UNIVERSITÀ DEGLI STUDI DI FOGGIA

HR EXCELLENCE IN RESEARCH

Reti Neurali Artificiali

Approfondimenti

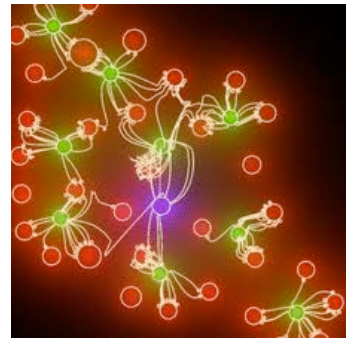
Dott. Crescenzio Gallo

Professore Aggregato di Sistemi di Elaborazione delle Informazioni

Dipartimento di Medicina Clinica e Sperimentale

CRESCENZIO.GALLO@UNIFG.IT

Le Reti Neurali: approfondimenti

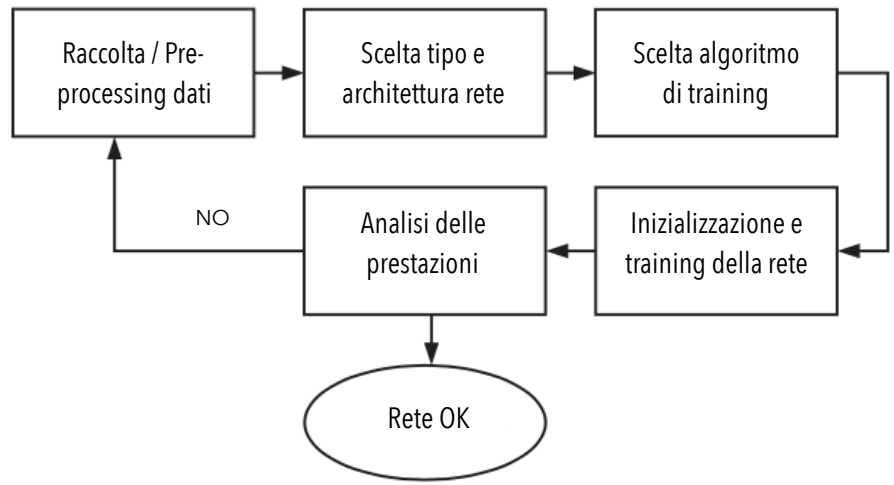


Nella prima parte "Concetti base" abbiamo visto le architetture utilizzate per le reti neurali e le tecniche di addestramento (*training*) da un punto di vista generale.

In questa seconda parte esamineremo alcuni aspetti pratici del training delle reti neurali, ed in particolare:

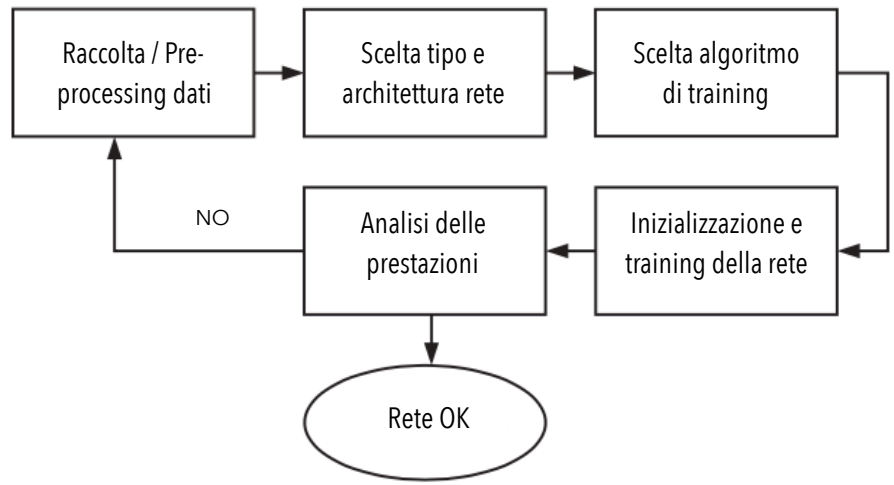
1. **raccolta** e **pre-elaborazione** dei dati, e **scelta dell'architettura** della rete;
2. **training** della rete;
3. **analisi post-training**.

Il processo di *training*



- Il processo di *training* di una rete neurale è una procedura iterativa che inizia con la **raccolta** dei dati e la loro **pre-elaborazione** (*preprocessing*) per rendere più efficiente il processo di addestramento. A questo stadio va anche stabilita la ripartizione dei dati negli insiemi di *training*, *validazione* (opzionale) e *test* finale.
- Dopo che i dati sono stati selezionati, va scelto il **tipo** della rete (MPL, dinamica, etc.) e la sua **architettura** (ad es. numero di strati, numero di neuroni nascosti).
- Infine va scelto l'**algoritmo di addestramento** appropriato per la rete ed il problema che si vuole risolvere.
- Dopo che la rete è stata addestrata (cioè i pesi sono stati definiti) occorre ***analizzarne le prestazioni***. Quest'analisi può portare a scoprire problemi con i dati, l'architettura della rete o l'algoritmo di addestramento. In tal caso va ripetuto l'intero processo fino a quando le prestazioni della rete sono ritenute soddisfacenti.

Il processo di *training*

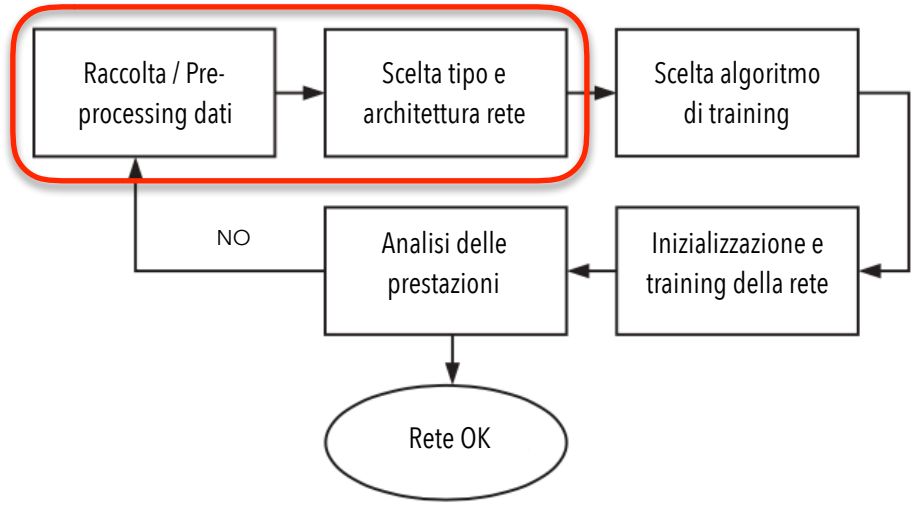


Nel seguito esamineremo ciascuna parte del processo di training in dettaglio: fasi di pre-training (raccolta/preprocessing dei dati), addestramento vero e proprio della rete, analisi post-training.

Una considerazione preliminare all'utilizzo di una rete neurale merita però di essere fatta a questo punto: ***La rete neurale è davvero la soluzione adatta al problema da risolvere?*** Spesso una semplice *soluzione lineare* è sufficiente per risolvere un problema non complesso.

Ad esempio non vi è alcuna necessità di una rete neurale per un problema di fitting se una regressione lineare standard produce già un risultato soddisfacente. Le reti neurali possiedono una maggiore capacità di modellizzazione, ma a scapito di requisiti di training (e di calcolo) molto più esigenti. Se i metodi lineari funzionano, essi rappresentano sempre la scelta più opportuna…

Fasi di pre-training



Le fasi preliminari all'addestramento della rete possono essere raggruppate in tre categorie:

- **raccolta** dei dati;
- pre-elaborazione (**preprocessing**) dei dati;
- scelta del **tipo** e dell'**architettura della rete**.

Fasi di pre-training

Raccolta dei dati

- La difficoltà principale di una rete risiede nell'acquisizione delle conoscenze legate al dominio del problema: la "bontà" di una rete sarà strettamente legata alla **qualità dei dati a disposizione** per l'addestramento.
- I dati di training devono rappresentare pienamente l'intero "*spazio degli input*" per i quali la rete sarà utilizzata. Vi sono metodi di training utilizzabili per assicurarsi che la rete interpoli (**generalizzi**) accuratamente sul range dei dati forniti.
- Non è comunque possibile garantire le prestazioni della rete quando i suoi input sono al di fuori del range del training set. Le reti neurali, come altri metodi non lineari, non estrapolano bene…
- Non è sempre facile essere sicuri che lo spazio di input sia **adeguatamente campionato** dai dati di training. Per molti problemi, la dimensione dello spazio di input è troppo grande e spesso le variabili di input sono dipendenti. Potrebbe non essere perciò possibile definire con precisione la regione utile dello spazio di input; tuttavia, spesso possiamo raccogliere dati rappresentativi di tutte le condizioni per le quali intendiamo utilizzare la rete e quindi impiegarli per un training corretto ed esaustivo.

Fasi di pre-training

Raccolta dei dati

Come possiamo essere sicuri che lo **spazio di input** sia stato **adeguatamente campionato** dai dati di training? Questo è difficile da fare prima del training, e ci sono molti casi in cui non abbiamo alcun controllo sul processo di raccolta dei dati (specie in campo clinico) e dobbiamo perciò utilizzare qualsiasi dato disponibile.

Analizzando la rete addestrata, possiamo spesso dire se i dati di training erano sufficienti. In questo caso, possiamo utilizzare tecniche che indicano quando una rete viene utilizzata al di fuori dell'intervallo dei dati su cui è stata addestrata. Questo non dimostrerà le prestazioni della rete, ma ci impedirà di utilizzare una rete in situazioni in cui non è affidabile.

Dopo averli raccolti, generalmente dividiamo (mediante campionamento casuale) i dati in tre gruppi: *training*, *validazione* e *test*, di solito nel rapporto 70% / 15% / 15%. È importante che ciascuno di questi gruppi sia "**stratificato**", cioè rappresentativo dell'insieme completo di dati.

Fasi di pre-training

Raccolta dei dati

Un'ultima questione sulla raccolta dei dati è: "Abbiamo abbastanza dati?". È difficile rispondere a questa domanda, soprattutto prima di addestrare la rete. La quantità di dati richiesti dipende dalla complessità della funzione sottostante che stiamo cercando di approssimare.

Se la funzione da approssimare è molto complessa, con molti punti di inflessione, allora questo richiede una grande quantità di dati.

Se la funzione è regolare, allora i dati richiesti sono significativamente ridotti (a meno che non siano molto rumorosi).

La scelta della dimensione del set di dati è strettamente correlata alla scelta del numero di neuroni nella rete neurale. Naturalmente, generalmente non sappiamo quanto sia complessa la funzione sottostante prima di iniziare il training. Per questo motivo il processo di addestramento della rete neurale è iterativo.

Al termine dell'addestramento si analizzano le prestazioni della rete: i risultati di tale analisi possono aiutarci a decidere se abbiamo abbastanza dati o meno.

Fasi di pre-training

Preprocessing dei dati

Lo scopo principale della fase di preprocessing dei dati è quello di **facilitare l'addestramento** della rete.

La fase di pre-elaborazione consiste in operazioni quali la *normalizzazione*, *trasformazioni non lineari*, *estrazione di feature*, *codifica* degli input/output discreti, gestione dei *dati mancanti*, ecc. L'idea è quella di eseguire un'elaborazione preliminare dei dati per facilitare l'estrazione delle informazioni rilevanti per l'addestramento della rete neurale.

Ad esempio, nelle reti multistrato, le funzioni di trasferimento sigmoidee sono spesso utilizzate negli strati nascosti. Queste funzioni diventano essenzialmente sature quando l'input netto è maggiore di tre ($e^{-3} \approx 0{,}05$). Se non vogliamo che questo accada all'inizio del processo di training, è pratica comune **normalizzare** gli ingressi (di solito nel range [-1,1]) prima di applicarli alla rete.

$$\frac{1}{1+e^{-x}}$$

Fasi di pre-training

Preprocessing dei dati → **Normalizzazione**

Esistono due metodi standard per la *normalizzazione*. Il primo metodo **scala** i dati in modo che rientrino in un intervallo standard, tipicamente da -1 a 1. Questo può essere fatto mediante la trasformazione:

$$x_i^{'} = 2 \cdot \frac{x_i - x_{min}}{x_{max} - x_{min}} - 1$$

Una procedura alternativa (detta **standardizzazione**) è quella di trasformare i dati in modo che abbiano una media e una varianza specifica, tipicamente 0 e 1:

$$x_i^{'} = \frac{x_i - \mu}{\sigma}$$

Generalmente, la fase di normalizzazione viene applicata sia agli ingressi che agli output del dataset.

Fasi di pre-training

Preprocessing dei dati → **Trasformazioni non lineari**

Oltre alla normalizzazione, che comporta una trasformazione lineare, a volte vengono eseguite *trasformazioni non lineari* come parte della fase di pre-elaborazione.

A differenza della normalizzazione, che può essere applicato a qualsiasi insieme di dati, queste trasformazioni non lineari sono **specifiche** per ogni caso.

Per esempio, molte variabili economiche mostrano una dipendenza logaritmica: in questo caso, potrebbe essere opportuno prendere il logaritmo dei valori di input prima di applicarli alla rete neurale.

Un altro esempio è la simulazione di dinamiche molecolari, in cui le forze atomiche sono calcolate come funzioni delle distanze tra gli atomi: poiché è noto che le forze sono inversamente correlate alle distanze, potremmo effettuare la trasformazione reciproca sugli ingressi, prima di applicarle alla rete.

Questo rappresenta un **modo di incorporare le conoscenze** pregresse nell'addestramento delle reti neurali. Se la trasformazione non lineare è scelta con intelligenza, può rendere più efficiente l'addestramento della rete. Il preprocessing alleggerirà parte del lavoro richiesto alla rete neurale per calcolare la funzione sottostante tra gli input e gli output.

Fasi di pre-training

Preprocessing dei dati → **Feature extraction**

Un'altra fase di pre-elaborazione dei dati è chiamata *feature extraction* (estrazione delle caratteristiche). Questa si applica generalmente a situazioni in cui la dimensione dell'input è molto grande e le componenti di input sono ridondanti.

L'idea è quella di **ridurre la dimensione dello spazio di input** calcolando dal dataset un piccolo insieme di feature (variabili caratteristiche) e utilizzandole come input per la rete neurale.

Ad esempio, le reti neurali possono essere utilizzate per analizzare i segnali ECG (elettrocardiogramma) per identificare i problemi cardiaci. L'ECG potrebbe coinvolgere 12 o 15 segnali misurati per diversi minuti ad alta frequenza di campionamento. Si tratta di troppi dati da inviare direttamente alla rete neurale. In alternativa, vengono estratte alcune caratteristiche del segnale ECG, come gli intervalli di tempo medio tra certe forme d'onda, le ampiezze medie di certe onde, ecc.

Fasi di pre-training

Preprocessing dei dati → **Componenti principali (PCA)**

Ci sono anche alcuni metodi di *feature extraction* di uso generale. Uno di questi è il **metodo di analisi delle componenti principali** (PCA).

Questo metodo trasforma i vettori di ingresso originali in modo che le componenti dei vettori trasformati non siano correlate. Inoltre, le componenti del vettore trasformato sono ordinate in modo tale che la prima componente abbia la maggiore varianza, la seconda la maggiore varianza successiva, ecc.

Generalmente si utilizzano **solo le prime componenti** del vettore trasformato, che rappresentano la maggior parte della varianza del vettore originale. Questo si traduce in una grande riduzione della dimensione del vettore di ingresso, se le componenti originali sono altamente correlate.

Lo **svantaggio** dell'uso di PCA è che considera solo le relazioni lineari tra le componenti del vettore di input. Quando si riduce la dimensione utilizzando una trasformazione lineare, si possono perdere alcune informazioni non lineari. Poiché lo scopo principale dell'uso delle reti neurali risiede nella loro potenza di mappatura non lineare, occorre stare attenti quando si usa la PCA prima di applicare gli input alla rete neurale.

Esistono anche versioni non lineari della PCA, come ad es. "Kernel PCA" (Schölkopf *et al.*, 1999) e "Feature Extraction with Nonlinear PCA" (Gallo & Capozzi, 2019).

Fasi di pre-training

Preprocessing dei dati → **Codifica dei dati discreti**

Un'altra importante fase di pre-elaborazione è necessaria quando gli ingressi o gli output assumono solo **valori discreti**. Se abbiamo ad esempio un problema di pattern recognition in cui ci sono quattro classi, vi sono almeno tre modi comuni per codificare gli output.

In primo luogo, possiamo avere **output scalari** che assumono quattro possibili valori (ad esempio, 1, 2, 3, 3, 4). In secondo luogo, possiamo avere **target bidimensionali**, che rappresentano un codice binario delle quattro classi (ad esempio, (0,0), (0,1), (1,0), (1,1), (1,1)). In terzo luogo, possiamo avere **output quadridimensionali**, in cui è attivo un solo neurone alla volta (ad esempio (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)): quest'ultimo metodo tende a dare i migliori risultati in generale. Si noti che gli input discreti possono essere codificati nello stesso modo degli output.

Quando si codificano gli output, dobbiamo anche considerare la **funzione di trasferimento** utilizzata nello strato di uscita della rete. Per i problemi di pattern recognition, usiamo tipicamente le funzioni sigmoidee: *logsig* o *tansig*. Se usiamo *tansig* nell'ultimo strato (il caso più comune) potremmo considerare l'assegnazione di valori di output a -1 o 1, che rappresentano gli asintoti della funzione. Tuttavia, questo tende a causare difficoltà all'algoritmo di addestramento, che cerca di saturare la funzione sigmoidea per raggiungere il valore di output.

E' meglio assegnare i valori target nel punto in cui la seconda derivata della funzione sigmoidea è massima. Per la funzione *tansig* ciò avviene quando l'ingresso è -1 e 1, che corrisponde a valori di uscita -0,76 e +0,76.

Fasi di pre-training

Preprocessing dei dati → **Codifica dei dati discreti**

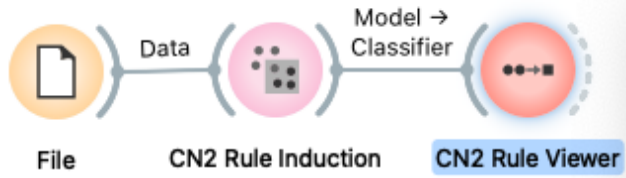
Un'altra funzione di trasferimento utilizzata nello strato di uscita delle reti multistrato per problemi di pattern recognition è la funzione ***softmax*** (o *funzione esponenziale normalizzata*), una generalizzazione della funzione logistica. Questa funzione di trasferimento ha la forma:

$$f : \mathbf{y} \in \mathbb{R}^N \rightarrow \left\{ \mathbf{z} \in \mathbb{R}^N | z_i > 0, \sum_{i=1}^{N} z_i = 1 \right\}$$

$$f(y_j) = \frac{e^{y_j}}{\sum_{i=1}^{N} e^{y_i}} \quad \text{per } j = 1, \ldots, N$$

Le uscite della funzione di trasferimento *softmax* possono essere interpretate come le probabilità associate ad ogni classe. Ogni uscita è compresa tra 0 e 1, e la somma delle uscite è uguale a 1.

Un esempio di applicazione lo ritroviamo nel widget Orange di classificazione "CN2 Rule Induction" e relativo widget "CN2 Rule Viewer" di visualizzazione dei risultati della classificazione.



CN2 Rule Viewer

	IF conditions		THEN class	Distribution	Probabilities [%]
0	petal length≤3.0 AND sepal width≥2.9	→	iris=Iris-setosa	[49, 0, 0]	96 : 2 : 2
1	petal width≥1.8 AND sepal length≥6.0	→	iris=Iris-virginica	[0, 0, 39]	2 : 2 : 95
2	sepal length≥4.9 AND sepal width≥3.1	→	iris=Iris-versicolor	[0, 8, 0]	9 : 82 : 9

Fasi di pre-training

Preprocessing dei dati → **Dati mancanti**

Un'altra questione pratica da considerare è la **mancanza di dati**, che può verificarsi per varie ragioni.

Ad esempio, potremmo avere dei dati clinici raccolti a intervalli mensili, con alcuni mesi in cui alcuni esami non sono stati eseguiti per alcuni pazienti. La soluzione più semplice a questo problema sarebbe quella di eliminare le registrazioni incomplete: tuttavia, la quantità di dati disponibili potrebbe essere molto limitata e potrebbe essere molto costoso raccogliere ulteriori dati. In tal caso, è opportuno fare pieno uso di tutti i dati in nostro possesso, anche se incompleti.

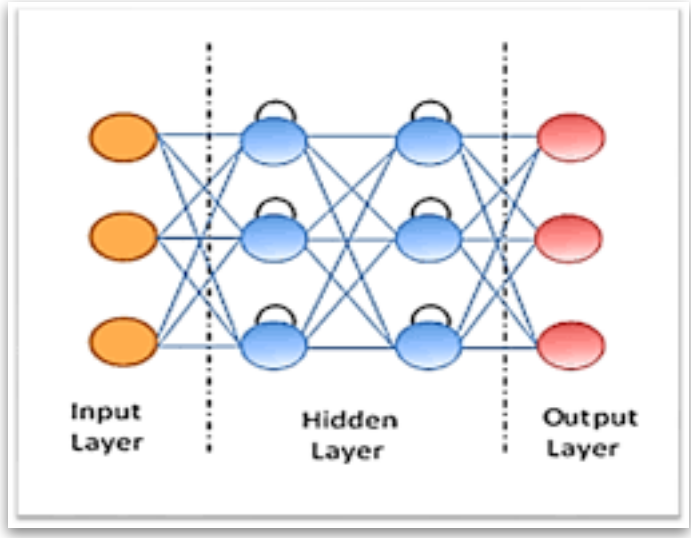
Esistono **diverse strategie** per gestire i dati mancanti. Se mancano dei dati per una variabile di input, una possibilità è quella di sostituire i valore mancanti con il ***valore medio*** per quella particolare variabile, con l'eventuale aggiunta di una variabile binaria di input che indichi il completamento di dati mancanti per quella variabile. Ciò fornirebbe al classificatore informazioni sulle variabili mancanti per una corretta elaborazione delle stesse.

Se i dati mancanti si verificano in output, l'indice di performance della rete può essere modificato in modo da non includere gli errori associati ai valori di output mancanti.

Fasi di pre-training

Scelta dell'architettura della rete

- Il passo successivo nel processo di addestramento della rete è la **scelta dell'architettura di rete**.
- Il tipo di base dell'architettura di rete è determinato dal tipo di problema che vogliamo risolvere.
- Una volta scelta l'architettura di base, bisogna decidere i dettagli specifici come il *numero di strati* e *di neuroni nascosti*, quanti *output* la rete dovrebbe avere e quale tipo di *indice di performance* utilizzare per il training.



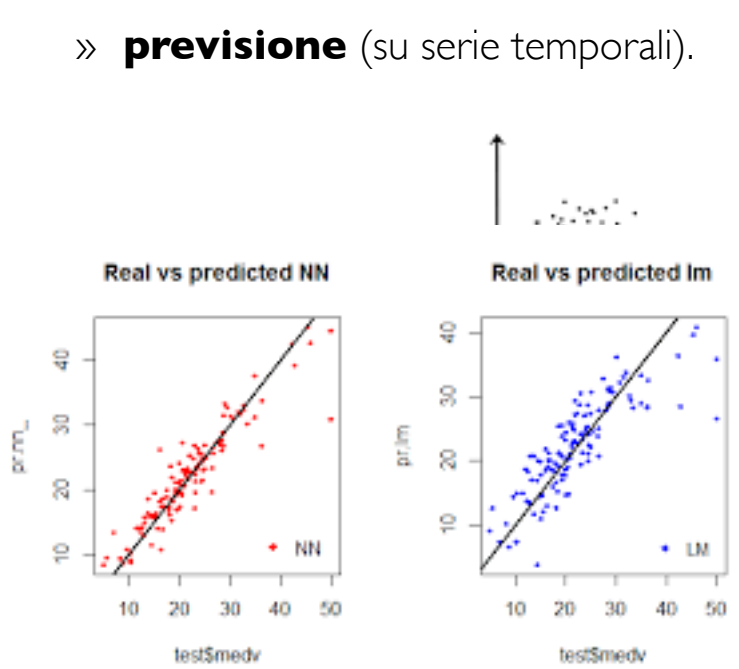
Fasi di pre-training

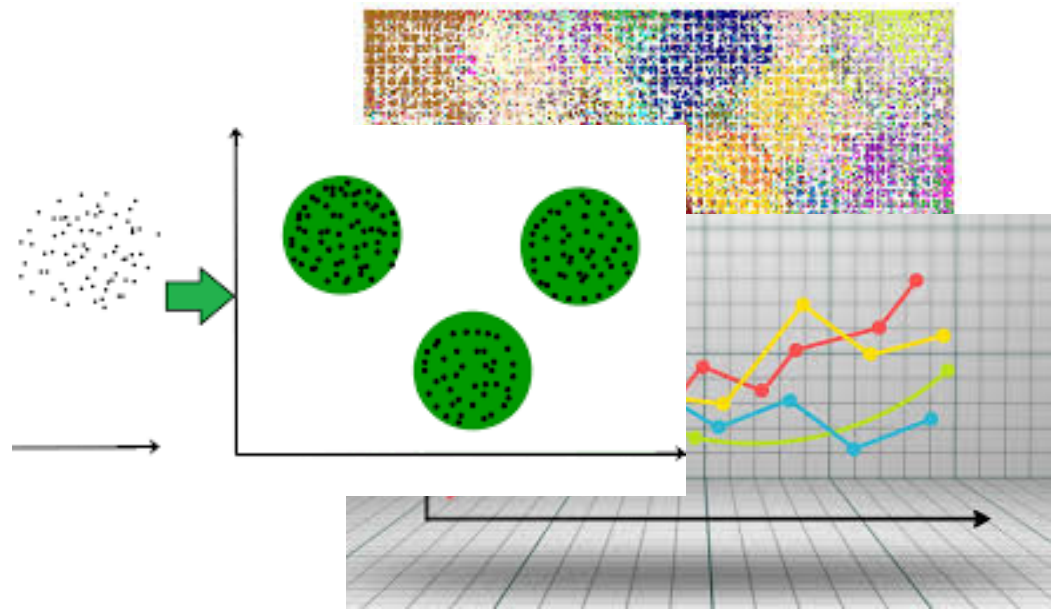
Scelta dell'architettura della rete → **Architettura di base**

Il primo passo nella scelta dell'architettura consiste nel **definire il problema** che si sta cercando di risolvere.

Per i nostri scopi, limiteremo la discussione a quattro tipi di problemi:

- **regressione** (*fitting*);
- **classificazione** (*pattern recognition*);
- **clustering**;
- **previsione** (su serie temporali).





Fasi di pre-training

Scelta dell'architettura della rete → Architettura di base → **Regressione**

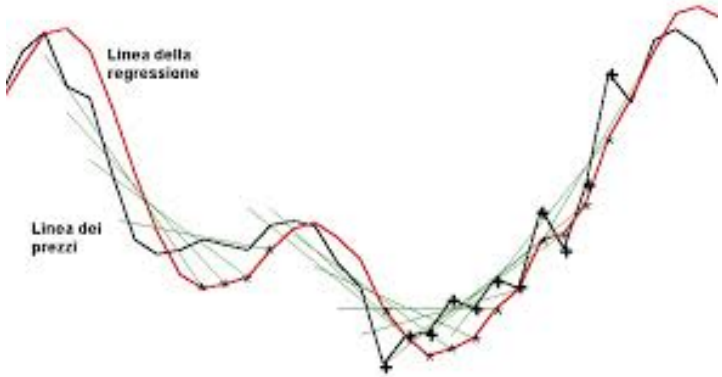
Nella **regressione** (*fitting*, anche definita come *approssimazione di funzioni*), si desidera che una rete neurale "mappi" un insieme di ingressi con i corrispondenti target.

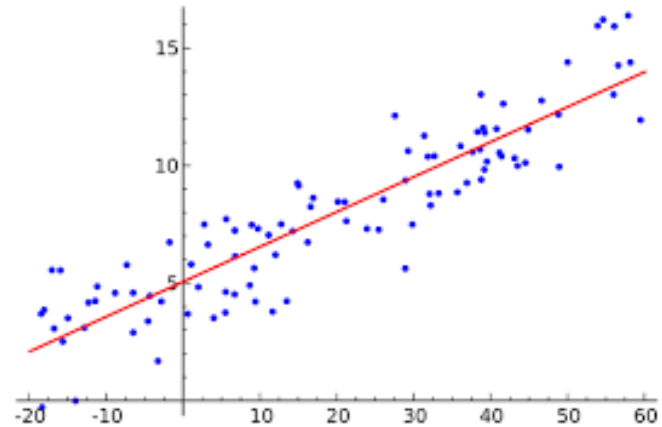
Ad esempio, un agente immobiliare potrebbe voler stimare i prezzi di una casa in base a variabili quali l'*aliquota fiscale*, il *rapporto alunni/insegnanti* nelle scuole del vicinato e il *tasso di criminalità* del quartiere.

Un ingegnere automobilistico potrebbe voler stimare i livelli di emissione del motore sulla base di misurazioni del *consumo di carburante* e della *velocità*.

Un medico potrebbe voler prevedere il livello di grasso corporeo di un paziente sulla base di misurazioni (variabili) quali il *peso* e l'*altezza*.

Per i problemi di fitting, la variabile target assume **valori numerici continui**.

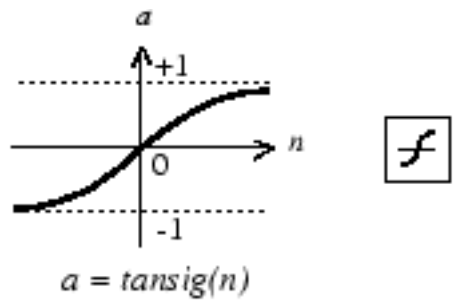




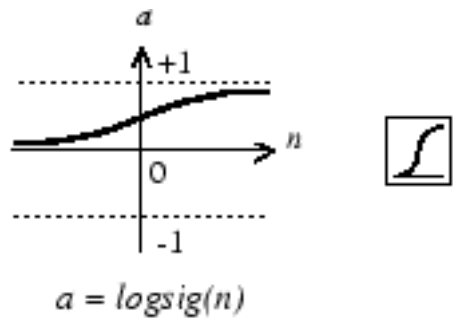
Fasi di pre-training

Scelta dell'architettura della rete → Architettura di base → **Regressione**

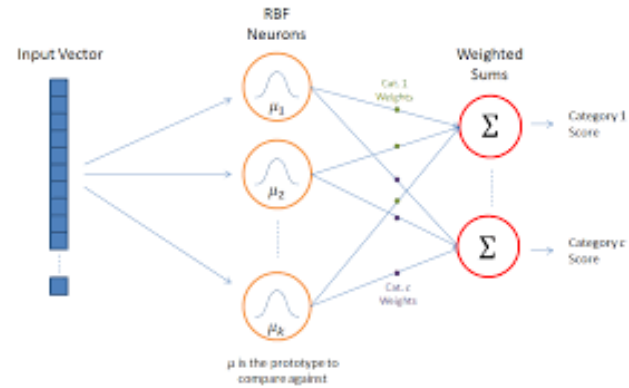
- L'**architettura di rete neurale standard** per i problemi di regressione è il **Multi Layer Perceptron** (MLP), con neuroni *tansig* negli strati nascosti, e neuroni *lineari* nello strato di uscita.
- La funzione di trasferimento *tansig* è generalmente preferita alla funzione di trasferimento *logsig* negli strati nascosti per lo stesso motivo per cui gli ingressi sono normalizzati: essa infatti produce uscite (che sono ingressi allo strato successivo) centrate intorno allo zero, mentre la funzione di trasferimento *logsig* produce sempre uscite positive.
- Per la maggior parte dei problemi di *fitting* è sufficiente un unico strato nascosto. Se i risultati con uno strato nascosto non sono soddisfacenti, a volte vengono utilizzati due strati; raramente vengono utilizzati più di due strati nascosti. Per problemi molto difficili, sono state utilizzate reti profonde (*deep network*) con molti strati.
- Le *funzioni di trasferimento lineari* sono utilizzate nello strato di uscita per i problemi di regressione perché **l'uscita è una variabile continua**.
- Le reti RBF (Radial Basis Function) possono essere utilizzate anche per problemi di regressione, con la funzione di trasferimento gaussiana usata nello strato nascosto e la funzione di trasferimento lineare nello strato di uscita.



Tan-Sigmoid Transfer Function



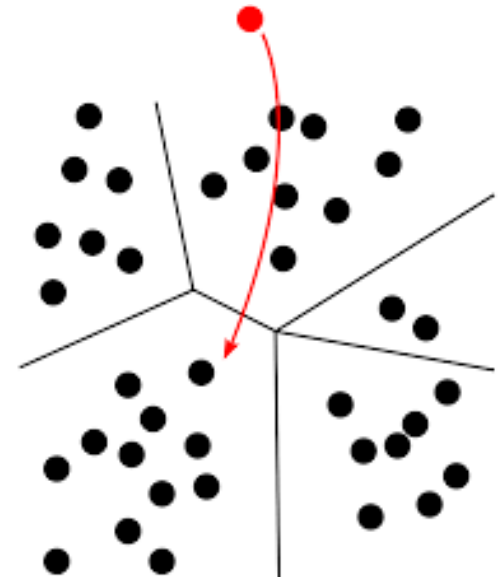
Log-Sigmoid Transfer Function



Fasi di pre-training

Scelta dell'architettura della rete → Architettura di base → **Classificazione**

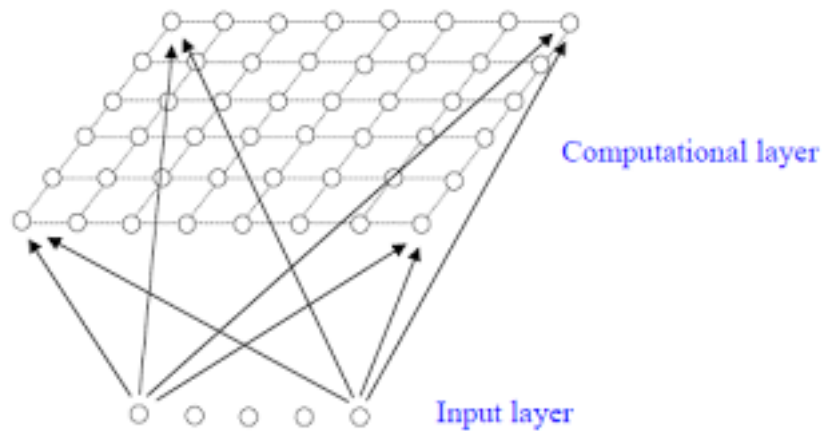
- Nei problemi di **classificazione** (o *pattern recognition*) si fanno corrispondere gli input ad una serie di categorie (classi) predefinite.
- *Ad esempio, un commerciante di vini potrebbe voler riconoscere il vigneto da cui proviene una particolare bottiglia di vino, sulla base di un'analisi chimica del vino. Un medico potrebbe voler classificare un tumore come benigno o maligno, sulla base dell'uniformità delle dimensioni delle cellule, dello spessore dei campioni e della mitosi cellulare.*
- Oltre che per i problemi di *fitting*, anche per il *pattern recognition* possono essere utilizzate le **reti MLP**.
- La differenza principale tra una rete per un problema di *fitting* e una per il pattern recognition sta nella funzione di trasferimento utilizzata nello strato di uscita.
- Per i problemi di *pattern recognition* generalmente si utilizza una **funzione sigmoidea nello strato di uscita**.
- Anche la rete RBF può essere utilizzata per il *pattern recognition*.



Fasi di pre-training

Scelta dell'architettura della rete → Architettura di base → **Clustering**

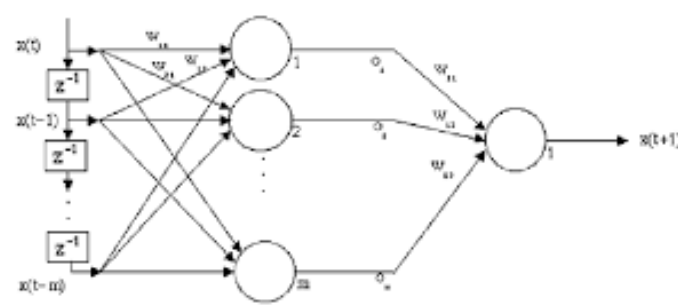
- Nei problemi di **clustering** si vuole realizzare una rete neurale per raggruppare i dati per similarità.
- *Ad esempio, le aziende possono voler eseguire la segmentazione del mercato, che viene effettuata raggruppando le persone in base ai loro modelli di acquisto. Gli informatici potrebbero voler eseguire il data mining suddividendo i dati in sottoinsiemi correlati. I biologi potrebbero voler eseguire analisi bioinformatiche, come il raggruppamento di geni con i relativi profili di espressione.*
- Una qualsiasi delle **reti competitive** potrebbe essere utilizzata per il clustering. La **rete SOM** (Self Organizing Map, o rete di Kohonen) è la rete più popolare per il clustering. Il vantaggio principale ti tale architettura è che permette la visualizzazione di spazi di input con molte dimensioni.



Fasi di pre-training

Scelta dell'architettura della rete → Architettura di base → **Previsione**

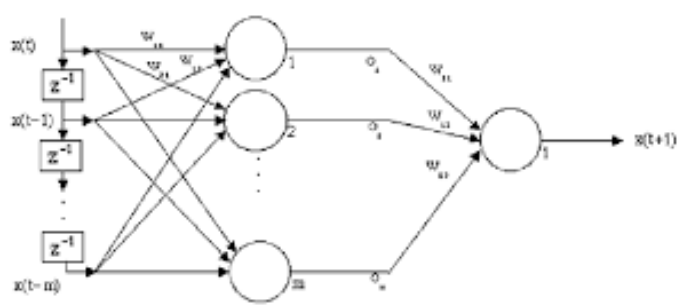
- La **previsione** rientra anche nelle categorie di analisi delle serie temporali, identificazione di sistemi, filtraggio o modellizzazione dinamica. L'idea è che si vuole prevedere il valore futuro di alcune serie temporali.
- *Un trader di azioni potrebbe voler prevedere il valore futuro di un titolo. Un chimico potrebbe voler prevedere il valore futuro della concentrazione di una certa sostanza in output da un impianto di lavorazione. Un ingegnere potrebbe voler pre-determinare le interruzioni della rete elettrica.*
- La previsione richiede l'uso di **reti neurali dinamiche**. La forma specifica della rete dipenderà dalla particolare applicazione.
- La rete più semplice per la previsione non lineare è la rete neurale a ritardo (temporale), che fa parte di una classe generale di reti in cui le dinamiche appaiono solo allo strato di input di una rete statica multistrato feedforward.
- Questa rete ha il vantaggio di poter essere addestrata utilizzando algoritmi di *backpropagation* statici, poiché la linea di ritardo all'ingresso della rete può essere sostituita da un vettore esteso di valori ritardati di input.



Fasi di pre-training

Scelta dell'architettura della rete → Architettura di base → **Previsione**

- Per problemi di modellizzazione e controllo dinamico viene spesso utilizzata la rete **NARX** (Non-linear AutoRegressive model with eXogenous input).
- *In questa rete il segnale di ingresso potrebbe rappresentare, ad esempio, la tensione applicata ad un motore e l'uscita potrebbe rappresentare la posizione angolare di un braccio di un robot.*
- Come con la rete neurale con ritardo, la rete NARX può essere addestrata con backpropagation statico. Le due linee a ritardo possono essere sostituite con vettori estesi di ingressi e uscita ritardati.
- *Si possono usare le uscite invece di re-immetterle nella rete (il che richiederebbe un addestramento con backpropagation dinamico) perché le uscite della rete dovrebbero corrispondere agli output richiesti al termine dell'addestramento.*



Fasi di pre-training

Scelta dell'architettura della rete → **Dettagli**

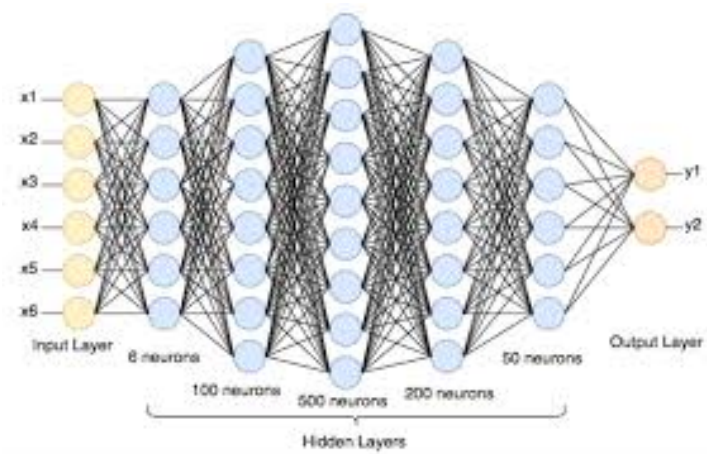
Dopo la struttura di base della rete, occorre selezionare i **dettagli dell'architettura** (ad esempio, il *numero di strati nascosti*, il *numero di neuroni*, ecc.)

In alcuni casi, la scelta dell'architettura di base determinerà automaticamente il numero di livelli. Ad esempio, se si è scelta l'architettura SOM per il clustering, allora la rete avrà un solo strato (la mappa caratteristica).

Nel caso della rete multistrato per problemi di regressione o classificazione, il numero di strati nascosti non è determinato dal problema. La procedura standard è quella di iniziare con un solo livello nascosto. Se le prestazioni della rete non sono soddisfacenti, si può aggiungere un ulteriore strato nascosto.

Raramente si utilizzano più di due strati nascosti, poiché l'addestramento della rete diventa più difficile. Questo perché ogni livello esegue un'operazione di "appiattimento" dovuto alle funzioni sigmoidee utilizzate, con relativo rallentamento della convergenza.

Per problemi molto difficili, tuttavia, è possibile utilizzare reti multistrato profonde (***deep neural network***) con diversi strati nascosti, per le quali si rende necessario il calcolo parallelo per la convergenza in un tempo ragionevole.



Fasi di pre-training

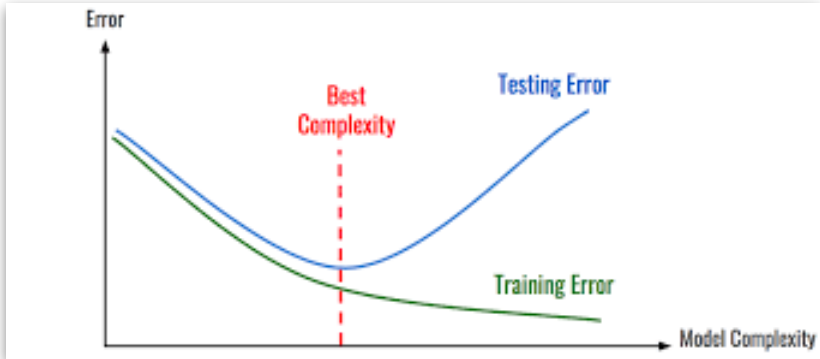
Scelta dell'architettura della rete → **Dettagli**

Occorre anche selezionare il **numero di neuroni in ogni strato**:

- il numero di neuroni dello strato di input è vincolato alla dimensione del vettore di input;
- il numero di neuroni nello strato di uscita è uguale alla dimensione del vettore di output;
- il numero di neuroni negli strati nascosti è determinato dalla complessità della funzione che si sta approssimando, che non è nota finché non proviamo ad addestrare la rete.

La procedura standard è quella di iniziare con più neuroni del necessario, per poi utilizzare la tecnica *early stopping* per prevenire l'**overfitting**.

Tuttavia, vi possono essere situazioni in cui **interessa limitare il tempo di calcolo o lo spazio di memoria** richiesti dalla rete (ad esempio, per l'implementazione in tempo reale su microcontrollori con risorse limitate). In questi casi si desidera trovare la rete più semplice che si adatta "ragionevolmente" ai dati.



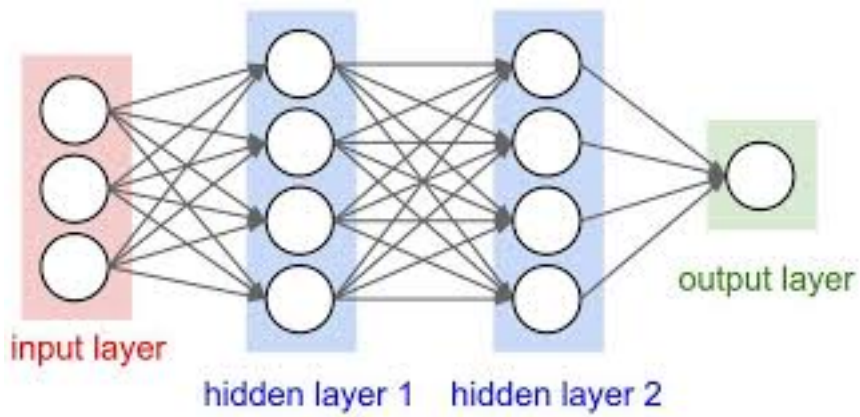
Fasi di pre-training

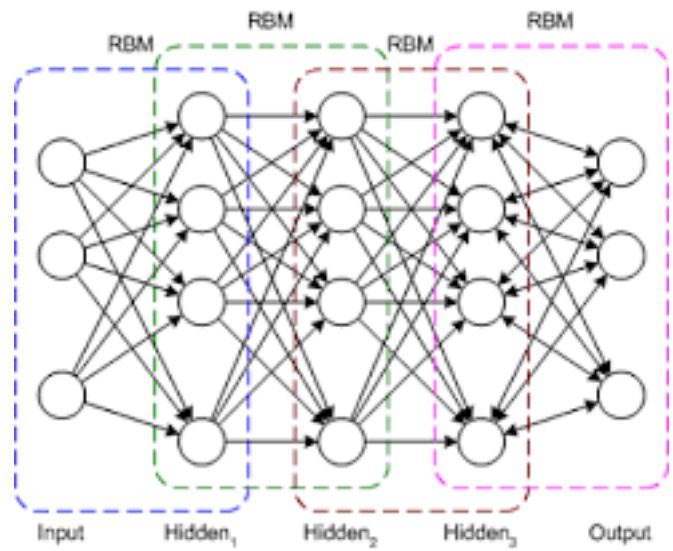
Scelta dell'architettura della rete → **Dettagli**

Quando vi sono **output multipli**, occorre decidere se avere una rete con più neuroni di output (uno per ciascun valore da calcolare) oppure implementare più reti, ognuna con un'unica uscita.

*Ad esempio, le reti neurali sono utilizzate per **stimare i livelli di colesterolo** LDL, VLDL e HDL, sulla base di un'analisi spettrale del sangue. È possibile avere una rete neurale con tre neuroni nello strato di uscita per stimare tutti e tre i livelli di colesterolo, oppure potremmo avere tre reti neurali, ognuna delle quali stima solo uno dei tre componenti.*

Teoricamente, entrambi i metodi dovrebbero funzionare, ma in pratica un metodo può funzionare meglio di un altro. Si inizia generalmente con una rete multi-output, e poi si passa a reti multiple single-output se i risultati originali non sono soddisfacenti.





Fasi di pre-training

Scelta dell'architettura della rete → **Dettagli**

Un'altra scelta architetturale è la **dimensione del vettore di ingresso**, che di solito è una scelta semplice determinata dai dati di input.

Tuttavia, vi sono situazioni in cui i dati di input hanno componenti (*feature*) ridondanti o irrilevanti, che può essere vantaggioso eliminare. Questo può ridurre il tempo di calcolo richiesto e può aiutare a prevenire l'*overfitting* durante l'addestramento.

Il processo di **scelta degli input** (*ranking*) per le reti non lineari può essere piuttosto difficile e non esiste una soluzione perfetta: le tecniche più utilizzate sono ad es. gli indici informativi (*Information Gain*, *Gain Ratio*), *Gini* (disuguaglianza tra i valori di una distribuzione di frequenza), *ANOVA* (differenza tra le medie di valori della feature in classi diverse), χ^2 (dipendenza tra la feature e la classe come misura della statistica del chi-quadro), *ReliefF* (capacità di un attributo di distinguere tra classi su istanze di dati simili), *FCBF* (Fast Correlation Based Filter, misura basata sull'entropia, che identifica la ridondanza dovuta a correlazioni a coppie tra feature).

Un'altra tecnica che può aiutare nella riduzione del vettore di ingresso è l'**analisi di sensitività** della rete addestrata, che sarà in seguito esaminata.

Training

Dopo la raccolta e pre-elaborazione dei dati e la scelta dell'architettura di rete, si passa alla fase di addestramento (**training**).

In questa sezione esamineremo alcune delle decisioni che devono essere prese nell'ambito del processo di addestramento.

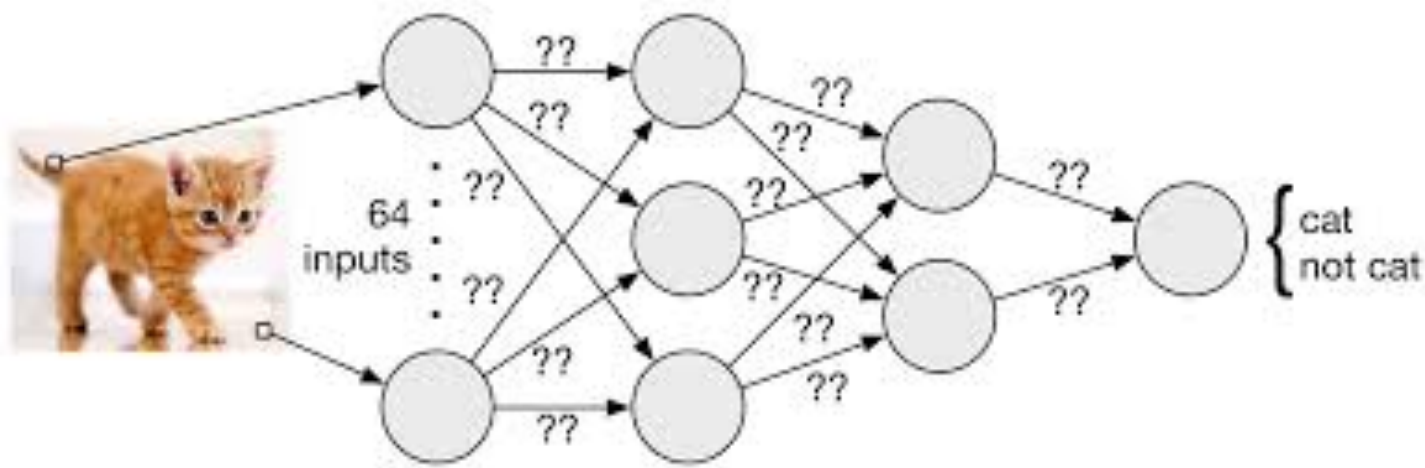
Questo include:

» il ***metodo di inizializzazione dei pesi***

» l'***algoritmo di training***

» l'***indice di prestazione*** della rete

» il ***criterio di arresto*** del training



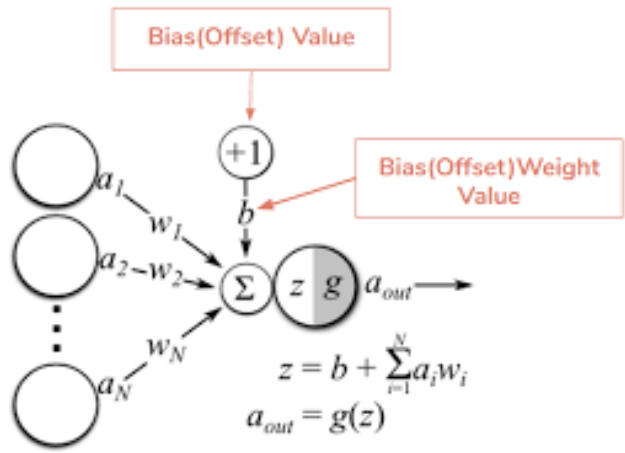
Training

Inizializzazione dei pesi

Prima di iniziare l'addestramento della rete, occorre **inizializzare i pesi e i bias**. Il metodo utilizzato dipende dal tipo di rete.

Per le reti multistrato, i pesi e i bias sono generalmente impostati a *piccoli valori casuali* (ad esempio, uniformemente distribuiti tra -0.5 e 0.5, se gli ingressi sono normalizzati tra -1 e 1) per evitare di cadere su un punto di sella della superficie di errore.

Per le reti competitive, i pesi possono essere impostati anche come piccoli numeri casuali. La dimensione iniziale dei neuroni vicini (*neighborhood*) è abbastanza grande in modo che tutti i neuroni abbiano l'opportunità di "imparare" durante le fasi iniziali di training. Questo sposterà tutti i vettori di peso nella regione appropriata dello spazio di input.



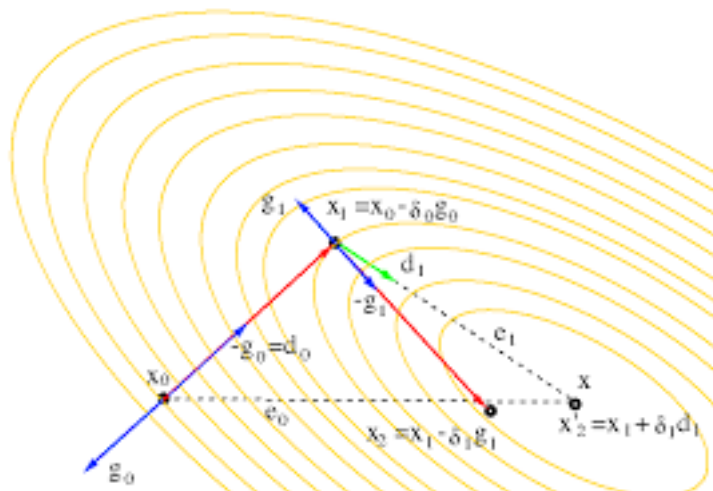
Training

Scelta dell'algoritmo di training

Per le reti multistrato si utilizzano generalmente algoritmi basati sul ***gradiente***. Questi algoritmi possono essere implementati in due **modalità**:

- **sequenziale**: i pesi vengono aggiornati dopo che ogni ingresso è presentato alla rete;
- **batch**: tutti gli ingressi sono presentati alla rete, e il gradiente totale è calcolato sommando i gradienti per ogni ingresso prima che i pesi siano aggiornati.

In alcune situazioni la forma sequenziale è preferibile (ad esempio quando è richiesta un'operazione on line). Tuttavia, molti degli algoritmi di ottimizzazione più efficienti (ad esempio, il gradiente coniugato e i il metodo di Newton) sono intrinsecamente algoritmi batch.

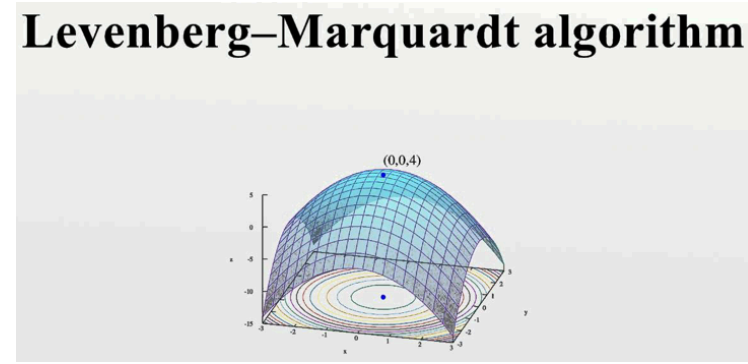


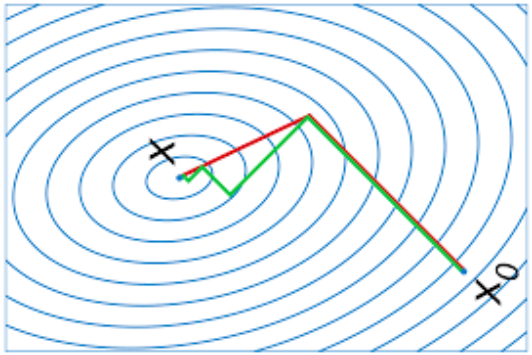
Training

Scelta dell'algoritmo di training

Per le reti multistrato con qualche centinaio di pesi e bias utilizzate per l'approssimazione di funzioni, l'algoritmo **Levenberg-Marquardt** è di solito il metodo di training più veloce.

Quando il numero di pesi raggiunge i mille o più, l'algoritmo Levenberg-Marquardt però non è efficiente, principalmente poiché il calcolo dell'inversa della matrice scala geometricamente con il numero di pesi.



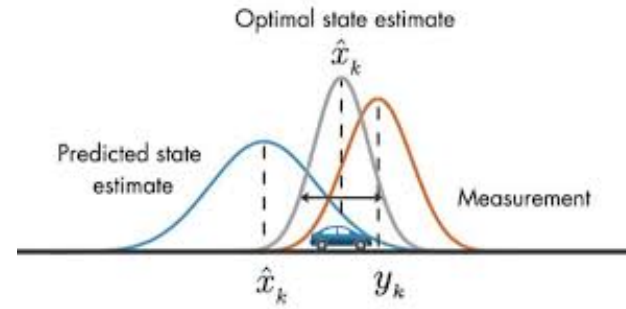


Per reti di grandi dimensioni, l'algoritmo **Scaled Conjugate Gradient** è molto efficiente.

Questo metodo è applicabile anche per problemi di pattern recognition.

Degli algoritmi che possono essere implementati in modalità sequenziale, i più veloci sono gli algoritmi estesi **Kalman filter** strettamente correlati alle implementazioni sequenziali dell'algoritmo di Gauss-Newton.

Essi però, a differenza della versione batch di Gauss-Newton, non richiedono un'inversione della matrice Hessiana approssimata.



Training

Criteri di stop

L'**errore di training** non converge quasi mai in modo identico a zero, specie per le reti multistrato. Per questo motivo occorre avere altri **criteri** per decidere quando interrompere l'addestramento della rete.

Possiamo fermare l'addestramento quando l'**errore raggiunge qualche limite predefinito** (anche se è difficile stabilire a priori un livello di errore accettabile).

Il criterio più semplice è quello di fermare il training dopo un numero prefissato di iterazioni, generalmente fissato ad un livello ragionevolmente alto. Se i pesi non convergono dopo che il numero massimo di iterazioni è stato raggiunto, si può **ripetere il training** utilizzando i pesi finali della prima epoca come condizioni iniziali

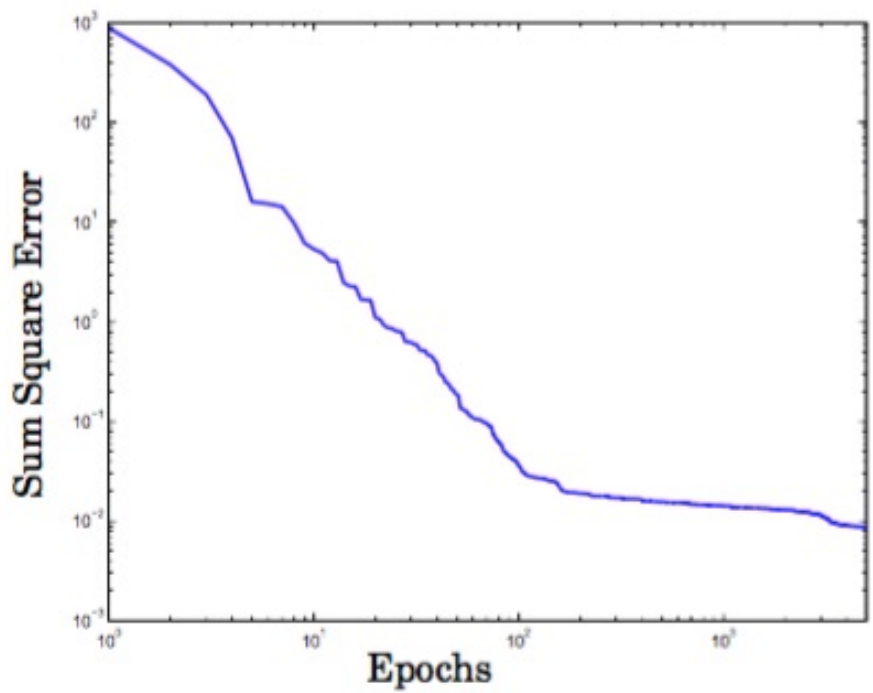
Un altro criterio di arresto è la **norma del gradiente** dell'indice di errore. Se questa norma raggiunge una soglia sufficientemente piccola, allora il training può essere fermato. Purtroppo la superficie di errore per reti multistrato può avere molte regioni "piatte", dove la norma del gradiente sarà piccola. Per questo motivo, la soglia per la norma minima dovrebbe essere impostata ad un valore molto piccolo (ad es. 10^{-6} per l'errore quadratico medio, con target normalizzati), in modo che il training non si concluda prematuramente.

Training

Criteri di stop

Possiamo anche interrompere il training quando la riduzione dell'indice di errore per iterazione diventa piccola. Come con la norma del gradiente, questo criterio può però fermare il training troppo presto.

Per le reti multistrato, l'errore può rimanere quasi costante per un certo numero di iterazioni prima di cadere improvvisamente. Quando il training è completo, è utile visualizzare la **curva dell'errore** su una scala log-log per verificare la convergenza.



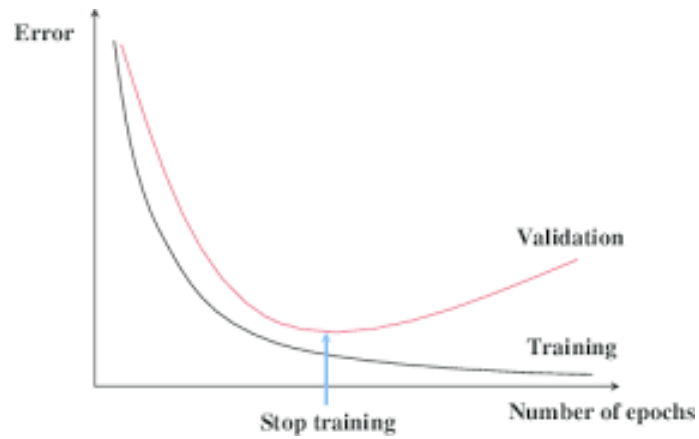
Training

Criteri di stop

Se si utilizza l'***early stopping*** per evitare l'*overfitting*, allora il training viene interrotto quando l'errore sul *validation set* aumenta per un certo numero di iterazioni. Oltre a prevenire l'overfitting, questa procedura di arresto fornisce anche una significativa riduzione del calcolo; per la maggior parte dei problemi pratici, l'errore di validazione aumenterà prima del raggiungimento di uno qualsiasi degli altri criteri di arresto.

L'addestramento di una rete neurale è un processo iterativo. Anche dopo la convergenza dell'algoritmo di training, l'**analisi post-training** può suggerire che la rete sia modificata e ri-addestrata.

Inoltre, dovrebbero essere effettuate diverse sessioni di training per assicurare che sia stato raggiunto un minimo globale per la rete.



Training

Criteri di stop

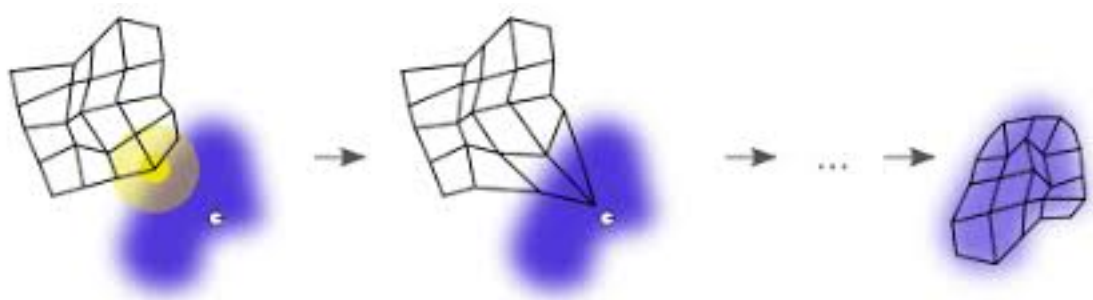
I precedenti criteri di arresto si applicano principalmente al training basato sul gradiente.

Quando si addestrano **reti competitive**, come SOM, non esiste un indice di errore o un gradiente esplicito da monitorare per la convergenza. Il training si ferma solo quando è stato raggiunto un **numero massimo di iterazioni** prestabilito.

Per le SOM, il tasso di apprendimento (*learning rate*) e la dimensione dei neuroni vicini (*neighborhood*) diminuiscono nel tempo. Quindi il numero massimo di iterazioni stabilisce la fine del training e rappresenta un parametro molto importante.

Generalmente lo si sceglie pari a più di dieci volte il numero di neuroni della rete. Si tratta di un numero approssimativo, e la rete deve essere analizzata al termine del training per determinare se le prestazioni sono soddisfacenti.

La rete potrebbe aver bisogno di essere addestrata più volte con diverse durate di training per ottenere un risultato soddisfacente.



Training

Scelta dell'indice di prestazione

Per le reti multistrato, l'indice di prestazione standard è l'**errore quadratico medio**:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Un altro indicatore è l'**errore assoluto medio**, generalmente meno sensibile ad outlier nei dati rispetto al precedente:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Questo concetto può essere esteso a qualsiasi potenza dell'errore assoluto, come segue:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|^K$$

dove K=2 corrisponde all'errore quadratico medio e K=1 all'errore assoluto medio. L'errore generale dato dalla precedente formula è anche noto come **errore di Minkowski**.

Training

Scelta dell'indice di prestazione → **Cross-entropy**

L'**errore quadrato medio** funziona bene per i problemi di approssimazione di funzioni, in cui i valori target sono continui.

Tuttavia, nei problemi di pattern recognition, dove le uscite assumono valori discreti, altri indici di prestazione sono più appropriati.

L'indice **cross-entropy** è definito come segue:

$$E(x) = -\sum_{i=1}^{N} y_i \ln \frac{\hat{y}_i}{y_i}$$

Si assume che i valori target siano 0 e 1 in corrispondenza delle due classi cui appartiene il vettore di input. La funzione di trasferimento *softmax* è generalmente utilizzata nell'ultimo strato della rete neurale, se si adopera l'indice di prestazione cross-enterpy.

Ricordiamo infine che l'algoritmo di *backpropagation* per il calcolo dei gradienti di training funzionerà per qualsiasi indice di prestazione differenziabile.

Training

Esecuzioni multiple e "comitati" di reti

Un singolo addestramento può non produrre prestazioni ottimali, a causa della possibilità di raggiungere un minimo locale della superficie di errore. È meglio **ripetere** (da cinque a dieci volte) **il training** in diverse condizioni iniziali e selezionare la rete che produce le migliori prestazioni.

Vi è un altro modo per eseguire più training e fare uso di tutte le reti ottenute. Questo metodo è chiamato **"comitato" di reti**.

Per ogni sessione di training, il *validation set* è selezionato a caso dal *training set*, e viene scelto un insieme casuale di pesi iniziali e bias. Dopo che le reti sono state addestrate, vengono usate tutte insieme per formare un output comune.

Per le reti di approssimazione di funzioni (regressione), l'output congiunto può essere una semplice media delle uscite di ogni rete.

Per le reti di classificazione, l'output congiunto può essere il risultato di una "votazione", in cui la classe scelta dalla maggioranza delle reti è selezionata come uscita.

Le **prestazioni** del "comitato" saranno di solito **migliori** anche della migliore delle singole reti. Inoltre, la variazione dei risultati delle singole reti può essere utilizzata per fornire *livelli di confidenza* per l'output ottenuto.

Analisi post-training

Prima di utilizzare una rete neurale addestrata, dobbiamo **analizzarla** per determinare se il training ha avuto successo.

Vi sono molte tecniche per l'**analisi post-training**; esamineremo quelle più comuni.

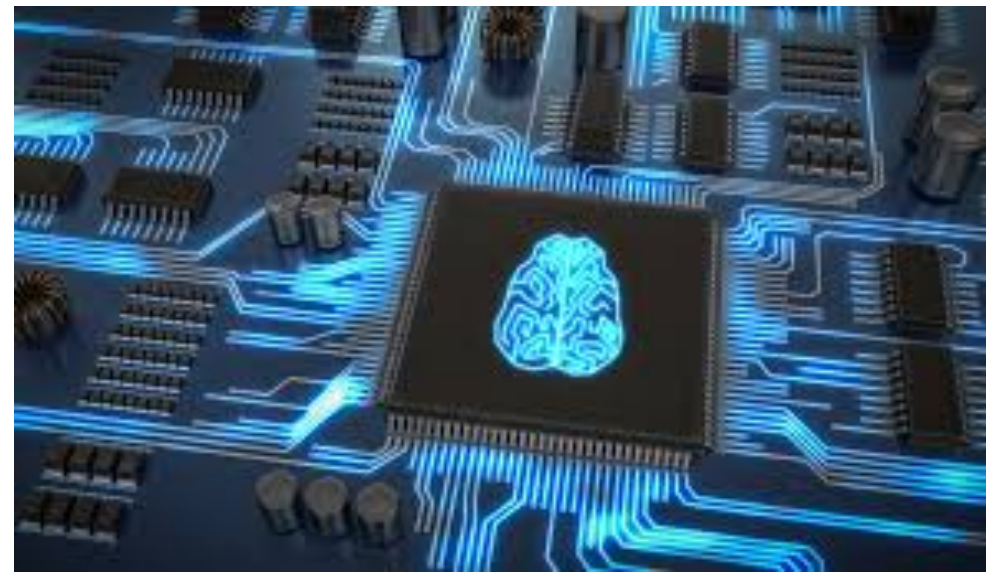
Poiché queste tecniche variano a seconda dell'applicazione, le organizzeremo in base a queste quattro aree di applicazione:

» **regressione** (*fitting*);

» **classificazione** (*pattern recognition*);

» **clustering**;

» **previsione** (su serie temporali).



Analisi post-training

Regressione (*fitting*)

Uno strumento utile per l'analisi delle reti neurali addestrate per problemi di *fitting* è la regressione tra gli output della rete addestrata e gli obiettivi (*target*) corrispondenti.

Si tratta di adattare una funzione lineare della forma:

$$\hat{y}_i = a \cdot y_i + b + \varepsilon_i$$

dove $\boldsymbol{a}$ e $\boldsymbol{b}$ sono rispettivamente il coefficiente angolare e l'ordinata all'origine della retta di regressione tra gli output $\boldsymbol{\hat{y}_i}$ ottenuti dalla rete ed i corrispondenti target $\boldsymbol{y_i}$ richiesti, con un errore residuo $\boldsymbol{\varepsilon_i}$.

I termini $\boldsymbol{a}$ e $\boldsymbol{b}$ della regressione possono essere stimati come segue:

$$\hat{a} = \frac{\sum_{i=1}^{N}(y_i - \overline{y})(\hat{y}_i - \overline{\hat{y}})}{\sum_{i=1}^{N}(y_i - \overline{y})^2}, \; \hat{b} = \overline{\hat{y}} - \hat{a} \cdot \overline{y}$$

dove $\overline{y}$ rappresenta il valore medio dei target e $\overline{\hat{y}}$ il valore medio degli output della rete.

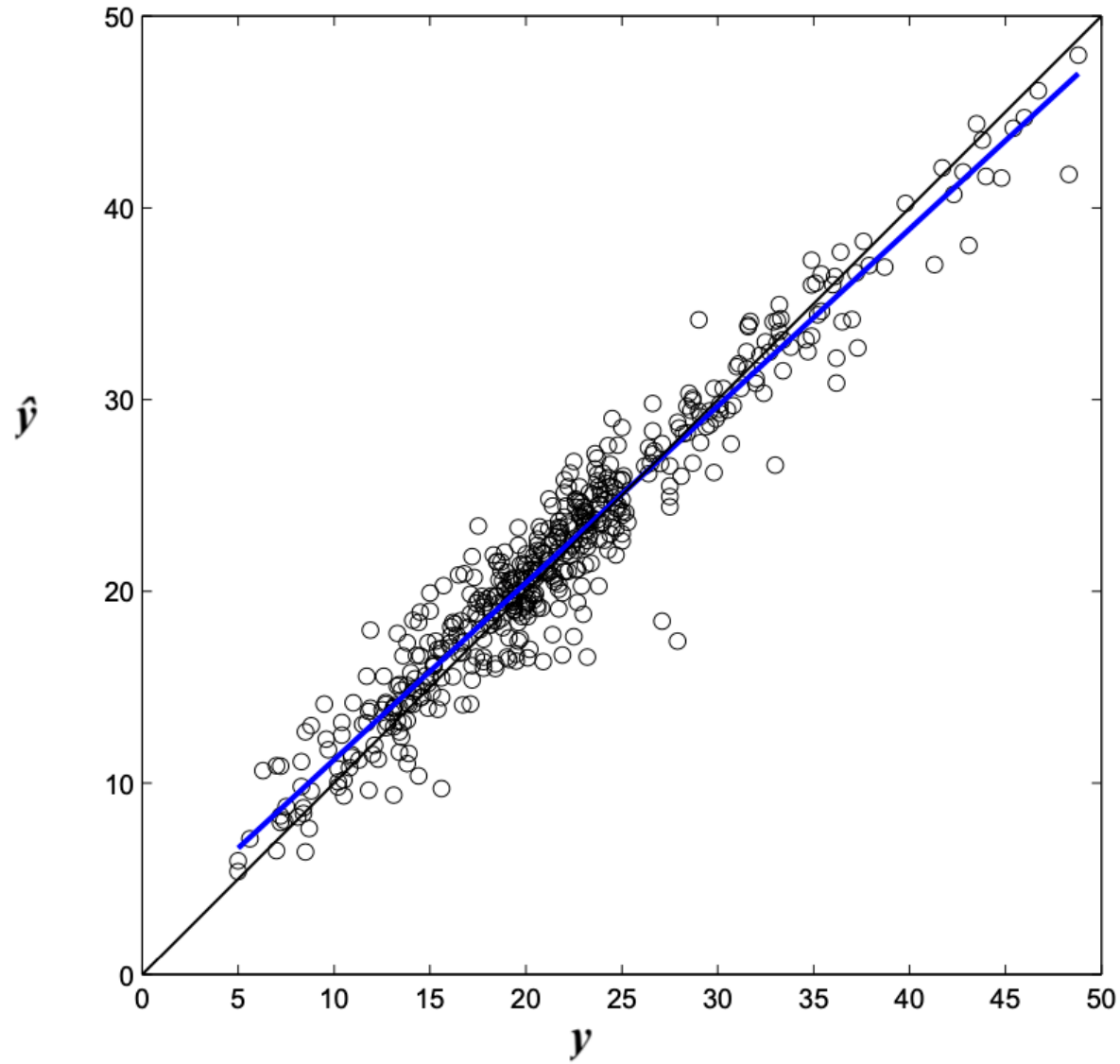
Analisi post-training

Regressione (*fitting*) → **Scatterplot (analisi di regressione)**

La figura mostra un esempio di **analisi di regressione** mediante *scatterplot* (diagramma di dispersione).

La linea blu rappresenta la regressione lineare, la sottile linea nera rappresenta la corrispondenza perfetta $\hat{y} = y$, e i cerchi rappresentano i dati.

In questo esempio, possiamo vedere che la corrispondenza è abbastanza buona, anche se non perfetta.



Analisi post-training

Regressione (*fitting*) → **Outliers**

Il passo successivo è quello di indagare i punti che cadono lontano dalla retta di regressione (**outliers**).

Per esempio, vi sono due punti intorno a (27;17) che sembrano essere outliers: sarebbe opportuno esaminarli in dettaglio per vedere se vi sono problemi con la raccolta e/o preprocessing dei dati.

Oltre ai coefficienti di regressione, spesso viene calcolato anche il **coefficiente di correlazione** (lineare) di Pearson $\boldsymbol{R}(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$:

$$R = \frac{\sum_{i=1}^{N}(y_i - \overline{y})(\hat{y}_i - \overline{\hat{y}})}{(N-1)\sigma_y \sigma_{\hat{y}}}$$

dove

$$\sigma_y = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})^2}\ , \quad \sigma_{\hat{y}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\hat{y}_i - \overline{\hat{y}})^2}$$

Analisi post-training

Regressione (*fitting*) → **Correlazione**

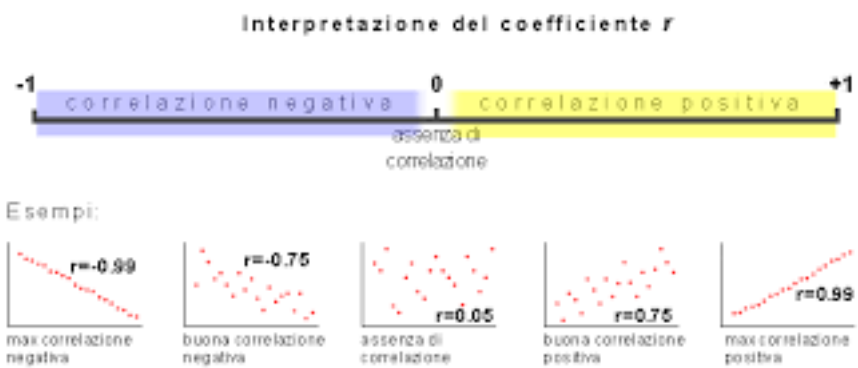
Il coefficiente R varia generalmente da **-1** (*correlazione negativa*) a 1 (*correlazione positiva*).

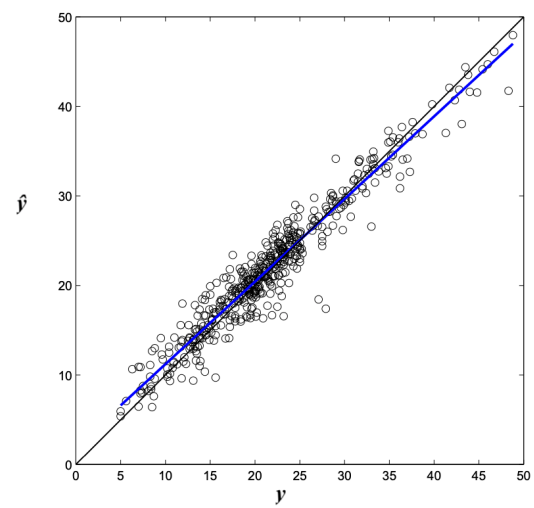
Se $R = 1$, allora tutti i dati cadranno esattamente sulla retta di regressione.

Se $R = 0$, allora i dati sono dispersi casualmente lontano dalla retta di regressione.

Per i dati della figura, abbiamo $R = 0{,}965$: possiamo osservare che i dati non cadono esattamente sulla linea di regressione, ma la variazione è relativamente piccola.

Il quadrato del coefficiente di correlazione, R^2, rappresenta la *proporzione della variabilità in un insieme di dati* che viene "catturata" dalla regressione lineare, e viene anche chiamato **coefficiente di determinazione**. In pratica, misura la percentuale della variabilità di $\hat{\boldsymbol{y}}$ spiegata dalla variabilità di $\boldsymbol{y}$. Per i dati della figura, $R^2 = 0{,}931$.





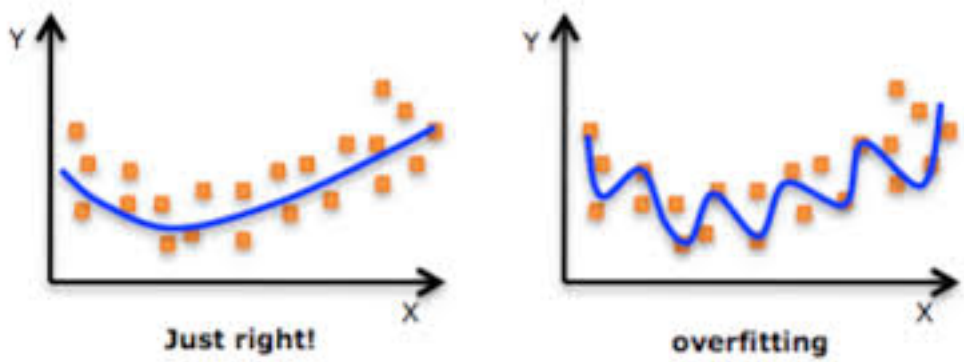
Analisi post-training

Regressione (*fitting*) → **Overfitting**

Quando i valori di R o R^2 sono significativamente **inferiori a 1**, allora la rete neurale non ha fatto un buon lavoro di approssimazione della funzione sottostante. Un'attenta analisi del diagramma di dispersione (*scatterplot*) può essere utile per determinare i problemi di *fitting*.

Ricordiamo che il set di dati originale viene diviso in *training*, *validazione* (se si usa l'early stopping) e *test*. L'analisi di regressione dovrebbe essere eseguita su ogni sottoinsieme individualmente, così come sull'intero set di dati. Le differenze tra i sottoinsiemi evidenzierebbero un overfitting o un'estrapolazione.

Ad esempio, se il training set mostra un ottimo adattamento, ma i risultati della validazione e dei test sono scarsi, ciò indica un ***overfitting****. In questo caso, potremmo ridurre le dimensioni della rete neurale (neuroni nascosti) e ripeterne l'addestramento.*



Analisi post-training

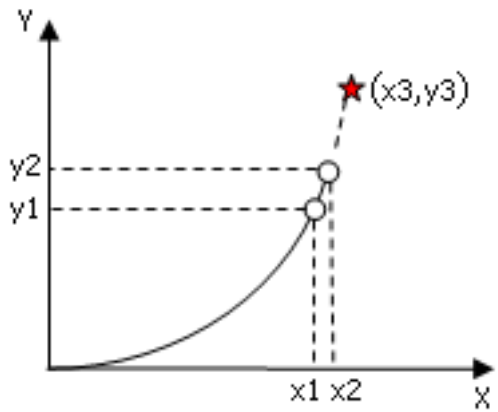
Regressione (*fitting*) → **Estrapolazione**

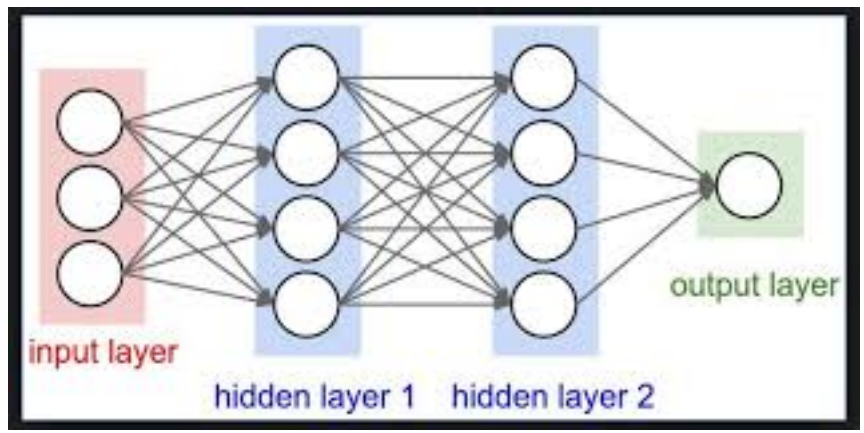
Se i risultati del training e della validazione sono buoni, ma i risultati dei test sono scarsi, allora questo potrebbe indicare un'**estrapolazione** (i dati di test cadono al di fuori dei dati di training e validazione). In questo caso, dobbiamo fornire più dati per l'addestramento e la convalida della rete.

Se i risultati per tutti e tre i set di dati sono scarsi, potrebbe essere necessario **aumentare il numero di neuroni** della rete.

Un'altra scelta è quella di **aumentare il numero di livelli nascosti** della rete.

Se si inizia con un singolo livello nascosto e i risultati sono scarsi, allora un secondo livello nascosto potrebbe essere utile. In primo luogo, conviene provare con più neuroni nel singolo livello nascosto, e quindi aumentare il numero di livelli se le prestazioni non sono soddisfacenti.





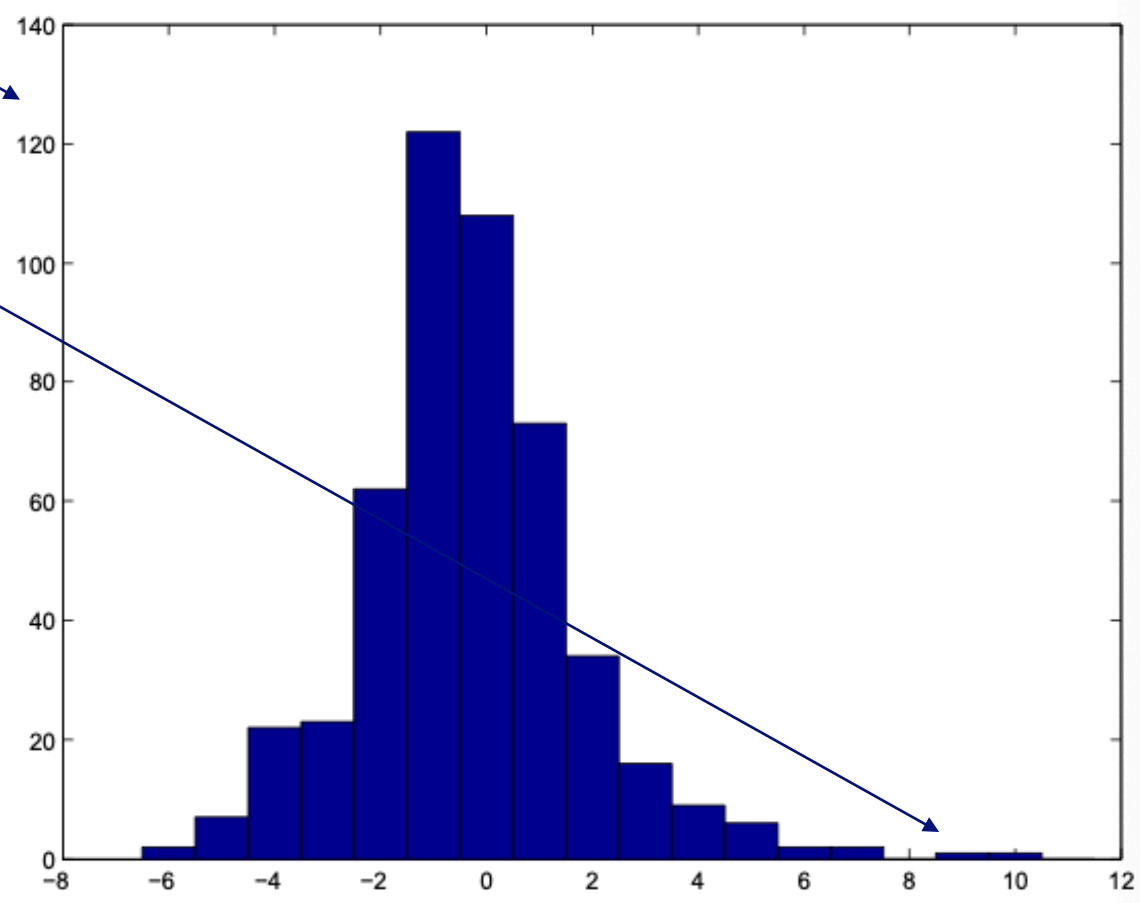
Analisi post-training

Regressione (*fitting*) → **Istogramma degli errori**

Oltre al diagramma di regressione (scatterplot), un altro strumento in grado di identificare i valori anomali (outliers) è l'**istogramma degli errori**, come mostrato in Figura.

L'asse verticale rappresenta il *numero di errori* che rientrano in ogni intervallo dell'asse orizzontale.

Qui possiamo vedere che due errori sono maggiori di 8: questi rappresentano gli stessi due errori identificati come outliers nel precedente scatterplot.



Analisi post-training

Classificazione → **Matrice di confusione**

Per i problemi di ***classificazione*** *(pattern recognition)*, l'analisi di regressione non è così utile come per i problemi di *fitting* poiché i valori target sono discreti.

Tuttavia, esiste uno strumento analogo: la **matrice di confusione**, una tabella le cui colonne rappresentano la *classe target* e le cui righe rappresentano la *classe di output della rete*.

La figura mostra un esempio di matrice di confusione con 447 osservazioni. Vi sono 54 input che appartengono alla classe 1 correttamente classificati come classe 1, e 358 input della classe 2 correttamente classificati come classe 2.

Gli input correttamente classificati sono mostrati nella diagonale principale della matrice di confusione. Le celle fuori dalla diagonale principale mostrano gli input erroneamente classificati.

*La cella in basso a sinistra mostra che 10 ingressi della classe 1 sono stati erroneamente classificati dalla rete come classe 2. Se la classe 1 è considerata un risultato positivo, allora la cella in basso a sinistra rappresenta i "**falsi negativi**", che sono anche chiamati errori di tipo II.*

*La cella in alto a destra mostra che 25 input della classe 2 sono stati erroneamente classificati come classe 1. Questo errore è chiamato "**falso positivo**" o errore di tipo I.*

		actual class: Positive (1)	actual class: Negative (2)
predicted class	Positive (1)	54 *true positive*	25 *false positive*
predicted class	Negative (2)	10 *false negative*	358 *true negative*

Analisi post-training

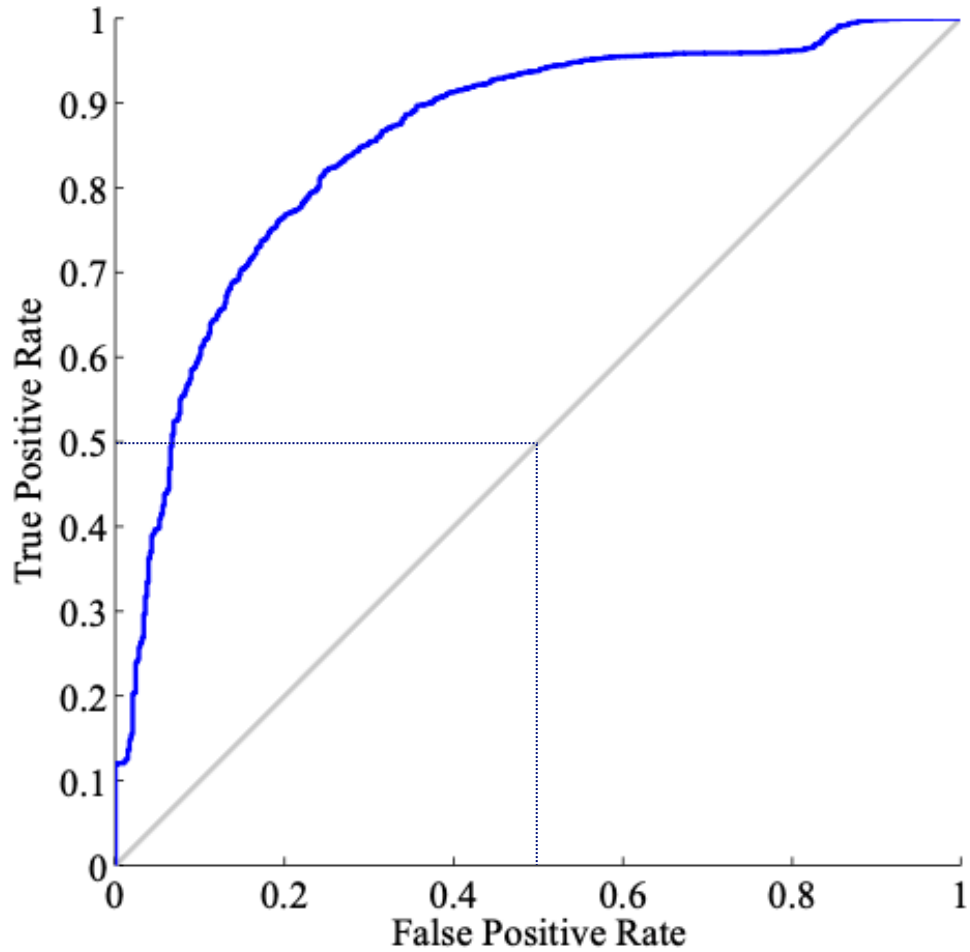
Classificazione → **Curva ROC (Receiver Operating Characteristic)**

Un altro strumento utile per l'analisi di una rete per il pattern recognition è la **curva ROC** (*Receiver Operating Characteristic*).

Per creare questa curva, si prende l'output della rete e lo si confronta con una soglia che va da -1 a +1 (assumendo una funzione di trasferimento tansig *nell'ultimo strato). Gli input che producono valori superiori alla soglia sono considerati come appartenenti alla Classe 1, mentre quelli con valori inferiori alla soglia sono considerati come appartenenti alla Classe 2.*

Per ogni valore di soglia, si conta la frazioni di falsi positivi (False Positive Rate) *e veri positivi* (True Positive Rate) *sul totale dei dati in esame. Questa coppia di numeri produce un punto sulla curva ROC. Al variare della soglia, si traccia la curva completa, come mostrato in figura.*

Il punto ideale per il passaggio della curva ROC sarebbe (0,1), che corrisponderebbe a nessun falso positivo e a tutti i veri positivi. Una curva ROC scadente rappresenterebbe un'ipotesi casuale, che è rappresentata dalla linea diagonale in figura che passa attraverso il punto (0.5,0.5).



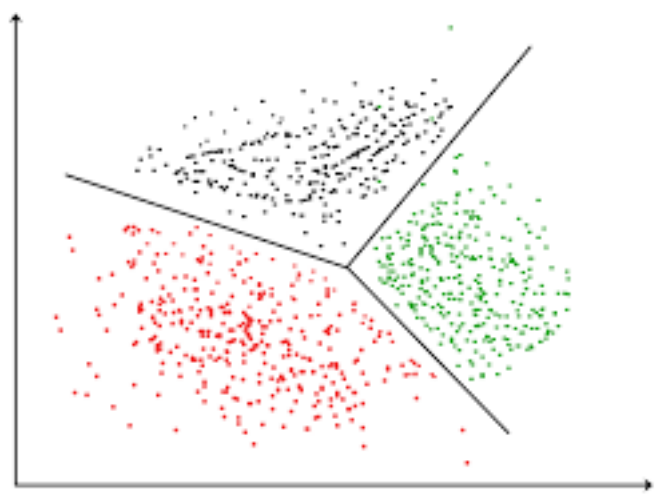
Analisi post-training

Clustering

L'architettura di rete SOM è quella più comunemente utilizzata per il ***clustering***: ci sono diverse *misure delle prestazioni della rete SOM*.

Una è l'*errore di quantizzazione*, corrispondente alla distanza media tra ogni vettore di ingresso e il vettore prototipo più vicino. Misura la "risoluzione" della mappa, e può essere reso artificialmente piccolo se si usa un gran numero di neuroni. Se ci sono tanti neuroni quanti sono i vettori di input nel set di dati, allora l'errore di quantizzazione potrebbe ridursi a zero: ciò rappresenterebbe un overfitting. Se il numero di neuroni non è significativamente inferiore al numero di vettori di input, allora l'errore di quantizzazione non è significativo.

Un'altra misura delle prestazioni della rete SOM è l'*errore topografico*. Questa è la proporzione di tutti i vettori di input per i quali i due vettori prototipo più vicini non sono vicini nella topologia della mappa caratteristica.



L'errore topografico misura la conservazione della topologia. In una SOM ben addestrata, i prototipi che sono vicini nella topologia dovrebbero anche essere vicini nello spazio di input. In questo caso, l'errore topografico dovrebbe essere zero.

Analisi post-training

Previsione

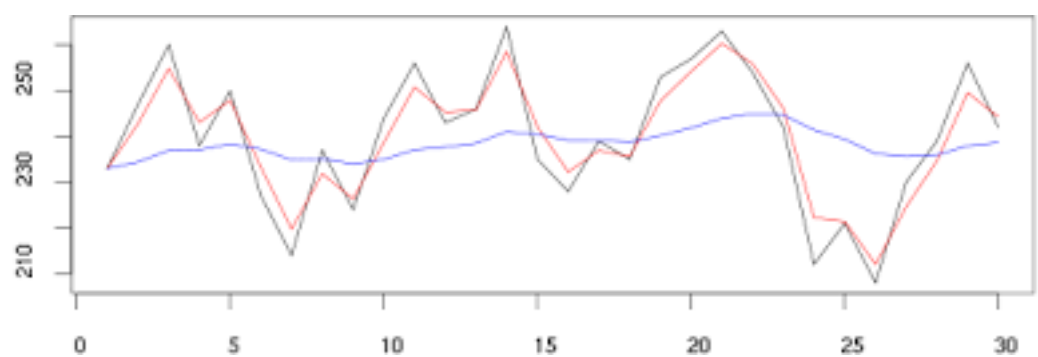
Come visto in precedenza, una delle applicazioni delle reti neurali è la ***previsione dei valori futuri*** di una serie temporale, per la quale si usano le *reti dinamiche*.

Vi sono due concetti importanti legati all'analisi di una rete predittiva addestrata:

1. gli errori di previsione non devono essere correlati nel tempo;
2. gli errori di previsione non devono essere correlati alla sequenza di input.

Se infatti gli errori di previsione fossero correlati nel tempo, allora saremmo in grado di prevedere gli errori di previsione e, quindi, migliorare la nostra previsione originale.

Inoltre, se gli errori di previsione fossero correlati con la sequenza di input, saremmo anche in grado di utilizzare questa correlazione per predire gli errori.



Analisi post-training

Overtitting e estrapolazione

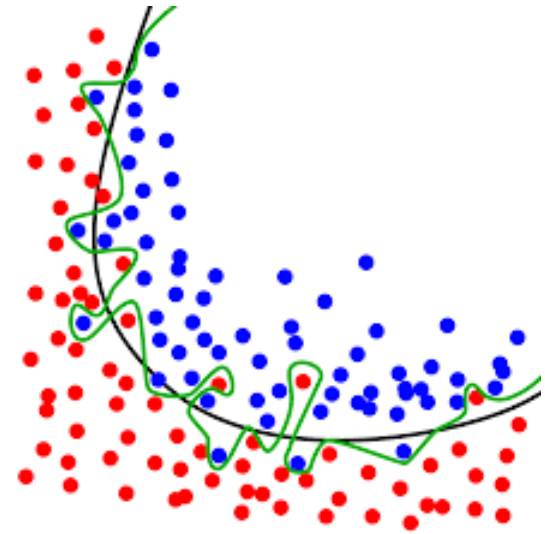
Ricordiamo che il dataset totale è normalmente diviso in tre parti: *training*, *validation* (opzionale) e *test*.

Il *training set* viene utilizzato per calcolare i gradienti e per determinare gli aggiornamenti dei pesi.

Il *validation set* viene utilizzato per interrompere l'addestramento prima che si verifichi l'*overfitting* (se si utilizza la regolarizzazione, può essere fuso con il training set).

Il *test set* viene utilizzato per valutare le prestazioni future della rete e ne misura quindi la qualità. Se, dopo che una rete è stata addestrata, le prestazioni del test set non sono adeguate, di solito vi sono quattro possibili cause:

1. la rete ha raggiunto un **minimo locale**;
2. la rete non ha abbastanza neuroni per "adattarsi" ai dati (**basso *fitting***);
3. la rete è sovradimensionata (***overfitting***);
4. la rete sta "**estrapolando**".



Analisi post-training

Overtitting e estrapolazione

Il problema del **minimo locale** può quasi sempre essere superato ri-addestrando la rete mediante delle serie casuali di pesi iniziali (di solito da 5 a 10): la rete con l'errore minimo rappresenta di solito un minimo globale.

Gli altri tre problemi possono essere generalmente distinti analizzando gli errori dei set di *training*, *validation* e *test*.

Ad esempio, se l'errore di validazione è molto più grande dell'errore di training, è probabile che si sia verificato un sovradimensionamento (***overfitting***). Anche se si utilizza l'arresto anticipato (*early stopping*), è possibile avere un po' di sovradimensionamento, se l'addestramento avviene troppo rapidamente. In questo caso, possiamo usare un algoritmo di training più lento per addestrare nuovamente la rete.

Se gli errori di training, validazione e test sono tutti di dimensioni simili, ma gli errori sono troppo grandi, è probabile che la rete non sia abbastanza potente per adattarsi ai dati (**basso *fitting***). In questo caso, dovremmo aumentare il numero di neuroni nello strato nascosto e ri-addestrare la rete.

Analisi post-training

Overtitting e estrapolazione

Se gli errori di *training* e di *validazione* sono di dimensioni simili, ma gli errori di *test* sono significativamente più grandi, allora la rete sta probabilmente **estrapolando**.

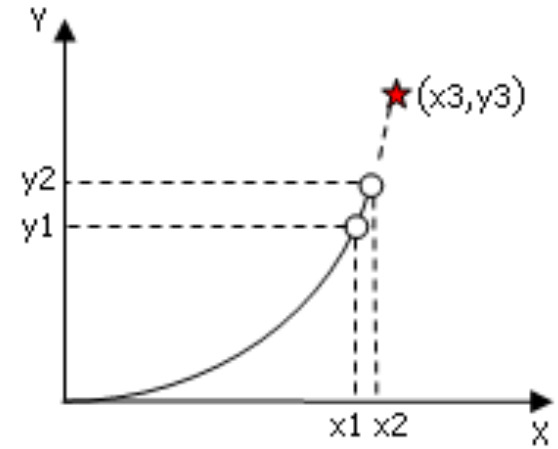
Ciò indica che i dati di test non rientrano nell'intervallo dei dati di training e di validazione.

In questo caso, **abbiamo bisogno di più dati**. È possibile unire i dati di test con i dati di training e validazione e quindi selezionare nuovi dati di test. Si dovrebbe continuare ad aggiungere dati di test fino a quando i risultati su tutti e tre i set di dati sono simili.

Se gli errori di *training*, *validazione* e *test* sono simili e gli errori sono abbastanza piccoli, allora **possiamo utilizzare la rete** ottenuta.

Tuttavia dobbiamo ancora fare attenzione alla possibilità di estrapolazione. Infatti, se viene fornito un input alla rete al di fuori dell'intervallo dei dati di training, avremo un'estrapolazione.

È difficile garantire che i dati di training comprendano tutti gli usi futuri di una rete neurale…



Analisi post-training

Analisi di sensitività

Dopo che una rete multistrato è stata addestrata, è spesso utile valutare l'importanza di ogni elemento (***feature***) del vettore di input. Se siamo in grado di determinare che una data *feature* non è importante, allora possiamo eliminarla. Questo può semplificare la rete, ridurre la quantità di calcolo e aiutare a prevenire l'overfitting.

Non esiste un solo metodo che possa con certezza determinare l'importanza di ogni *feature*, ma un'analisi di sensitività può essere utile a questo proposito. L'analisi calcola le derivate della risposta della rete rispetto a ciascuna *feature*. Se una derivata è piccola, allora quell'elemento può essere eliminato dal vettore di input.

Poiché la rete multistrato non è lineare, la derivata dell'output della rete rispetto ad un elemento di input non sarà costante. Per ogni vettore di ingresso nel training set, le derivate saranno diverse. Per questo motivo, non possiamo usare una singola derivata per determinare la sensitività.

Un'opzione sarebbe quella di prendere la media delle derivate assolute, oppure gli RMS delle derivate, sull'intero training set. Se alcune di queste derivate sono molto più piccole della derivata massima, allora possiamo prendere in considerazione la rimozione delle corrispondenti *feature*.

Dopo aver rimosso gli input potenzialmente irrilevanti, ri-addestriamo la rete e confrontiamo le prestazioni con la rete originale. Se le prestazioni sono simili, accettiamo la rete semplificata.