2 \cdot \mathbf{w}^T(n) \cdot \mathbf{x}(n)$$

Ora abbiamo presupposto che il percettrone classifica in modo errato gli input con n=1,2,... quindi, $\mathbf{w}^T(n) \cdot \mathbf{x}(n) < 0$ e di conseguenza:

$$\|\mathbf{w}(n+1)\|^2 \leq \|\mathbf{w}(n)\|^2 + \|\mathbf{x}(n)\|^2 \leq \sum_{i=1}^{n} \|\mathbf{x}(i)\|^2 \leq n \cdot \beta$$

dove $\beta>0$ è definito come:

$$\beta = \max_{x(n)\in X_1} \|\mathbf{x}(n)\|^2$$

Quindi il peso aggiornato $\mathbf{w}(n+1)$ è limitato superiormente ed inferiormente e di conseguenza esiste un numero massimo di iterazioni n_{max} tale per cui:

$$\frac{n_{\max}^2 \cdot \alpha^2}{\|\mathbf{w}_0\|^2} = \|\mathbf{w}(n+1)\|^2 = n_{\max} \cdot \beta$$

da cui ricaviamo n_{max}:

$$n_{\max} = \frac{\beta \cdot \|w_0\|^2}{\alpha^2}$$

Quindi abbiamo dimostrato che se X_1 e X_2 sono due insiemi linearmente separabili, il percettrone converge con al massimo n_{max} iterazioni utilizzando il metodo fixed increment adaptation.
Nel caso di $\eta(n)$ variabile, consideriamo $\eta(n)$ come il più piccolo intero per cui:

$$\eta(n) \cdot \mathbf{x}^T(n) \cdot \mathbf{x}(n) > \left|\mathbf{w}^T(n) \cdot \mathbf{x}(n)\right|$$

avremo che se all'interazione n $\mathbf{w}^T(n) \cdot \mathbf{x}(n)$ ha un segno errato, se ripresentiamo lo stesso input all'iterazione successiva avremo che il segno diventa corretto infatti:

1. se $\mathbf{w}^T(n) \cdot \mathbf{x}(n) \leq 0$ e $\mathbf{x}(n) \in X_1$ abbiamo:
   $$\mathbf{w}^T(n+1) \cdot \mathbf{x}(n) = \mathbf{w}^T(n) \cdot \mathbf{x}(n) + \eta(n) \cdot \mathbf{x}^T(n) \cdot \mathbf{x}(n) > 0$$
2. se $\mathbf{w}^T(n) \cdot \mathbf{x}(n) > 0$ e $\mathbf{x}(n) \in X_2$ abbiamo:
   $$\mathbf{w}^T(n+1) \cdot \mathbf{x}(n) = \mathbf{w}^T(n) \cdot \mathbf{x}(n) - \eta(n) \cdot \mathbf{x}^T(n) \cdot \mathbf{x}(n) < 0$$

In altre parole ogni pattern viene presentato più volte di seguito finché non viene classificato correttamente.

6.2 Algoritmo di apprendimento per percettrone

1. **Inizializzione**: si setta $\mathbf{w}(0)=\mathbf{0}$ ed n=0;

2. **Attivazione**: al tempo n si attiva il percettrone applicando l'input **x**(n) e la risposta desiderata d(n);
3. **Calcolo dell'output**: si calcola l'output secondo la sequente formula:
$y(n) = \text{sgn}\left(\mathbf{w}^T(n) \cdot \mathbf{x}(n)\right)$
dove sgn è la funzione segno definita come: $\text{sgn}(x) = \begin{cases} +1 & \text{se } x > 0 \\ -1 & \text{se } x \le 0 \end{cases}$
4. **Aggiornamento dei pesi**: si aggiornano i pesi secondo la seguente formula:
$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta \cdot \left[d(n) - y(n)\right] \cdot \mathbf{x}(n)$
dove $d(n) = \begin{cases} +1 & \text{se } x(n) \in X_1 \\ -1 & \text{se } x(n) \in X_2 \end{cases}$
5. **Continuazione**: si incrementa n e si torna a 2.

Il processo termina quando si è giunti ad uno step n_0 tale per cui $\mathbf{w}(n_0)=\mathbf{w}(n_0+1)=\mathbf{w}(n_0+2)=$ ... ovvero finché tutti i pattern sono classificati correttamente senza modifiche ai pesi.

Il parametro di apprendimento è una costante reale positiva limitata all'intervallo (0,1], tenendo conto di queste considerazioni:

- più piccola è η più si tiene in considerazione il peso passato, garantendo stime stabili, a scapito della velocità di convergenza;
- più è grande e maggiore è la velocità di convergenza.

6.3 Esempio di applicazione: operatori booleani

Abbiamo visto che il percettrone è in grado di classificare input che sono linearmente separabili. Possiamo costruire percettroni che risolvono l'AND, l'OR, il NOT, ma non possiamo costruire un percettrone che risolve lo XOR.

6.3.1 NOT

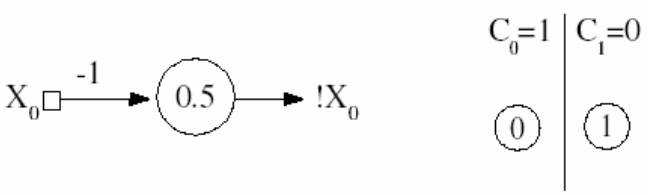

Figura 14: percettrone per operatore NOT

6.3.2 AND

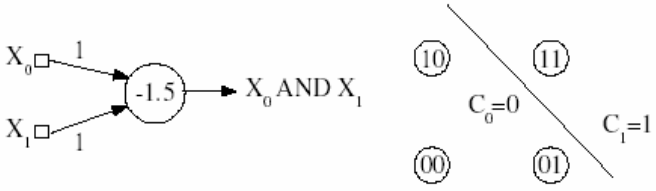

Figura 15: percettrone per operatore AND

6.3.3 OR

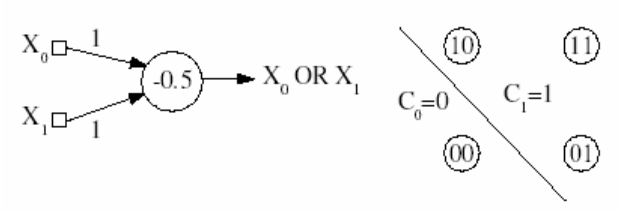

Figura 16: percettrone per operatore OR

6.3.4 XOR

Non è linearmente separabile infatti possiamo vedere dalla figura che necessitiamo di 2 rette per separare le 2 classi.

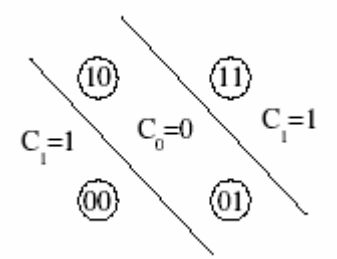

Figura 17: Non separabilità lineare dell'operatore XOR

7 Multilayer Perceptrons

La rete 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 3 caratteristiche distintive:

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

Un metodo efficiente per l'addestramento della rete è quello del **back-propagation**.

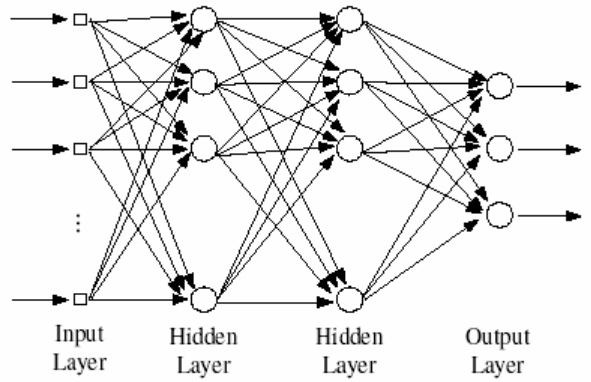

Figura 18: multilayer perceptrons

In questa rete troviamo 2 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 i layer 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 multilayer perceptron esegue 2 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.

7.1 Algoritmo di back-propagation

Distinguiamo 2 casi:
1. neurone del layer di output;
2. neurone di un hidden layer;

7.1.1 Neurone j di output

Il **segnale d'errore** del neurone di output j all'iterazione n (presentazione dell'n-esimo esempio del training set) è definito da

$$e_j(n) = d_j(n) - y_j(n)$$

dove
- $d_j(n)$ è l'output atteso del neurone j;
- $y_j(n)$ è l'output del neurone j.

L'**errore totale** dell'output layer presentando l'n-esimo esempio del training set è definito come

$$E(n) = \frac{1}{2} \cdot \sum_{j \in C} e_j^2(n)$$

dove C è l'insieme dei neuroni che formano l'output layer.

Sia N il numero totale di patterns contenuti nel training set. L'errore quadrato medio rappresenta la funzione costo ed è dato da

$$E_{AV} = \frac{1}{N} \cdot \sum_{i=1}^{N} E(n)$$

L'obiettivo dell'addestramento è quello di minimizzare E_{av} aggiustando opportunamente i parametri liberi della rete neurale.
Considereremo un metodo semplice che aggiorna i pesi di pattern in pattern fino al raggiungimento di un'epoca. Un'**epoca** è una presentazione completa dell'intero training set.
L'aggiustamento dei pesi viene fatto in base all'errore calcolato per ogni pattern presentato alla rete.
La media aritmetica di queste variazioni di peso su pattern singoli del training set sono una stima della variazione reale che risulterebbe da modifiche di peso che minimizzino la funzione costo E_{av} sull'intero training set.

Per minimizzare la funzione costo si adotta il metodo della discesa del gradiente. Nel nostro caso il gradiente è dato da $\frac{\partial E(n)}{\partial w_{ji}(n)}$ e gli aggiornamenti di peso avvengono in verso opposto al gradiente.

$$\Delta w_{ji}(n) = -\eta \cdot \frac{\partial E(n)}{\partial w_{ji}(n)}$$

dove η è il fattore di apprendimento.

Calcoliamo il gradiente:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

applicando la regola della catena abbiamo che il gradiente è pari al prodotto di 4 derivate parziali.

Calcoliamo ciascuna derivata:

$$\frac{\partial E(n)}{\partial e_j(n)} = \frac{1}{2} \cdot \sum_{k \in C} \frac{\partial e_k^2(n)}{\partial e_j(n)} = e_j(n)$$

dove dal momento che consideriamo un neurone di output abbiamo che j ∈ C.

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial \left(d_j(n) - y_j(n)\right)}{\partial y_j(n)} = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \varphi\left[v_j(n)\right]}{\partial v_j(n)} = \varphi'\left[v_j(n)\right]$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \sum_{k=0}^{m} y_k(n) \cdot \frac{\partial w_{jk}(n)}{\partial w_{ji}(n)} = y_i(n)$$

Da cui ricaviamo che

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -\delta_j(n) \cdot y_i(n)$$

dove $\delta_j(n)$ viene definito **gradiente locale** ed equivale a:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \cdot \varphi'\left[v_j(n)\right]$$

Quindi la variazione di peso sinaptico tra il neurone i e j relativo all'n-esimo esempio del training set è la seguente:

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$

7.1.2 Neurone j nascosto

Quando il neurone appartiene ad uno strato nascosto, non abbiamo una specifica risposta desiderata. Il segnale di errore deve essere determinato ricorsivamente dal segnale di errore di tutti i neuroni ai quali questo neurone nascosto è connesso.
Eseguiamo i calcoli considerando il nodo appartenente all'ultimo strato nascosto.

Ridefiniamo a questo proposito il gradiente locale come

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)} \cdot \varphi'\left[v_j(n)\right]$$

Calcoliamo la derivata:

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{1}{2} \cdot \sum_{k \in C} \frac{\partial e_k^2(n)}{\partial e_k(n)} \cdot \frac{\partial e_k(n)}{\partial y_k(n)} \cdot \frac{\partial y_k(n)}{\partial v_k(n)} \cdot \frac{\partial v_k(n)}{\partial y_j(n)} = -\sum_{k \in C} e_k(n) \cdot \varphi'\left[v_k(n)\right] \cdot w_{kj}(n)$$

dove trattandosi di un neurone nascosto j ∉ C.
A questo punto abbiamo che $\delta_k(n) = e_k(n) \cdot \varphi'\left[v_k(n)\right]$ e quindi:

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k \in C} \delta_k(n) \cdot w_{kj}(n)$$

Abbiamo supposto che j fosse collegato a neuroni di output ma in generale questa formula vale anche se j fosse collegato ad un altro strato nascosto. In questo caso però la sommatoria non è estesa ai neuroni di uscita ma ai neuroni dello strato successivo. Per fare questo possiamo considerare la sommatoria estesa a tutti i neuroni perché $w_{kj}=0$ se j non è adiacente a k; quindi in generale:

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k} \delta_k(n) \cdot w_{kj}(n)$$

A questo punto il gradiente locale del neurone nascosto j relativo all'n-esimo esempio del training set è

$$\delta_j(n) = \varphi'\left[v_j(n)\right] \cdot \sum_k \delta_k(n) \cdot w_{kj}(n)$$

7.1.3 In generale

In generale la variazione di peso sinaptico tra il neurone i e j relativo all'n-esimo esempio del training set è dato da:

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$

dove $\delta_j(n) = \begin{cases} e_j(n) \cdot \varphi'\left[v_j(n)\right] & \text{se j è un neurone di output} \\ \varphi'\left[v_j(n)\right] \cdot \sum_k \delta_k(n) \cdot w_{kj}(n) & \text{altrimenti} \end{cases}$

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 del training set alla volta, anche se questo è meno coerente al modello biologico. Per l'apprendimento off-line abbiamo la seguente funzione costo:

$$E = \frac{1}{2} \cdot \sum_n \sum_{j \in C} e_j^2(n)$$

da cui deriva la seguente regola di apprendimento:

$$\Delta w_{ji}(n) = \eta \cdot \sum_n \delta_j(n) \cdot y_i(n)$$

7.1.4 Fattore di apprendimento

La scelta del fattore di apprendimento influenza molto il comportamento dell'algoritmo, infatti se scegliamo valori troppo piccoli, la convergenza sarà lenta, mentre se scegliamo 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 è quello di modificare la regola di aggiornamento inserendo il **momento**.

$$\Delta w_{ji}(n) = \alpha \cdot \Delta w_{ji}(n-1) + \eta \cdot \delta_j(n) \cdot y_i(n)$$

dove α è un numero positivo.
Se espandiamo ricorsivamente la formula otteniamo

$$\Delta w_{ji}(n) = \eta \cdot \sum_{t=0}^{n} \alpha^{n-t} \cdot \delta_j(t) \cdot y_i(t)$$

ricordando che $\frac{\partial E(t)}{\partial w_{ji}(t)} = -\delta_j(t) \cdot y_i(t)$ abbiamo

$$\Delta w_{ji}(n) = -\eta \cdot \sum_{t=0}^{n} \alpha^{n-t} \cdot \frac{\partial E(t)}{\partial w_{ji}(t)}$$

che converge per valori di [0,1). Inoltre se la derivata parziale tende a mantenere lo stesso segno su iterazioni consecutive, grazie alla sommatoria, l'aggiornamento sarà per valori più ampi e quindi tende ad **accelerare nelle discese**. Se però la derivata ha segni opposti ad iterazioni consecutive la

sommatoria tende a diminuire l'ampiezza dell'aggiornamento facendo in modo di avere un **effetto stabilizzante** quando abbiamo oscillazioni.
Oltre a questo l'uso del momento ha il vantaggio di **prevenire** che il processo di apprendimento incappi in **minimi locali** della funzione d'errore.

7.1.5 Criteri d'arresto

Non esistono criteri d'arresto ben definiti; ne citiamo 3.

1. L'algoritmo di back-propagation può essere interrotto quando è nelle vicinanze di un minimo locale, ovvero quando la norma euclediana del vettore gradiente è inferiore di una soglia sufficientemente piccola.
2. L'algoritmo di back-propagation può essere interrotto quando la percentuale di variazione dell'errore quadrato medio tra due epoche consecutive è sufficientemente piccola.
3. L'algoritmo di back-propagation può essere interrotto quando la capacità di generalizzazione è adeguata.

Per il terzo punto vediamo più avanti nel dettaglio come si stima la capacità di generalizzazione di una rete.

7.1.6 Euristiche per migliorare l'algoritmo di back-propagation

Vediamo alcune regole che consentono di migliorare le prestazioni dell'algoritmo di back-propagation:

- Se abbiamo un training set grande e altamente ridondante è consigliato l'uso del metodo di back-propagation online.
- Ogni pattern scelto da presentare alla rete dovrebbe massimizzare il contenuto informativo ovvero essere radicalmente differente dagli esempi usati precedentemente e in generale portare la rete a sbagliare. In questo modo presentiamo alla rete un training set che cerca di coprire in modo completo lo spazio degli input.
- Se la rete viene addestrata con una funzione di attivazione antisimmetrica, questa apprende più velocemente. Una funzione è antisimmetrica se $\varphi(-x) = -\varphi(x)$. Un esempio ne è la tangente iperbolica definita come: $\varphi(x) = a \cdot \tanh(b \cdot x)$, dove a, b sono costanti. Buoni valori per le due costanti sono: a=1.7159; b=$\frac{2}{3}$.
- E' importante che la risposta desiderata venga scelta all'interno del codominio della funzione di attivazione perché altrimenti l'algoritmo tende a portare i parametri liberi della rete a infinito.
- Ogni vettore di input dovrebbe venire preprocessato in modo che la media calcolata sull'intero traning set tenda a 0 o altrimenti sia piccola se comparato con la sua deviazione standard. Se supponiamo vettori di input con componenti positive avremo che i pesi del primo layer nascosto si incrementano/decrementano insieme. Se il vettore dei pesi di un neurone deve cambiare direzione lo può fare solo zigzagando attraverso la superficie dell'errore, cosa che tipicamente è lenta e quindi va evitata. Questo non può accadere se gli input vengono normalizzati come precedentemente detto. Ulteriori normalizzazioni che possono essere fatte sono la scorrelazione delle variabili di input seguita da una scalatura in modo che le loro covarianze siano approssimativamente uguali. In questo modo ci si assicura che i diversi pesi sinaptici della rete apprendano approssimativamente alla stessa velocità.
- Se inizializziamo i pesi sinaptici con valori grandi avremo che la rete giunge alla saturazione; se ciò accade il gradiente locale assumerebbe valori piccoli che rallenterebbero il processo di apprendimento. Se viceversa i pesi fossero troppo piccoli, l'algoritmo si troverebbe a dover operare su una area molto piatta attorno all'origine della superficie d'errore e sfortunatamente l'origine è un punto stazionario né di minimo né di massimo.

Vediamo allora come inizializzare la rete facendo le seguenti considerazioni. Supponiamo di avere una rete con funzione di attivazione la tangente iperbolica, e con valore soglia settata a 0, quindi $w_{j0}=0$, $\forall$ j.
Esprimiamo l'input netto di un neurone come:

$$v_j = \sum_{i=1}^{m} w_{ji} \cdot y_i$$

dove m è il numero di neuroni della rete.
Supponiamo inoltre che :

- la media degli i-esimi componenti dell'input y_i di ciascun neurone sia nulla e la loro varianza sia unitaria:

$$\mu_y = E[y_i] = 0, \quad \forall i$$
$$\sigma_y^2 = E[(y_i - \mu_y)^2] = E[y_i^2] = 1, \quad \forall i$$

- gli input siano non correlati con gli altri input e con i pesi sinaptici e poniamo:

$$E[y_i \cdot y_k] = \begin{cases} 1 & \text{se } k = j \\ 0 & \text{se } k \neq j \end{cases}$$

- i pesi siano scelti da un insieme uniformemente distribuito con media zero

$$\mu_w = E[w_{ji}] = 0, \quad \forall (i, j)$$

  e varianza

$$\sigma_w^2 = E[(w_{ji} - \mu_w)^2] = E[w_{ji}^2], \quad \forall (i, j)$$

Definiamo a questo punto la media e la varianza dell'input netto v_j .

$$\mu_v = E[v_j] = \sum_{i=1}^{m} E[w_{ji} \cdot y_i] = \sum_{i=1}^{m} E[w_{ji}] \cdot E[y_i]$$

dove $E[w_{ji} \cdot y_i] = E[w_{ji}] \cdot E[y_i]$ in quanto scorrelati.

$$\sigma_v^2 = E[(v_j - \mu_v)^2] = E[v_j^2] = \sum_{i=1}^{m} \sum_{k=1}^{m} E[w_{ji} \cdot w_{jk} \cdot y_i \cdot y_k]$$

$$= \sum_{i=1}^{m} \sum_{k=1}^{m} E[w_{ji} \cdot w_{jk}] \cdot E[y_i \cdot y_k] = \sum_{i=1}^{m} E[w_{ji}^2] = m \cdot \sigma_w^2$$

dove $E[w_{ji} \cdot w_{jk} \cdot y_i \cdot y_k] = E[w_{ji} \cdot w_{jk}] \cdot E[y_i \cdot y_k]$ in quanto i pesi e gli input sono scorrelati.
Ora vogliamo che la deviazione standard dell'input netto stia nella zona di transazione tra l'area lineare e quella satura della funzione di attivazione. Poniamo quindi $\sigma_v = 1$ quindi abbiamo che

$$m \cdot \sigma_w^2 = 1$$
$$\sigma_w = \frac{1}{\sqrt{m}}$$

Anziché vedere m come il numero di neuroni consideriamolo come numero di connessioni sinaptiche della rete.
Quindi è desiderabile che i pesi sinaptici siano scelti da una distribuzione uniforme che abbia media zero e varianza pari al reciproco del numero di connessioni sinpatiche della rete neurale.

7.2 Generalizzazione

Una rete neurale **generalizza** bene se la mappa input/output che genera la rete è corretta (o quasi) per esempi di test mai presentati alla rete in fase di training. Si assume che gli esempi di test siano tratti dalla stessa popolazione usata per generare il training set.

Il problema di apprendimento può essere visto come un problema di "approssimazione di una curva". La rete di per sé può essere considerata semplicemente una mappa input/output non lineare, quindi una buona generalizzazione può essere 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 addestra con troppi esempi, la rete rischia di memorizzare il training set. Questo fenomeno è detto **overfitting** o **overtraining**. Una rete "sovra-addestrata" perde la capacità di generalizzazione.

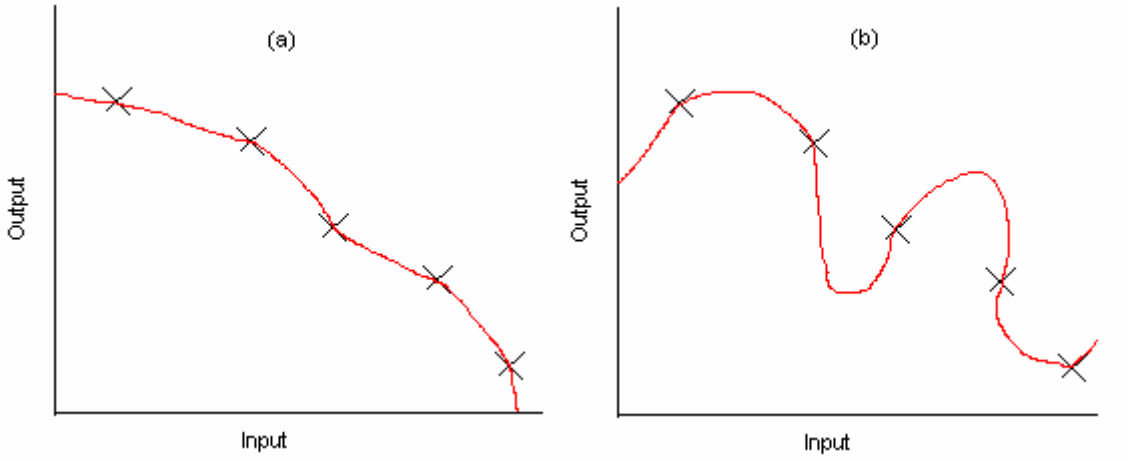

Figura 19: Esempio di buona generalizzazione (a), e di una cattiva generalizzazione (overfitted) (b)

La generalizzazione è influenzata da 3 fattori:

1. le dimensioni del training set;
2. l'architettura della rete neurale;
3. la complessità del problema.

Alla luce del fatto che non abbiamo controllo sul punto (3), possiamo vedere il problema della generalizzazione da 2 punti di vista:

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.

Vediamo un'euristica per il punto (1).
Nella pratica si è visto che si ha una buona generalizzazione, se la dimensione del training set N soddisfa la condizione

$$N = O\left(\frac{W}{\varepsilon}\right)$$

dove W è il numero totale di parametri liberi nella rete e ε denota la frazione di errori di classificazione permessi sui dati di test.

7.3 Approssimazione di funzioni

Un percettrone multilayer addestrato con back-propagation può essere visto come un approssimatore di funzioni universale.

Teorema dell'approssimazione universale: sia φ una funzione continua, non costante, limitata e monotona crescente. Sia I_{m_0} un ipercubo unitario m_0-dimensionale $[0,1]^{m_0}$. Lo spazio delle funzioni

continue su I_{m_0} lo denotiamo con $C(I_{m_0})$. Allora dati una qualunque funzione f $\in$ $C(I_{m_0})$ e ε>0, esiste un intero m_1 e insiemi di costanti reali α_i, β_i e w_{ij}, dove i= 1, …, m_1 e j=1, …, m_0 tale che possiamo definire

$$F(x_1,\ldots,x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \cdot \varphi\left(\sum_{j=1}^{m_0} w_{ij} \cdot x_j + b_i \right)$$

come una realizzazione approssimata della funzione f; ovvero

$$\left| F(x_1,\ldots,x_{m_0}) - f(x_1,\ldots,x_{m_0}) \right| < \varepsilon$$

per ogni $x_1,\ldots x_{m_0}$ che giace nello spazio di input.

Il teorema è un teorema di esistenza perché non ci dice come calcolare l'insieme di costanti necessari a creare F.
Il teorema dice quindi che uno strato nascosto è sufficiente ad un percettrone multilayer per una ε-approssimazione di un generico training set rappresentato dall'insieme di inputs $x_1,\ldots x_{m_0}$ e outputs desiderati $f(x_1,\ldots x_{m_0})$.

7.4 Cross-validation

Vediamo 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 ripartito in 2 insiemi disgiunti:

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

La motivazione è quella di validare il modello su un set di dati diverso da quello usato per l'addestramento. In questo modo possiamo 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 è incappato in un overfitting del validation set. Per evitare questo si ricorre al test set per verificare la capacità di generalizzazione della rete.

7.4.1 Selezione del modello

Il problema della selezione del modello è essenzialmente quello di scegliere il percettrone con il miglior valore di W, il numero di parametri liberi. Più precisamente posto che la riposta desiderata relativa ad un input **x** sia d={0,1}, definiamo l'**errore di generalizzazione** come

$$\varepsilon_g(F) = P(F(\mathbf{x}) \neq d), \quad \mathbf{x} \in X$$

Supponiamo di avere a disposizione il seguente training set

$$Ts = \{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$$

L'obiettivo è di selezionare la particolare funzione F(**x**,**w**) che minimizza l'errore di generalizzazione che risulta presentando alla rete inputs dal test set.

Per ogni dimensione del test set N, possiamo sempre trovare un percettrone multilayer con un numero sufficiente di parametri liberi $W_{max}(N)$ tale per cui il training set T_S è interpolato adeguatamente. Chiamiamo $W_{max}(N)$ **fitting number**. Una procedura ragionevole di selezione del modello sceglierebbe una funzione F(**x**,**w**) che richiede $W \leq W_{max}(N)$.

Sia r $\in$ [0,1] un parametro che determina il partizionamento di T_S nell'estimation subset e validation subset. L'estimation subset denotato con T_S' è usato per addestrare una sequenza di percettroni multilayer di complessità crescente con un numero di parametri liberi $W \le W_{\max}\left[(1-r)\cdot N\right]$.
L'uso del cross-validation porta alla scelta

$$F_{cv} = \min_{k=1,2,\ldots,v}\left\{e_t''(F_k)\right\}$$

dove v corrisponde a $W_v \le W_{\max}\left[(1-r)\cdot N\right]$ e $e_t''(F_k)$ è l'errore di classificazione prodotto da F_k se testato sul validation subset T_S'', consistente in $r \cdot N$ esempi.
Il problema che ci poniamo ora è come determinare il parametro r. Identifichiamo diverse proprietà qualitative del valore ottimo di r:

- se la complessità della funzione target è piccola comparata con il numero di esempi N, la prestazione del cross-validation non è influenzata dalla scelta di r;
- se la complessità della funzione target è maggiore comparata col numero di esempi N, la scelta di r ha un effetto pronunciato sulle prestazioni del cross-validation;
- un valore di r fisso opera quasi in maniera ottimale per un largo numero di funzioni target complesse.

Un valore fisso di r che risulta essere buono è 0.2, ovvero 80% estimation subset, 20% validation subset.

7.4.2 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 di training se non si ferma l'addestramento al punto giusto.
Possiamo evitare il fenomeno dell'overfitting ricorrendo al metodo di cross-validation; 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.

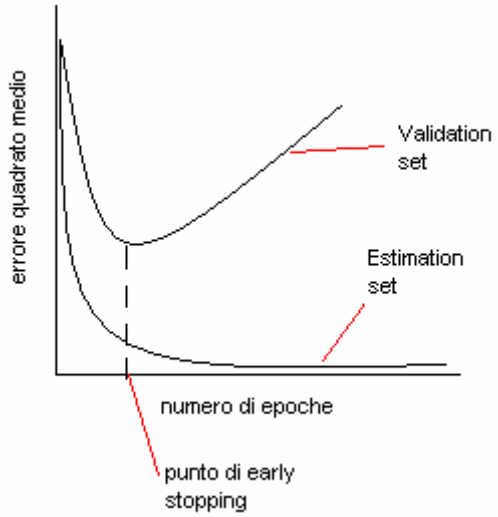

Figura 20: metodo dell' "early-stopping" basato sul cross-validation

Se guardiamo la sola curva dell'errore quadrato medio dell'estimation set, 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 però ciò che la rete apprende dopo quel punto è il rumore contenuto nei dati dell'estimation set. Questa euristica suggerisce quindi di fermare l'addestramento in corrispondenza del minimo della curva relativa al validation set.

Una teoria statistica sul fenomeno dell'overfitting mette in guardia dall'utilizzo del metodo dell'"early stopping". Vengono distinti 2 casi dipendenti dalla dimensione del training set:

- **modalità non asintotica**: N<W, dove N è la dimensione del training set e W è il numero di parametri liberi. L'utilizzo dell'"early stopping" migliora la capacità di generalizzazione rispetto al training esaustivo. L'overfitting potrebbe verificarsi quando $N < 30 \cdot W$ e ci sono vantaggi pratici ad usare il cross-validation per bloccare il training. Il valore ottimo per il parametro r è definito da:

$$r_{opt} = 1 - \frac{\sqrt{2 \cdot W - 1} - 1}{2 \cdot (W - 1)}$$

  Per valori grandi di W questa formula si approssima a

$$r_{opt} \approx 1 - \frac{1}{\sqrt{2 \cdot W}}$$

- **modalità asintotica**: $N > 30 \cdot W$. I miglioramenti nella capacità di generalizzazione sono scarsi rispetto al training esaustivo, in altre parole il training esaustivo è soddisfacente se la dimensione del training set è elevata se comparata col numero di parametri liberi.

7.4.3 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 richede molta computazione dal momento che il modello deve essere addestrato K volte.
Quando il numero di esempi è piccolo si può ricorrere ad una forma estrema di questa tecnica detta **leave-one-out** in cui K=N.

7.5 Tecniche di network pruning

Le tecniche di network pruning hanno lo scopo di minimizzare le dimensioni della rete mantenendo buone prestazioni. Una rete più piccola non ha la tendenza ad apprendere il rumore nei dati e generalizza meglio di dati nuovi. Possiamo raggiungere questo obiettivo in 2 modi:

1. **Network growing**: dove partiamo da un percettrone multilayer piccolo e lo espandiamo;
2. **Network pruning**: dove partiamo da un percettrone multilayer sufficientemente grande per poi ridurre la rete eliminando connessioni sinaptiche o neuroni interi.

Analizzeremo techiche relative al punto (2).

7.5.1 OBS (Optimal Brain Surgeon)

Utilizziamo l'informazione della derivata seconda della superficie dell'errore per predire gli effetti dovuti a perturbazioni nei pesi sinaptici della rete.
L'approssimazione locale della funzione costo E_{av} nel punto **w** usando un espansione di Taylor fino al secondo ordine è data da

$$E_{av}(\mathbf{w}+\Delta\mathbf{w}) = E_{av}(\mathbf{w}) + g^T(\mathbf{w})\cdot\Delta\mathbf{w} + \frac{1}{2}\cdot\Delta\mathbf{w}^T\cdot H(\mathbf{w})\cdot\Delta\mathbf{w}$$

dove $\Delta\mathbf{w}$ è la perturbazione applicata al punto $\mathbf{w}$ (il vettore di pesi sinaptici della rete), g($\mathbf{w}$) è il vettore gradiente valutato nel punto $\mathbf{w}$ e infine H($\mathbf{w}$) è l'hessiano valutato nel punto $\mathbf{w}$.
Lo scopo è quello di trovare un insieme di parametri la cui eliminazione dal percettrone multilayer causi il minor incremento nella funzione costo. Per risolvere il problema introduciamo delle approssimazioni:

1. assumiamo che il vettore dei pesi $\mathbf{w}$ sia il risultato di una fase di training completa, e corrisponda quindi ad un minimo locale o globale sulla superficie dell'errore. Quindi il gradiente è nullo in quel punto e quindi lo è anche il termine $g^T(\mathbf{w})\cdot\Delta\mathbf{w}$;
2. assumiamo che la superficie dell'errore attorno ad un minimo locale abbia circa un andamento quadratico, in modo da poter giustificare l'espansione di Taylor fino al secondo ordine.

Sotto queste assunzioni abbiamo che

$$\Delta E_{av} = E_{av}(\mathbf{w}+\Delta\mathbf{w}) - E_{av}(\mathbf{w}) = \frac{1}{2}\cdot\Delta\mathbf{w}^T\cdot H(\mathbf{w})\cdot\Delta\mathbf{w}$$

L'obiettivo del OBS è di settare un peso sinaptico a 0 e minimizzare l'incremento prodotto nella funzione costo. Sia $w_i(n)$ questo particolare peso la cui eliminazione corrisponde alla condizione

$$\Delta w_i + w_i = 0 \quad \text{oppure} \quad \mathbf{1}_i^T\cdot\Delta\mathbf{w} + w_i = 0$$

dove $\mathbf{1}_i$ è un vettore di tutti 0 tranne l'i-esima componente che vale 1.
Dobbiamo a questo punto risolvere un problema di minimo vincolato. Per questo ricorriamo al Lagrangiano:

$$S = \frac{1}{2}\cdot\Delta\mathbf{w}^T\cdot H(\mathbf{w})\cdot\Delta\mathbf{w} - \lambda\cdot(\mathbf{1}_i^T\cdot\Delta\mathbf{w} + w_i)$$

dove λ è il moltiplicatore di Lagrange.
Per trovare il valore di $\Delta\mathbf{w}$ dobbiamo risolvere il seguente sistema:

$$\begin{cases}\dfrac{\partial S}{\partial \mathbf{w}} = 0\\ \dfrac{dS}{d\lambda} = 0\end{cases} \Rightarrow \begin{cases}H(\mathbf{w})\cdot\Delta\mathbf{w} - \lambda\cdot\mathbf{1}_i = 0\\ \mathbf{1}_i^T\cdot\Delta\mathbf{w} + w_i = 0\end{cases} \Rightarrow \begin{cases}\Delta\mathbf{w} = \lambda\cdot H^{-1}(\mathbf{w})\cdot\mathbf{1}_i\\ \lambda\cdot\mathbf{1}_i^T\cdot H^{-1}(\mathbf{w})\cdot\mathbf{1}_i + w_i = 0\end{cases}$$

a questo punto abbiamo che $\mathbf{1}_i^T\cdot H^{-1}(\mathbf{w})\cdot\mathbf{1}_i = \left[H^{-1}(\mathbf{w})\right]_{i,i}$ ovvero l'elemento in posizione (i,i) dell'hessiano inverso

$$\Rightarrow \quad \begin{cases}\Delta\mathbf{w} = \dfrac{-w_i}{\left[H^{-1}(\mathbf{w})\right]_{i,i}}\cdot H^{-1}(\mathbf{w})\cdot\mathbf{1}_i\\ \lambda = \dfrac{-w_i}{\left[H^{-1}(\mathbf{w})\right]_{i,i}}\end{cases}$$

e il corrispondente valore ottimo della Langragiana S per l'elemento w_i è

$$S_i = \frac{1}{2}\cdot\Delta\mathbf{w}^T\cdot H(\mathbf{w})\cdot\Delta\mathbf{w} = \frac{w_i^2}{2\cdot\left[H^{-1}(\mathbf{w})\right]_{i,i}^2}\cdot\mathbf{1}_i^T\cdot\left[H^{-1}(\mathbf{w})\right]^T\cdot H(\mathbf{w})\cdot H^{-1}(\mathbf{w})\cdot\mathbf{1}_i$$

$$= \frac{w_i^2}{2\cdot\left[H^{-1}(\mathbf{w})\right]_{i,i}^2}\cdot\mathbf{1}_i^T\cdot\left[H^{-1}(\mathbf{w})\right]^T\cdot\mathbf{1}_i$$

a questo punto abbiamo che $\mathbf{1}_i^T\cdot\left[H^{-1}(\mathbf{w})\right]^T\cdot\mathbf{1}_i = \left\{\left[H^{-1}(\mathbf{w})\right]^T\right\}_{i,i} = \left[H^{-1}(\mathbf{w})\right]_{i,i}$ e quindi

$$S_i = \frac{w_i^2}{2\cdot\left[H^{-1}(\mathbf{w})\right]_{i,i}}$$

Il Lagrangiano S_i è detto **saliency** (importanza) di w_i ovvero l'incremento di errore quadratico medio risultato dall'eliminazione del peso w_i .
Il peso che ha la più piccola saliency è quello che verrà eliminato.

La complessità del metodo è $O(M \cdot v^2)$ dove M è la cardinalità del training set e v è il numero di pesi della rete.

7.5.2 CFP

A differenza del metodo OBS è un metodo che agisce localmente al neurone di cui si vuole eliminare un peso sinaptico.
Supponiamo di avere una rete addestrata con uno strato hidden e supponiamo di voler eliminare il peso w_{hi} che esce dal neurone h ed entra nel neurone i. L'idea del metodo è quella di rimuovere la connessione sinaptica e aggiustare i rimanenti pesi w_{ji}, $\forall$ j $\neq$ h in modo tale da mantenere inalterato l'input netto del neurone i.
Definiamo R_i il **campo recettivo** del neurone i, ovvero l'insieme di neuroni con connessione entrante in i.
L'input netto del neurone i a seguito della presentazione alla rete di un generico pattern μ prima della rimozione del peso w_{hi} è dato da

$$v_i^{(\mu)} = \sum_{j \in R_i} w_{ji} \cdot y_j^{(\mu)}$$

Vogliamo a questo punto che l'input netto del neurone i dopo la rimozione del peso sia uguale a $v_i^{(\mu)}$ perturbando opportunamente i pesi sinaptici

$$v_i^{(\mu)} = \sum_{j \in R_i} w_{ji} \cdot y_j^{(\mu)} = \sum_{j \in R_i / \{h\}} \left(w_{ji} + \Delta w_{ji}\right) \cdot y_j^{(\mu)}, \quad \forall \eta \in Ts$$

dove Ts indica il training set.
A questo punto possiamo trovare i valori delle perturbazioni risolvendo il seguente sistema

$$\sum_{j \in R_i} w_{ji} \cdot y_j^{(\mu)} = \sum_{j \in R_i / \{h\}} \left(w_{ji} + \Delta w_{ji}\right) \cdot y_j^{(\mu)}$$

$$w_{hi} \cdot y_h^{(\mu)} = \sum_{j \in R_i / \{h\}} \Delta w_{ji} \cdot y_j^{(\mu)}$$

Possiamo riscriverlo in forma compatta ponendo:

- $\mathbf{z}_h = w_{hi} \cdot \mathbf{y}_h$, dove $\mathbf{y}_h$ è un vettore colonna che ha come i-esima componente $y_h^{(\mu_i)}$ con μ_i l'i-esimo pattern del training set;
- $\mathbf{\Delta w}_i$, vettore colonna dei pesi sinaptici entranti nell'i-esimo neurone escluso w_{hi};
- $\mathbf{Y}_h$, matrice che ha come (i,j)-esima componente $y_j^{(\mu_i)}$ con μ_i l'i-esimo pattern del training set, dove escludiamo però il caso j=h.

$$\mathbf{Y}_h \cdot \mathbf{\Delta w}_i = \mathbf{z}_h$$

Questo sistema può essere risolto ai minimi quadrati cioè trovando il vettore $\mathbf{\Delta w}_i$ che minimizza

$$\left\| \mathbf{z}_h - \mathbf{Y}_h \cdot \mathbf{\Delta w}_i \right\|^2$$

oppure ricorrendo a metodi di riduzione dei residui che partono da un valore x_0 iniziale e calcolano il residuo con la formula $r_i = \left\| \mathbf{A} \cdot \mathbf{x}_i - \mathbf{b} \right\|^2$, quindi trovano un nuovo valore $\mathbf{x}_{i+1}$ tale per cui $r_{i+1} < r_i$.
Altrimenti si può ricorrere a una soluzione euristica che cerca semplicemente il neurone h tale per cui lo scarto r_0 è minimo il quale, scegliendo $\mathbf{x}_0 = \mathbf{0}$, vale $r_0 = \left\| \mathbf{z}_h \right\|^2$.

Quindi l'algoritmo non fa altro che scegliere la connessione che minimizza r_0 (supponiamo che entri nel neurone i) la elimina, aggiusta i pesi rimanenti entranti in i e itera il procedimento finché non si raggiungono le prestazioni desiderate

La complessità del metodo è $O(M \cdot r_i)$ con M la cardinalità del training set e r_i il numero di connessioni entranti nel neurone i.

Una variante del metodo può essere quella di rimuovere neuroni interi, basta fare in modo che gli input netti entranti nello strato successivo non cambino.

8 Neurodinamica

8.1 Introduzione

Un modo implicito per introdurre in una rete neurale la variabile tempo è mediante l'utilizzo di reti con feedback o ricorrenti. Ci sono 2 modi per rendere ricorrente una rete:

1. introducendo **feedback locale** che interessa solamente il singolo neurone;
2. introducendo **feedback globale** che interessa l'intera rete.

Porremo l'attenzione sulle reti di tipo (2).

Lo studio delle reti neurali viste come sistemi dinamici non lineari, con particolare attenzione al problema della loro stabilità è chiamato **neurodinamica**.

Nel contesto dei sistemi dinamici non lineari quando parliamo di stabilità ci riferiamo alla stabilità vista in senso di Lyapuanov, vedremo in seguito di cosa si tratta.

Lo studio della neurodinamica segue due strade a seconda dell'interesse applicativo:

1. **neurodinamica deterministica**: nella quale la rete neurale ha un comportamento deterministico ed è descritto attraverso un insieme di equazioni differenziali non lineari;
2. **neurodinamica statistica**: nella quale la rete neurale è perturbata dalla presenza di rumore e il sistema è descritto da un insieme di equazioni differenziali non lineari stocastiche.

8.2 Sistemi dinamici

Un modello matematico per descrivere le dinamiche di un sistema non lineare è quello dello **spazio degli stati**. Con questo modello pensiamo 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 $\mathbf{x}(t)=\{x_1(t), x_2(t), \ldots, x_N(t)\}$ un vettore contenente le variabili di stato di un sistema dinamico non lineare detto **vettore di stato**, in cui la variabile indipendente è il tempo t e N è l'ordine del sistema. Possiamo allora descrivere un largo numero di sistemi dinamici non lineari mediante un sistema di equazioni differenziali di primo ordine scritte come segue:

$$\frac{d}{dt} \cdot \mathbf{x}(t) = \mathbf{F}\big(\mathbf{x}(t)\big)$$

dove $\mathbf{F}(\mathbf{x})$ è una **funzione vettore** che se applicata ad un vettore **x** ritorna un vettore che ha come j-esima componente $F_j(x_j)$ con F_j una qualche funzione. Un sistema in cui la funzione vettore **F** non dipende esplicitamente dal tempo è detto **autonomo** altrimenti è **non autonomo**. Noi tratteremo solo sistemi autonomi.

Possiamo definire un **sistema dinamico**, un sistema il cui stato varia nel tempo.

Inoltre possiamo pensare al vettore $\frac{d}{dt}\cdot\mathbf{x}(t)$ come ad un **vettore di velocità.**

8.2.1 Spazio degli stati

Possiamo vedere l'equazione dello spazio degli stati $\frac{d}{dt}\cdot\mathbf{x}(t)=\mathbf{F}\left(\mathbf{x}(t)\right)$ 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**(**x**) è definita; nel caso di sistemi autonomi per ogni punto dello spazio esiste una sola traiettoria che vi passa.

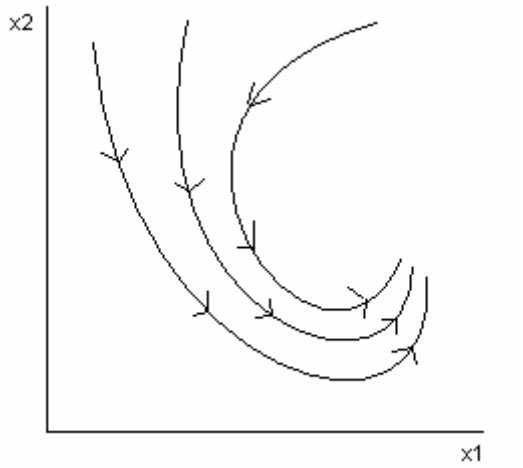

Figura 21: un ritratto degli stati 2-dimensionale di un sistema dinamico

8.2.2 Condizione di Lipschitz

Affinché l'equazione dello spazio degli stati abbia soluzione e questa sia unica dobbiamo imporre alcune restrizioni sul vettore **F**(**x**). Condizione sufficiente affinché vi sia soluzione è che **F**(**x**) sia continua in tutti i suoi argomenti; tuttavia questa condizione non garantisce l'unicità della soluzione. Dobbiamo quindi introdurre una restrizione aggiuntiva conosciuta come **condizione di Lipschitz**.
Sia $\|\mathbf{x}\|$ la norma euclediana del vettore **x**. Siano **x** e **u** una coppia di vettori di un insieme aperto M in uno spazio vettoriale degli stati. Allora in base alla condizione di Lipschitz, esiste una costante K tale che

$$\|\mathbf{F}(\mathbf{x})-\mathbf{F}(\mathbf{u})\|\le K\cdot\|\mathbf{x}-\mathbf{u}\|$$

per ogni **u** e **x** in M. Una funzione vettore **F**(**x**) che soddisfa la condizione è detta Lipschitz e K è detta **costante di Lipschitz** per **F**(**x**). La condizione di Lipschitz garantisce l'esistenza e l'unicità della soluzione per l'equazione dello spazio degli stati. In particolare se tutte le derivate parziali $\frac{\partial F_i}{\partial x_i}$ sono finite ovunque, allora la funzione **F**(**x**) soddisfa la condizione di Lipschitz.

8.3 Stabilità degli stati di equilibrio

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

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

dove **0** è il vettore nullo. Il vettore velocità si annulla nel punto di equilibrio e quindi $\mathbf{x}(t) = \bar{\mathbf{x}}$ 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.

8.3.1 Definizioni di stabilità

Definizione 1: Lo stato di equilibrio $\bar{\mathbf{x}}$ è detto **uniformemente stabile** se per ogni ε positivo, esiste un δ positivo tale che la condizione

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

implica

$$\left\|\mathbf{x}(t) - \bar{\mathbf{x}}\right\| < \varepsilon$$

per ogni t>0.

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

Definizione 2: Lo stato di equilibrio $\bar{\mathbf{x}}$ è detto **convergente** se esiste un δ positivo tale che la condizione

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

implica che

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

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

Definizione 3: Lo stato di equilibrio $\bar{\mathbf{x}}$ è detto **asintoticamente stabile** se è stabile e convergente.

Definizione 4: Lo stato di equilibrio $\bar{\mathbf{x}}$ è detto **globalmente asintoticamente stabile** se è stabile e tutte le traiettorie del sistema convergono a $\bar{\mathbf{x}}$ per t che tende ad infinito.

8.3.2 Teorema di Lyapunov

Teorema 1: Lo stato di equilibrio $\bar{\mathbf{x}}$ è **stabile** se in un piccolo intorno di $\bar{\mathbf{x}}$ esiste una funzione definita positiva V(**x**) tale che la sua derivata ripetto al tempo è minore o uguale a 0 in quella regione.

Teorema 2: Lo stato di equilibrio $\bar{\mathbf{x}}$ è **asintoticamente stabile** se in un piccolo intorno di $\bar{\mathbf{x}}$ esiste una funzione definita positiva V(**x**) tale che la sua derivata rispetto al tempo 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 $\bar{\mathbf{x}}$.
La funzione V(**x**) è **definita positiva** nello spazio degli stati L, se ∀ **x** ∈ L, soddisfa le seguenti proprietà:

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

3. $V(\mathbf{x})>0$ se $\mathbf{x} \neq \bar{\mathbf{x}}$.

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

$$\frac{d}{dt} \cdot V(\mathbf{x}) \leq 0, \quad \text{se} \left\| \mathbf{x} - \bar{\mathbf{x}} \right\| < \varepsilon$$

con ε un numero positivo piccolo.
Lo stato di equilibrio è asintoticamente stabile se

$$\frac{d}{dt} \cdot V(\mathbf{x}) < 0, \quad \text{se} \left\| \mathbf{x} - \bar{\mathbf{x}} \right\| < \varepsilon$$

Il teorema non da indicazione su come trovare una funzione di Lyapunov e la sua esistenza è sufficiente ma non necessaria per la stabilità.

8.4 Attrattori

Lo spazio degli stati di sistemi dinamici non lineari sono caratterizzati dalla presenza di **attrattori**. Questi possono essere **punti** o orbite periodiche (**cicli**) a cui le vicine traiettorie tendono all'aumentare del tempo.
Gli attrattori rappresentano gli unici stati di equilibrio di un sistema dinamico osservabili sperimentalmente. Facciamo notare che uno stato di equilibrio non implica un equilibrio statico, ne è prova il fatto che possono esserci cicli in cui lo stato varia nel tempo.
Un attrattore è racchiuso in una regione distinta detta **bacino di attrazione** e qualunque traiettoria abbia come stato iniziale un punto del bacino di attrazione, tende all'attrattore che questo contiene.

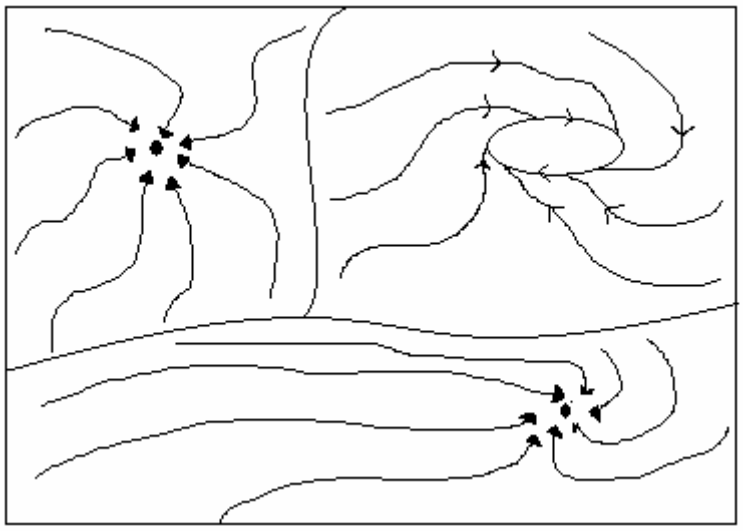

Figura 22: esempi di attratori puntuali e ciclici.

8.5 Reti di Hopfield

Le reti di Hopfield sono:
- **reti ricorrenti** ad uno strato in cui ogni neurone è connesso a tutti gli altri (sono assenti connessioni con sé stesso);
- **simmetriche**: perché hanno la matrice dei pesi sinaptici simmetrica, quindi $w_{ji} = w_{ij}$, $\forall i,j$;
- **non lineari**: ogni neurone ha una funzione di attivazione non lineare invertibile.

Possiamo distinguere diverse tecniche di aggiornamento dello stato dei neuroni:
- **aggiornamento asincrono:** in cui si aggiorna un neurone alla volta;
- **aggiornamento sincrono:** tutti i neuroni vengono aggiornati allo stesso istante;
- **aggiornamento continuo:** in cui tutti i neuroni si aggiornano continuamente.

Vedremo inoltre le reti di Hopfield nel caso discreto con aggiornamento asincrono e nel caso continuo con aggiornamento continuo.

8.5.1 Caso discreto

Nel caso discreto si usano neuroni di McCulloch-Pitts in cui però l'input netto del generico neurone i è dato da

$$v_i = \sum_{j \neq i} w_{ij} \cdot x_j + I_i$$

dove x_j è lo stato del neurone j e I_i è una corrente esterna (input esterno) applicata al neurone i. Se I_i=0 abbiamo il tradizionale neurone di McCulloch-Pitts.

Ogni neurone aggiorna il proprio stato in modo asincrono nel seguente modo

$$x_i(t) = \begin{cases} -1 & \text{se } v_i(t) < 0 \\ x_i(t-1) & \text{se } v_i(t) = 0 \\ +1 & \text{se } v_i(t) > 0 \end{cases}$$

Abbiamo detto la matrice dei pesi deve essere simmetrica perché questa è una condizione sufficiente alla convergenza, in quel caso infatti il sistema ha una funzione di energia di Lyapunov che viene minimizzata con l'evolvere del processo. Inoltre abbiamo detto che non ci devono essere autoloop, quindi settiamo $w_{ii} = 0$, $\forall$ i

La funzione quadratica di energia è la seguente:

$$E(t) = -\frac{1}{2} \cdot \mathbf{x}(t)^T \cdot \mathbf{W} \cdot \mathbf{x}(t) - \mathbf{x}(t)^T \cdot \mathbf{I}$$

dove **x** è il vettore di stato del sistema, **W** è la matrice simmetrica dei pesi con gli elementi diagonali nulli e **I** e il vettore di input esterno.

Possiamo verificare che la funzione è strettamente monotona decrescente verificando che

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

Abbiamo detto che i neuroni si aggiornano in modo asincrono, quindi supponiamo sia h il neurone che ha cambiato stato, allora:

$$\Delta E(t) = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot x_i(t+1) \cdot x_j(t+1) - \sum_i I_i \cdot x_i(t+1) + \frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot x_i(t) \cdot x_j(t) + \sum_i I_i \cdot x_i(t)$$

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

$$x_i(t+1) = x_i(t), \quad \forall i \neq h$$

quindi

$$\Delta E(t) = -\frac{1}{2} \cdot \left\{ \sum_j w_{hj} \cdot \left[x_h(t+1) \cdot x_j(t+1) - x_h(t) \cdot x_j(t) \right] + \sum_j w_{jh} \cdot \left[x_h(t+1) \cdot x_j(t+1) - x_h(t) \cdot x_j(t) \right] \right\} -$$

$$+ I_h \cdot \Delta x_h(t) = -\sum_j w_{hj} \cdot x_j(t) \cdot \Delta x_h(t) - I_h \cdot \Delta x_h(t) = -\Delta x_h(t) \cdot \left[\sum_j w_{kj} \cdot x_j(t) + I_h \right] = -\Delta x_h(t) \cdot v_h(t+1) < 0$$

dove sfruttando la simmetricità della matrice dei pesi e il fatto che ha gli elementi diagonali nulli, abbiamo che le sommatorie del primo passaggio sono uguali; nel terzo passaggio abbiamo sfruttato il fatto che $v_h(t+1) = \sum_{j \neq h} w_{hj} \cdot x_j(t) + I_h$ e $\Delta x_h(t) = x_h(t+1) - x_h(t)$ e nell'ultimo passaggio $-\Delta x_h(t) \cdot v_h(t+1)$ è sempre negativo in quanto per la regola di aggiornamento del neurone $\text{sgn}[x_h(t+1)] = \text{sgn}[v_h(t+1)]$ e un cambio di stato nel neurone h implica che $x_h(t+1) = -x_h(t)$, quindi

$$\operatorname{sgn}(\Delta E) = -\operatorname{sgn}[\Delta x_h(t)] \cdot \operatorname{sgn}[v_k(t+1] = -\operatorname{sgn}\{x_h(t+1) - x_h(t)\} \cdot \operatorname{sgn}[v_k(t+1] =$$
$$= -\operatorname{sgn}\{x_h(t+1) + x_h(t+1)\} \cdot \operatorname{sgn}[x_h(t+1)] = -\operatorname{sgn}[x_h(t+1)]^2 = -1 < 0$$

8.5.2 Caso continuo

La regola di aggiornamento dello stato di un generico neurone i è nel caso continuo la seguente:

$$x_i + C_i \cdot \frac{dx_i}{dt} = \varphi_i[v_i] = \varphi_i\left[\sum_j w_{ij} \cdot x_j + I_i\right]$$

da cui segue

$$C_i \cdot \frac{dx_i}{dt} = -x_i + \varphi_i\left[\sum_j w_{ij} \cdot x_j + I_i\right]$$

dove φ_i è una funzione di attivazione continua, non lineare e crescente associata al neurone i e C_i è una costante.
Possiamo vedere come il sistema raggiunge la stabilità solo quando il vettore velocità del sistema si azzera ovvero quando

$$\frac{dx_i}{dt} = 0, \quad \forall i$$

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

$$v_i + C_i \cdot \frac{dv_i}{dt} = \sum_j w_{ij} \cdot x_j + I_i = \sum_j w_{ij} \cdot \varphi_j(v_j) + I_i$$

da cui segue

$$C_i \cdot \frac{dv_i}{dt} = -v_i + \sum_j w_{ij} \cdot \varphi_j(v_j) + I_i$$

La funzione di energia del sistema nel caso continuo è la seguente

$$E = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot x_i \cdot x_j + \sum_i \int_0^{x_i} \varphi_i^{-1}(x) \cdot dx - \sum_i I_i \cdot x_i$$

Così come nel caso discreto se $W=W^T$ (matrice dei pesi simmetrica) allora la funzione è Lyapunoviana e quindi $\frac{dE}{dt} \le 0$, dove in caso di uguaglianza ci troviamo in un punto stazionario. Dimostriamo ora quest'ultima affermazione.

Facciamo notare innanzitutto che $v_i = \varphi_i^{-1}(x_i)$ e quindi

$$\int_0^{x_i} \varphi_i^{-1}(x) \cdot dx = \int_0^{x_i} \varphi_i^{-1}(x) \cdot dx = \int_0^{x_i} v_i \cdot dx = v_i \cdot x_i$$

$$\frac{dE}{dt} = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot \frac{d(x_i \cdot x_j)}{dt} + \sum_i v_i \cdot \frac{dx_i}{dt} - \sum_i I_i \cdot \frac{dx_i}{dt} =$$

$$= -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot \left(x_i \cdot \frac{dx_j}{dt} + x_j \cdot \frac{dx_i}{dt}\right) + \sum_i v_i \cdot \frac{dx_i}{dt} - \sum_i I_i \cdot \frac{dx_i}{dt} = -\sum_i \left(-v_i + \sum_j w_{ij} \cdot x_j + I_i\right) \cdot \frac{dx_i}{dt}$$

$$= -\sum_i C_i \cdot \frac{dv_i}{dt} \cdot \frac{dx_i}{dt} = -\sum_i C_i \cdot \left[\frac{d}{dt} \cdot \varphi_i^{-1}(x_i)\right] \cdot \frac{dx_i}{dt} = -\sum_i C_i \cdot \left[\frac{d}{dx_i} \cdot \varphi_i^{-1}(x_i)\right] \cdot \left(\frac{dx_i}{dt}\right)^2$$

Essendo φ una funzione monotona crescente lo è anche la sua inversa φ^{-1}, quindi la sua derivata prima è positiva in ogni punto, di conseguenza possiamo affermare che

$$\frac{d}{dx_i} \cdot \varphi_i^{-1}(x_i) \geq 0, \ \forall i$$

Inoltre banalmente $\left(\frac{dx_i}{dt}\right)^2 \geq 0$ e C_i è una costante positiva, di conseguenza $\frac{dE}{dt} \leq 0$.

Da ciò deduciamo che E è una funzione di Lyapunov stabile secondo il primo Teorema.

Facciamo notare che non è possibile che un sistema con funzione di Lyapunov entri in ciclo, infatti trattandosi di una funzione monotona decrescente ad ogni istante temporale t abbiamo che $E(t+1) \leq E(t), \ \forall t \geq 0$., dove con E(t) indichiamo il valore della funzione di energia relativa allo stato del sistema al tempo t.
Supponiamo ora di avere un ciclo nella sequenza degli stati cui corrisponde la seguente relazione della funzione di energia $E(k) \leq E(k+1) \leq \ldots \leq E(k+m) = E(k)$, dove lo stato x(k+m)=x(k). Questa condizione non soddisfa la condizione di monotonia a meno che non valga l'uguaglianza per tutto il ciclo infatti risulterebbe $E(k) \leq E(k)$. Ma se vale l'uguaglianza non avremo un ciclo ma un punto stazionario, quindi non possiamo avere cicli.

8.5.3 Reti di Hopfield come CAM

Le reti di Hopfield hanno attratto l'attenzione in letteratura come **CAM** (memorie indirizzabili per contenuto), in cui conosciamo a priori i punti di attrazione della rete, ma i pesi sono sconosciuti e il problema è di trovarli.
La funzione primaria di una CAM è di recuperare pattern memorizzati in risposta alla presentazione di un pattern incompleto o rumoroso; per questo motivo vengono anche dette **error-correcting** (correttori di errore) nel senso che ricostruiscono il pattern originale da uno "difettato".
L'essanza di una CAM è di mappare un pattern ξ_μ in un punto stabile $\mathbf{x}_\mu$ del sistema dinamico; matematicamente possiamo esprimere questa relazione nella forma

$$\xi_\mu \leftrightarrow x_\mu$$

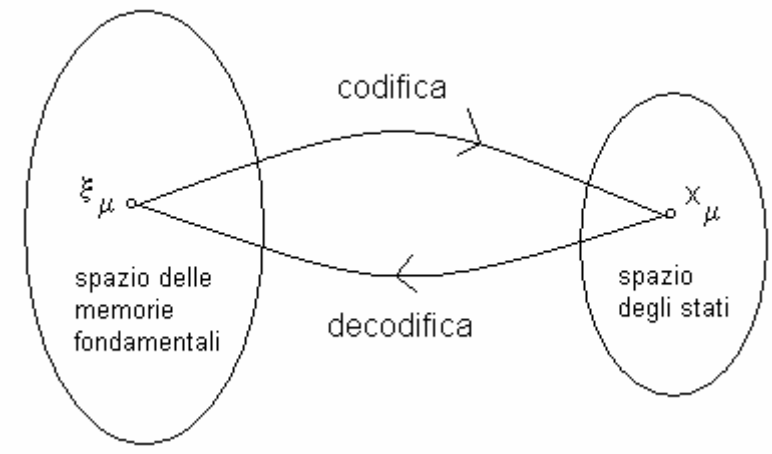

Figura 23: operazioni di codifica e decodifica

La freccia verso destra è detta **operazione di codifica**, mentre quella verso sinistra è detta **operazione di decodifica**. Gli attrattori nello spazio degli stati della rete sono **memorie fondamentali** o **stati prototipo** della rete.

Supponiamo a questo punto che alla rete venga presentato un pattern contenente sufficienti informazioni parziali su una memoria fondamentale. Possiamo rappresentare questo pattern come uno stato iniziale nello spazio degli stati e, se questo è sufficientemente vicino allo stato che

rappresenta la memoria fondamentale, la traiettoria che descriverà il sistema convergerà a quest'ultima.

8.6 Ottimizzazione ricorrendo alle reti di Hopfield

Se riuscissimo a ricondurre un problema P ad un problema di ottimizzazione quadratico, possiamo usare una rete di Hopfield per risolverlo.

8.6.1 Travelling Salesman Problem TSP

La soluzione di un problema TSP è rappresentabile tramite una matrice di permutazione di dimensione n^2 con n il numero di città da visitare. Ogni cella della matrice è un neurone e i loro output sono 0 oppure 1.
Indichiamo con $\mathbf{V} \in \{0,1\}^{n \times n}$ la matrice di stato del sistema di dove la (x,i)-esima componente è a 1 se la x-esima città viene attraversata al i-esimo step del tour, 0 altrimenti.
Indichiamo con $\mathbf{D} \in \Re^{n \times n}$ la matrice delle distanze dove la (x,y)-esima componente indica la distanza tra la x-esima e la y-esima città e con diagonale nulla, perché la distanza da una città a se stessa è 0.
Indicheremo con $v_{x,i}$ una generica componente di **V** e $d_{x,y}$ la generica componente di **D**.

Definiamo ora la funzione costo del sistema

$$E_1 = \frac{C_1}{2} \cdot \sum_x \sum_y \sum_{i=0}^{n} d_{x,y} \cdot v_{x,i} \cdot \left(v_{y,i-1} + v_{y,i+1} \right)$$

dove C_1 è una costante positiva e l'indice i è inteso modulo n.
La matrice **V** senza costrizioni di alcun genere può descrivere zero o più percorsi di lunghezza arbitraria (quindi può non toccare tutte le città).

Ora se la matrice **V** descrivesse solo percorsi unici hamiltoniani, allora minimizzando la funzione costo E_1 otterremmo la soluzione del TSP; quindi a questo punto dobbiamo introdurre dei vincoli che facciano in modo che la matrice **V** soddisfi le proprietà appena descritte.

Vogliamo innanzitutto che ogni città venga visitata al massimo 1 volta, vogliamo quindi che le righe della matrice **V** abbiano soltanto un elemento settato ad 1 e il resto a 0. Per far questo introduciamo un'ulteriore funzione costo da minimizzare

$$E_2 = \frac{C_2}{2} \cdot \sum_x \sum_i \sum_{j \neq i} v_{x,i} \cdot v_{x,j}$$

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

Vogliamo inoltre che ogni step del tour contenga al massimo una città, ovvero ogni colonna della matrice **V** abbia un elemento settato ad 1 ed il resto a 0. Per fare questo introduciamo una terza funzione costo da minimizzare

$$E_3 = \frac{C_3}{2} \cdot \sum_i \sum_x \sum_{y \neq x} v_{x,i} \cdot v_{y,i}$$

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

Come ultima condizione imponiamo che in tutto devono essere attraversate n città. Per far questo introduciamo un'ultima funzione costo da minimizzare

$$E_4 = \frac{C_4}{2} \cdot \left(\sum_x \sum_i v_{x,i} - n \right)^2$$

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

A questo punto una soluzione ottima del TSP richiederebbe che le 4 funzioni energia vengano minimizzate, noi abbiamo però bisogno di un'unica funzione energia, per questo esprimiamo la funzione costo totale come combinazione lineare delle funzioni energia E_i

$$E = E_1 + E_2 + E_3 + E_4$$

dove il peso da attribuire a ciascun addendo è determinato dalle costanti positive C_i .

Dobbiamo ora ricondurci alla funzione di energia nella forma vista al paragrafo 8.5.1, ovvero

$$E = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot x_i \cdot x_j - \sum_i I_i \cdot x_i$$

Per fare questo consideriamo **v** un vettore e indichiamo con $v_{x,i}$ la $(x + i \cdot n)$-esima componente del vettore, analogamente facciamo la stessa considerazione per **I** e consideriamo **W** una matrice quadrata e indichiamo con $w_{(x,i),(y,j)}$ la $(x + i \cdot n, y + j \cdot n)$-esima componente della matrice. Quindi la forma a cui vorremmo arrivare è:

$$E = -\frac{1}{2} \cdot \sum_x \sum_y \sum_i \sum_j w_{(x,i),(y,j)} \cdot v_{x,i} \cdot v_{y,j} - \sum_x \sum_i I_{x,i} \cdot v_{x,i}$$

Introduciamo innanzitutto una notazione:

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

e trasformiamo ciascuna delle funzioni energia E_i nella forma sopra indicata.

$$E_1 = \frac{C_1}{2} \cdot \sum_x \sum_y \sum_{i=0}^{n} d_{x,y} \cdot v_{x,i} \cdot \left(v_{y,i-1} + v_{y,i+1}\right) = -\frac{1}{2} \cdot \sum_x \sum_y \sum_i \sum_j -C_1 \cdot d_{x,y} \cdot v_{x,i} \cdot v_{y,j} \cdot \left(\delta_{j,i-1} + \delta_{j,i+1}\right)$$

$$E_2 = \frac{C_2}{2} \cdot \sum_x \sum_i \sum_{j \neq i} v_{x,i} \cdot v_{x,j} = -\frac{1}{2} \cdot \sum_x \sum_y \sum_i \sum_j -C_2 \cdot v_{x,i} \cdot v_{y,j} \cdot \delta_{x,y} \cdot \left(1 - \delta_{i,j}\right)$$

$$E_3 = \frac{C_3}{2} \cdot \sum_i \sum_x \sum_{y \neq x} v_{x,i} \cdot v_{y,i} = -\frac{1}{2} \cdot \sum_x \sum_y \sum_i \sum_j -C_3 \cdot v_{x,i} \cdot v_{y,j} \cdot \delta_{i,j} \cdot \left(1 - \delta_{x,y}\right)$$

$$E_4 = \frac{C_4}{2} \cdot \left(\sum_x \sum_i v_{x,i} - n \right)^2 = -\frac{1}{2} \cdot \sum_x \sum_y \sum_i \sum_j -C_4 \cdot v_{x,i} \cdot v_{y,j} - \sum_x \sum_i C_4 \cdot n \cdot v_{x,i} - \frac{C_4}{2} \cdot n^2$$

in E_4 tralasciamo il termine costante alla fine in quanto non modifica il punto in cui abbiamo il minimo, ma solo il valore della funzione E_4 in corrispondenza del minimo, cosa che a noi non interessa, dal momento che il nostro scopo è minimizzare le 4 funzioni.
Mettendo il tutto assieme otteniamo

$$E = -\frac{1}{2} \cdot \sum_x \sum_y \sum_i \sum_j v_{x,i} \cdot v_{y,j} \cdot \left[-C_1 \cdot d_{x,y} \cdot \left(\delta_{j,i-1} + \delta_{j,i+1}\right) - C_2 \cdot \delta_{x,y} \cdot \left(1 - \delta_{i,j}\right) - C_3 \cdot \delta_{i,j} \cdot \left(1 - \delta_{x,y}\right) - C_4 \right] -$$

$$+ \sum_x \sum_i C_4 \cdot n \cdot v_{x,i}$$

da cui segue che possiamo risolvere in modo approssimato il problema del TSP utilizzando la matrice dei pesi **W** che ha come $(x + i \cdot n, y + j \cdot n)$-esima componente

$$w_{(x,i),(y,j)} = -C_1 \cdot d_{x,y} \cdot \left(\delta_{j,i-1} + \delta_{j,i+1}\right) - C_2 \cdot \delta_{x,y} \cdot \left(1 - \delta_{i,j}\right) - C_3 \cdot \delta_{i,j} \cdot \left(1 - \delta_{x,y}\right) - C_4$$

e come vettore corrente esterna il vettore **I** che ha come $(x + i \cdot n)$-esima componente

$$I_{(x,i)} = n \cdot C_4$$

8.6.2 Maximum Clique Problem (MCP)

Sia G=(V,E) un grafo non orientato con V l'insieme dei vertici ed E quello degli archi, tale per cui V={1,…,n} e E $\subseteq$ V×V. Definiamo **clique** C$\subseteq$V, un sottoinsieme di vertici di G che formano un grafo completo, ovvero tale che $\forall$ i,j$\in$C con i$\neq$j, (i,j) $\in$ E.

Definiamo **clique massimale** di G una clique di G che non è contenuta in nessun'altra clique di G.
Definiamo **clique massima** di G una clique massimale di G di cardinalità massima.

Il problema **MCP** consiste nel cercare in un grafo una clique massima. Questo problema è classificato come NP-completo.

Sia C$\subseteq$V chiamiamo **vettore caratteristico di C, $\mathbf{x}^C$** un vettore che soddisfa la seguente relazione

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

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

Definiamo **simplesso standard** in $\Re^n$ l'insieme S_n definito come

$$S_n = \left\{ x \in \Re^n \left| \sum_{i=1}^{n} x_i = 1,\ x_i \geq 0\ \ \forall i \right. \right\}$$

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

Sia A=(a_{ij}) la **matrice di adiacenza** di G=(V,E) ovvero una matrice tale per cui

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

Consideriamo a questo punto la seguente funzione

$$f(\mathbf{x}) = \mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{x} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j$$

Sia C$\subseteq$V e $\mathbf{x}^C$ il suo vettore caratteristico allora C è clique massima di G sse $\mathbf{x}^C$ è un massimo globale di f in S_n . In questo caso

$$|C| = \frac{1}{1 - f(\mathbf{x}^C)}$$

C'è una clique massimale di G sse $\mathbf{x}^C$ è un massimo locale di f in S_n . (**Teorema di Motzkin-Strauss**)

Quello che abbiamo fatto finora è prendere il problema P di clique massima nel discreto e lo abbiamo trasformato in un problema P' di ottimizzazione quadratica nel continuo, a questo punto dobbiamo poter, data una soluzione al problema P' riuscire a mapparla in una soluzione del problema originario P. Questo è possibile se la soluzione ottenuta in P' è un vettore caratteristico.

La funzione f(**x**) ha il difetto di avere massimi locali anche in presenza di soluzioni non nella forma caratteristica (**soluzioni spurie**).
Per ovviare a ciò modifichiamo la funzione nel seguente modo

$$f(\mathbf{x}) = \mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{x} + \frac{1}{2} \cdot \mathbf{x}^T \cdot \mathbf{x} = \mathbf{x}^T \cdot \left(\mathbf{A} + \frac{1}{2} \cdot \mathbf{I} \right) \cdot \mathbf{x}$$

dove con **I** indichiamo la matrice identica.

Sia $C \subseteq V$ e sia $\mathbf{x}^C$ il su vettore caratteristico allora

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

8.6.3 Problemi

I problemi che si hanno utilizzando una rete di Hopfield come strumento di ottimizzazione sono:

- in molti casi abbiamo un numero di connessione dell'ordine di n^4 ;
- abbiamo un numero di unità dell'ordine di n^2 ;
- le soluzioni ottenute sono raramente ammissibili;
- difficoltà nel determinare i parametri per risolvere il problema;
- difficoltà di evitare minimi locali.

8.7 Teoria dei giochi evoluzionistici

Assumiamo di avere una grande popolazione della stessa specie in cui gli individui competono per delle risorse limitate. Definiamo **gioco** un conflitto tra 2 individui. I giochi tra ogni coppia di individui vengono assegnati a caso. I giocatori non agiscono in modo razionale ma hanno una loro **strategia** preprogrammata. La riproduzione è asessuata. Si ottiene un guadagno in caso di vittoria o successo riproduttivo.

Siano:

- J={1,…,n} l'insieme delle strategie;
- lo stato della popolazione al tempo t è il vettore **x**(t) dove la i-esima componente indica la frazione di popolazione che usa la strategia i al tempo t;
- **W** la matrice di utilità (**fitness**) dove w_{ij} rappresenta il guadagno medio che un giocatore che gioca la strategia i ottiene contro un giocatore che gioca la strategia j con $i,j \in J$ e supponiamo senza perdita di generalità che $w_{ij} \geq 0$

Dato uno stato **x**, defininiamo **supporto di x** l'insieme

$$\sigma(\mathbf{x}) = \left\{ i \in J \middle| x_i > 0 \right\}$$

Se la popolazione si trova nello stato **x** il guadagno di un giocatore che gioca la strategia i al tempo t è

$$\pi_i(t) = \sum_{j=1}^{n} w_{ij} \cdot x_j(t) = \left(\mathbf{W} \cdot \mathbf{x}(t) \right)_i$$

Il guadagno medio della popolazione ad un certo istante t è

$$\pi(t) = \sum_{i=1}^{n} x_i(t) \cdot \pi_i(t) = \mathbf{x}^T(t) \cdot \mathbf{W} \cdot \mathbf{x}(t)$$

Vediamo ora come aggiornare, nel caso discreto, lo stato della popolazione.
Per fare ciò vogliamo che la frazione di popolazione che utilizza la strategia i sia, al tempo t+1, pari al rapporto fra il guadagno ottenuto da coloro che hanno giocato la strategia i e il guadagno totale della popolazione. In pratica se una strategia risulta efficace allora la porzione di popolazione che la utilizzerà sarà maggiore. Giungiamo così ad un **equazione di replicazione** discreta

$$x_i(t+1) = \frac{x_i(t) \cdot \pi_i(t)}{\pi(t)}$$

Ogni neurone per aggiornarsi deve conoscere lo stato di tutti gli altri neuroni.
Vediamo ora come ottenere la stessa relazione nel continuo. Dell'equazione di replicazione discreta ci interessano soprattutto i punti stazionari tali per cui $x_i(t+1)=x_i(t)$. Sia x un punto fisso dell'equazione allora

$$x = \frac{x \cdot \pi_i(t)}{\pi(t)} \Rightarrow 0 = \frac{x \cdot \pi_i(t) - x \cdot \pi(t)}{\pi(t)} \Rightarrow 0 = x \cdot \left[\pi_i(t) - \pi(t)\right]$$

Dal momento che un punto è stazionario in un equazione dinamica se la sua derivata rispetto al tempo è nulla, un equazione di replicazione continua equivalente a quella discreta la otteniamo ponendo

$$\frac{dx}{dt} = x \cdot \left[\pi_i(t) - \pi(t)\right]$$

Le due equazioni dinamiche (discreto e continuo) hanno quindi gli stessi punti stazionari ed inoltre S_n è invariante rispetto ad entrambe le equazioni, ovvero qualunque traiettoria nasca in S_n rimane in S_n.

Se $W=W^T$ (matrice dei pesi è simmetrica) allora $\pi(t)$ è strettamente crescente, ovvero $\frac{d\pi}{dt} > 0$ per il sistema continuo e $\pi(t+1)>\pi(t)$ per quello discreto a meno di punti stazionari.

Se scegliamo come peso $\mathbf{W} = \mathbf{A} + \frac{1}{2} \cdot \mathbf{I}$ dove **A** è la matrice di adiacenza e **I** è la matrice identica possiamo risolvere il problema MCP in modo approssimato trovando un punto fisso del sistema.

Una misura di qualità della soluzione è

$$Q = \frac{f_{ave} - f_{re}}{f_{ave} - \alpha}$$

dove f_{ave} è il valore medio della funzione calcolato su n campioni, f_{re} è il valore della funzione in corrispondenza della soluzione ottenuta e α è il valore della funzione in corrispondenza della soluzione ottimale.

Se Q tende ad 1 allora l'algoritmo è buono.