



UNIVERSITÀ DI PISA

DIPARTIMENTO DI ECONOMIA E MANAGEMENT

CORSO DI LAUREA IN BANCA, FINANZA AZIENDALE E MERCATI FINANZIARI

TESI DI LAUREA MAGISTRALE

RETI NEURALI E TRADING SYSTEMS:
APPLICAZIONE AL TASSO DI CAMBIO EUR/USD

CANDIDATO:
MATTEO MARCHETTI

RELATORE:
RICCARDO CAMBINI

ANNO ACCADEMICO 2016 - 2017

Indice

Prefazione

CAPITOLO 1: FOREIGN EXCHANGE MARKET

- 1.1 Introduzione
- 1.2 Storia
- 1.3 Dimensione del mercato e liquidità
- 1.4 Partecipanti al mercato
- 1.5 Caratteristiche del trading
- 1.6 Determinanti del tasso di cambio
- 1.7 Strumenti finanziari
- 1.8 Efficienza del foreign exchange market

CAPITOLO 2: RETI NEURALI ARTIFICIALI

- 2.1 Introduzione
- 2.2 Modello del neurone e architettura della rete
- 2.3 Perceptron learning rule
- 2.4 Apprendimento basato sulla performance
- 2.5 Steepest descent
- 2.6 Apprendimento di Widrow-Hoff
- 2.7 Backpropagation e relative variazioni
- 2.8 Generalizzazione e metodologie per incrementare la generalizzazione della rete neurale
- 2.9 Reti neurali dinamiche
- 2.10 L'allenamento della rete neurale: tecniche pratiche

CAPITOLO 3: CASO PRATICO

- 3.1 Introduzione
- 3.2 Tipo di reti neurali utilizzate nel trading system
 - 3.2.1 Rete neurale net2
 - 3.2.2 Rete neurali net3 e net4
- 3.3 Trading system utilizzati
- 3.4 Descrizione dei risultati
- 3.5 Conclusione e possibili sviluppi della metodologia

Conclusione

Prefazione

Ho deciso di scrivere questa tesi sulla creazione di trading system attraverso le reti neurali in quanto, durante tutto il mio percorso di studi, mi sono sempre interessato al trading in generale e soprattutto al trading attraverso sistemi computerizzati e automatizzati che potessero permettere al trader di operare eliminando totalmente quella che è una delle cause principali di fallimento per chi opera sui mercati finanziari, ovvero il fattore psicologico, l'emozionalità.

Tra tutte le procedure che possono essere automatizzate per creare trading system mi sono concentrato su quelle basate su questi nuovi modelli non lineari di previsione, le reti neurali appunto, e ho cercato di applicarle ad un mercato molto veloce e dinamico come quello del Forex, in particolare al cambio EUR/USD. Un altro dei motivi che mi ha spinto a studiare questo argomento è senza dubbio la crescita esponenziale che ha avuto il settore del trading "fai da te" pubblicizzato enormemente dai media (internet in primis), con le nuove piattaforme online che promettono grandi e soprattutto facili guadagni e che permettono di negoziare con capitali veramente esigui.

Tuttavia basta guardare queste pubblicità per rendersi conto di come queste piattaforme considerano i propri clienti: per esempio non esiste nessuna differenza sostanziale tra la pubblicità di una piattaforma di negoziazione online e una piattaforma di scommesse sportive, e ciò significa che per i gestori essere uno scommettitore o un trader è essenzialmente la stessa cosa. Essere un operatore che negozia sui mercati finanziari deve però necessariamente essere qualcosa di diverso, qualcosa di più: esso non può basarsi su istinti o sensazioni, esso deve basarsi, almeno per far sì che la propria attività sia redditizia nel lungo periodo, su un continuo studio del mercato, su un continuo aggiornamento e soprattutto su una metodologia rigida che coinvolga il meno possibile l'emozionalità. L'elaborato è suddiviso in tre capitoli: nel primo viene descritto il mercato di riferimento dello studio, ovvero il Foreign exchange market; nel secondo vengono spiegate nel dettaglio le reti neurali ed il loro funzionamento dal punto di vista matematico; nel terzo vengono descritti i trading system realizzati e commentati i relativi risultati ottenuti in una fase di test simulata sul mercato.

Capitolo 1: Foreign Exchange Market

1.1 Introduzione

Il foreign exchange market, meglio conosciuto come Forex, è di gran lunga il mercato più grande e più liquido del mondo. È un mercato over the counter (OTC) caratterizzato da operatività continua, 24 ore al giorno, cinque giorni su sette: è quindi possibile operare dalle 22:00 GMT della domenica alle 22:00 GMT del venerdì. Ovviamente questo mercato è fondamentale per gli investimenti e per il commercio internazionale dato che permette la conversione delle valute. Per esempio, possiamo considerare un'azienda americana che importa beni dall'Inghilterra: attraverso il foreign exchange market, l'azienda potrà pagare i beni in Sterline inglesi, anche se il suo reddito è in Dollari americani. Una particolarità di questo mercato è inoltre il fatto che non viene stabilito il valore assoluto di una valuta, ma piuttosto, dato che le valute vengono sempre scambiate in coppie, si stabilisce il valore relativo di una valuta in termini di un'altra.

1.2 Storia

All'inizio del ventesimo secolo la moneta più scambiata a livello globale era la Sterlina inglese; durante questo periodo Londra, Parigi, New York e Berlino diventarono le città più attive nel commercio valutario, e il numero di foreign exchange brokers crebbe rapidamente. Successivamente, nel 1944, 730 delegati di 44 nazioni siglarono gli accordi di Bretton Woods, che in sintesi possono essere considerati il risultato di un compromesso tra due progetti: quello di Harry Dexter White (delegato degli Stati Uniti d'America) e quello di John Maynard Keynes (delegato dell'Inghilterra). I punti chiave del compromesso erano:

- La creazione dell'IMF (International Monetary Fund) con l'obiettivo della stabilità;
- La creazione della WB (World Bank) con l'obiettivo dell'equità;
- La creazione dell'ITO (International Trade Organization), con l'obiettivo dell'efficienza.

Gli accordi di Bretton Woods possono essere considerati come il primo esempio di ordine monetario completamente determinato. Essi rappresentavano un sistema a cambi fissi: ogni paese doveva stabilire la parità della valuta nazionale nei confronti della valuta di riserva e quindi mantenere il tasso di cambio all'interno di un range di $\pm 1\%$ intervenendo nel mercato del forex. In teoria la valuta di riserva sarebbe dovuta essere il "bancor" (valuta ideata da Keynes e mai creata) ma in pratica questo ruolo venne assunto dal Dollaro americano. Questi accordi rappresentavano perciò il cosiddetto regime del gold exchange standard: un sistema a tassi di cambio fissi nel quale solo il Dollaro americano poteva essere scambiato direttamente in oro e tutte le altre valute dovevano essere scambiate prima in Dollari americani e successivamente in oro (un oncia d'oro, pari a circa 28 grammi, valeva 35 Dollari americani). Tuttavia, durante la Guerra Fredda, alcuni problemi legati a questo sistema vennero alla luce. Gli Stati Uniti infatti dovevano fronteggiare gli elevati costi legati alla guerra in Vietnam, e per far questo iniziarono a stampare moneta, ma il risultato di questa azione fu semplicemente un innalzamento dell'offerta di Dollari americani. A quel punto, alcuni dei paesi che siglarono l'accordo nel 1944 iniziarono a credere che gli Stati Uniti non avessero abbastanza oro per cambiare tutti i Dollari americani in circolazione ed infatti, quando chiesero alla FED di cambiare le loro riserve di valuta estera in oro la risposta fu negativa, proprio perché le riserve auree americane erano troppo contenute rispetto ai Dollari in circolazione. Il 15 Agosto del 1971 il presidente Richard Nixon fu quindi costretto a dichiarare l'inconvertibilità del Dollaro americano in oro. Questa decisione rese il sistema di Bretton Woods inoperante e per questo, nel Dicembre del 1971, il Gruppo dei Dieci (Belgio, Canada, Francia, Germania, Italia, Giappone, Olanda, Svezia, Svizzera, Regno Unito, Stati Uniti) si riunì allo Smithsonian Institution a Washington D.C. per siglare lo Smithsonian Agreement, ponendo fine agli accordi di Bretton Woods: gli Stati Uniti si impegnarono a fissare il Dollaro americano a 38\$/oncia con una banda di oscillazione del 2.25% e svalutarono la propria moneta. Dal 1973 ogni legame tra il Dollaro americano e le altre valute terminò definitivamente e il regime del gold exchange standard fu rimpiazzato da un

sistema a cambi flessibili. Per concludere quindi, è possibile affermare come il 1973 possa essere considerato l'anno di inizio del moderno mercato del forex.

1.3 Dimensione del mercato e liquidità

Come già esposto in precedenza, il forex è il mercato più liquido e più grande del mondo. È un mercato in continua crescita: secondo la Bank of International Settlement (BIS), il turnover (volume totale delle transazioni) giornaliero medio è aumentato da 1381 miliardi di Dollari nel 2001 a 5067 miliardi di dollari nel 2016, cioè ha subito un incremento del 266.91%, che certamente è molto significativo. Inoltre, di questi 5067 miliardi di dollari, 1652 miliardi di dollari sono relativi alle transazioni spot, 700 agli outright forwards, 2378 ai foreign exchange swaps, 82 ai currency swaps e 254 alle opzioni ed altri prodotti.

OTC foreign exchange turnover						
Net-net basis, ¹ daily averages in April, in billions of US dollars						Table 1
Instrument	2001	2004	2007	2010	2013	2016
Foreign exchange instruments	1,239	1,934	3,324	3,973	5,357	5,067
Spot transactions	386	631	1,005	1,489	2,047	1,652
Outright forwards	130	209	362	475	679	700
Foreign exchange swaps	656	954	1,714	1,759	2,240	2,378
Currency swaps	7	21	31	43	54	82
Options and other products ²	60	119	212	207	337	254
<i>Memo:</i>						
Turnover at April 2016 exchange rates ³	1,381	1,884	3,123	3,667	4,917	5,067
Exchange-traded derivatives ⁴	12	25	77	145	145	115

¹ Adjusted for local and cross-border inter-dealer double-counting (ie "net-net" basis). ² The category "other FX products" covers highly leveraged transactions and/or trades whose notional amount is variable and where a decomposition into individual plain vanilla components was impractical or impossible. ³ Non-US dollar legs of foreign currency transactions were converted into original currency amounts at average exchange rates for April of each survey year and then reconverted into US dollar amounts at average April 2016 exchange rates. ⁴ Sources: Euromoney Tradedata; Futures Industry Association; The Options Clearing Corporation; BIS derivatives statistics. Foreign exchange futures and options traded worldwide.

Per quanto riguarda la distribuzione geografica del turnover del mercato del forex, nel 2016 il Regno Unito è stato il paese con il più alto livello di trading (36.9%), seguito da Stati Uniti (19.5%), Singapore (7.9%), Hong Kong (6.7%) e Giappone (6.1%). L'elevato tasso di crescita di questo mercato può essere ricondotto a vari fattori: in primo luogo la crescente attività degli High Frequency Traders (HFT), che grazie ad algoritmi computerizzati sono in grado di eseguire una quantità

impressionante di ordini ogni secondo; in secondo luogo la crescita del numero di piccole banche che partecipano a questo mercato; in terzo luogo la crescita del numero di investitori retail che partecipano a questo mercato. Riguardo a quest'ultimo fattore possiamo senza dubbio dire che esso è stato una naturale

Geographical distribution of OTC foreign exchange turnover¹

Net-gross basis,² daily averages in April, in billions of US dollars and percentages

Table 6

Country	2001		2004		2007		2010		2013		2016	
	Amount	%	Amount	%	Amount	%	Amount	%	Amount	%	Amount	%
Argentina	1	0.0	1	0.0	2	0.0	1	0.0	1	0.0
Australia	54	3.2	107	4.1	176	4.1	192	3.8	182	2.7	121	1.9
Austria	8	0.5	15	0.6	19	0.4	20	0.4	15	0.2	19	0.3
Bahrain	3	0.2	3	0.1	3	0.1	5	0.1	9	0.1	6	0.1
Belgium	10	0.6	21	0.8	50	1.2	33	0.6	22	0.3	23	0.4
Brazil	6	0.3	4	0.1	6	0.1	14	0.3	17	0.3	20	0.3
Bulgaria	1	0.0	1	0.0	2	0.0	2	0.0
Canada	44	2.6	59	2.3	64	1.5	62	1.2	65	1.0	86	1.3
Chile	2	0.1	2	0.1	4	0.1	6	0.1	12	0.2	7	0.1
China	1	0.0	9	0.2	20	0.4	44	0.7	73	1.1
Chinese Taipei	5	0.3	9	0.4	16	0.4	18	0.4	26	0.4	27	0.4
Colombia	0	0.0	1	0.0	2	0.0	3	0.1	3	0.0	4	0.1
Czech Republic	2	0.1	2	0.1	5	0.1	5	0.1	5	0.1	4	0.1
Denmark	24	1.4	42	1.6	88	2.1	120	2.4	117	1.8	101	1.5
Estonia	0	0.0	1	0.0	1	0.0	0	0.0
Finland	2	0.1	2	0.1	8	0.2	31	0.6	15	0.2	14	0.2
France	50	2.9	67	2.6	127	3.0	152	3.0	190	2.8	181	2.8
Germany	91	5.4	120	4.6	101	2.4	109	2.2	111	1.7	116	1.8
Greece	5	0.3	4	0.2	5	0.1	5	0.1	3	0.0	1	0.0
Hong Kong SAR	68	4.0	106	4.1	181	4.2	238	4.7	275	4.1	437	6.7
Hungary	1	0.0	3	0.1	7	0.2	4	0.1	4	0.1	3	0.1
India	3	0.2	7	0.3	38	0.9	27	0.5	31	0.5	34	0.5
Indonesia	4	0.2	2	0.1	3	0.1	3	0.1	5	0.1	5	0.1
Ireland	9	0.5	7	0.3	11	0.3	15	0.3	11	0.2	2	0.0
Israel	1	0.1	5	0.2	8	0.2	10	0.2	8	0.1	8	0.1
Italy	18	1.0	23	0.9	38	0.9	29	0.6	24	0.4	18	0.3
Japan	153	9.0	207	8.0	250	5.8	312	6.2	374	5.6	399	6.1
Korea	10	0.6	21	0.8	35	0.8	44	0.9	48	0.7	48	0.7
Latvia	2	0.1	3	0.1	2	0.0	2	0.0	1	0.0
Lithuania	1	0.0	1	0.0	1	0.0	1	0.0	0	0.0
Luxembourg	13	0.8	15	0.6	44	1.0	33	0.7	51	0.8	37	0.6
Malaysia	1	0.1	2	0.1	3	0.1	7	0.1	11	0.2	8	0.1
Mexico	9	0.5	15	0.6	15	0.4	17	0.3	32	0.5	20	0.3
Netherlands	31	1.8	52	2.0	25	0.6	18	0.4	112	1.7	85	1.3
New Zealand	4	0.2	7	0.3	13	0.3	9	0.2	12	0.2	10	0.2
Norway	13	0.8	14	0.6	32	0.7	22	0.4	21	0.3	40	0.6
Peru	0	0.0	0	0.0	1	0.0	1	0.0	2	0.0	1	0.0
Philippines	1	0.1	1	0.0	2	0.1	5	0.1	4	0.1	3	0.0
Poland	5	0.3	7	0.3	9	0.2	8	0.2	8	0.1	9	0.1
Portugal	2	0.1	2	0.1	4	0.1	4	0.1	4	0.1	2	0.0
Romania	3	0.1	3	0.1	3	0.1	3	0.0
Russia	10	0.6	30	1.1	50	1.2	42	0.8	61	0.9	45	0.7
Saudi Arabia	2	0.1	2	0.1	4	0.1	8	0.1	7	0.1	8	0.1
Singapore	104	6.1	134	5.1	242	5.6	266	5.3	383	5.7	517	7.9
Slovakia	1	0.0	2	0.1	3	0.1	0	0.0	1	0.0	2	0.0
Slovenia	0	0.0	0	0.0	0	0.0
South Africa	10	0.6	10	0.4	14	0.3	14	0.3	21	0.3	21	0.3
Spain	8	0.5	14	0.5	17	0.4	29	0.6	43	0.6	33	0.5
Sweden	25	1.5	32	1.2	44	1.0	45	0.9	44	0.7	42	0.6
Switzerland	76	4.5	85	3.3	254	5.9	249	4.9	216	3.2	156	2.4
Thailand	2	0.1	3	0.1	6	0.1	7	0.1	13	0.2	11	0.2
Turkey	1	0.1	3	0.1	4	0.1	17	0.3	27	0.4	22	0.3
United Kingdom	542	31.8	835	32.0	1,483	34.6	1,854	36.7	2,726	40.8	2,406	36.9
United States	273	16.0	499	19.1	745	17.4	904	17.9	1,263	18.9	1,272	19.5
Total	1,705	100.0	2,608	100.0	4,281	100.0	5,045	100.0	6,686	100.0	6,514	100.0

¹ Data may differ slightly from national survey data owing to differences in aggregation procedures and rounding. The data for the Netherlands are not fully comparable over time due to reporting improvements in 2013. ² Adjusted for local inter-dealer double-counting (ie "net-gross" basis).

conseguenza della nascita, negli ultimi anni, di innumerevoli piattaforme online che consentono facilmente agli investitori retail di operare in questo mercato con bassi costi di negoziazione. Tutti questi fattori hanno quindi contribuito ad aumentare la dimensione e la liquidità del mercato, a abbassare i costi di transazione e ad attrarre sempre più partecipanti di ogni tipo.

1.4 Partecipanti al mercato

Il mercato del forex è costituito da diversi partecipanti. Essi negoziano direttamente tra loro e possono essere suddivisi in due categorie: i partecipanti al mercato del forex interbancario e i partecipanti al mercato del forex retail. Il mercato interbancario è relativo alle transazioni effettuate tra banche centrali, commercial banks e financial institution.

- Banche centrali

Le banche centrali sono i partecipanti più importanti al mercato del forex. Esse giocano un ruolo essenziale per questo mercato e per l'intera economia cercando di raggiungere il loro obiettivo principale, dato dalla stabilità dei prezzi e dalla crescita economica. Le banche centrali intervengono nel mercato del forex con transazioni di ammontare significativo: per esempio, attraverso le operazioni di mercato aperto, comprano o vendono bond governativi per aumentare o diminuire la base monetaria in seguito ad una decisione di ridurre o aumentare i tassi di interesse, e ciò può portare a riflessi più o meno forti sul mercato valutario nel complesso.

- Commercial banks

Attraverso il volume di scambi che gestiscono ogni giorno, le commercial banks provvedono ad aumentare significativamente la liquidità giornaliera del mercato del forex. Esse operano principalmente per due motivi: per scambiare valuta straniera per conto dei propri clienti e per speculare sulla variazione dei tassi di cambio. Solo per menzionare i primi cinque maggiori trader in valuta a livello globale, secondo una ricerca dell'Euromoney FX, nel 2016 Citigroup ha realizzato il 12.9% delle operazioni totali, JP Morgan e UBS l'8.8%, Deutsche bank il 7.9%

e Bank of America Merrill Lynch il 6.4%.¹

Top 10 currency traders % of overall volume, May 2016		
Rank	Name	Market share
1	 Citi	12.9 %
2	 JP Morgan	8.8%
3	 UBS	8.8%
4	 Deutsche Bank	7.9%
5	 Bank of America Merrill Lynch	6.4%
6	 Barclays	5.7%
7	 Goldman Sachs	4.7%
8	 HSBC	4.6%
9	 XTX Markets	3.9%
10	 Morgan Stanley	3.2%

- Financial institution

Le istituzioni finanziarie che operano nel mercato del forex sono i fondi pensione, i fondi di investimento e le compagnie di brokeraggio. Esse partecipano a questo mercato per diversificare i loro portafogli di investimento e per cercare le migliori opportunità di investimento per i loro clienti.

Il mercato retail è invece costituito dalle transazioni effettuate tra brokers, che agiscono come intermediari tra il mercato retail e il mercato interbancario. I partecipanti sono gli hedge funds, le commercial companies e i retail traders.

- Hedge funds

Gli hedge funds sono fondi di investimento che operano nel mercato del forex esclusivamente per speculare sull'andamento futuro dei tassi di cambio. Essi sono investitori istituzionali, e come tutti gli investitori istituzionali operano solo dopo attente e accurate analisi macroeconomiche, che permettono loro di avere un'idea esatta sul valore di una particolare valuta nei confronti di un'altra. Inoltre, per aumentare i propri profitti, utilizzano la leva finanziaria, e date le loro strategie

¹ Foreign exchange market, Wikipedia. Ultimo aggiornamento 19 Agosto 2017.
https://en.wikipedia.org/wiki/Foreign_exchange_market

aggressive e l'elevata liquidità di cui dispongono possono avere un forte impatto sul mercato.

- **Commercial companies**

Le commercial companies partecipano a questo mercato in quanto prendono parte a operazioni di import/export con aziende straniere. Solitamente l'ammontare delle transazioni effettuate da questo tipo di partecipanti è sensibilmente più basso rispetto alle categorie precedenti, ma rappresenta comunque un fattore molto importante nel trend di lungo periodo di un determinato tasso di cambio. In particolare, le commercial companies che possono generare l'impatto più forte su questo tipo di mercato sono le multinazionali, perché dispongono di elevata liquidità e perché, data la loro attività internazionale, effettuano transazioni di rilevante significatività.

- **Retail traders**

I retail traders sono trader individuali che negoziano i propri capitali nel mercato del forex cercando di trarre profitto dalla speculazione sull'andamento futuro dei tassi di cambio. Oggigiorno, essi operano principalmente tramite piattaforme di trading online, che offrono spread contenuti e immediata esecuzione degli ordini.

1.5 Caratteristiche del trading

A causa della natura OTC del mercato del forex, questo mercato è privo di quei meccanismi di trasparenza e standardizzazione tipici dei mercati regolamentati; è costituito da una serie di mercati interconnessi dove vengono scambiati i diversi strumenti finanziari. Ciò comporta che non vi sia un singolo, univoco tasso di cambio, ma piuttosto vari tassi di cambio, che dipendono da quale banca o da quale market maker sta operando, e da dove esso è localizzato. In pratica i contratti vengono scambiati direttamente tra le controparti senza nessuna cassa di compensazione e senza nessuna piattaforma di contrattazione che standardizzi i contratti stessi, garantisca i prezzi e il rischio di controparte. In ogni caso, è necessario evidenziare come anche se effettivamente sono presenti sul mercato vari tassi di cambio, le differenze tra i tassi negoziati su piazze diverse sono molto

limitate, se non inesistenti, a causa dell'arbitraggio. Come già accennato, è un mercato caratterizzato da operatività continua durante tutta la settimana ad esclusione del sabato e della domenica (anche se un'importante eccezione in questo senso è data dai mercati di Tel Aviv e Abu Dhabi, che restano aperti anche la domenica). Ogni giorno, una volta terminata la sessione asiatica inizia quella europea, seguita da quella nord americana, per tornare poi ad una nuova sessione asiatica il giorno seguente. Le valute vengono sempre scambiate in coppia, ed ogni coppia rappresenta quindi un singolo prodotto di investimento. La notazione che viene utilizzata per indicare le coppie di valute scambiate nel mercato del forex è univoca e standardizzata a livello internazionale: esse vengono indicate con la notazione XXX/YYY, dove XXX e YYY sono le tre lettere che identificano le varie valute a livello internazionale con il codice "ISO 4217 international three letter code". La prima valuta è chiamata base currency mentre la seconda è chiamata counter currency o quote currency. Per esempio, la quotazione EUR/USD 1.1250 esprime il prezzo di un Euro in termini di Dollari americani, e significa che per acquistare 1 Euro sono necessari 1.1250 Dollari americani. Analizzando poi alcuni dati quantitativi, sempre secondo la Bank of International Settlement, nel 2016 la valuta più coinvolta negli scambi è stata il Dollaro americano (coinvolta nell'87.6% delle transazioni), seguita dall'Euro (31.3%), dallo Yen (21.6%) e dalla Sterlina inglese (12.8%).

Currency distribution of OTC foreign exchange turnover

Net-net basis,¹ percentage shares of average daily turnover in April²

Table 2

Currency	2001		2004		2007		2010		2013		2016	
	Share	Rank	Share	Rank	Share	Rank	Share	Rank	Share	Rank	Share	Rank
USD	89.9	1	88.0	1	85.6	1	84.9	1	87.0	1	87.6	1
EUR	37.9	2	37.4	2	37.0	2	39.0	2	33.4	2	31.4	2
JPY	23.5	3	20.8	3	17.2	3	19.0	3	23.0	3	21.6	3
GBP	13.0	4	16.5	4	14.9	4	12.9	4	11.8	4	12.8	4

Se si guardano poi le coppie più scambiate notiamo come le prime cinque coppie siano EUR/USD (che rappresenta il 23.1% degli scambi), USD/JPY (17.8%), GBP/USD (9.3%), AUD/USD (5.2%) e USD/CAD (4.3%).

OTC foreign exchange turnover by currency pair

Net-net basis,¹ daily averages in April, in billions of US dollars and percentages

Table 3

Currency pair	2001		2004		2007		2010		2013		2016	
	Amount	%	Amount	%	Amount	%	Amount	%	Amount	%	Amount	%
USD / EUR	372	30.0	541	28.0	892	26.8	1,099	27.7	1,292	24.1	1,172	23.1
USD / JPY	250	20.2	328	17.0	438	13.2	567	14.3	980	18.3	901	17.8
USD / GBP	129	10.4	259	13.4	384	11.6	360	9.1	473	8.8	470	9.3
USD / AUD	51	4.1	107	5.5	185	5.6	248	6.3	364	6.8	262	5.2
USD / CAD	54	4.3	77	4.0	126	3.8	182	4.6	200	3.7	218	4.3
USD / CNY	31	0.8	113	2.1	192	3.8
USD / CHF	59	4.8	83	4.3	151	4.5	166	4.2	184	3.4	180	3.6
USD / MXN	128	2.4	90	1.8
USD / SGD	65	1.2	81	1.6
USD / KRW	58	1.5	60	1.1	78.0	1.5
USD / NZD	82	1.5	77.6	1.5
USD / HKD	85	2.1	69	1.3	77	1.5
USD / SEK	57	1.7	45	1.1	55	1.0	66	1.3
USD / TRY	63	1.2	64	1.3
USD / INR	36	0.9	50	0.9	56	1.1
USD / RUB	79	1.5	53	1.1
USD / NOK	49	0.9	48	0.9
USD / BRL	25	0.6	48	0.9	45	0.9
USD / ZAR	24	0.6	51	1.0	40	0.8
USD / TWD	22	0.4	31	0.6
USD / PLN	22	0.4	19	0.4
USD / OTH	199	16.0	307	15.9	612	18.4	446	11.2	214	4.0	215	4.2
EUR / GBP	27	2.1	47	2.4	69	2.1	109	2.7	102	1.9	100	2.0
EUR / JPY	36	2.9	61	3.2	86	2.6	111	2.8	148	2.8	79	1.6
EUR / CHF	13	1.1	30	1.6	62	1.9	71	1.8	71	1.3	44	0.9
EUR / SEK	24	0.7	35	0.9	28	0.5	36	0.7
EUR / NOK	20	0.4	28	0.6
EUR / AUD	1	0.1	4	0.2	9	0.3	12	0.3	21	0.4	16	0.3
EUR / CAD	1	0.1	2	0.1	7	0.2	14	0.3	15	0.3	14	0.3
EUR / PLN	14	0.3	13	0.3
EUR / DKK	13	0.2	13	0.2
EUR / HUF	10	0.2	5	0.1
EUR / TRY	6	0.1	4	0.1
EUR / CNY	1	0.0	2	0.0
EUR / OTH	20	1.6	38	1.9	83	2.5	102	2.6	51	0.9	65	1.3
JPY / AUD	24	0.6	46	0.9	31	0.6
JPY / CAD	6	0.1	7	0.1
JPY / NZD	4	0.1	5	0.1	5	0.1
JPY / TRY	1	0.0	3	0.1
JPY / ZAR	4	0.1	3	0.1
JPY / BRL	3	0.1	1	0.0
JPY / OTH	15	1.2	28	1.4	66	2.0	50	1.3	88	1.7	45	0.9
Other currency pairs	13	1.1	22	1.1	74	2.2	71	1.8	44	0.8	116	2.3
All currency pairs	1,239	100.0	1,934	100.0	3,324	100.0	3,973	100.0	5,357	100.0	5,067	100.0

¹ Adjusted for local and cross-border inter-dealer double-counting (ie "net-net" basis).

Dall'analisi di questi dati emerge come la maggior parte delle transazioni nel mercato del forex ruoti attorno a sei valute: Dollaro americano, Euro, Yen, Sterlina inglese, Dollaro australiano e Dollaro canadese. Esse infatti costituiscono circa il 60% degli scambi che avvengono nel forex a livello globale.

1.6 Determinanti del tasso di cambio

In questo paragrafo verranno trattate brevemente le teorie e i fattori che cercano di spiegare l'andamento dei tassi di cambio. Per quanto riguarda le prime, possiamo senza dubbio riferirci da una parte alla teoria della parità del potere d'acquisto, a quella della parità dei tassi di interesse e all'effetto Fisher e dall'altra a quella che guarda ai tassi di cambio con un approccio di portafoglio. Per quanto riguarda i secondi invece, essi possono essere suddivisi in fattori economici, condizioni politiche e psicologia del mercato.

La teoria della parità del potere d'acquisto (purchasing power parity-PPP) afferma che il tasso di cambio tra le valute di due paesi è uguale al rapporto tra i livelli dei prezzi nei due paesi, misurati prendendo in considerazione un opportuno paniere di beni come riferimento. Espressa in termini formali, essa afferma che:

$$E = \frac{P_2}{P_1}$$

dove E è il tasso di cambio tra la valuta del paese 2 e la valuta del paese 1; P_1 è il livello generale dei prezzi nel paese 1 e P_2 è il livello generale dei prezzi nel paese 2. Questa teoria è una teoria della determinazione del tasso di cambio di lungo periodo e permette di capire se una determinata valuta è sopravvalutata o sottovalutata. Se per esempio dal confronto dei prezzi in Europa e negli Stati Uniti risulta un tasso di cambio EUR/USD di 1.20 mentre quello effettivo sul mercato è 1.30, ciò significa che l'Euro è sopravvalutato rispetto al dollaro. Questa appena esposta è la teoria della parità del potere d'acquisto in termini assoluti. Tale teoria infatti può essere formulata anche in termini relativi, e in tal caso essa afferma che, su qualsiasi intervallo di tempo, le variazioni percentuali del tasso di cambio delle valute dei due paesi sono uguali alle variazioni percentuali dei livelli dei prezzi nazionali. In termini formali risulta:

$$\Delta E = \pi_2 - \pi_1$$

dove π_2 e π_1 sono rispettivamente il tasso di inflazione nei paesi 2 e 1. Da questa equazione è possibile notare come se il tasso di inflazione del paese 2 aumenta (diminuisce) o il tasso di inflazione del paese 1 diminuisce (aumenta), il tasso di cambio E aumenta (diminuisce). La parità del potere d'acquisto tuttavia non è spesso in grado di spiegare in modo corretto e preciso il formarsi dei tassi di cambio sul mercato, e le motivazioni sono da ricercare nelle ipotesi, fin troppo forti e poco realistiche, che stanno alla base di questa teoria, da individuare nell'uguaglianza del paniere di beni preso come riferimento per il calcolo dell'inflazione dai vari paesi; nell'assenza di costi di trasporto; nell'assenza di barriere allo scambio, come i dazi doganali ecc.

La teoria della parità dei tassi di interesse (interest parity condition-parità scoperta) è una condizione fondamentale di equilibrio che deve valere sia nel breve che nel lungo periodo e che esprime che per ogni coppia di valute i rendimenti attesi sui depositi, se misurati in termini della stessa valuta, devono risultare uguali. Essa è una condizione di non arbitraggio e per capire come funziona è sufficiente fare riferimento al seguente esempio: supponiamo che il tasso di interesse sulle attività denominate in Dollari sia del 10% e quello sulle attività denominate in Euro sia del 5%; supponiamo inoltre che vi sia un deprezzamento atteso del Dollaro nei confronti dell'Euro pari al 9%; in questo caso, il tasso di rendimento atteso sulle attività denominate in Euro supera quello delle attività denominate in Dollari di quattro punti percentuali. Per questo motivo nessuno sarà disposto a detenere depositi in Dollari, e anche coloro che li detengono vorranno scambiarli con i depositi in Euro. Ciò porterà ad un eccesso di offerta sui depositi in Dollari e un eccesso di domanda sui depositi in euro e la condizione di parità verrà ristabilita solo grazie all'aggiustamento del tasso di cambio: quando i depositi in Dollari offrono un rendimento più alto dei depositi in Euro il Dollaro si apprezzerà nei confronti dell'Euro; viceversa, quando i depositi in Euro offrono un rendimento più elevato dei depositi in Dollari, sarà l'Euro ad apprezzarsi nei confronti del Dollaro. Chiedendosi poi come questa condizione possa combinarsi con la parità del potere d'acquisto (in termini relativi) è possibile notare come dato che gli

agenti conoscono la relazione della PPP secondo la quale la variazione percentuale del tasso di cambio è data dalla differenza dei tassi di inflazione dei due paesi, essi si aspetteranno che la differenza tra i tassi di interesse sui depositi denominati nelle due valute sia pari alla differenza tra i tassi attesi di inflazione. Questa relazione di lungo periodo tra tassi di interesse e tassi di inflazione è nota come Effetto Fisher e stabilisce che a parità di condizioni, un aumento del tasso atteso di inflazione di un paese causa un aumento dello stesso ammontare del tasso di interesse sui depositi denominati nella valuta di quel paese, e viceversa.

La teoria che cerca di spiegare il formarsi dei tassi di cambio sul mercato con un approccio di portafoglio parte dal presupposto che le valute siano un'importante asset class che gli investitori utilizzano per costruire i loro portafogli di investimento. Secondo questa teoria il tasso di cambio tra due valute è semplicemente il risultato dell'incontro tra domanda e offerta di assets denominati in quelle valute. Per esempio, un aumento della domanda di assets denominati in Dollari da parte degli operatori per costruire i loro portafogli di investimento provocherà un aumento della domanda di Dollari, con conseguente apprezzamento del Dollaro nei confronti delle altre valute.

In ogni caso, sebbene tutte queste teorie vengano spesso utilizzate per spiegare come si formano i tassi di cambio, esse non sono quasi mai state in grado di raggiungere il loro scopo, per vari motivi. Il mercato valutario infatti è un mercato molto complesso, influenzabile da svariati fattori, da variabili macroeconomiche a variabili microeconomiche, dalla psicologia degli operatori alle aspettative sui futuri tassi di cambio, e per questo, soprattutto nel lungo periodo, una sola teoria, per quanto accurata possa essere, difficilmente potrà riuscire nell'intento di spiegare come viene determinato un tasso di cambio in un preciso istante di tempo.

Passando poi ai fattori che cercano di spiegare l'andamento dei tassi di cambio, come già ricordato essi si suddividono in fattori economici, condizioni politiche e psicologia del mercato. I fattori economici comprendono le politiche fiscali e le politiche monetarie dello Stato; i surplus o i deficit che i governi realizzano, dato che generalmente ad un ampliamento del deficit i mercati reagiscono

negativamente e viceversa; la bilancia dei pagamenti, poiché una bilancia positiva, dove le esportazioni sono maggiori delle importazioni, genera una maggiore domanda di valuta del paese; il livello di inflazione, perché generalmente alti livelli di inflazione sono tali da far percepire al mercato che la valuta del paese possa perdere gradualmente potere d'acquisto, e ciò non fa altro che diminuire la domanda di valuta del paese; il grado di salute dell'economia, dato che report positivi per quanto riguarda il Pil, il livello di occupazione, le vendite al dettaglio ecc. sono sinonimi di un'economia in salute, e non fanno altro che aumentare la domanda di valuta del paese.

Le condizioni politiche invece comprendono tutti gli avvenimenti politici di un paese: l'instabilità politica per esempio influenza negativamente l'andamento della valuta di un paese (un esempio lampante è rappresentato proprio in questi giorni dalle elezioni in Gran Bretagna dove, a seguito di un risultato in cui la premier May non ha raggiunto il numero di seggi necessari per avere la maggioranza assoluta, la Sterlina è arrivata a perdere quasi il 2% nei confronti del Dollaro in apertura di contrattazioni).

La psicologia del mercato infine comprende tutti quei comportamenti più o meno inconsci che portano i trader a comportarsi in un determinato modo, andando ad influenzare l'andamento dei tassi di cambio. Un esempio classico può essere quello in cui in un momento di incertezza a livello globale, gli investitori vanno ad aumentare sempre di più la domanda dei beni considerati "rifugio" (Dollaro americano, Franco svizzero, oro ecc.)

1.7 Strumenti finanziari

I principali strumenti finanziari negoziati nel foreign exchange market sono i foreign exchange spot, i forward contract, i foreign exchange swap, i futures e le foreign exchange option.

Il foreign exchange spot è una transazione a pronti, un accordo tra due parti per comprare un certo ammontare di una valuta e vendere lo stesso ammontare di un'altra a un prezzo stabilito; il tasso di cambio al quale viene effettuata la transazione viene definito tasso di cambio spot o spot exchange rate e la

liquidazione avviene solitamente entro due giorni lavorativi (tranne per alcune coppie di valute per le quali la liquidazione avviene entro un solo giorno lavorativo).

I forward contract sono transazioni in cui la liquidazione avviene ad una data futura stabilita dalle parti e in cui quindi il denaro non passa di mano fino al sopraggiungere della data futura specificata nel contratto; sono accordi sottoscritti oggi che obbligano le due controparti ad acquistare o vendere un certo ammontare di valuta estera in un momento futuro specificato; il tasso di cambio al quale viene effettuata la transazione viene definito tasso di cambio a termine o forward exchange rate e la data alla quale le valute vengono effettivamente scambiate è detta giorno di valuta o value date.

I foreign exchange swap sono la combinazione di una vendita e un acquisto dello stesso ammontare di una valuta per un'altra in giorni di valuta diversi (generalmente spot e forward); questo tipo di strumento viene utilizzato da molti operatori in quanto il costo di transazione di uno swap è sensibilmente più contenuto rispetto a quello derivante dall'effettuare due transazioni separate di vendita di una valuta a pronti e acquisto di una valuta a termine. Un esempio di utilizzo di uno swap può essere dato dalla seguente situazione: un'azienda riceve un incasso di un certo ammontare di Dollari in seguito alla vendita di alcuni prodotti ma sa che tra tre mesi dovrà pagare lo stesso ammontare in Dollari a causa dell'acquisto di alcune forniture; nel frattempo però vuole investire i Dollari incassati in obbligazioni denominate in Euro; anziché effettuare due transazioni separate –vendita di Dollari spot e acquisto di Dollari forward a tre mesi- può negoziare uno swap a tre mesi da Dollari a Euro limitando i costi di transazione.

I futures sono contratti forward standardizzati che specificano l'ammontare, il prezzo e la data futura alla quale la valuta vengono effettivamente scambiate; proprio come avviene con i forward contract, quando si acquista un contratto future si acquista la promessa che un certo ammontare di valuta estera verrà consegnato ad una data futura, ma mentre con il contratto a termine non è possibile non completare la transazione a scadenza, con il futures è possibile rivendere il contratto sul mercato organizzato dei futures, realizzando subito un guadagno o

una perdita. Tra i futures e i contratti forward esistono tuttavia sostanziali differenze: in primo luogo, mentre nei contratti forward gli scambi sono decentrati (vengono cioè scambiati sul mercato interbancario, che è disperso geograficamente e aperto 24 ore al giorno), nei futures gli scambi sono centralizzati (sono scambiati su apposite piattaforme elettroniche come EUREX o GLOBEX); in secondo luogo, mentre i contratti forward sono transazioni “customized”, cioè i termini del contratto sono stabiliti di volta in volta dalle parti, i contratti futures sono altamente standardizzati, con date di scadenza, condizioni di consegna, quantità dell’attività trattata determinate a priori e uguali per tutti i contratti; in terzo luogo, mentre nel caso di contratti forward il rischio di controparte è altamente variabile e derivante dalla controparte stessa, nel caso dei futures è presente la clearinghouse che, assumendo legalmente responsabilità per le obbligazioni derivanti dal contratto, rende il rischio di controparte fisso; infine nei futures, al contrario dei contratti forward, è presente come garanzia un meccanismo di marginazione, ovvero si stabilisce un margin requirement che varia continuamente secondo l’andamento del sottostante e che funge da vera e propria garanzia contro il rischio di default della controparte.

Le foreign exchange option infine sono degli strumenti che danno il diritto (ma non comportano l’obbligo) al proprietario di acquistare (call option) o vendere (put option) un determinato ammontare di valuta estera a un certo prezzo (definito strike price) ad una data futura prefissata o entro una data futura prefissata (nel primo caso si parla di european option mentre nel secondo caso di american option); la controparte, cioè il venditore dell’opzione, è invece obbligato a vendere o acquistare la valuta a discrezione del possessore dell’opzione.

1.8 Efficienza del foreign exchange market

Lo studio dell’efficienza dei mercati finanziari in genere è stato da sempre uno dei temi più analizzati dagli economisti. Sebbene vi siano varie definizioni di efficienza (tra cui allocativa, operativa, di portafoglio), quella sulla quale si sono concentrati il maggior numero di studi è senza dubbio l’efficienza informativa. Un tipo di efficienza informativa molto studiata negli ultimi anni l’Efficient Market Hypothesis elaborata da Eugene Fama, secondo la quale un mercato si dice

efficiente (dal punto di vista informativo) se riflette interamente e correttamente tutta l'informazione rilevante. Quest'ipotesi può essere spiegata prendendo come riferimento le informazioni "superate", cioè le informazioni passate su un determinato titolo o strumento finanziario, che, essendo già note a tutti gli investitori, non sono in grado di influenzare il prezzo corrente perché già riflesse in esso. Se per avere questo tipo di informazioni nessuno sarebbe disposto a pagare, gli investitori sarebbero invece ben disposti a pagare per avere informazioni nuove, che possano influenzare i prezzi futuri. Un mercato è quindi tanto più efficiente quanto più le informazioni divengono "superate" velocemente, quanto più cioè vengono incorporate velocemente nel prezzo. Fama parlò di tre tipi di efficienza: l'efficienza in forma debole, l'efficienza in forma semi forte e l'efficienza in forma forte. Un mercato si dice efficiente in forma debole se l'insieme delle informazioni riflesse nei prezzi è formato dai prezzi correnti e dai prezzi passati di un determinato titolo; si dice invece efficiente in forma semi forte se l'insieme delle informazioni è costituito da tutta l'informazione disponibile pubblicamente; si dice infine efficiente in forma forte se l'insieme dell'informazione è costituito anche dall'inside information, ovvero dall'informazione privata. L'Efficient Market Hypothesis ha alla base le seguenti ipotesi: i mercati sono competitivi e tutti gli investitori sono price takers (ovvero nessuno è in grado di influenzare il prezzo di un determinato titolo), gli scambi avvengono solo a prezzo di equilibrio, i prezzi si aggiustano istantaneamente in modo da eguagliare domanda e offerta, tutti gli investitori sono razionali, l'informazione è uguale per tutti e non è costosa, il modello teorico usato per tradurre l'insieme informativo in previsioni sui prezzi è uguale per tutti gli investitori. Se valgono queste ipotesi, ovvero se l'Efficient Market Hypothesis è verificata, i prezzi delle attività di qualunque mercato si muovono come un random walk e quindi, non essendo per definizione prevedibili, il comportamento ottimale di qualsiasi investitore dovrebbe essere quello di adottare una politica di gestione passiva, acquistando il portafoglio di mercato, perché nessuna politica di gestione attiva dovrebbe essere in grado di ottenere extra rendimenti costanti nel tempo. Tuttavia, alcune delle ipotesi alla base dell'Efficient Market Hypothesis sono

molto forti e scarsamente verificate nella realtà, in particolare il fatto che gli individui sono razionali, che l'informazione è uguale per tutti e che tutti gli investitori sono price takers. Proprio per questo, nel momento in cui questa teoria è stata sottoposta a validazione empirica, i risultati sono stati contraddittori: sono venute alla luce evidenti anomalie che hanno mostrato come molto spesso i prezzi non si comportino come un random walk. Queste anomalie sono state studiate fortemente dalla finanza comportamentale, che, soprattutto negli ultimi anni, sta cercando di dimostrare come la presenza di noise traders (soggetti che operano in modo non razionale, che prendono le loro decisioni di investimento attraverso regole del pollice e che sono influenzati fortemente dal modo in cui un problema decisionale viene loro posto) sul mercato causi un costante mispricing dei titoli e degli strumenti finanziari. Sebbene anche la finanza tradizionale contemplasse la presenza di noise traders sul mercato, essa riteneva che gli scostamenti dai fondamentali da essi provocati fossero qualche volta scostamenti in positivo e qualche volta in negativo, e che si annullassero in media (i mercati erano comunque efficienti). La finanza comportamentale invece ritiene che questi scostamenti possano persistere nel tempo tanto da far sì che i mercati finanziari in genere non siano efficienti.

Il punto di partenza di tutta la trattazione è proprio questo: dato che nel tempo si sono riscontrate moltissime anomalie che hanno portato sempre più ad un rigetto dell'ipotesi di efficienza dei mercati, si cercherà, nell'ultimo capitolo, di utilizzare delle metodologie di previsione dei tassi di cambio per ottenere dei rendimenti positivi, dato che se i mercati non sono efficienti e se i prezzi non si muovono secondo un random walk ciò dovrebbe essere, almeno in teoria, possibile.

Capitolo 2: Reti neurali artificiali

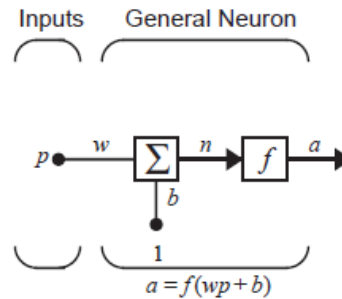
2.1 Introduzione

Come appena ricordato, nell'ultimo capitolo della trattazione si cercherà di creare un metodo di previsione dei tassi di cambio capace di ottenere rendimenti positivi sul mercato. In particolare si cercherà di raggiungere questo obiettivo attraverso una nuova classe di modelli non lineari la cui architettura cerca di riprodurre il funzionamento del cervello umano: le reti neurali artificiali (o reti neuronali artificiali). Queste reti nascono dalla volontà di ricreare artificialmente il complesso funzionamento delle reti neurali biologiche, e ultimamente vengono utilizzate nei campi più disparati: si va dall'applicazione a livello militare (riconoscimento facciale, nuovi sensori e radar, tracciamento dell'obiettivo) a quella a livello di intrattenimento (effetti speciali, animazioni), da quella a livello medico (analisi di EEG e ECG, disegno di protesi) a quella a livello economico e finanziario (previsione dei tassi di cambio, creazione di portafogli di investimento). La caratteristica fondamentale di questi modelli, sulla quale ci si soffermerà più nel dettaglio in seguito, è quella di essere capaci di apprendere dall'esperienza, a differenza di altri modelli statistici o econometrici. La rete neurale artificiale (di seguito semplicemente rete neurale) infatti, dati una serie di input e target, cerca di stabilire quei legami e quelle relazioni che sussistono tra i primi e che generano i secondi. Una volta determinati questi legami la rete impara dai propri errori e cerca di cambiare e adattare la propria struttura e la propria organizzazione tramite regole di apprendimento, fino ad arrivare ad un certo grado di ottimizzazione predefinito.

2.2 Modello del neurone e architettura della rete

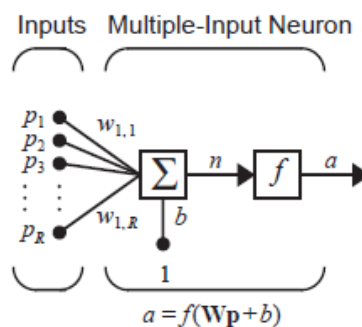
All'inizio di questo paragrafo verranno trattati il modello del neurone ad un solo input e il modello del neurone a input multipli, che sono le forme elementari che costituiscono l'architettura della rete, spiegata nella parte finale del paragrafo.

Il modello del neurone ad un solo input, mostrato in figura², ha il seguente funzionamento: l'input p (scalare) viene moltiplicato per il peso w (anch'esso scalare) e il prodotto wp è inviato all'operatore sommatoria. Un altro input, 1, è moltiplicato per il *bias* b (molto simile al peso w , tranne per il fatto che ha sempre come input la costante 1), e anche in questo caso il prodotto viene inviato



all'operatore sommatoria. In seguito il risultato della somma n (detto input netto) viene passato da una funzione di trasferimento f (detta anche funzione di attivazione), che produce l'output a (scalare). I parametri fondamentali sono il peso w e il bias b : essi infatti sono aggiustabili e generalmente, una volta scelta la funzione di trasferimento, vengono adattati tramite specifici processi di apprendimento (learning rules) cosicché la relazione tra input e target raggiunga un determinato obiettivo.

Il modello del neurone a input multipli si differenzia dal precedente semplicemente per il fatto che invece che avere un solo input p , questo neurone ha R input (p_1, p_2, \dots, p_R), ognuno ponderato per il corrispondente elemento della matrice dei pesi W ($w_{1,1}, w_{1,2}, \dots, w_{1,R}$), che in questo caso ha una sola riga³.












² Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 2, pp. 3

³ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 2, pp. 7

In questo caso inoltre l'input netto n è il risultato del prodotto tra il vettore degli input p e la matrice dei pesi W . Per quanto riguarda la notazione, per esempio con l'elemento $w_{1,2}$ della matrice dei pesi W si indica che quel peso è relativo alla connessione al primo neurone (che in questo caso è anche l'unico) dalla seconda fonte.

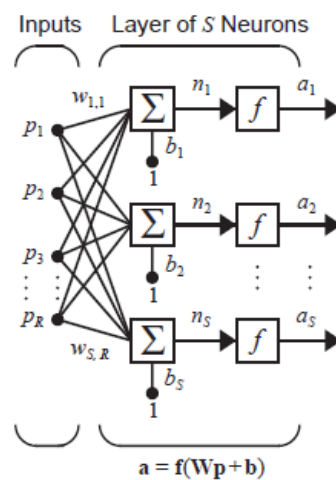
Come affermato in precedenza, affinché la rete funzioni correttamente, è necessario specificare una funzione di trasferimento (transfer function), che deve essere cambiata di volta in volta secondo lo specifico problema che la rete neurale tenta di risolvere. In letteratura e nei vari test empirici sulle reti neurali sono state utilizzate varie funzioni di trasferimento, e le principali sono riassunte nella seguente tabella⁴:

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1 \quad \text{neuron with max } n$ $a = 0 \quad \text{all other neurons}$		compet

⁴ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 2, pp. 6

La funzione Hard Limit per esempio pone l'output a del neurone a 0 o a 1 rispettivamente nel caso in cui l'argomento della funzione sia minore di 0 o maggiore o uguale a 0. Questa funzione viene spesso utilizzata nei casi in cui è necessario classificare elementi in due diverse categorie.

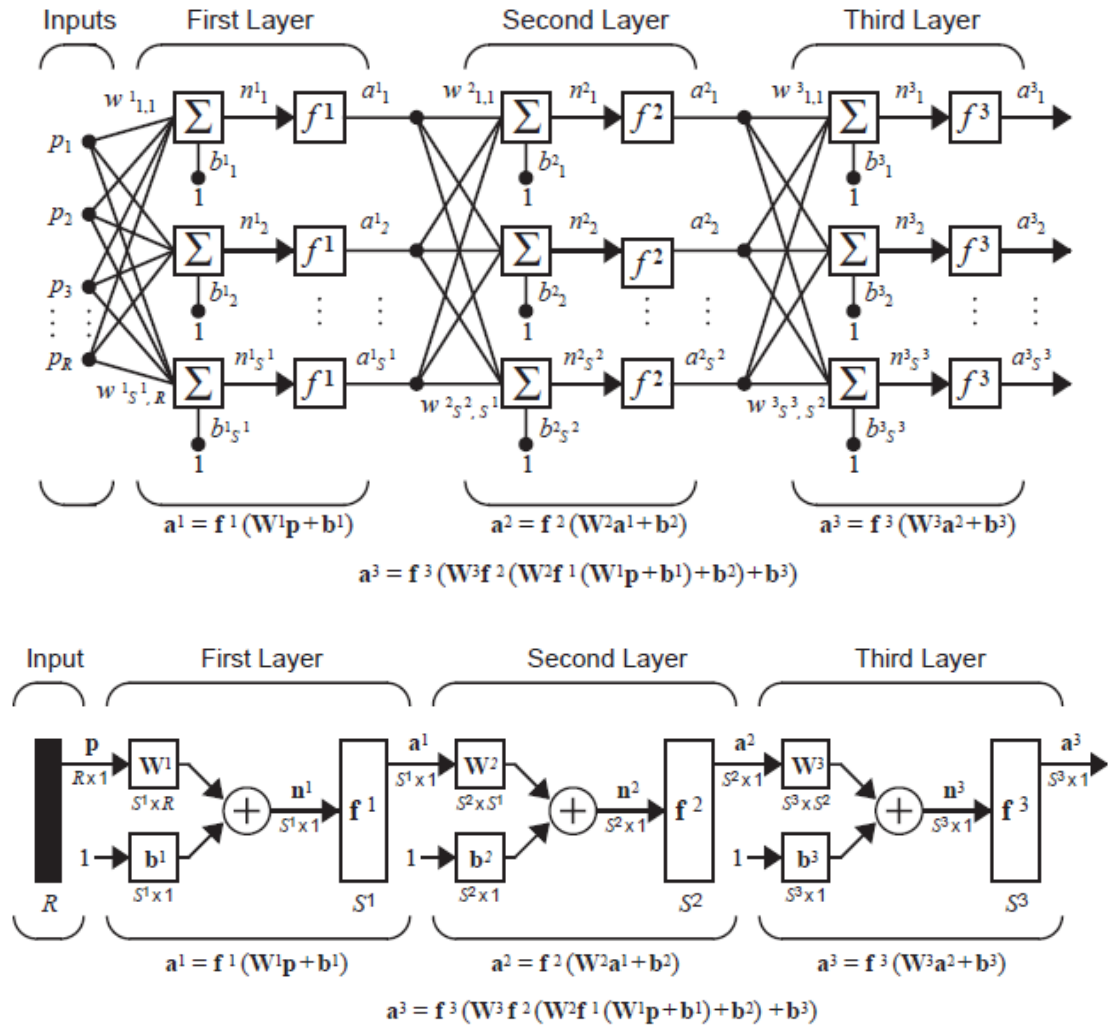
Passando ora all'architettura della rete, possiamo senza dubbio affermare che solitamente un singolo neurone, per quanti input possa avere, non è mai sufficiente per risolvere problemi complessi. Per questo motivo vengono utilizzati più neuroni, che lavorando tra loro in parallelo vanno a formare uno strato della rete neurale (neural network layer), composto in questo caso da S neuroni.



Come si può notare dalla figura⁵ ogni strato della rete, proprio come avveniva in precedenza nei modelli con un singolo neurone, contiene la matrice dei pesi, gli operatori somma, il vettore dei bias, le funzioni di trasferimento e il vettore degli output. In questo caso però ogni elemento del vettore degli input è connesso a ogni neurone dello strato attraverso la matrice dei pesi W . Inoltre nella maggior parte delle reti neurali si utilizzano più strati, ognuno con la propria matrice dei pesi, con il proprio vettore di bias, con il proprio vettore degli input netti e con il proprio vettore degli output. Prima di rappresentare la rete nella sua totalità è tuttavia necessario introdurre una notazione aggiuntiva per distinguere tra le varie matrici e i vari vettori degli strati. In particolare verranno utilizzati gli apici per riferirsi ai vari strati della rete: per esempio W^1 sarà la matrice dei pesi del primo strato, S^2

⁵ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 2, pp. 9

sarà il numero di neuroni del secondo strato, e così via. Di seguito viene rappresentata la rete in forma estesa, con notazione completa e abbreviata⁶.



Lo strato i cui output sono gli output di tutta la rete viene chiamato strato di output (output layer), mentre gli altri vengono chiamati strati nascosti (hidden layers). Inoltre è necessario specificare che nelle reti multistrato le funzioni di trasferimento non devono necessariamente essere uguali per tutti gli strati.

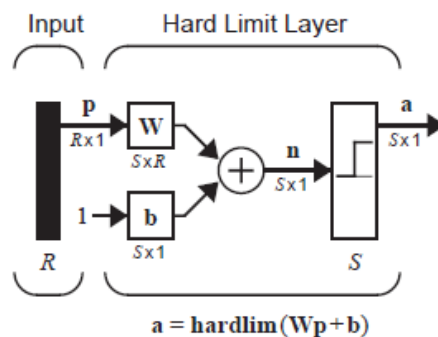
2.3 Perceptron learning rule

Prima di mostrare la regola di apprendimento del perceptrone è opportuno specificare che cosa si intende per regole di apprendimento in generale. Una regola

⁶ Hagan M. T., Debut H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 2, pp. 11-12

di apprendimento è una procedura che va a modificare i pesi e i bias di una rete neurale. Nonostante vi siano molte regole di apprendimento per le reti neurali, esse possono essere raggruppate in tre categorie principali: regole di apprendimento non supervisionato, supervisionato e per rinforzo. Nell'apprendimento non supervisionato i pesi e i bias sono modificati soltanto in risposta agli input che vengono presentati alla rete; nell'apprendimento supervisionato invece, alla rete vengono presentati gli input e i target corrispondenti, cosicché possa compararli e aggiustare i pesi e i bias in maniera opportuna (la perceptron learning rule appartiene a questa categoria); nell'apprendimento per rinforzo infine viene semplicemente fornito un rinforzo al sistema, che lo utilizza come un segnale positivo o negativo riguardo al comportamento della rete e aggiusta i parametri di conseguenza (questo tipo di apprendimento si utilizza quando non è possibile specificare coppie input-target come avviene nell'apprendimento supervisionato). Nel resto della trattazione verranno utilizzate solamente regole di apprendimento supervisionato.

L'architettura del percettrone è la seguente⁷:



Gli output della rete sono raggruppati in un vettore, ma talvolta risulta conveniente riferirsi ad ogni singolo output. Per questo motivo possiamo scrivere l'i-esimo output come

$$a_i = \text{hardlim}(\sum_j w_{ij} p_j + b_i)$$

Dove $\sum_j w_{ij} p_j$ non è altro che la i-esima riga della matrice dei pesi W. Per esempio, in un percettrone con un singolo neurone e due input l'output è dato da:

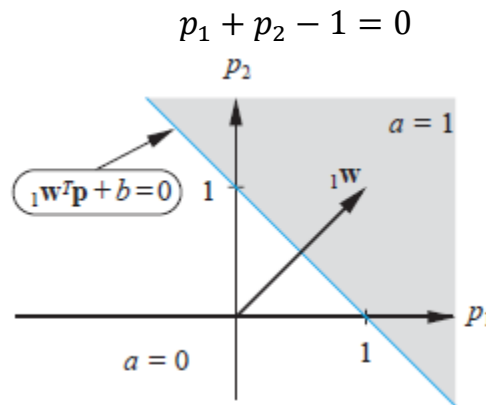
⁷ Hagan M. T., Debut H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 4, pp. 4

$$a = \text{hardlim}({}_1w^T p + b) = \text{hardlim}(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

Importante è definire inoltre il cosiddetto confine di decisione, che non è altro che il confine oltre il quale l'output assume valore 0 o 1. In questo caso (perceptrone con un singolo neurone), tale confine è determinato dal vettore di input per il quale l'input netto n è uguale a zero:

$$n = (w_{1,1}p_1 + w_{1,2}p_2 + b) = 0$$

Ponendo per esempio i due pesi e il bias rispettivamente uguali a 1,1,-1, il confine di decisione risulterebbe⁸:



Nel perceptrone con più neuroni invece vi sarà un confine di decisione per ognuno dei neuroni, e quindi questo tipo di rete sarà in grado di classificare gli input in più categorie, al contrario del perceptrone ad un solo neurone, che li classificherà soltanto in due categorie. Come mostrato in precedenza il confine di decisione è determinato dal valore dei pesi e dei bias, e dato che quest'ultimi vengono inizializzati casualmente, non sarà mai in grado di classificare correttamente gli input, almeno inizialmente. Per far sì che la rete classifichi correttamente gli input è quindi necessario utilizzare la perceptron learning rule, aggiustando i parametri secondo la seguente equazione:

$${}_1w^{nex} = {}_1w^{old} + ep = {}_1w^{old} + (t - a)p$$

⁸ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 4, pp. 6

Dove e è l'errore, dato da $(t-a)$, ovvero dalla differenza tra il target e l'output generato dalla rete. Il bias invece viene aggiornato secondo la seguente equazione:

$$b^{new} = b^{old} + e$$

Questa regola, valida per un percettrone con un singolo neurone, può essere generalizzata anche per un percettrone con neuroni multipli. Scritte in forma matriciale, le equazioni per l'aggiornamento dei pesi e dei bias risultano le seguenti:

$$W^{new} = W^{old} + ep^T$$

$$b^{new} = b^{old} + e$$

Nella pratica, questo processo attraverso il quale si vanno ad aggiustare i parametri inizializzati casualmente al fine di classificare ogni vettore di input in modo corretto è un processo iterativo: una volta inizializzati casualmente i parametri, si presenta il primo vettore di input alla rete e si calcola il relativo output; se questo corrisponde al target, il processo continua mostrando alla rete il secondo vettore di input, il terzo ecc.; se questo non corrisponde invece si utilizzano le equazioni precedenti per aggiornare i parametri e si controlla che il secondo vettore di input dia il giusto output; il processo quindi continua fino a che tutti gli input non vengono classificati nel modo corretto. Solo a questo punto infatti la rete può dirsi “allenata”, ed è pronta a classificare altri input mai visti prima.

Sebbene questo algoritmo sia molto semplice, esso è allo stesso tempo molto potente. Si può infatti mostrare come questa regola, assunto che una soluzione esista, sia sempre in grado di convergere a pesi e bias che risolvano il problema di classificazione dato⁹. Nonostante ciò, questo tipo di regola di apprendimento ha una forte limitazione, data dal fatto che può classificare solo input separabili linearmente. Ciò significa che può classificare solo input separabili da un confine di decisione lineare (come la retta nel precedente esempio). Per superare tale limite

⁹ Per la dimostrazione formale si veda Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 4, pp. 15-18

sono necessarie reti con neuroni multipli e percettroni multistrato, che saranno presentati nel seguito della trattazione.

2.4 Apprendimento basato sulla performance

Continuando a parlare delle regole di apprendimento di una rete neurale, alcune tra le tecniche più utilizzate sono certamente quelle che appartengono alla classe performance learning. Utilizzando questo tipo di tecniche i parametri fondamentali della rete (pesi e bias) vengono aggiustati per ottimizzare la performance della rete stessa, data da un opportuno indice di performance (ad esempio l'errore quadratico medio), che avrà un risultato tanto più basso quanto più la rete funziona correttamente. Proprio per questo motivo risulta di fondamentale importanza cercare di risolvere questo problema di ottimizzazione, dato dalla ricerca di pesi e bias che minimizzano l'indice di performance scelto. La risoluzione del problema verrà affrontata in due parti: nella prima parte si indagheranno le condizioni per l'esistenza dei minimi dell'indice di performance, rappresentato da una certa funzione $F(x)$; nella seconda parte, ovvero nel prossimo paragrafo, si mostrerà una tecnica iterativa per calcolare i valori di x che minimizzano la $F(x)$.

Per cominciare è possibile rappresentare la funzione $F(x)$ tramite la sua serie di Taylor nel punto x_0 :

$$F(x) = F(x_0) + \frac{1}{1!}F'(x_0)(x - x_0) + \frac{1}{2!}F''(x_0)(x - x_0)^2 + \dots \\ + \frac{1}{n!}F^n(x_0)(x - x_0)^{n10}$$

Nelle reti neurali però la $F(x)$ non è funzione di uno scalare x , ma di tutti i parametri della rete, e per questo l'indice di performance sarà dato da:

$$F(x) = F(x_1, x_2, \dots, x_n)$$

E la sua serie di Taylor nel punto $x_0=(x_{01}, x_{02}, \dots, x_{0n})$ risulterà:

¹⁰ Con $F'(x_0)$ si intende la derivata prima della F calcolata nel punto x_0

$$\begin{aligned}
F(x) = F(x_0) &+ \frac{1}{1!} \frac{dF}{dx_1}(x_0)(x_1 - x_{01}) + \frac{1}{1!} \frac{dF}{dx_2}(x_0)(x_2 - x_{02}) + \dots \\
&+ \frac{1}{1!} \frac{dF}{dx_n}(x_0)(x_n - x_{0n}) + \frac{1}{2!} \frac{d^2F}{dx_1^2}(x_0)(x_1 - x_{01})^2 \\
&+ \frac{1}{2!} \frac{d^2F}{dx_1 dx_2}(x_0)(x_1 - x_{01})(x_2 - x_{02}) + \dots
\end{aligned}$$

Ovvero, scritta in forma matriciale:

$$F(x) = F(x_0) + \nabla F(x)^T(x_0)(x - x_0) + \frac{1}{2!}(x - x_0)^T \nabla^2 F(x)(x_0)(x - x_0) + \dots$$

Dove $\nabla F(x)$ è il gradiente, definito come:

$$\nabla F(x) = \left[\frac{d}{dx_1} F(x) \frac{d}{dx_2} F(x) \dots \frac{d}{dx_n} F(x) \right]^T$$

E $\nabla^2 F(x)$ è la matrice Hessiana:

$$\nabla^2 F(x) = \begin{bmatrix} \frac{d^2}{dx_1^2} F(x) & \frac{d^2}{dx_1 dx_2} F(x) & \dots & \frac{d^2}{dx_1 dx_n} F(x) \\ \frac{d^2}{dx_2 dx_1} F(x) & \frac{d^2}{dx_2^2} F(x) & \dots & \frac{d^2}{dx_2 dx_n} F(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{d^2}{dx_n dx_1} F(x) & \frac{d^2}{dx_n dx_2} F(x) & \dots & \frac{d^2}{dx_n^2} F(x) \end{bmatrix}$$

Ogni elemento i -esimo del gradiente e della diagonale principale della matrice Hessiana è molto importante in quanto rappresenta rispettivamente la derivata prima e seconda lungo l'asse x_i -esima dell'indice di performance dato dalla $F(x)$. Attraverso il gradiente e la matrice Hessiana è inoltre possibile calcolare la derivata della funzione in una qualsiasi direzione. Sia p è il vettore nella direzione verso la quale vogliamo calcolare la derivata. La derivata direzionale della $F(x)$ lungo p può essere calcolata tramite la formula seguente:

$$\frac{p^T \nabla F(x)}{\|p\|}^{11}$$

E, analogamente, la derivata seconda lungo p può essere calcolata tramite la formula:

$$\frac{p^T \nabla^2 F(x) p}{\|p\|^2}$$

Ricordando che l'obiettivo principale è quello di ricercare i punti di ottimo, ovvero i punti di minimo della funzione che rappresenta l'indice di performance, passiamo ora a definire che cosa si intende per punti di minimo.

Minimo locale

Un punto x_0 è un minimo locale della $F(x)$ se esiste uno scalare $k > 0$ tale che $F(x_0) < F(x_0 + \Delta x)$ per ogni Δx tale che $k > \|\Delta x\| > 0$. Cioè, muovendosi da un minimo locale verso un intorno del minimo stesso la funzione assumerà un valore maggiore.

Minimo globale

Un punto x_0 è un unico minimo globale della $F(x)$ se $F(x_0) < F(x_0 + \Delta x)$ per ogni $\Delta x \neq 0$. In altre parole, se per un minimo locale la funzione può anche assumere valori più piccoli al di fuori dell'intorno di x_0 , ciò non può avvenire nel caso di minimo globale, dove la funzione assumerà valori più elevati in ogni punto dello spazio.

Minimo debole

Un punto x_0 è un minimo debole della $F(x)$ se non è un minimo locale e se esiste uno scalare $k > 0$ tale che $F(x_0) \leq F(x_0 + \Delta x)$ per ogni Δx tale che $k > \|\Delta x\| > 0$.

Una volta definito che cosa si intende per punti di minimo, è possibile identificare alcune condizioni che tali punti devono soddisfare al fine di essere effettivamente

¹¹ Dove $\|p\|$ è la Norma del vettore p , ovvero una funzione che assegna ad ogni vettore di uno spazio vettoriale, escluso lo zero, una lunghezza positiva. È calcolata come radice quadrata della somma delle componenti del vettore al quadrato.

punti di minimo. Queste condizioni sono dette condizioni di primo e secondo ordine, e vengono derivate ancora una volta dalla serie di Taylor della funzione.

$$F(x) = F(x_0 + \Delta x) = F(x_0) + \nabla F(x)^T(x_0)(\Delta x) + \frac{1}{2!}(\Delta x)^T \nabla^2 F(x)(x_0)(\Delta x) + \dots$$

Dove $\Delta x = (x - x_0)$.

Condizioni del primo ordine

Tornando alla serie di Taylor appena vista, se $\|\Delta x\|$ è molto piccolo allora i termini di grado più alto della serie sono trascurabili ed essa può essere approssimata come:

$$F(x_0 + \Delta x) \cong F(x_0) + \nabla F(x)^T(x_0)(\Delta x)$$

Il punto x_0 è un candidato minimo, e questo significa che la funzione dovrà crescere di valore se l'incremento Δx è diverso da 0. Ma affinché questo accada è necessario che il secondo termine del membro di destra dell'equazione precedente sia maggiore o uguale a zero, ovvero:

$$\nabla F(x)^T(x_0)(\Delta x) \geq 0$$

Tuttavia, se questo termine è strettamente positivo risulta:

$$F(x_0 - \Delta x) \cong F(x_0) - \nabla F(x)^T(x_0)(\Delta x) < F(x_0)$$

Ma quest'ultima è una contraddizione, dal momento che il punto x_0 dovrebbe essere un punto di minimo. Per questo motivo l'unica alternativa è che il secondo termine dell'equazione iniziale sia uguale a 0 per ogni Δx :

$$\nabla F(x)(x_0) = 0$$

Questa è quindi la condizione del primo ordine: il gradiente deve essere uguale a zero nel punto di minimo. Essa è una condizione necessaria (e non sufficiente) affinché il punto sia un minimo locale.

Condizioni del secondo ordine

Ponendo di avere in x_0 un punto stazionario (un punto in cui il gradiente è 0), la

serie di Taylor risulterà:

$$F(x_0 + \Delta x) = F(x_0) + \frac{1}{2!}(\Delta x)^T \nabla^2 F(x)(x_0)(\Delta x) + \dots$$

Ripetendo lo stesso procedimento, considerando cioè solo quei punti tali per cui $\|\Delta x\|$ è molto piccolo e per cui la $F(x)$ può essere approssimata dai primi due termini dell'equazione precedente, un minimo locale esisterà in x_0 se:

$$(\Delta x)^T \nabla^2 F(x)(x_0)(\Delta x) > 0$$

Affinché questo sia verificato per ogni Δx diverso da 0 è necessario che la matrice Hessiana sia definita positiva. Questa è una delle condizioni del secondo ordine, sufficiente a far sì che un minimo locale esista. Tuttavia non è una condizione necessaria, perché può comunque esistere un minimo locale anche se il termine di secondo grado della serie di Taylor è 0 ma il termine di terzo grado è positivo. Proprio per questo motivo l'altra condizione del secondo ordine (necessaria) è che la matrice Hessiana sia semi-definita positiva.¹²

Riassumendo, le condizioni necessarie affinché x_0 sia un punto di minimo della $F(x)$ sono che il gradiente sia uguale a 0 e che la matrice Hessiana sia semi-definita positiva; le condizioni sufficienti affinché x_0 sia un punto di minimo locale della $F(x)$ sono invece che il gradiente sia uguale a 0 e che la matrice Hessiana sia definita positiva.

Prima di passare ad affrontare la seconda parte del problema, ovvero cercare una tecnica iterativa per calcolare i valori di x che minimizzano la $F(x)$, in quest'ultima parte del paragrafo verranno analizzate particolari funzioni che possono essere utilizzate come indice di performance, ovvero le funzioni quadratiche, che hanno caratteristiche particolari e che sono le più utilizzate dalle reti neurali. La forma generale di una funzione quadratica è:

$$F(x) = \frac{1}{2}x^T A x + d^T x + c$$

¹² Si ricorda che per definizione una matrice è definita positiva se $z^T A z > 0$ per ogni vettore z diverso da 0 e che è semi-definita positiva se $z^T A z \geq 0$ per ogni vettore z

Dove la matrice A è una matrice simmetrica.

Al fine di calcolare il gradiente di questo tipo di funzioni, è possibile utilizzare due importanti proprietà, date dalle seguenti equazioni:

$$\nabla(h^T x) = \nabla(x^T h) = h$$

$$\nabla x^T Q x = Q x + Q^T x = 2Q x$$

Dove h è un vettore costante e Q è una matrice simmetrica. Nel nostro caso quindi risulterà:

$$\nabla(d^T x) = \nabla(x^T d) = d$$

$$\nabla x^T A x = A x + A^T x = 2A x$$

Ed il gradiente e la matrice Hessiana della $F(x)$ saranno dati da:

$$\nabla F(x) = A x + d$$

$$\nabla^2 F(x) = A$$

Poiché tutte le derivate di grado più elevato della funzione quadratica sono uguali a 0, i primi tre termini della serie di Taylor daranno un'esatta approssimazione della funzione. Cercando poi di capire qual è la forma di una funzione quadratica, è possibile affermare come quest'ultima dipenda essenzialmente dagli autovalori e dagli autovettori¹³ della matrice Hessiana. In particolare:

- Se gli autovalori della matrice Hessiana sono tutti positivi, la funzione avrà un singolo minimo locale;
- Se gli autovalori della matrice Hessiana sono tutti negativi, la funzione avrà un singolo massimo locale;

¹³ Data una trasformazione lineare $A: X \rightarrow X$, quei vettori $z \in X$ (diversi da 0) e quegli scalari λ che soddisfano l'equazione $A(z) = \lambda z$ (che, scritta in forma matriciale, risulta $Az = \lambda z$ o $[A - \lambda I]z = 0$) sono detti rispettivamente autovettori e autovalori della trasformazione lineare. Per calcolare gli autovalori si risolve l'equazione $|[A - \lambda I]| = 0$ rispetto a λ ; per calcolare gli autovettori si risolve invece l'equazione $[A - \lambda I]z = 0$ rispetto a z utilizzando i valori di λ trovati in precedenza. In pratica un autovettore di una trasformazione lineare rappresenta la direzione tale per cui ogni vettore in quella direzione, una volta trasformato, continuerà a puntare in quella direzione, ma scalato dell'autovalore.

- Se alcuni autovalori della matrice Hessiana sono positivi e altri negativi la funzione avrà un unico punto di sella (un punto in cui la matrice hessiana risulta indefinita, cioè non è né semi definita positiva né semi definita negativa);
- Se gli autovalori sono non negativi, ma alcuni risultano uguali a 0, la funzione potrà sia avere un minimo debole sia non avere un punto stazionario;
- Se gli autovalori sono non negativi, ma alcuni risultano uguali a 0, la funzione potrà sia avere un massimo debole sia non avere un punto stazionario.¹⁴

2.5 Steepest Descent

Andando ora ad affrontare la seconda parte del problema è necessario ricercare degli algoritmi che ottimizzano l'indice di performance, ovvero che sono in grado di trovare i valori di x che minimizzano la $F(x)$. Nonostante vi siano vari metodi per risolvere tale problema di ottimizzazione, quello che verrà mostrato in questo paragrafo è forse il più semplice, ma anche uno dei più utilizzati: lo steepest descent. Come tutti questi tipi di algoritmi anche lo steepest descent è un algoritmo iterativo, si basa cioè su una stima iniziale di un valore x_0 che viene aggiornato per step successivi secondo un'equazione nella forma:

$$x_{k+1} = x_k + \alpha_k p_k$$

o

$$\Delta x_k = (x_{k+1} - x_k) = \alpha_k p_k$$

Dove p_k è la direzione nella quale si va a ricercare il valore che minimizza la $F(x)$ e α_k è il tasso di apprendimento, che determina la lunghezza tra uno step e l'altro dell'algoritmo e che viene fissato arbitrariamente. Ovviamente, dal momento che

¹⁴ Per un'esposizione più esaustiva si veda Hagan M. T., DeButh H. B., Beale M.H., De Jous O., 2014. Neural Network Design, 2nd edition, cap. 8, pp. 12-19

l'obiettivo è quello di cercare un punto di minimo, ad ogni iterazione la funzione dovrà assumere un valore più basso rispetto all'iterazione precedente:

$$F(x_{k+1}) < F(x_k)$$

Risulta quindi di fondamentale importanza scegliere una direzione p_k verso la quale, per tassi di apprendimento opportunamente ridotti, la disequazione precedente sia verificata. Considerando ancora una volta la serie di Taylor del primo ordine (nel punto x_k) si ha:

$$F(x_{k+1}) = F(x_k + \Delta x_k) \approx F(x_k) + g_k^T \Delta x_k$$

dove g_k non è altro che il gradiente calcolato nel punto x_k , cioè:

$$g_k \equiv \nabla F(x)(x_k)$$

Affinché la disequazione sia verificata, il secondo termine del lato destro della serie di Taylor deve essere necessariamente negativo:

$$g_k^T \Delta x_k = \alpha_k g_k^T p_k < 0$$

Dato che α_k , per quanto piccolo, è sempre maggiore di 0, risulterà:

$$g_k^T p_k < 0$$

Ogni vettore p_k che soddisfa questa disequazione viene detto descent direction, e ciò significa che procedendo in quella direzione per opportuni tassi di apprendimento la funzione assumerà valori più bassi. Tuttavia, quello che risulta più interessante è cercare di capire qual è la direzione verso la quale la funzione decresce più rapidamente (direction of steepest descent), e questa è la direzione per la quale il prodotto $g_k^T p_k$ assume il massimo valore negativo. Assumendo che p_k non possa variare la propria lunghezza ma soltanto la propria direzione, dalla definizione di derivata direzionale data nel paragrafo precedente si nota come questo prodotto assuma il massimo valore negativo quando il vettore p_k è il negativo del gradiente, ovvero quando:

$$p_k = -g_k^{15}$$

Grazie a questa equazione è possibile derivare l'algoritmo steepest descent per la ricerca dei punti di minimo. Combinando infatti quest'ultima equazione con quella iniziale $x_{k+1} = x_k + \alpha_k p_k$ è possibile determinare l'equazione del metodo che ci permette di aggiornare i valori di x che minimizzano la $F(x)$ come:

$$x_{k+1} = x_k - \alpha_k g_k$$

Riassumendo, per applicare il metodo steepest descent alla funzione $F(x)$ che rappresenta l'indice di performance è necessario innanzitutto partire da una stima del valore x_0 ; successivamente si calcola il gradiente della funzione in quel punto e si stabilisce il tasso di apprendimento; infine, procedendo per iterazioni successive e utilizzando quest'ultima equazione si calcola il punto di minimo dell'indice di performance.

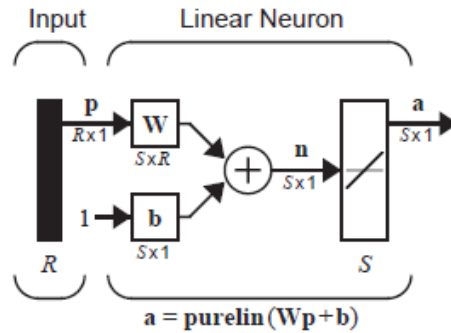
2.6 Apprendimento di Widrow-Hoff

L'apprendimento di Widrow-Hoff è un'approssimazione dell'algoritmo steepest descent nel quale come indice di performance viene utilizzato l'errore quadratico medio. Questo metodo di apprendimento è molto importante in quanto si pone come precursore dell'algoritmo di backpropagation per le reti multistrato che verrà presentato nel prossimo paragrafo. Questo tipo di apprendimento è stato ideato da Bernard Widrow e Marcian Hoff alla metà del secolo scorso: questi studiosi hanno introdotto la cosiddetta rete ADALINE (ADaptive Linear NEuron) e la regola di apprendimento che hanno chiamato algoritmo LMS (Least Mean Square). Questa rete neurale è molto simile al perceptrone visto in precedenza, con l'unica differenza data dal fatto che la funzione di trasferimento è lineare anziché di tipo hard limit. Per questo motivo ha lo stesso, grande, punto di debolezza, ovvero non riesce a risolvere problemi non separabili linearmente. Nonostante ciò l'algoritmo LMS è molto più potente della perceptron learning rule in quanto minimizzando

¹⁵ La derivata direzionale è infatti data da $\frac{p^T \nabla F(x)}{\|p\|}$. Dato che al numeratore si ha un prodotto tra vettori, esso assumerà valore 0 quando il vettore di direzione sarà ortogonale al gradiente e assumerà valore massimo quando il vettore di direzione è lo stesso del gradiente.

l'errore quadratico medio muove il confine decisionale più lontano possibile dai punti con i quali è stata allenata la rete, evitando che la rete risultante sia troppo sensibile al rumore (il perceptron è molto sensibile al rumore, cioè crea problemi quando i pattern con i quali è stata allenata la rete sono molto vicini al confine decisionale).

La rete ADALINE è rappresentata nella figura seguente¹⁶:



Sebbene gli output della rete siano dati dal vettore:

$$a = \text{purelin}(Wp + b) = Wp + b$$

risulta spesso utile, come esposto nel caso del perceptron, andare ad identificare il singolo output, che in questo caso è dato da:

$$a_i = \text{purelin}(n_i) = \text{purelin}(w_i^T p + b_i) = w_i^T p + b_i$$

dove come in precedenza w_i^T non è altro che la i-esima riga della matrice dei pesi W . A questo punto, prima di spiegare nel dettaglio il funzionamento dell'algoritmo di apprendimento LMS (che come la regola di apprendimento del perceptrone è un tipo di apprendimento supervisionato nella quale alla rete vengono forniti gli input e i corrispondenti target da comparare poi con gli output della rete) è necessario soffermarsi su quella che è la particolarità che lo differenzia dalla regola di apprendimento del perceptrone: l'errore quadratico medio. Tramite questo algoritmo infatti i parametri fondamentali della rete ADALINE vengono aggiustati in modo da minimizzare l'errore quadratico medio. Nel caso in cui questa rete abbia un singolo neurone, è possibile, al fine di indagare meglio le caratteristiche

¹⁶ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 4, pp. 5

di questo indice di performance, raggruppare tutti i parametri da aggiustare in un unico vettore x :

$$x = \begin{bmatrix} {}_1w \\ b \end{bmatrix}$$

e includere l'input "1" del bias nel vettore degli input z :

$$z = \begin{bmatrix} p \\ 1 \end{bmatrix}$$

Adesso l'output della rete, che in questo caso sarà uno scalare e che generalmente è dato da:

$$a = {}_1w^T p + b$$

può essere riscritto come:

$$a = x^T z$$

A questo punto è possibile esprimere l'errore quadratico medio della rete ADALINE come:

$$F(x) = E[e^2] = E[(t - a)^2] = E[(t - x^T z)^2]$$

che, scritto in forma estesa risulta:

$$F(x) = E[t^2 - 2tx^T z + x^T z z^T x] = E[t^2] - 2x^T E[tz] + x^T E[zz^T]x$$

Ponendo ora $c = E[t^2]$, $h = E[tz]$ e $R = E[zz^T]$ ¹⁷ è possibile riscrivere l'equazione nel modo seguente:

$$F(x) = c - 2x^T h + x^T R x$$

Si nota facilmente come quest'ultima equazione sia una funzione di tipo quadratico dove $d = -2h$ e $A = 2R$. Questo è un risultato molto importante in quanto come esposto nel paragrafo 2.4 le caratteristiche di una funzione di tipo quadratico dipendono essenzialmente dalla matrice Hessiana A , che in questo caso è data dal doppio della matrice di correlazione R . Essendo possibile dimostrare che tutte le

¹⁷ Il vettore h è il vettore di correlazione incrociata tra il vettore degli input e quello dei target mentre la matrice R è la matrice di correlazione.

matrici di correlazione sono definite positive o semi-definite positive (ovvero non possono mai avere autovalori negativi), rimangono solo due possibilità: se R ha solo autovalori positivi allora la $F(x)$ avrà un unico minimo globale; se invece R ha qualche autovalore uguale a zero la $F(x)$ potrà sia avere un minimo debole, sia non avere un minimo (e ciò dipenderà dal vettore d). A questo punto, volendo calcolare il punto stazionario dell'indice di performance dato dalla funzione quadratica e sapendo dal paragrafo 2.4 che esso si trova nel punto in corrispondenza del quale il gradiente della funzione è uguale a zero, si avrà:

$$\nabla F(x) = d + Ax = -2h + 2Rx$$

$$-2h + 2Rx = 0$$

Perciò se la matrice R è definita positiva avrà un unico punto stazionario, che sarà anche un minimo locale, in corrispondenza del punto:

$$x^* = R^{-1}h$$

Passando ora all'algoritmo LMS, il cui obiettivo è ovviamente quello di localizzare il punto di minimo della funzione $F(x)$, appare superfluo specificare come avendo a disposizione l'inversa di R e h sarebbe possibile calcolare il punto di minimo direttamente dall'equazione precedente. In genere tuttavia non è conveniente calcolare la matrice di correlazione R e il vettore h , e per questo si utilizza un'approssimazione dell'algoritmo steepest descent nel quale il gradiente è sostituito con un gradiente stimato. Il punto di svolta nel lavoro di Widrow e Hoff è proprio questo, ovvero stimare l'errore quadratico medio $F(x)$ come:

$$\hat{F}(x) = (t(k) - a(k))^2 = e^2(k)$$

In questo caso si può notare come l'aspettativa E sull'errore quadratico sia stata sostituita dai due autori con l'errore quadratico all'iterazione k . Ad ogni iterazione quindi il gradiente sarà un gradiente stimato, nella forma:

$$\hat{\nabla}F(x) = \nabla e^2(k)$$

I primi R elementi di questo gradiente saranno le derivate rispetto ai pesi mentre l'elemento R+1 sarà la derivata rispetto al bias, ovvero:

$$[\nabla e^2(k)]_j = \frac{de^2(k)}{dw_{1,j}} = 2e(k) \frac{de(k)}{dw_{1,j}} \quad \text{per } j = 1, \dots, R$$

e

$$[\nabla e^2(k)]_{R+1} = \frac{de^2(k)}{db} = 2e(k) \frac{de(k)}{db} \quad 18$$

Considerando le derivate parziali alla fine di queste equazioni si ha:

$$\begin{aligned} \frac{de(k)}{dw_{1,j}} &= \frac{d[t(k) - a(k)]}{dw_{1,j}} = \frac{d}{dw_{1,j}} [t(k) - (w^T p(k) + b)] \\ &= \frac{d}{dw_{1,j}} \left[t(k) - \left(\sum_{i=1}^R w_{1,i} p_i(k) + b \right) \right] \end{aligned}$$

dove $p_i(k)$ è l'i-esimo elemento del vettore degli input alla k-esima iterazione. Ciò può essere semplificato come:

$$\frac{de(k)}{dw_{1,j}} = -p_j(k)$$

E allo stesso modo l'elemento R+1 sarà dato da:

$$\frac{de(k)}{db} = -1$$

$p_j(k)$ e 1 sono gli elementi del vettore degli input z, e quindi il gradiente dell'errore quadratico all'iterazione k può essere scritto come:

$$\hat{\nabla} F(x) = \nabla e^2(k) = -2e(k)z(k)$$

o analogamente, in forma estesa,

¹⁸ Queste due equazioni, ed in particolare i passaggi $\frac{de^2(k)}{dw_{1,j}} = 2e(k) \frac{de(k)}{dw_{1,j}}$ e $\frac{de^2(k)}{db} = 2e(k) \frac{de(k)}{db}$ sono stati ottenuti tenendo conto della regola della derivata del prodotto di due funzioni, cioè $d(e(x) \cdot e(x)) = de(x) \cdot e(x) + e(x) \cdot de(x) = 2e(x) \cdot de(x)$

$$\hat{\nabla}F(x) = \nabla e^2(k) = \begin{bmatrix} -2e(k)p_1(k) \\ -2e(k)p_2(k) \\ \vdots \\ -2e(k)p_j(k) \\ +2e(k)(-1) \end{bmatrix}$$

Da qui è possibile notare la funzionalità dell'approssimare l'errore quadratico medio con il singolo errore all'iterazione k: in tal modo infatti per calcolare il gradiente approssimato basta semplicemente moltiplicare l'errore per il vettore degli input. A questo punto si utilizza l'approssimazione del gradiente nell'algoritmo steepest descent. Ipotizzando che il tasso di apprendimento sia costante, come visto nel paragrafo 2.5 l'equazione fondamentale dell'algoritmo è:

$$x_{k+1} = x_k - \alpha g_k$$

Sostituendo il gradiente all'iterazione k con $\hat{\nabla}F(x)$ si ottiene:

$$x_{k+1} = x_k + 2\alpha e(k)z(k)$$

oppure:

$${}_1w(k+1) = {}_1w(k) + 2\alpha e(k)p(k)$$

e

$$b(k+1) = b(k) + 2\alpha e(k)$$

Queste equazioni, che formano l'algoritmo LMS (spesso detto anche delta rule), possono poi essere generalizzate nel caso in cui vi siano output multipli, e quindi neuroni multipli. Per far ciò è conveniente scrivere le equazioni di aggiornamento della matrice dei pesi e del vettore dei bias in forma matriciale nel modo seguente:

$$W(k+1) = W(k) + 2\alpha e(k)p^T(k)$$

e

$$b(k+1) = b(k) + 2\alpha e(k)$$

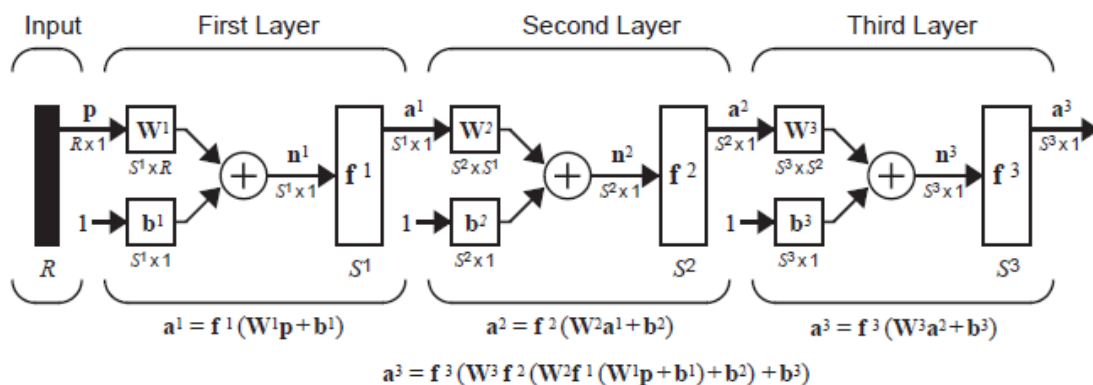
dove b ed e sono adesso dei vettori.

2.7 Backpropagation e relative variazioni

Il backpropagation è un algoritmo che come il precedente fa parte delle regole di apprendimento basate sulla performance e che può essere senza dubbio visto come il naturale sviluppo dell'algoritmo LMS. Anche questa regola è infatti un'approssimazione dello steepest descent in cui come indice di performance viene utilizzato l'errore quadratico medio. Tra l'algoritmo LMS e quello backpropagation tuttavia sussiste una sostanziale differenza data dal modo in cui vengono calcolate le derivate: nel primo infatti si ha una rete lineare (cioè con una funzione di trasferimento lineare) con un singolo strato e l'errore è una funzione lineare esplicita dei pesi della rete (le derivate possono cioè essere facilmente calcolate); nel secondo invece si ha una rete multistrato con funzioni di trasferimento non necessariamente lineari e per questo la relazione tra i pesi della rete e l'errore è molto più complessa e per calcolare le derivate è necessario utilizzare una particolare regola, detta regola della catena, che verrà trattata nel prosieguo del paragrafo.

Gli studiosi di reti neurali hanno iniziato ad approfondire lo studio dell'algoritmo backpropagation solo a partire dalla metà degli anni '80, per risolvere il problema principale delle reti utilizzate precedentemente, che come già ricordato più volte era quello di riuscire a risolvere solo problemi separabili linearmente. Da quel momento in poi, viste le capacità enormemente superiori di questa nuova tecnica, il perceptrone multistrato allenato con l'algoritmo backpropagation è diventato senza dubbio una delle reti neurali più utilizzate.

Il perceptrone multistrato è rappresentato, con la notazione abbreviata, dalla figura seguente (che è la stessa del paragrafo 2.2 e viene qui riportata solo per comodità):



In questo tipo di rete l'output di uno strato diventa l'input per lo strato successivo, secondo l'equazione:

$$a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1}) \quad \text{per } m = 0, \dots, M-1$$

dove M sono gli strati della rete. Il primo strato della rete riceve l'input esterno, cioè p, mentre l'output dell'ultimo strato è considerato l'output della rete nella sua totalità. L'algoritmo backpropagation, essendo una generalizzazione dell'algoritmo LMS, utilizza anch'esso come indice di performance l'errore quadratico medio dato da:

$$F(x) = E[e^2] = E[(t - a)^2]$$

dove come nel paragrafo precedente x è il vettore dei parametri della rete (pesi e bias). Nel caso in cui la rete abbia output multipli questa equazione può essere riscritta come:

$$F(x) = E[e^T e] = E[(t - a)^T (t - a)]$$

dove ovviamente adesso anche e, t ed a sono vettori. Esattamente come avviene per l'algoritmo LMS l'errore quadratico medio viene approssimato nel modo seguente:

$$\hat{F}(x) = ((t(k) - a(k))^T (t(k) - a(k))) = e^T(k)e(k)$$

E le equazioni per l'aggiornamento dei pesi e dei bias diventano:

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \frac{d\hat{F}}{dw_{i,j}^m}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha \frac{d\hat{F}}{db_i^m}$$

Fino ad ora la procedura di aggiornamento dei parametri è identica a quella dell'algoritmo LMS. Il problema principale è che se nelle reti ad un singolo strato (come la rete ADALINE) le derivate parziali delle equazioni precedenti potevano essere facilmente calcolate, in questo caso il calcolo di queste derivate è più complesso, in quanto l'errore è una funzione indiretta dei pesi e dei bias degli strati

nascosti. Per questo motivo si utilizza la regola della catena, che permette di calcolare la derivata della funzione composta di due funzioni derivabili. Data una funzione $f(n(w))$, è possibile calcolare la derivata di questa funzione rispetto a w nel modo seguente:

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \cdot \frac{dn(w)}{dw}$$

Utilizzando questa regola per calcolare le derivate parziali $\frac{d\hat{F}}{dw_{i,j}^m}$ e $\frac{d\hat{F}}{db_i^m}$ risulta:

$$\frac{d\hat{F}}{dw_{i,j}^m} = \frac{d\hat{F}}{dn_i^m} \cdot \frac{dn_i^m}{dw_{i,j}^m}$$

$$\frac{d\hat{F}}{db_i^m} = \frac{d\hat{F}}{dn_i^m} \cdot \frac{dn_i^m}{db_i^m}$$

dove $\frac{dn_i^m}{dw_{i,j}^m} = a_j^{m-1}$ e $\frac{dn_i^m}{db_i^m} = 1$ (tali derivate possono essere facilmente calcolate tenendo presente il fatto che l'input netto n dello strato m è una funzione esplicita dei pesi e dei bias di quello stesso strato, ovvero $n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$. Definendo poi la cosiddetta sensitività come:

$$s_i^m \equiv \frac{d\hat{F}}{dn_i^m}$$

le due equazioni precedenti possono essere così semplificate:

$$\frac{d\hat{F}}{dw_{i,j}^m} = s_i^m \cdot a_j^{m-1}$$

$$\frac{d\hat{F}}{db_i^m} = s_i^m \cdot (1) = s_i^m$$

Grazie a questo passaggio è possibile esprimere le equazioni di aggiornamento dei pesi e dei bias come:

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha s_i^m a_j^{m-1}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m$$

o, in forma matriciale:

$$W^m(k+1) = W^m(k) - \alpha s^m (a^{m-1})^T$$

$$b^m(k+1) = b^m(k) - \alpha s^m$$

dove a , b e s sono ora vettori, ed in particolare:

$$s^m \equiv \frac{d\hat{F}}{dn^m} = \begin{bmatrix} \frac{d\hat{F}}{dn_1^m} \\ \frac{d\hat{F}}{dn_2^m} \\ \vdots \\ \frac{d\hat{F}}{dn_{s^m}^m} \end{bmatrix}$$

A questo punto ciò che rimane da fare per derivare l'algoritmo backpropagation è calcolare questo vettore della sensitività s^m , che richiede un'altra applicazione della regola della catena. Questo calcolo è un processo ricorrente in quanto la sensitività di ogni strato è calcolata a partire dallo strato successivo, ed è proprio per questo motivo che l'algoritmo è detto backpropagation, perché è richiesta la propagazione all'indietro della sensitività per aggiornare i parametri della rete. Per derivare questa relazione ricorrente per la sensitività viene utilizzata la seguente matrice jacobiana (che non è altro che la matrice che ha per elementi le derivate parziali prime di una funzione):

$$\frac{dn^{m+1}}{dn^m} \equiv \begin{bmatrix} \frac{dn_1^{m+1}}{dn_1^m} & \frac{dn_1^{m+1}}{dn_2^m} & \dots & \frac{dn_1^{m+1}}{dn_{s^m}^m} \\ \frac{dn_2^{m+1}}{dn_1^m} & \frac{dn_2^{m+1}}{dn_2^m} & \dots & \frac{dn_2^{m+1}}{dn_{s^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dn_{s^{m+1}}^{m+1}}{dn_1^m} & \frac{dn_{s^{m+1}}^{m+1}}{dn_2^m} & \dots & \frac{dn_{s^{m+1}}^{m+1}}{dn_{s^m}^m} \end{bmatrix}$$

L'obiettivo è quello di trovare un'espressione per questa matrice, perché successivamente grazie a questa espressione sarà possibile calcolare la sensitività s^m . Considerando l'elemento i,j -esimo della matrice risulta:

$$\begin{aligned}\frac{dn_i^{m+1}}{dn_j^m} &= \frac{d(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1})}{dn_j^m} = w_{i,j}^{m+1} \frac{da_j^m}{dn_j^m} = w_{i,j}^{m+1} \frac{df^m(n_j^m)}{dn_j^m} \\ &= w_{i,j}^{m+1} f^m(n_j^m)\end{aligned}$$

dove:

$$f^m(n_j^m) = \frac{df^m(n_j^m)}{dn_j^m}$$

Grazie a questo la matrice Jacobiana può essere riscritta come:

$$\frac{dn^{m+1}}{dn^m} = W^{m+1} F^m(n^m)$$

dove:

$$F^m(n^m) = \begin{bmatrix} f^m(n_1^m) & 0 & \dots & 0 \\ 0 & f^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f^m(n_{S^m}^m) \end{bmatrix}$$

A questo punto la sensitività s^m , sempre scritta in forma matriciale e utilizzando la regola della catena diventa:

$$\begin{aligned}s^m &= \frac{d\hat{F}}{dn^m} = \left(\frac{dn^{m+1}}{dn^m} \right)^T \frac{d\hat{F}}{dn^{m+1}} = F^m(n^m) (W^{m+1})^T \frac{d\hat{F}}{dn^{m+1}} \\ &= F^m(n^m) (W^{m+1})^T s^{m+1}\end{aligned}$$

In questo modo è possibile calcolare la sensitività di uno strato a partire dalla sensitività dello strato successivo, ma per determinare esattamente l'algoritmo backpropagation è necessario un ultimo passo, ovvero quello di stabilire il punto di partenza per iniziare a propagare all'indietro la sensitività, ovvero s^M , la sensitività dell'ultimo strato. Questa sarà data da:

$$s_i^M = \frac{d\hat{F}}{dn_i^M} = \frac{d(t-a)^T(t-a)}{dn_i^M} = \frac{d\sum_{j=1}^M (t_j - a_j)^2}{dn_i^M} = -2(t_i - a_i) \frac{da_i}{dn_i^M}^{19}$$

¹⁹ $-2(t_i - a_i)$ deriva dalla regola della catena: è $\frac{d\sum_{j=1}^M (t_j - a_j)^2}{da_i} = -2(t_i - a_i)$

Adesso, poiché:

$$\frac{da_i}{dn_i^M} = \frac{da_i^M}{dn_i^M} = \frac{df^M(n_i^M)}{dn_i^M} = f^M(n_i^M)$$

È possibile scrivere la sensitività come:

$$s_i^M = -2(t_i - a_i)f^M(n_i^M)$$

che in forma matriciale diventa:

$$s^M = -2F^M(n^M)(t - a)$$

dove s , t , a e n sono adesso vettori. Volendo fare una sorta di riassunto dell'algoritmo backpropagation è possibile affermare che il primo passo da fare è quello di fornire alla rete l'input e calcolare l'output finale:

$$a^0 = p$$

$$a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1}) \quad \text{per } m = 0, \dots, M-1$$

$$a = a^M$$

Successivamente si calcola la sensitività dello strato finale e si propaga all'indietro fino allo strato iniziale:

$$s^M = -2F^M(n^M)(t - a)$$

$$s^m = F^m(n^m)(W^{m+1})^T s^{m+1} \quad \text{per } m = M-1, \dots, 2, 1$$

Infine, si aggiornano i parametri fondamentali della rete, cioè i pesi e i bias, secondo le equazioni mostrate in precedenza:

$$W^m(k+1) = W^m(k) - \alpha s^m (a^{m-1})^T$$

$$b^m(k+1) = b^m(k) - \alpha s^m$$

Come già ricordato, l'algoritmo presentato fino ad ora è un'approssimazione dello steepest descent. Esso tuttavia, pur avendo rappresentato un punto di svolta per tutti gli studi successivi sulle reti neurali, talvolta risulta molto lento nel convergere ad una soluzione per la maggior parte delle applicazioni pratiche. Per questo

motivo in letteratura si è cercato di modificare l'algoritmo backpropagation come variazione dello steepest descent al fine di trovare procedure che convergessero ad una soluzione più rapidamente. L'algoritmo in esame infatti funziona bene fino a che la rete ha un singolo strato: in tal caso infatti l'errore quadratico medio è una funzione quadratica e, avendo una matrice Hessiana costante, la curvatura della funzione in una data direzione non cambia (inoltre il profilo della funzione è ellittico). Nel caso in cui invece l'algoritmo venga applicato a reti neurali multistrato, esso risulta molto lento a causa del fatto che la funzione dell'errore quadratico medio è molto più complessa, caratterizzata da più punti di minimo locale, da una curvatura che può variare ampiamente a seconda del punto dello spazio in cui ci si trova e da un profilo che può essere dei più disparati. Per risolvere questa problematica, una delle soluzioni migliori proposte dagli studiosi è senza dubbio l'utilizzo di tecniche di ottimizzazione numerica già esistenti da applicare alle reti neurali multistrato. In particolare una delle tecniche che hanno avuto più successo può essere considerata il cosiddetto algoritmo Levenberg-Marquardt, che, continuando ad utilizzare la stessa rete multistrato, sostituisce l'algoritmo backpropagation come approssimazione dello steepest descent con questo nuovo algoritmo che non è altro che una variazione del classico metodo di ottimizzazione numerica di Newton.

Mentre l'algoritmo dello steepest descent trattato nel paragrafo 2.5 si basa sulla serie di Taylor del primo ordine, il metodo di Newton si basa sulla serie di Taylor del secondo ordine:

$$F(x_{k+1}) = F(x_k + \Delta x_k) \approx F(x_k) + g_k^T \Delta x_k + \frac{1}{2} \Delta x_k^T A_k \Delta x_k$$

Il principio che sta dietro questo metodo è cercare di localizzare il punto stazionario di questa approssimazione quadratica della $F(x)$. Quest'ultima equazione è infatti chiaramente una funzione quadratica del tipo

$$F(x) = \frac{1}{2} x^T A x + d^T x + c$$

dove $c = 0$. Dal paragrafo 2.4 è noto come il gradiente di questo tipo di funzioni sia:

$$\nabla F(x) = Ax + d$$

ovvero, in questo caso,

$$\nabla F(x) = A_k \Delta x_k + g_k$$

A questo punto uguagliando a zero il gradiente e risolvendo per Δx_k risulta:

$$\Delta x_k = (x_{k+1} - x_k) = -A_k^{-1} g_k$$

E quindi il metodo di Newton è definito dall'equazione seguente:

$$x_{k+1} = x_k - A_k^{-1} g_k$$

dove:

$$g_k \equiv \nabla F(x)(x_0)$$

$$A_k \equiv \nabla^2 F(x)(x_0)$$

Questo metodo ha il vantaggio di essere molto più rapido dell'algoritmo steepest descent: esso infatti molto spesso (per funzioni quadratiche con un minimo locale) convergerà ad una soluzione in un solo step. Come già detto l'algoritmo Levenberg- Marquardt è una variazione di questo metodo, ed in particolare è una variazione ideata per minimizzare funzioni che sono la somma di quadrati o altre funzioni non lineari. L'algoritmo quindi si presenta come ideale per allenare reti neurali multistrato in cui l'indice di performance è dato dall'errore quadratico medio. Assumendo che la $F(x)$ sia una funzione di somma di quadrati del tipo:

$$F(x) = \sum_{i=1}^N v_i^2(x) = v^T(x)v(x)$$

dove le v a destra dell'equazione sono vettori, così come la x , il j -esimo elemento del gradiente sarà dato da (dalla regola della catena):

$$[\nabla F(x)]_j = \frac{dF(x)}{dx_j} = 2 \sum_{i=1}^N v_i(x) \frac{dv_i(x)}{dx_j}$$

Il gradiente può quindi essere riscritto in forma matriciale nel seguente modo:

$$\nabla F(x) = 2J^T(x)v(x)$$

dove

$$J(x) = \begin{bmatrix} \frac{dv_1(x)}{dx_1} & \frac{dv_1(x)}{dx_2} & \dots & \frac{dv_1(x)}{dx_n} \\ \frac{dv_2(x)}{dx_1} & \frac{dv_2(x)}{dx_2} & \dots & \frac{dv_2(x)}{dx_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dv_N(x)}{dx_1} & \frac{dv_N(x)}{dx_2} & \dots & \frac{dv_N(x)}{dx_n} \end{bmatrix}$$

Per quanto riguarda la matrice Hessiana invece, il suo elemento k,j- esimo sarà dato da:

$$[\nabla^2 F(x)]_{k,j} = \frac{d^2 F(x)}{dx_k dx_j} = 2 \sum_{i=1}^N \left\{ \frac{dv_i(x)}{dx_k} \frac{dv_i(x)}{dx_j} + v_i(x) \frac{d^2 v_i(x)}{dx_k dx_j} \right\}$$

ed essa potrà quindi essere scritta in forma matriciale nel modo seguente:

$$\nabla^2 F(x) = 2J^T(x)J(x) + 2S(x)$$

dove

$$S(x) = \sum_{i=1}^N v_i(x) \nabla^2 v_i(x)$$

Assumendo inoltre che $S(x)$ sia piccolo, la matrice Hessiana può essere approssimata come:

$$\nabla^2 F(x) \approx 2J^T(x)J(x)$$

Sostituendo poi le equazioni del gradiente e della matrice Hessiana in quella del metodo di Newton si ottiene il cosiddetto metodo Newton-Gauss:

$$\begin{aligned} x_{k+1} &= x_k - [2J^T(x_k)J(x_k)]^{-1} 2J^T(x_k)v(x_k) \\ &= x_k - [J^T(x_k)J(x_k)]^{-1} J^T(x_k)v(x_k) \end{aligned}$$

Da questo metodo emerge tuttavia un problema, cioè il fatto che la matrice $H = J^T J$ potrebbe non essere invertibile. Nonostante ciò questo problema può essere

facilmente superato andando a modificare la matrice Hessiana con la seguente approssimazione:

$$G = H + \mu I$$

In questo modo è garantita l'invertibilità della matrice H^{20} ed è possibile esprimere l'equazione finale dell'algoritmo Levenberg- Marquardt:

$$x_{k+1} = x_k - [J^T(x_k)J(x_k) + \mu_k I]^{-1} J^T(x_k) v(x_k)$$

o

$$\Delta x_k = -[J^T(x_k)J(x_k) + \mu_k I]^{-1} J^T(x_k) v(x_k)$$

L'algoritmo quindi inizia fissando μ_k ad un certo valore (per esempio 0.02). Se lo step non produce un valore più piccolo per la $F(x)$, questo viene ripetuto moltiplicando μ_k per un certo fattore $\vartheta > 1$. Ad un certo punto la $F(x)$ decrescerà, ed allo step successivo si moltiplicherà μ_k per un certo fattore $\vartheta < 1$. Andando infatti ad analizzare come l'algoritmo venga utilizzato per allenare le reti neurali multistrato è necessario innanzitutto ricordare come per queste reti l'indice di performance sia dato dall'errore quadratico medio e come se ogni target si presenta con uguale probabilità questo errore è proporzionale alla somma dei singoli errori quadratici sui Q target che vengono presentati alla rete:

$$F(x) = \sum_{q=1}^Q (t_q - a_q)^T (t_q - a_q) = \sum_{q=1}^Q e_q^T e_q = \sum_{q=1}^Q \sum_{j=1}^{s^M} (e_{j,q})^2 = \sum_{i=1}^N (v_i)^2$$

dove $e_{j,q}$ non è altro che il j -esimo elemento dell'errore per la q -esima coppia input/output che viene presentata alla rete.

Dall'equazione finale dell'algoritmo appare inoltre subito chiaro come il punto fondamentale del metodo Levenberg- Marquardt sia rappresentato dal calcolo della matrice Jacobiana. Al fine di effettuare questo calcolo si utilizza una variazione dell'algoritmo backpropagation. Tuttavia, mentre in quest'ultimo era

²⁰ Per la dimostrazione si veda Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 12, pp. 21

necessario calcolare le derivate degli errori quadratici rispetto ai pesi e ai bias della rete, in questo caso è necessario calcolare le derivate degli errori (anziché quindi degli errori quadratici). Prima di presentare la metodologia di calcolo della matrice Jacobiana, attraverso uno sguardo più attento alla sua forma si nota come il vettore degli errori e quello dei parametri siano dati rispettivamente da:

$$v^T = [v_1 \ v_2 \ \dots \ v_N] = [e_{1,1} \ e_{2,1} \ \dots \ e_{s^M,1} \ e_{1,2} \ \dots \ e_{s^M,Q}]$$

$$x^T = [x_1 \ x_2 \ \dots \ x_n] = [w_{1,1}^1 \ w_{1,2}^1 \ \dots \ w_{s^1,R}^1 \ b_1^1 \ \dots \ b_{s^1}^1 \ w_{1,1}^2 \ \dots \ b_{s^M}^M]$$

dove

$$N = Q \times S^M \text{ e } n = S^1(R + 1) + S^2(S^1 + 1) + \dots + S^M(S^{M-1} + 1).$$

Sostituendo ora tali espressioni nella matrice Jacobiana scritta precedentemente, otteniamo la matrice Jacobiana necessaria per allenare le reti neurali multistrato:

$$J(x) = \begin{bmatrix} \frac{de_{1,1}}{dw_{1,1}^1} & \frac{de_{1,1}}{dw_{1,2}^1} & \dots & \frac{de_{1,1}}{dw_{s^1,R}^1} & \frac{de_{1,1}}{db_1^1} & \dots \\ \frac{de_{1,2}}{dw_{1,1}^1} & \frac{de_{1,2}}{dw_{1,2}^1} & \dots & \frac{de_{1,2}}{dw_{s^1,R}^1} & \frac{de_{1,2}}{db_1^1} & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \dots \\ \frac{de_{s^M,1}}{dw_{1,1}^1} & \frac{de_{s^M,1}}{dw_{1,2}^1} & \dots & \frac{de_{s^M,1}}{dw_{s^1,R}^1} & \frac{de_{s^M,1}}{db_1^1} & \dots \\ \frac{de_{1,2}}{dw_{1,1}^1} & \frac{de_{1,2}}{dw_{1,2}^1} & \dots & \frac{de_{1,2}}{dw_{s^1,R}^1} & \frac{de_{1,2}}{db_1^1} & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \end{bmatrix}$$

Ogni termine della stessa può essere calcolato tramite una variazione dell'algoritmo backpropagation. In questo caso infatti, per gli elementi della matrice Jacobiana dell'algoritmo Levenberg- Marquardt, anziché calcolare

$$\frac{d\hat{F}(x)}{dx_l} = \frac{de_q^T e_q}{dx_l} \text{ è necessario calcolare:}$$

$$[J]_{h,l} = \frac{dv_h}{dx_l} = \frac{de_{k,q}}{dx_l}$$

Inoltre, se nell'algoritmo backpropagation si calcolava:

$$\frac{d\hat{F}}{dw_{i,j}^m} = \frac{d\hat{F}}{dn_i^m} \cdot \frac{dn_i^m}{dw_{i,j}^m}$$

dove il primo termine del membro di destra era la sensitività definita come:

$$s_i^m \equiv \frac{d\hat{F}}{dn_i^m}$$

che veniva calcolata a partire dall'ultimo strato fino al primo, nell'algoritmo Levenberg- Marquardt è possibile utilizzare un procedimento analogo definendo la cosiddetta sensitività di Marquardt come:

$$s_{i,h}^{\sim m} \equiv \frac{dv_h}{dn_{i,q}^m} = \frac{de_{k,q}}{dn_{i,q}^m}$$

dove dall'equazione del vettore degli errori risulta $h = (q - 1)s^M + k$. Adesso è possibile calcolare l'elemento h,l -esimo della matrice Jacobiana attraverso le due equazioni seguenti, che si riferiscono rispettivamente al caso in cui x_l sia un peso o un bias:

$$[J]_{h,l} = \frac{dv_h}{dx_l} = \frac{de_{k,q}}{dw_{i,j}^m} = \frac{de_{k,q}}{dn_{i,q}^m} \times \frac{dn_{i,q}^m}{dw_{i,j}^m} = s_{i,h}^{\sim m} \times \frac{dn_{i,q}^m}{dw_{i,j}^m} = s_{i,h}^{\sim m} \times a_{j,q}^{m-1}$$

$$[J]_{h,l} = \frac{dv_h}{dx_l} = \frac{de_{k,q}}{db_i^m} = \frac{de_{k,q}}{dn_{i,q}^m} \times \frac{dn_{i,q}^m}{db_i^m} = s_{i,h}^{\sim m} \times \frac{dn_{i,q}^m}{db_i^m} = s_{i,h}^{\sim m}$$

La sensitività di Marquardt può essere calcolata attraverso la stessa relazione utilizzata per l'algoritmo backpropagation, con una sola modifica per quanto riguarda lo strato finale. Per la sensitività di Marquardt alla strato finale si ha infatti:

$$s_{i,h}^{\sim M} \equiv \frac{dv_h}{dn_{i,q}^M} = \frac{de_{k,q}}{dn_{i,q}^M} = \frac{d(t_{k,q} - a_{k,q}^M)}{dn_{i,q}^M} = \frac{da_{k,q}^M}{dn_{i,q}^M} = \begin{cases} -f^M(dn_{i,q}^M) & \text{per } i = k \\ 0 & \text{per } i \neq k \end{cases}$$

Quindi ogni volta che l'input p_q è stato presentato alla rete e il corrispondente output a_q^M è stato calcolato, l'algoritmo backpropagation Levenberg- Marquardt è inizializzato con:

$$S_q^M = -F^M(n_q^M)$$

dove, come per il backpropagation standard,

$$F^m(n^m) = \begin{bmatrix} f^m(n_1^m) & 0 & \dots & 0 \\ 0 & f^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f^m(n_{s^m}^m) \end{bmatrix}$$

Ogni colonna della matrice S_q^M deve essere propagata all'indietro attraverso la rete tramite l'equazione seguente, già incontrata in precedenza:

$$\begin{aligned} s^m &= \frac{d\hat{F}}{dn^m} = \left(\frac{dn^{m+1}}{dn^m} \right)^T \frac{d\hat{F}}{dn^{m+1}} = F^m(n^m)(W^{m+1})^T \frac{d\hat{F}}{dn^{m+1}} \\ &= F^m(n^m)(W^{m+1})^T s^{m+1} \end{aligned}$$

Le matrici della sensitività totale di Marquardt per ogni strato sono quindi create aumentando le matrici calcolate per ogni input:

$$s^M = [S_1^m | S_2^m | \dots | S_Q^m]$$

Da notare come per ogni input che viene presentato alla rete è necessario propagare all'indietro s^M vettori della sensitività. Ciò avviene perché, come detto in precedenza, in questa variante dell'algoritmo si calcolano le derivate degli errori individuali piuttosto che le derivate della somma dei quadrati degli errori. Per ogni input fornito alla rete ci saranno s^M errori (uno per ogni elemento degli output della rete), e per ogni errore ci sarà una riga della matrice Jacobiana. Alla fine, dopo che le sensitività sono state propagate all'indietro, sarà possibile calcolare ogni singolo elemento della matrice.

Volendo a questo punto fare un riassunto dell'algoritmo Levenberg- Marquardt è possibile affermare che esso si sviluppa nei quattro punti seguenti:

- Si presentano tutti gli input alla rete e si calcolano i corrispondenti output usando le equazioni:

$$\begin{aligned} a^0 &= p \\ a^{m+1} &= f^{m+1}(W^{m+1}a^m + b^{m+1}) \quad \text{per } m = 0, \dots, M-1 \end{aligned}$$

Si calcolano poi gli errori $e_q = t_q - a_q^M$ e la somma degli errori quadrati su tutti gli input, $F(x)$ tramite l'equazione

$$F(x) = \sum_{q=1}^Q (t_q - a_q)^T (t_q - a_q) = \sum_{q=1}^Q e_q^T e_q = \sum_{q=1}^Q \sum_{j=1}^{s^M} (e_{j,q})^2 = \sum_{i=1}^N (v_i)^2$$

- Si calcola la matrice Jacobiana. Si calcola la sensitività tramite l'equazione che permette di propagare all'indietro tutte le colonne della matrice s_q^M insieme, ovvero $s_q^M = F^m(n_q^m)(W^{m+1})^T s_q^{m+1}$ dopo aver inizializzato con l'equazione $s_q^M = -F^M(n_q^M)$. Si aumentano le matrici individuali utilizzando l'equazione $s^M = [s_1^M | s_2^M | \dots | s_Q^M]$ e si calcolano i singoli elementi della matrice Jacobiana.
- Si ottiene $\Delta x_k = -[J^T(x_k)J(x_k) + \mu_k I]^{-1} J^T(x_k) v(x_k)$
- Si ricalcola la somma degli errori al quadrato utilizzando $x_k + \Delta x_k$. Se la nuova somma degli errori al quadrato è minore rispetto a quella calcolata al punto 1, allora si divide μ per ϑ , cosicché $x_{k+1} = x_k + \Delta x_k$ e si torna al punto 1. Se invece la somma degli errori al quadrato è maggiore si moltiplica μ per ϑ e si torna al punto 3.

2.8 Generalizzazione e metodologie per incrementare la generalizzazione della rete neurale

Uno dei problemi più difficili da risolvere quando si costruisce una certa rete neurale è lo stabilire qual è il numero adeguato di neuroni da utilizzare in ogni strato. Da questa scelta infatti dipendono la capacità della rete di generalizzare adeguatamente e soprattutto il problema dell'overfitting. Una rete con un numero di neuroni troppo elevato infatti quasi certamente incorrerà in questo tipo di problema, dato dal fatto che l'errore della rete sarà molto ridotto per i dati del training set, ma allo stesso tempo sarà molto elevato quando alla rete verranno presentati nuovi dati (la rete generalizzerà quindi in modo non adeguato). Perciò l'obiettivo che ci si prefissa quando si va a costruire una rete neurale è quello di crearne una che si comporti con i nuovi dati che le vengono presentati così come

si era comportata con i dati sui quali era stata allenata. Solo in questo modo infatti la rete potrà generalizzare adeguatamente. La strategia chiave per ottenere una buona generalizzazione è quella di trovare il modello più semplice che spieghi i dati, ovvero la rete con il più piccolo numero di parametri (pesi e bias), o, analogamente, con il più piccolo numero di neuroni. Volendo meglio definire il problema, si supponga che il training set di una certa rete sia dato dalle seguenti coppie di input/target:

$$\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_Q, t_Q\}$$

e che il target sia generato da:

$$t_q = g(p_q) + \varepsilon_q$$

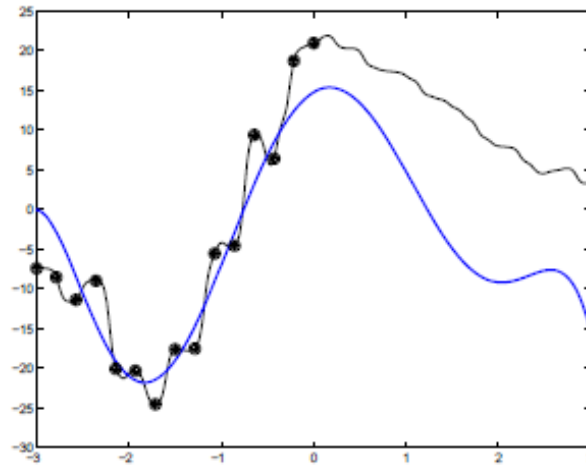
dove g è una certa funzione e ε_q è una fonte di errore indipendente con media zero (rumore). L'obiettivo dell'allenamento è quello di produrre una rete che approssimi la funzione g ignorando l'errore casuale. Com'è noto, l'indice di performance per l'allenamento delle reti neurali è la somma degli errori quadratici del training set:

$$F(x) = E_D = \sum_{q=1}^Q (t_q - a_q)^T (t_q - a_q)$$

dove a_q è l'output della rete corrispondente all'input p_q . In questo caso si è utilizzato la variabile E_D per rappresentare la somma degli errori quadratici del training set solamente perché in seguito l'indice di performance verrà modificato per comprendere un termine aggiuntivo.

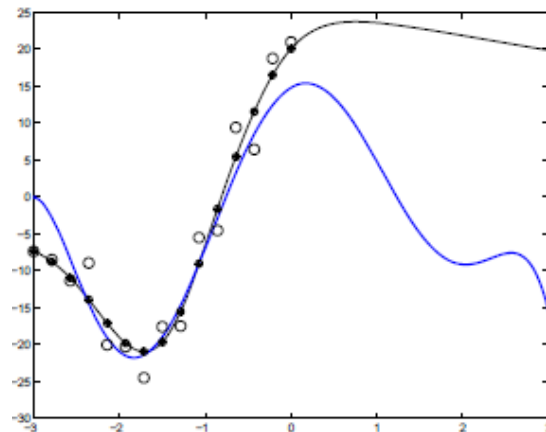
Una volta definito il problema, è possibile mostrare concretamente cosa si intende per overfitting dalla figura seguente²¹:

²¹ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 13, pp. 4



La curva blu rappresenta la funzione g ; i cerchi più larghi rappresentano i punti target (compreso l'errore casuale); i cerchi più piccoli rappresentano la risposta della rete ai punti con i quali è stata allenata ed infine la curva nera rappresenta la risposta della rete neurale una volta che quest'ultima è stata allenata. Come si può notare dalla figura la risposta della rete combacia esattamente con i punti con i quali è stata allenata, ma non riesce a combaciare con la funzione sottostante: ha quindi un problema di overfitting. Andando ad analizzare la questione in maniera più accurata è possibile notare come in realtà da questo esempio emergano due problemi. Il primo, causato dall'overfitting, si ha quando i valori di input sono compresi tra -3 e 0. In questo caso la rete ha appunto un problema di overfitting sui training points perché non riesce a lavorare adeguatamente con input che non sono presenti nel training set. La rete non riesce a interpolare correttamente, non riesce cioè ad approssimare accuratamente la funzione sottostante vicino ai punti con i quali è stata allenata. Il secondo si ha quando i valori di input sono compresi tra 0 e 3. In questo caso la rete non riesce ad approssimare la funzione non per un problema di overfitting, ma semplicemente perché non ci sono punti con i quali è stata allenata in quell'intervallo. La rete cioè sta estrapolando oltre il range dei dati di input. Ovviamente vi sono dei metodi per evitare (o quantomeno limitare) l'overfitting, ma non ci sono metodi per evitare il problema della scorretta estrapolazione, poiché la rete non può sapere com'è la funzione in un range in cui non ci sono dati (l'unico modo è quello di allenare la rete con un insieme di dati che coprano tutte le regioni degli spazio degli input dove la rete verrà utilizzata).

Un caso invece in cui la rete è stata allenata per generalizzare correttamente è il seguente²²:



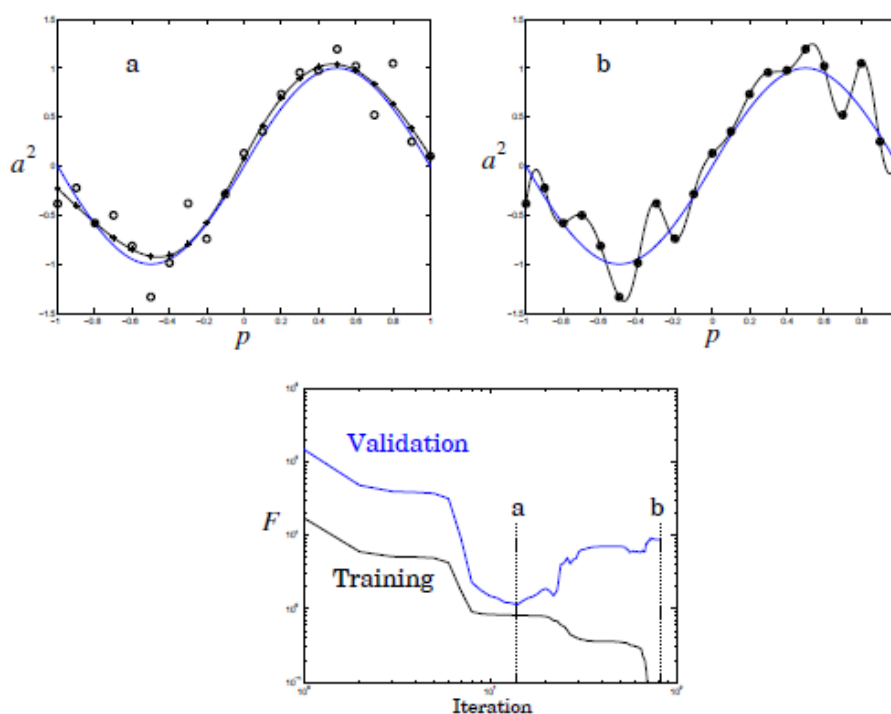
Come si nota dalla figura la rete in questo caso approssima la funzione nel modo migliore possibile nel range -3, 0, ma è caratterizzata dallo stesso problema di estrapolazione della rete precedente, proprio perché non è stata allenata con dati che vanno oltre il valore 0.

Passando ora ai metodi per incrementare la generalizzazione della rete, due tra i più utilizzati sono il metodo early stopping e il metodo regularization. Prima di trattare questi metodi in modo approfondito è tuttavia necessario assumere che il numero di dati a disposizione per allenare la rete sia limitato. Se infatti il numero di dati fosse illimitato, cioè se l'ammontare di dati fosse significativamente più ampio dei parametri della rete, probabilmente non vi sarebbero problemi di overfitting. Inoltre con un numero limitato di dati, è importante tenerne da parte un sottoinsieme da utilizzare come test set, poiché l'errore che la rete produrrà sul test set è un'indicazione della capacità della rete di generalizzare.

Il metodo early stopping è un metodo basato su un'idea molto semplice: poiché man a mano che il processo di allenamento della rete procede essa utilizza sempre più parametri (pesi, bias) fino a che non li utilizza tutti quando raggiunge l'ottimizzazione dell'indice di performance, se l'allenamento viene fermato prima che questo minimo venga raggiunto la rete utilizzerà effettivamente meno parametri e sarà meno probabile il problema dell'overfitting. Tuttavia, al fine di utilizzare questo metodo, è necessario determinare quando effettivamente fermare

²² Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 13, pp. 4

l'allenamento. Ciò si realizza tramite la cosiddetta cross-validation, cioè si utilizza un validation set per decidere quando fermarsi. I dati disponibili dopo aver rimosso la parte necessaria per il test set vengono suddivisi in due parti: il training set e il validation set. Successivamente, mentre il training set viene usato per calcolare il gradiente o la matrice Jacobiana e per determinare l'aggiornamento dei parametri ad ogni iterazione, il validation set viene utilizzato come indicatore di cosa succede alla funzione della rete “nel mezzo” ai training points e il suo errore viene monitorato costantemente durante tutto il processo di allenamento. Quando l'errore sul validation set aumenta per un certo numero di iterazioni l'allenamento viene fermato e i pesi che hanno prodotto il minimo errore sul validation set vengono utilizzati come i pesi finali della rete neurale. Per meglio comprendere questo metodo tuttavia è necessaria la seguente figura²³:



Il grafico in basso mostra l'andamento del processo di allenamento con l'indice di performance F (somma degli errori quadratici) sul training set e sul validation set in funzione del numero di iterazioni. In particolare è possibile notare come anche se l'errore sul training set continui a diminuire ad ogni iterazione, il minimo errore sul validation set si ha in corrispondenza della quattordicesima iterazione (punto

²³ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 13, pp. 7

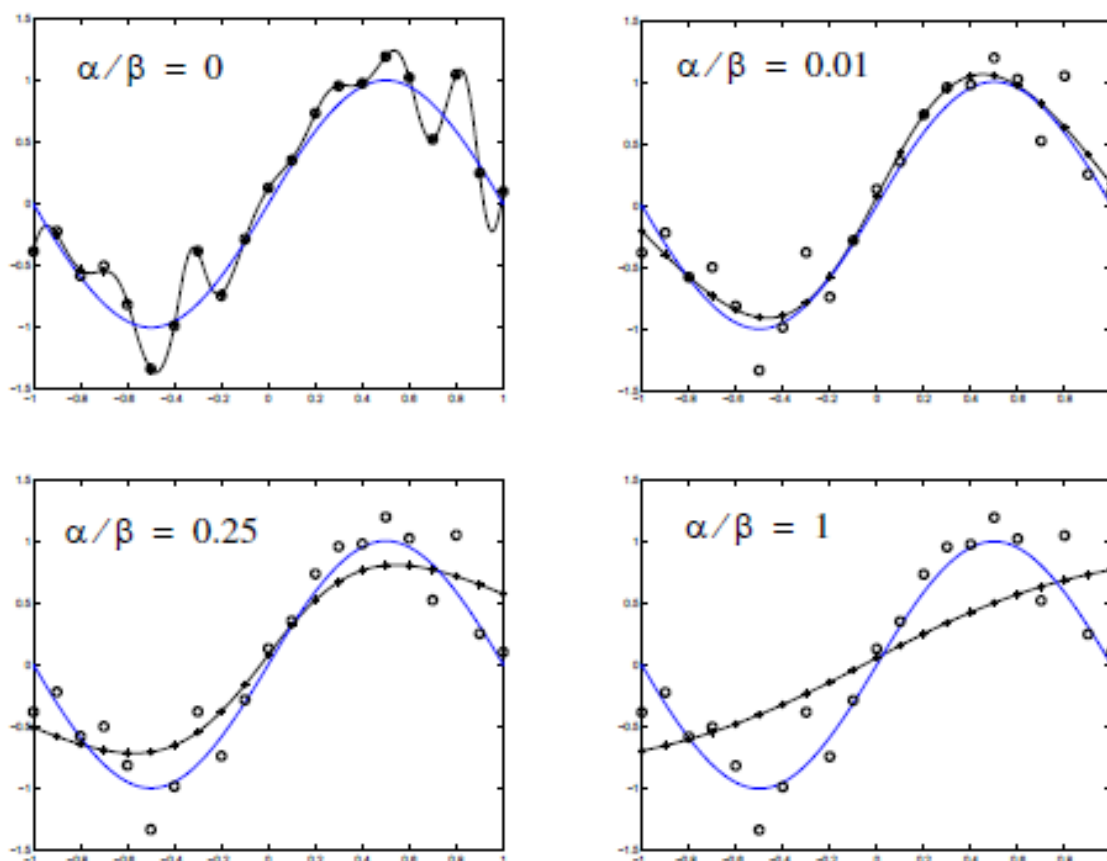
a). Il grafico in alto a sinistra mostra quindi la risposta della rete a questo punto di early stopping mentre il grafico in alto a destra mostra la risposta della rete nel caso l'allenamento continui a procedere fino al punto b. Questi tre grafici, analizzati nel complesso, dimostrano come il problema dell'overfitting, chiaro nel caso del grafico di destra, possa essere limitato dall'utilizzo di questa metodologia. Nonostante la semplicità dell'early stopping, per utilizzare questo metodo è necessario scegliere attentamente i tre data sets: validation set, training set e test set infatti devono essere rappresentativi di tutte le situazioni per le quali la rete verrà utilizzata. Ogni set cioè, anche se avrà dimensione diversa, dovrà essere equivalente come copertura dello spazio degli input.

Con il metodo regularization invece si va a modificare l'indice di performance in modo che vi sia incluso un termine che penalizza la complessità della rete. Si aggiunge cioè un termine di penalità (o di regularization) che riguarda le derivate della rete neurale e che "obbliga" la funzione risultante ad essere più liscia. Sotto certe condizioni, questo termine di regularization può essere scritto come somma dei quadrati dei pesi della rete:

$$F(x) = \beta E_D + \alpha E_W = \beta \sum_{q=1}^Q (t_q - a_q)^T (t_q - a_q) + \alpha \sum_{i=1}^N x_i^2$$

dove il rapporto α/β controlla l'effettiva complessità delle soluzioni della rete. In particolare, dato che più elevato sarà questo rapporto più la risposta della rete sarà liscia e viceversa, il successo di questo metodo risiede interamente nella corretta scelta del parametro α/β , che determina la capacità della rete di generalizzare adeguatamente evitando il problema dell'overfitting. È possibile notare come la scelta del parametro influenzi effettivamente la risposta della rete dalla figura seguente, nella quale viene mostrata una rete del tipo 1-20-1 allenata su 21 campioni (rumorosi) della funzione seno²⁴:

²⁴ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 13, pp. 10

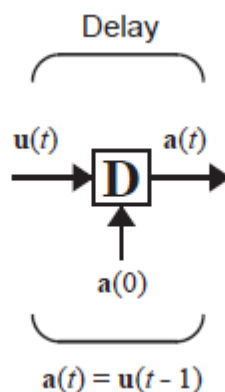


La linea blu rappresenta la funzione seno, i cerchi vuoti rappresentano i dati (con rumore), la linea nera rappresenta la risposta della rete e i cerchi neri rappresentano la risposta della rete nei punti con i quali è stata allenata. Come si vede, il rapporto α/β che produce la migliore risposta della rete è $\frac{\alpha}{\beta} = 0.01$. Negli altri casi la rete ha una performance peggiore, ed in particolare per rapporti più elevati la risposta della rete è troppo liscia mentre nel caso in cui il rapporto è più basso si manifesta chiaramente il problema dell'overfitting. Per concludere appare doveroso specificare come vi siano diversi metodi per determinare questo parametro di regularization: un primo approccio è quello di utilizzare un validation set (proprio come nell'early stopping) e di prendere come parametro di regularization quello che minimizza gli errori quadratici sul validation set; un altro approccio è quello di utilizzare la cosiddetta Bayesian regularization. In ogni caso, queste metodologie non verranno approfondite nella trattazione.

2.9 Reti neurali dinamiche

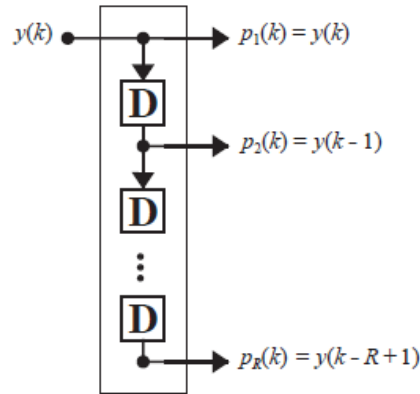
Le reti neurali che sono state affrontate fino a questo punto sono tutte reti neurali statiche, ovvero reti in cui l'output è calcolato direttamente dall'input tramite le connessioni feedforward. A questo punto è però utile introdurre anche un altro tipo di rete, quella dinamica, in cui l'output non dipende solo dall'input corrente ma anche dai precedenti input, output e stati della rete. In altre parole le reti dinamiche, avendo non solo connessioni del tipo feedforward ma anche connessioni ricorrenti di tipo feedback, calcolano l'output corrente come funzione degli output agli istanti di tempo precedenti. Nonostante l'allenamento di queste reti avvenga con le stesse tecniche di ottimizzazione descritte in precedenza (algoritmi steepest descent e Levenberg- Marquardt), la differenza tra allenare una rete neurale statica e una dinamica consiste essenzialmente nel modo in cui il gradiente o la matrice Jacobiana sono calcolate. Nel caso di reti dinamiche infatti non è possibile utilizzare l'algoritmo backpropagation standard, e per questo si usano fondamentalmente altri due algoritmi per il calcolo del gradiente e della matrice Jacobiana: il backpropagation-through-time (BPTT) e il real-time-recurrent learning (RTRL). Prima di trattare questi algoritmi in dettaglio è tuttavia conveniente introdurre una notazione aggiuntiva che permetta una migliore comprensione dell'architettura di questo tipo di reti.

In primo luogo è necessario presentare il cosiddetto delay block mostrato in figura²⁵:

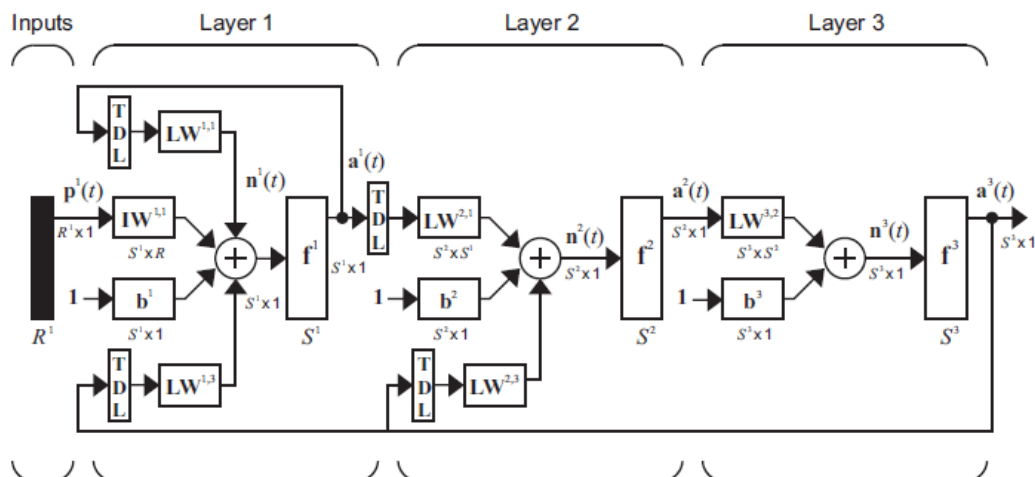


²⁵ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 2, pp. 13

dove l'output $a(t)$ è calcolato dal suo input $u(t)$ tramite l'equazione $a(t) = u(t - 1)$ (qui si assume che il tempo sia aggiornato in modo discreto e che assuma solo valori interi). Il secondo blocco da introdurre è la tapped delay line, mostrata dalla figura seguente²⁶ (con R output):



L'output della tapped delay line è un vettore di dimensione R, che consiste nell'input corrente e un ritardo che comprende gli step di tempo da 1 a R-1. Per rappresentare le reti neurali dinamiche si utilizza una particolare cornice, detta Layered Digital Dynamic Network (LDDN), che è un'estensione della notazione utilizzata per le reti neurali multistrato statiche e che permette di rappresentare reti con connessioni ricorrenti multiple (feedback) e tapped delay lines. Per introdurre la notazione LDDN si consideri la seguente rete²⁷:



²⁶ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 10, pp. 14

²⁷ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 14, pp. 3

L'equazione fondamentale per il calcolo dell'input netto $n^m(t)$ per lo strato m di una LDDN è:

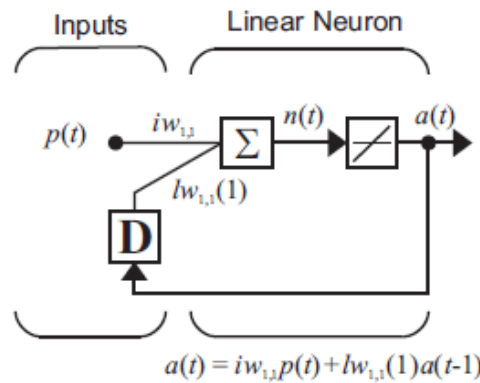
$$n^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} LW^{m,l}(d) a^l(t-d) + \sum_{l \in I_m} \sum_{d \in DI_{m,l}} IW^{m,l}(d) p^l(t-d) + b^m$$

Dove $p^l(t)$ è l' l -esimo vettore di input della rete al tempo t , $IW^{m,l}$ è il peso dell'input tra l'input l e lo strato m , $LW^{m,l}$ è il peso dello strato tra lo stato l e lo strato m , b^m è il vettore dei bias per lo strato m , $DL_{m,l}$ è l'insieme di tutti i ritardi della tapped delay line tra l'input l e lo strato m , I_m è l'insieme di indici dei vettori degli input che connettono allo strato m , L_m^f è l'insieme degli indici degli starti che connettono direttamente in modo forward allo strato m . L'output dello strato m sarà quindi calcolato tramite l'equazione:

$$a^m(t) = f^m(n^m(t))$$

Come si nota a differenza di quanto accadeva con le reti statiche le reti dinamiche possono avere diversi strati che sono connessi con uno strato m (anche in modo ricorrente tramite le tapped delay line), ovvero gli strati non sono più collegati tra loro in ordine numerico, ma al contrario ogni strato può essere connesso a qualsiasi altro strato, compreso se stesso. Inoltre questo tipo di reti possono avere vettori di input multipli connessi con qualunque strato della rete (nelle reti statiche invece si aveva un solo vettore di input connesso solamente al primo strato della rete stessa). Tuttavia, per poter utilizzare l'equazione precedente e calcolare così l'output della rete, è necessario calcolare gli output dei vari strati in uno specifico ordine, detto ordine di simulazione (quest'ordine non è però unico, dal momento che vi possono essere più ordini di simulazione validi). Per propagare all'indietro le derivate per il calcolo del gradiente invece si procede nell'ordine inverso, detto ordine di backpropagation. Per esempio nella rete dinamica mostrata in precedenza uno degli ordini di simulazione è 1- 2- 3, mentre l'ordine di backpropagation è 3- 2- 1. Il problema è che dal momento che l'output di una LDDN non è funzione solo dei pesi, dei bias e degli input correnti della rete ma anche di alcuni output degli strati

agli istanti di tempo precedenti, non risulta semplice calcolare il gradiente dell'output della rete rispetto ai pesi e ai bias. In particolare i pesi e i bias hanno un duplice, differente effetto sull'output della rete neurale: hanno in primo luogo un effetto diretto, che si calcola semplicemente tramite l'algoritmo backpropagation standard, ed hanno poi un effetto indiretto, poiché tutti o alcuni degli input della rete sono output precedenti, i quali sono a loro volta funzione dei pesi e dei bias. Al fine di comprendere meglio quali sono questi effetti e come incidono sul calcolo del gradiente è utile considerare la seguente, semplice rete neurale²⁸:



Considerando come indice di performance l'errore quadratico medio

$$F(x) = \sum_{t=1}^Q e^2(t) = \sum_{t=1}^Q (t(t) - a(t))^2$$

Gli elementi del gradiente saranno dati da:

$$\frac{dF(x)}{dlw_{1,1}(1)} = \sum_{t=1}^Q \frac{de^2(t)}{dlw_{1,1}(1)} = -2 \sum_{t=1}^Q e(t) \frac{da(t)}{dlw_{1,1}(1)}$$

$$\frac{dF(x)}{diw_{1,1}} = \sum_{t=1}^Q \frac{de^2(t)}{diw_{1,1}} = -2 \sum_{t=1}^Q e(t) \frac{da(t)}{diw_{1,1}}$$

Nel caso di una rete neurale statica, come è emerso dai paragrafi precedenti, i termini chiave di queste equazioni dati da $\frac{da(t)}{dlw_{1,1}(1)}$ e da $\frac{da(t)}{diw_{1,1}}$ sarebbero relativamente semplici da calcolare, poiché sapendo che $a(t) = iw_{1,1}p(t) +$

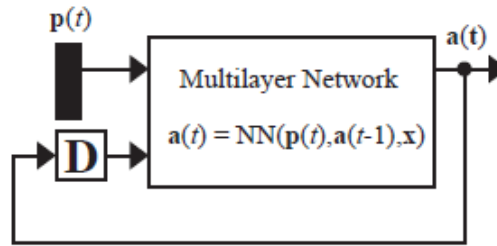
²⁸ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 14, pp. 7

$lw_{1,1}(1)a(t-1)$ essi risulterebbero pari rispettivamente a $a(t-1)$ e $p(t)$. Tuttavia, com'è stato appena ricordato, nelle reti dinamiche i pesi hanno un duplice effetto, diretto e indiretto, e per questo motivo i termini chiave dell'equazione precedente devono essere calcolati nel modo seguente:

$$\frac{da(t)}{dlw_{1,1}(1)} = a(t-1) + lw_{1,1}(1) \frac{da(t-1)}{dlw_{1,1}(1)}$$

$$\frac{da(t)}{diw_{1,1}} = p(t) + lw_{1,1}(1) \frac{da(t-1)}{diw_{1,1}}$$

Dove i primi termini rappresentano l'effetto diretto e i secondi invece quello indiretto. Abbandonando poi questa semplice rete con un singolo neurone e andando a considerarne una leggermente più complessa come la seguente²⁹ (dove come sempre x rappresenta il vettore dei parametri della rete e $a(t)$ l'output al tempo t), è possibile notare quali siano le modifiche da apportare all'algoritmo backpropagation per calcolare il gradiente:



Al fine di risolvere questo problema esistono due diversi approcci, che utilizzano entrambi la regola della catena:

$$\frac{dF}{dx} = \sum_{t=1}^Q \left[\frac{da(t)}{dx^T} \right]^T \times \frac{d^e F}{da(t)}$$

o

$$\frac{dF}{dx} = \sum_{t=1}^Q \left[\frac{d^e a(t)}{dx^T} \right]^T \times \frac{dF}{da(t)}$$

²⁹ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 14, pp. 10

dove l'apice "e" sta per "esplicita", ovvero la derivata con l'apice "e" è quella che rappresenta l'effetto diretto e che può essere calcolata tramite l'algoritmo backpropagation standard e dove ovviamente adesso sia x che $a(t)$ sono vettori. Per calcolare le derivate precedenti complete è tuttavia necessario utilizzare altre due equazioni:

$$\frac{da(t)}{dx^T} = \frac{d^e a(t)}{dx^T} + \frac{d^e a(t)}{da^T(t-1)} \times \frac{da(t-1)}{dx^T}$$

e

$$\frac{dF}{da(t)} = \frac{d^e F}{da(t)} + \frac{d^e a(t-1)}{da^T(t)} \times \frac{dF}{da(t+1)}$$

La prima e la terza delle ultime quattro equazioni costituiscono l'algoritmo RTLR, mentre la seconda e la quarta costituiscono l'algoritmo BPTT.

Volendo ora affrontare nel dettaglio gli algoritmi real-time-recurrent learning (RTRL) e backpropagation-through-time (BPTT) è opportuno in primo luogo considerare alcuni strati della LDDN come strati di input e altri come strati di output. In particolare, uno strato è considerato strato di input se ha una matrice che contiene i pesi da assegnare agli input o se contiene qualche ritardo per ognuna delle matrici dei pesi; è considerato invece strato di output se il suo output è comparato con un target durante l'allenamento o se è connesso a uno strato di input attraverso una matrice che ha qualche ritardo associato con esso. Per esempio, la nella rete riportata all'inizio del paragrafo vi sono 2 strati di input (1 e 2) e due strati di output (1 e 3). In secondo luogo si definisce U come l'insieme dei numeri che corrispondono agli strati di output e X come l'insieme dei numeri che corrispondono agli strati di input (esempio $U=(1,3)$ e $X=(1,2)$). Le equazioni per risolvere una LDDN generiche sono quelle sopra riportate, ovvero:

$$n^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} LW^{m,l}(d) a^l(t-d) + \sum_{l \in I_m} \sum_{d \in DI_{m,l}} IW^{m,l}(d) p^l(t-d) + b^m$$

e

$$a^m(t) = f^m(n^m(t))$$

Ad ogni istante di tempo esse vengono iterate in avanti attraverso gli strati, con m che viene incrementato tramite l'ordine di simulazione e t che viene incrementato da 1 a Q .

L'algoritmo RTRL si ottiene dalla generalizzazione delle equazioni precedenti:

$$\frac{dF}{dx} = \sum_{t=1}^Q \left[\frac{da(t)}{dx^T} \right]^T \times \frac{d^e F}{da(t)}$$

$$\frac{da(t)}{dx^T} = \frac{d^e a(t)}{dx^T} + \frac{d^e a(t)}{da^T(t-1)} \times \frac{da(t-1)}{dx^T}$$

Tenendo sempre presente che lo sviluppo per generare questo algoritmo è simile a quello utilizzato per l'algoritmo backpropagation, il primo passo da compiere è quello di generalizzare la prima equazione, calcolando i termini del gradiente grazie alla regola della catena,

$$\frac{dF}{dx} = \sum_{t=1}^Q \sum_{u \in U} \left[\left[\frac{da^u(t)}{dx^T} \right]^T \times \frac{d^e F}{da^u(t)} \right]$$

dove è stato aggiunto un termine nella somma per ogni strato di output (tuttavia, se l'indice di performance $F(x)$ non è una funzione esplicita di uno specifico output $a^u(t)$, allora la derivata esplicita sarà zero). Il secondo passo è quello di generalizzare la seconda equazione, sempre grazie alla regola della catena,

$$\frac{da(t)}{dx^T} = \frac{d^e a^u(t)}{dx^T} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u'}} \frac{d^e a^u(t)}{dn^x(t)^T} \times \frac{d^e n^x(t)}{da^{u'}(t-d)^T} \times \frac{da^{u'}(t-d)}{dx^T}$$

dove è stato preso in considerazione il fatto che in questo caso, non essendoci un solo ritardo nel sistema, bisogna tenere conto di ogni output e del numero di volte in cui ogni output è ritardato prima che esso sia un input per un altro strato (ecco la ragione delle prime due sommatorie). Per applicare questa equazione è necessario calcolare i termini:

$$\frac{d^e a^u(t)}{dn^x(t)^T} \times \frac{d^e n^x(t)}{da^{u'}(t-d)^T}$$

Per il secondo termine possiamo usare l'equazione seguente:

$$n_k^x(t) = \sum_{l \in L_x^f} \sum_{d' \in DL_{x,l}} \left[\sum_{i=1}^{S^l} lw_{k,i}^{x,l}(d') a_i^l(t - d') \right] + \sum_{l \in I_x} \sum_{d \in DI_{x,l}} \left[\sum_{i=1}^{R^l} iw_{k,i}^{x,l}(d') p_i^l(t - d') \right] + b_k^x$$

Dalla quale risulta:

$$\frac{d^e n_k^x(t)}{da_j^{u'}(t - d)} = lw_{k,i}^{x,l}(d)$$

Adesso, come è stato fatto per l'algoritmo backpropagation, si definisce la sensitività come:

$$s_{k,i}^{u,m}(t) \equiv \frac{d^e a_k^u(t)}{dn_i^m(t)}$$

E si utilizza per costruire una matrice del tipo:

$$S^{u,m}(t) = \frac{d^e a^u(t)}{dn^m(t)^T} = \begin{bmatrix} s_{1,1}^{u,m}(t) & s_{1,2}^{u,m}(t) & \cdots & s_{1,m}^{u,m}(t) \\ s_{2,1}^{u,m}(t) & s_{2,2}^{u,m}(t) & \cdots & s_{2,m}^{u,m}(t) \\ \vdots & \vdots & \ddots & \vdots \\ s_{S_u,1}^{u,m}(t) & s_{S_u,2}^{u,m}(t) & \cdots & s_{S_u,m}^{u,m}(t) \end{bmatrix}$$

A questo punto si ha:

$$\left[\frac{d^e a^u(t)}{dn^x(t)^T} \times \frac{d^e n^x(t)}{da^{u'}(t - d)^T} \right]_{i,j} = \sum_{k=1}^{S^x} s_{i,k}^{u,x}(t + d) \times lw_{k,j}^{x,u'}(d)$$

o analogamente, in forma matriciale:

$$\frac{d^e a^u(t)}{dn^x(t)^T} \times \frac{d^e n^x(t)}{da^{u'}(t - d)^T} = S^{u,x}(t) \times LW^{x,u'}(d)$$

La generalizzazione della seconda equazione iniziale può quindi essere riscritta come:

$$\frac{da(t)}{dx^T} = \frac{d^e a^u(t)}{dx^T} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u'}} S^{u,x}(t) \times LW^{x,u'}(d) \times \frac{da^{u'}(t-d)}{dx^T}$$

Molti dei termini di destra di quest'equazione tuttavia saranno uguali a zero e perciò non dovranno essere calcolati. Per questo motivo è utile introdurre una serie di insiemi che indicano per quali strati i pesi e le sensitività sono diversi da zero. Il primo contiene tutti gli strati di output che collegano ad uno strato di input x con almeno un ritardo diverso da zero:

$$E_{LW}^U(x) = \{u \in U | \exists (LW^{x,u}(d) \neq 0, d \neq 0)\}$$

Il secondo contiene gli strati di input che hanno una sensitività diversa da zero con uno strato specificato u :

$$E_S^X(u) = \{x \in X | \exists (S^{u,x} \neq 0)\}$$

Infine il terzo contiene gli strati che hanno una sensitività diversa da zero con uno strato specificato u :

$$E_S(u) = \{x | \exists (S^{u,x} \neq 0)\}$$

Utilizzando ora questi insiemi è possibile riscrivere ancora una volta l'equazione sommando solo gli elementi diversi da zero:

$$\frac{da(t)}{dx^T} = \frac{d^e a^u(t)}{dx^T} + \sum_{x \in E_S^X(u)} S^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} LW^{x,u'}(d) \times \frac{da^{u'}(t-d)}{dx^T}$$

Gli ultimi due passi che rimangono da compiere per ottenere l'algoritmo RTLRL nella sua completezza sono il calcolo delle matrici della sensitività $S^{u,m}(t)$ e le derivate esplicite $\frac{d^e a^u(t)}{dw}$.

Per calcolare gli elementi della matrice della sensitività si utilizza un approccio simile all'algoritmo backpropagation standard. Infatti le sensitività agli output della rete risultano:

$$s_{k,i}^{u,u}(t) = \frac{d^e a_k^u(t)}{dn_i^u(t)} = \begin{cases} f^u(n_i^u(t)) & \text{per } i = k \\ 0 & \text{per } i \neq k \end{cases}, u \in U$$

o, in forma matriciale:

$$S^{u,u}(t) = F^u(n^u(t))$$

dove $F^u(n^u(t))$ è definita come:

$$F^u(n^u(t)) = \begin{bmatrix} f^u(n_1^u(t)) & 0 & \cdots & 0 \\ 0 & f^u(n_2^u(t)) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f^u(n_{s^u}^u(t)) \end{bmatrix}$$

Le matrici $S^{u,m}(t)$ si calcolano attraverso la propagazione all'indietro nella rete, da ognuno degli strati di output, grazie alla seguente equazione:

$$S^{u,m}(t) = \left[\sum_{l \in E_s(u) \cap L_m^b} S^{u,l}(t) L W^{l,m}(0) \right] F^m(n^m(t)), u \in U$$

dove m è incrementato da u tramite l'ordine di backpropagation, L_m^b è l'insieme degli indici degli strati che sono direttamente connessi all'indietro con lo strato m e che non contengono ritardi nella connessione.

Infine, per calcolare le derivate esplicite $\frac{d^e a^u(t)}{dx^T}$ si utilizza ancora una volta la regola della catena. Così, per i pesi relativi agli input risulta:

$$\frac{d^e a_k^u(t)}{dw_{i,j}^{m,l}(d)} = \frac{d^e a_k^u(t)}{dn_i^m(t)} \times \frac{d^e n_i^m(d)}{dw_{i,j}^{m,l}(d)} = s_{k,i}^{u,m}(t) \times p_j^l(t-d)$$

che in forma vettoriale può essere scritta come:

$$\frac{d^e a^u(t)}{dw_{i,j}^{m,l}(d)} = s_i^{u,m}(t) \times p_j^l(t-d)$$

(dove a e s sono ora vettori) e in forma matriciale come:

$$\frac{d^e a^u(t)}{dvec(IW^{m,l}(d))^T} = [p^l(t-d)]^T \otimes S^{u,m}(t)$$

(dove \mathbf{a} e \mathbf{p} sono vettori, vec è un operatore che trasforma la matrice in un vettore prendendo le colonne e posizionandole una sopra l'altra e \otimes rappresenta il prodotto di Kronecker).

Con lo stesso procedimento si ottengono le derivate rispetto ai pesi relativi agli strati e rispetto ai bias:

$$\frac{d^e \mathbf{a}^u(t)}{d\text{vec}(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$$

$$\frac{d^e \mathbf{a}^u(t)}{d(\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t)$$

Per riassumere l'algoritmo RTLRL è possibile servirsi del seguente schema³⁰:

Real-Time Recurrent Learning Gradient

Initialize:
 $\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \mathbf{0}, t \leq 0$, for all $u \in U$,

For $t = 1$ to Q ,
 $U' = \emptyset, E_S(u) = \emptyset$ and $E_S^X(u) = \emptyset$ for all $u \in U$.

For m decremented through the BP order
For all $u \in U'$, if $E_S(u) \cap L_m^b \neq \emptyset$

$$\mathbf{S}^{u,m}(t) = \left[\sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t) \mathbf{LW}^{l,m}(0) \right] \mathbf{F}^m(\mathbf{n}^m(t))$$

add m to the set $E_S(u)$
if $m \in X$, add m to the set $E_S^X(u)$

EndFor u

If $m \in U$
 $\mathbf{S}^{m,m}(t) = \mathbf{F}^m(\mathbf{n}^m(t))$
add m to the sets U' and $E_S(m)$
if $m \in X$, add m to the set $E_S^X(m)$

EndIf m

EndFor m

For $u \in U$ incremented through the simulation order
For all weights and biases (\mathbf{x} is a vector containing all weights and biases)

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \text{vec}(\mathbf{IW}^{m,l}(d))^T} = [\mathbf{p}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \text{vec}(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t)$$

EndFor weights and biases

$$\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T} + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in D_{Lx,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial \mathbf{x}^T}$$

EndFor u

EndFor t

Compute Gradients

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^Q \sum_{u \in U} \left[\left[\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right]$$

³⁰ Da Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 14, pp 17.

Per quanto riguarda l'algoritmo BPTT invece esso si ottiene dalla generalizzazione delle equazioni precedenti qui riportate:

$$\frac{dF}{dx} = \sum_{t=1}^Q \left[\frac{d^e a(t)}{dx^T} \right]^T \times \frac{dF}{da(t)}$$

$$\frac{dF}{da(t)} = \frac{d^e F}{da(t)} + \frac{d^e a(t-1)}{da^T(t)} \times \frac{dF}{da(t+1)}$$

Come per l'algoritmo RTRL il primo passo da compiere è quello di generalizzare la prima equazione, calcolando i termini del gradiente grazie alla regola della catena:

$$\frac{dF}{dlw_{i,j}^{m,l}(d)} = \sum_{t=1}^Q \left[\sum_{u \in U} \sum_{k=1}^{S^u} \frac{dF}{da_k^u(t)} \times \frac{d^e a_k^u(t)}{dn_i^m(t)} \right] \frac{dn_i^m(t)}{dlw_{i,j}^{m,l}(d)}$$

(questo per i pesi relativi agli strati), dove u è uno strato di output, U è l'insieme di tutti gli strati di output e S^u è il numero di neuroni nello strato u . Inoltre risulta:

$$\frac{dn_i^m(t)}{dlw_{i,j}^{m,l}(d)} = a_j^l(t-d)^{31}$$

Definendo poi:

$$d_i^m(t) = \sum_{u \in U} \sum_{k=1}^{S^u} \frac{dF}{da_k^u(t)} \times \frac{d^e a_k^u(t)}{dn_i^m(t)}$$

I termini del gradiente per i pesi relativi agli strati possono essere scritti come

$$\frac{dF}{dlw_{i,j}^{m,l}(d)} = \sum_{t=1}^Q d_i^m(t) a_j^l(t-d)$$

³¹ Dall'equazione precedente

$$n^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} LW^{m,l}(d) a^l(t-d) + \sum_{l \in L_m} \sum_{d \in DI_{m,l}} IW^{m,l}(d) p^l(t-d) + b^m$$

Ricordando infine la definizione di sensitività $s_{k,i}^{u,m}(t) \equiv \frac{d^e a_k^u(t)}{dn_i^m(t)}$, gli elementi $d_i^m(t)$ possono essere scritti come:

$$d_i^m(t) = \sum_{u \in U} \sum_{k=1}^{s^u} \frac{dF}{da_k^u(t)} \times s_{k,i}^{u,m}(t)$$

o, in forma matriciale:

$$d^m(t) = \sum_{u \in U} [S^{u,m}(t)]^T \times \frac{dF}{da^u(t)}$$

(dove ora a e d sono vettori e dove $\frac{dF}{da^u(t)} = \left[\frac{dF}{da_1^u(t)} \frac{dF}{da_2^u(t)} \dots \frac{dF}{da_{s_u}^u(t)} \right]^T$). A questo punto è possibile scrivere il gradiente in forma matriciale:

$$\frac{dF}{dLW^{m,l}(d)} = \sum_{t=1}^Q d^m(t) \times [a^l(t-d)]^T$$

Seguendo lo stesso procedimento è possibile calcolare le derivate rispetto ai bias e ai pesi relativi agli input:

$$\frac{dF}{dIW^{m,l}(d)} = \sum_{t=1}^Q d^m(t) \times [p^l(t-d)]^T$$

$$\frac{dF}{db^m} = \sum_{t=1}^Q d^m(t)$$

Continuando con il procedimento di generalizzazione delle equazioni iniziali e generalizzando la seconda utilizzando ancora una volta la regola della catena si ottiene:

$$\begin{aligned} \frac{dF}{da^u(t)} &= \frac{d^e F}{da^u(t)} \\ &+ \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u}} \left[\frac{d^e a^{u'}(t+d)}{dn^x(t+d)^T} \times \frac{d^e n^x(t+d)}{da^u(t)^T} \right]^T \times \frac{dF}{da^{u'}(t+d)} \end{aligned}$$

La ragione delle tre sommatorie risiede nel fatto che adesso, al contrario di quanto avveniva in precedenza, è necessario tenere conto di ogni output della rete, di come questo è connesso all'indietro attraverso l'input della rete e del numero di volte con il quale ogni output è ritardato prima che diventi un input. Questa equazione va poi aggiornata all'indietro nel tempo, ed infatti t varia da Q a 1 . Se poi si considera la matrice tra parentesi nel membro di destra, è possibile scrivere:

$$\frac{d^e a^{u'}(t+d)}{dn^x(t+d)^T} \times \frac{d^e n^x(t+d)}{da^u(t)^T} = S^{u',x}(t+d) \times LW^{x,u}(d)^{32}$$

A questo punto, l'equazione diventa:

$$\frac{dF}{da^u(t)} = \frac{d^e F}{da^u(t)} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u}} [S^{u',x}(t+d) \times LW^{x,u}(d)]^T \times \frac{dF}{da^{u'}(t+d)}$$

Inoltre, come avveniva per l'algoritmo RTRL, molti dei termini di destra saranno uguali a zero e non necessiteranno di essere calcolati. Per questo motivo, analogamente a quanto già fatto in precedenza, si introducono gli insiemi seguenti:

$$E_{LW}^X(u) = \{x \in X | \exists (LW^{x,u}(d) \neq 0, d \neq 0)\}$$

$$E_S^U(x) = \{u \in U | \exists (S^{u,x} \neq 0)\}$$

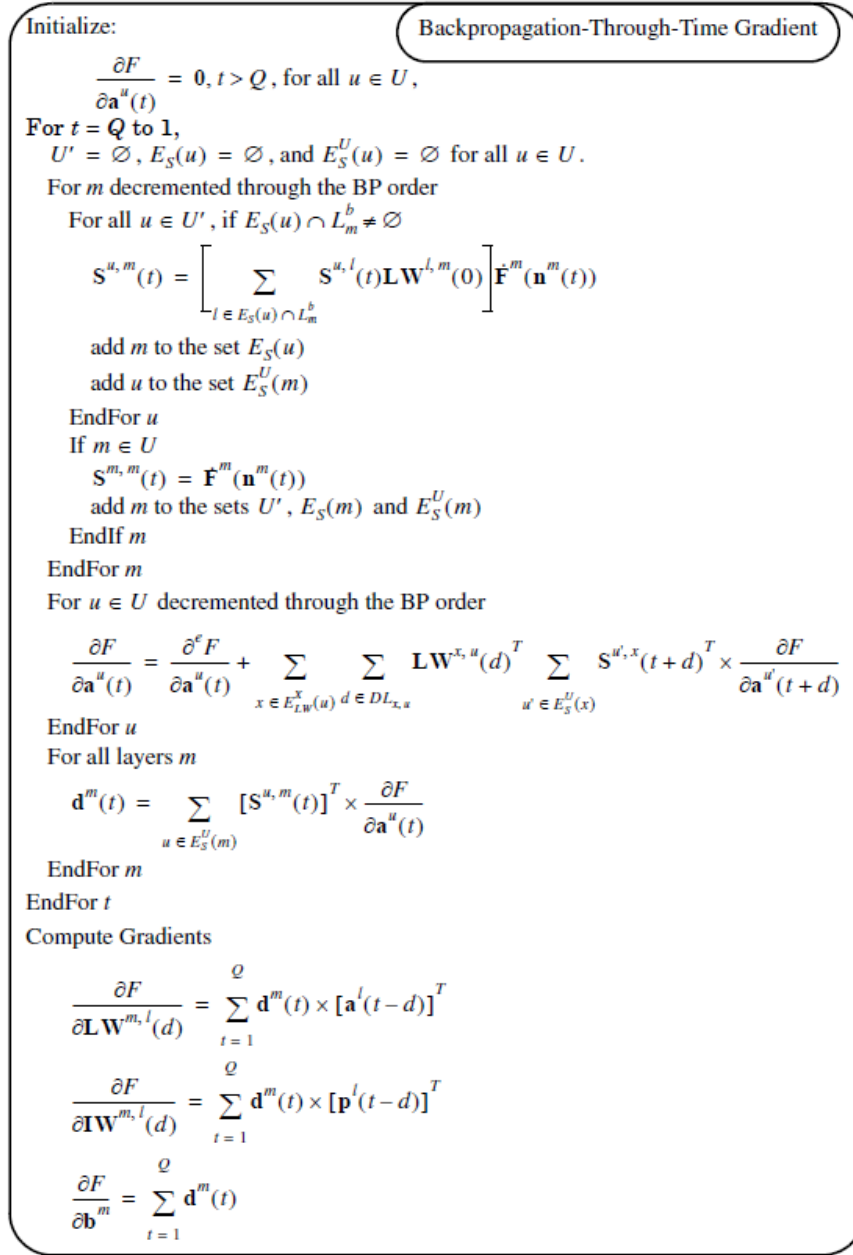
Il primo insieme contiene tutti gli strati di input che hanno una connessione da uno strato di output u con almeno un ritardo diverso da zero; il secondo contiene gli strati di output che hanno una sensitività diversa da zero con lo strato di input x . Adesso è possibile riscrivere l'equazione nella sua forma finale:

$$\begin{aligned} \frac{dF}{da^u(t)} &= \frac{d^e F}{da^u(t)} \\ &+ \sum_{x \in E_{LW}^X(u)} \sum_{d \in DL_{x,u}} LW^{x,u}(d)^T \sum_{u' \in E_S^U(x)} S^{u',x}(t+d)^T \times \frac{dF}{da^{u'}(t+d)} \end{aligned}$$

³² Questa equazione deriva dall'equazione dell'algoritmo RTRL

$$\frac{d^e a^u(t)}{dn^x(t)^T} \times \frac{d^e n^x(t)}{da^{u'}(t-d)^T} = S^{u,x}(t) \times LW^{x,u'}(d)$$

Per riassumere l'algoritmo BPTT è possibile servirsi del seguente schema³³:



Gli algoritmi RTRL e BPTT sono quindi due metodologie che permettono di calcolare il gradiente per le reti neurali dinamiche e, con qualche variazione, anche la matrice jacobiana necessaria per esempio nell'algoritmo Levenberg- Marquardt. Una volta che questi elementi sono stati calcolati è possibile iniziare ad allenare la rete anche se, per varie ragioni che non verranno trattate, l'allenamento di questo tipo di reti è molto più complesso rispetto a quello di una rete statica, e per questo

³³ Da Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 14, pp. 26

effettuato solo ed esclusivamente tramite programmi di calcolo computerizzati che permettono di gestire con rapidità l'enorme quantità di dati.

2.10 L'allenamento della rete neurale: tecniche pratiche

In quest'ultimo paragrafo del capitolo verranno trattate le tecniche pratiche da utilizzare nell'allenamento della rete, o meglio in tutto il processo di creazione della rete. Questo processo si suddivide in tre parti: la fase pre-allenamento, la fase dell'allenamento e la fase post-allenamento. In particolare, quando ci si trova ad affrontare e risolvere un problema attraverso una rete neurale gli step che devono essere affrontati sono la raccolta e l'elaborazione dei dati, la scelta dell'architettura della rete, la scelta dell'algoritmo per il suo allenamento, l'inizializzazione dei parametri e l'allenamento vero e proprio, l'analisi della performance ed infine il suo utilizzo. Ovviamente questi step si susseguono in un processo iterativo fino a che la rete non produce dei risultati soddisfacenti.

- Fase pre-allenamento

In questa fase sono necessari tre step molto importanti che di solito influiscono pesantemente sulla performance della rete: la raccolta dei dati, la loro rielaborazione e la scelta del tipo e dell'architettura della rete stessa.

Per quanto riguarda la raccolta dei dati, come già affermato in precedenza, quelli sui quali la rete viene allenata devono necessariamente abbracciare tutto il range dello spazio degli input sul quale verrà utilizzata la rete, altrimenti la performance non sarà soddisfacente. Tuttavia, a meno di non aver a che fare con problemi molto semplici, capire se lo spazio degli input è stato adeguatamente campionato nei dati con i quali è stata allenata la rete non è una cosa affatto semplice, poiché molto spesso i dati raccolti sono complessi e dipendenti tra loro. Inoltre, dopo aver raccolto i dati, essi vanno suddivisi in training set, validation set e test set, e tutti questi insiemi devono essere rappresentativi di tutto l'insieme dei dati. Al tal fine la procedura migliore da adottare è quella di selezionare casualmente i dati per ogni insieme dal totale dei dati disponibili, anche se in una fase successiva è sempre opportuno controllare i tre insiemi per rilevare delle eventuali incongruenze.

Lo step successivo alla raccolta dei dati è quello della rielaborazione degli stessi, grazie al quale è possibile rendere l'allenamento della rete molto più semplice. Gli elementi più comuni di questo passaggio sono la normalizzazione degli input, l'applicazione di trasformazioni non lineari, la codifica dei target e tutti quei processi utili a risolvere le problematiche connesse ai dati mancanti. La normalizzazione degli input può essere effettuata in due modi. Il primo normalizza i dati in modo che siano ricompresi in un range fisso (generalmente -1;1) attraverso la seguente equazione:

$$p^n = 2(p - p^{min})./(p^{max} - p^{min}) - 1$$

dove p^{min} è il vettore che contiene il minimo valore di ogni elemento dei vettori di input nell'insieme dei dati, p^{max} quello che contiene i valori massimi, ./ un rapporto elemento per elemento tra due vettori e p^n è il vettore normalizzato risultante. Il secondo normalizza i dati in modo che abbiano media 0 e varianza 1 tramite l'equazione:

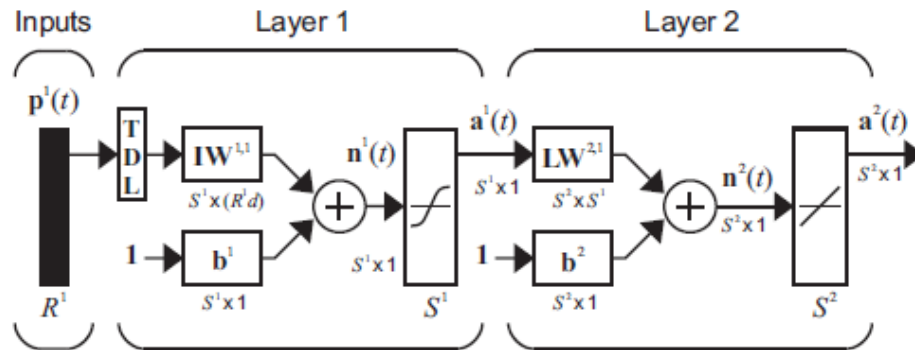
$$p^n = (p - p^{mean})./p^{std}$$

dove p^{mean} è la media dei vettori di input nell'insieme dei dati e p^{std} è il vettore che contiene la deviazione standard di ogni elemento dei vettori di input. Generalmente la normalizzazione viene fatta anche sui target. L'altra operazione che viene effettuata molto spesso è rappresentata dall'applicazione di trasformazioni non lineari che tuttavia, al contrario della normalizzazione che può essere sempre fatta, può essere applicata solo in casi specifici, dati per esempio dalle variabili economiche, per le quali, mostrando spesso una dipendenza logaritmica, è utile prendere il logaritmo degli input iniziali. La codifica dei target invece è particolarmente utile quando gli input o i target assumono solo valori discreti. Ciò avviene per esempio nei problemi di classificazione, dove se per esempio si ha un problema di riconoscimento di pattern nel quale si hanno solo quattro classi, ci possono essere fondamentalmente 3 modi per codificare i target: il primo è assegnare alle classi uno scalare (es. 1, 2, 3, 4); il secondo è utilizzare un target bidimensionale che rappresenti con un codice binario le quattro classi (es. (0,0), (0,1), (1,0), (1,1)); il terzo è utilizzare un target quadridimensionale nel

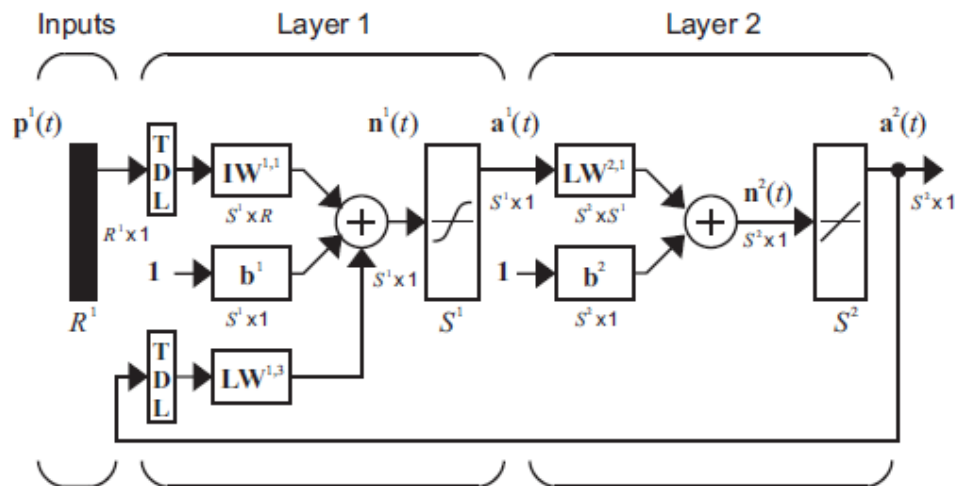
quale si attiva solo un neurone per volta (es. (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)). Nel compiere questa operazione è anche necessario tenere conto della funzione di trasferimento che si utilizza nello strato di output della rete. Per i problemi di classificazione infatti si utilizza spesso una funzione di tipo log-sigmoid o tangent-sigmoid e nel secondo caso si tende a assegnare ai target un valore di -1 e 1, che sono gli asintoti della funzione. Tuttavia questo potrebbe causare vari problemi all'algoritmo di allenamento della rete, e per questo da caso a caso è necessario scegliere adeguatamente i valori tenendo conto anche della funzione di trasferimento utilizzata. Talvolta infine nella raccolta dei dati può accadere che vi siano dei dati mancanti, soprattutto quando si ha a che fare con i dati di tipo economico, e per questo motivo sono necessari degli accorgimenti per risolvere il problema. Uno di questi è per esempio quello di rimpiazzare il dato mancante con il valore medio di quel particolare valore di input e allo stesso tempo aggiungere una flag al vettore degli input che indica che il dato originale è stato sostituito con un valore medio. Allo stesso modo, se l'elemento mancante appartiene ai target, è possibile modificare l'indice di performance in modo che l'errore associato a quel target mancante non venga considerato.

L'ultimo step della fase pre-allenamento è la scelta del tipo e dell'architettura della rete neurale. Ovviamente questa scelta dipende strettamente dal tipo di problema che ci si appresta a risolvere e coinvolge non solo la scelta tra le architetture di base ma anche tutte quelle caratteristiche come numero di strati, numero di neuroni in ogni strato, tipo di funzione di trasferimento da utilizzare, ecc. Se ad esempio è necessario risolvere un problema di classificazione, molto spesso si utilizza come architettura di base il perceptrone multistrato con uno strato nascosto e una funzione di trasferimento di tipo tansig nell'hidden layer e una sigmoid function nello strato di output. Se invece è necessario risolvere un problema di previsione (come avviene per una serie storica di tipo economico) sono necessarie le reti neurali dinamiche. In questo caso l'architettura più semplice da utilizzare è la cosiddetta focused time-delay neural network, nella quale il ritardo si ha solamente nello strato di input di una rete multistrato feedforward. Questo tipo di rete,

mostrata nella figura seguente³⁴, ha il vantaggio di poter essere allenata con l'algoritmo backpropagation statico, poiché la tapped-delay-line può essere



rimpiazzata con un vettore esteso comprendente i valori ritardati degli input. Un altro tipo di architettura molto utilizzata in questo caso è anche la rete di tipo NARX (Nonlinear AutoRegressive model with eXogenous input). Come si vede dalla figura³⁵ anche questo tipo di rete può essere allenata con l'algoritmo backpropagation statico, ed in questo caso le due tapped-delay-line possono essere



rimpiazzate con due vettori estesi che comprendono i valori ritardati degli input e dei target. Una volta chiara l'architettura di base della rete, il passo successivo da compiere consiste nello scegliere per esempio il numero di strati. In questo caso la procedura standard è quella, per i problemi di classificazione, di iniziare con un solo strato nascosto, per poi passare a due strati nascosti nel caso in cui la performance della rete non sia adeguata. È possibile utilizzare anche più di due

³⁴ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 10

³⁵ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 11

strati nascosti, ma ciò avviene solo in rari casi, dal momento che più hidden layer vi sono più l'allenamento della rete risulterà lento e complesso. Un'altra scelta cruciale è quella relativa al numero di neuroni in ogni strato: per quanto riguarda lo strato di output esso avrà un numero di neuroni pari alla dimensione del vettore dei target, mentre il numero di neuroni negli strati nascosti dipenderà strettamente dalla complessità del problema. La procedura standard consiste comunque con l'utilizzare un numero di neuroni più alto rispetto a quanto necessario per poi utilizzare una procedura come l'early stopping o la regularization per prevenire la problematica dell'overfitting. Un'ultima problematica da affrontare in questa fase è la scelta della dimensione del vettore di input. Qualche volta quest'ultima risulta decisamente semplice in quanto determinata esclusivamente dai dati su cui si va ad allenare la rete, ma altre volte risulta più complicata, perché tra i dati ve ne possono essere alcuni ridondanti o irrilevanti, che devono essere riconosciuti ed eliminati al fine di ridurre la probabilità di overfitting durante l'allenamento della rete.

- Fase dell'allenamento

La fase di allenamento della rete neurale comprende fondamentalmente quattro step, dati da scelte relative a: metodi per l'inizializzazione dei parametri, algoritmo di allenamento, criteri per lo stop dell'allenamento e indice di performance. Per quanto riguarda i metodi per l'inizializzazione dei parametri, sebbene ve ne siano molti, il più utilizzato è senza dubbio quello che consiste nell'inizializzare pesi e bias della rete a piccoli valori casuali, per esempio a valori uniformemente distribuiti tra -0.5 e 0.5 se gli input sono normalizzati tra -1 e 1. La scelta dell'algoritmo con cui allenare la rete invece per la maggior parte delle volte ricade su uno degli algoritmi visti in precedenza e generalmente dipende dal numero dei parametri coinvolti nella rete. Per le reti multistrato per esempio si usano gli algoritmi basati sul gradiente o sulla matrice Jacobiana, e per quelle che hanno fino a poche centinaia di pesi e bias l'algoritmo Levenberg- Marquardt si rileva spesso come il più veloce ed efficiente, mentre se il numero di parametri aumenta fino a qualche migliaia anche questo algoritmo diventa inefficiente e quindi è necessario

utilizzarne altri di più complessi. Inoltre, dal momento che non è possibile che in ogni allenamento di una rete neurale l'errore converga esattamente a zero, sono necessari dei criteri per fermare l'allenamento stesso. Anche in questo caso ve ne sono dei più disparati: il più semplice consiste nel fermare l'allenamento dopo un numero fisso (generalmente abbastanza elevato) di iterazioni; un altro criterio consiste nel controllare la norma del gradiente dell'indice di performance e fermare l'allenamento al momento che quest'ultima non raggiunge una certa soglia (questo metodo si basa sul fatto che il gradiente sarà zero nel punto di minimo, quindi questo criterio fermerà l'allenamento in un punto vicino al punto di minimo); l'ultimo criterio, e forse il più utilizzato, è il già discusso early stopping, che fermando l'allenamento non appena l'errore sul validation set cresce per un numero prefissato di iterazioni, contribuisce ad evitare il problema dell'overfitting e permette un notevole risparmio in termini di calcolo. L'ultimo step della fase di allenamento consiste nella scelta dell'indice di performance. Fino ad ora è stato trattato un solo indice di performance, ovvero nell'errore quadratico medio: esso, quando tutti gli input del training set hanno la stessa probabilità di verificarsi, è dato da:

$$F(x) = \frac{1}{QS^M} \sum_{q=1}^Q (t_q - a_q)^T (t_q - a_q)$$

$$F(x) = \frac{1}{QS^M} \sum_{q=1}^Q \sum_{i=1}^{S^M} (t_{i,q} - a_{i,q})^2$$

Il fattore di scala al di fuori della sommatoria non incide sui pesi ottimi, perciò l'indice di performance dato dalla somma degli errori al quadrato produce gli stessi parametri di quello dato dall'errore quadratico medio. Nonostante questo tipo di indice sia il più utilizzato, ve ne sono anche altri che a seconda dei casi risultano più adeguati. Uno di questi è l'errore medio assoluto, dato da:

$$F(x) = \frac{1}{QS^M} \sum_{q=1}^Q \sum_{i=1}^{S^M} |t_{i,q} - a_{i,q}|$$

Il vantaggio che può avere questo indice di performance è quello di essere meno sensibile a qualche errore elevato nell'insieme dei dati, e quindi allo stesso tempo più robusto a eventuali anomalie rispetto all'errore quadratico medio. Un altro indice molto utilizzato, soprattutto nei problemi di classificazione in cui i target assumono solo valori discreti è la cross entropy, definita come:

$$F(x) = - \sum_{q=1}^Q \sum_{i=1}^{S^M} t_{i,q} \ln \frac{a_{i,q}}{t_{i,q}}$$

dove si è assunto che i valori del target possano essere 1 e 0. Per concludere poi la fase dell'allenamento è necessario affermare che spesso un singolo allenamento della rete non produce la performance ottima. Per questo motivo è utile far ripartire l'allenamento più volte a varie condizioni iniziali e scegliere la rete che produce la performance migliore (solitamente un numero di allenamenti da cinque a dieci produce sempre un minimo globale). Un altro metodo molto interessante da utilizzare nella pratica è il cosiddetto committee of networks: si portano a termine varie sessioni di allenamento scegliendo il validation set e i parametri iniziali casualmente; successivamente si utilizzano tutte le diverse reti che sono state allenate per formare e dare come unico risultato un solo input congiunto (che può essere per esempio una media semplice degli output ottenuti dalle varie reti oppure, in un problema di classificazione, la classe che è stata scelta dalla maggior parte delle reti allenate).

- Fase post-allenamento

Nella fase post-allenamento è necessario controllare l'allenamento della rete per capire se quest'ultimo ha avuto successo oppure no. A tal fine esistono diverse metodologie a seconda del problema per il quale la rete è stata creata. Se per esempio è stata realizzata per risolvere un problema di approssimazione di una funzione, uno strumento molto utile è rappresentato dalla regressione tra gli output della rete e i target. La retta di regressione avrà la forma:

$$a_q = mt_q + c + \varepsilon_q$$

dove m e c sono l'inclinazione e l'intercetta, t_q è un valore target e a_q è un valore di output della rete. I termini della regressione possono essere calcolati come:

$$\hat{m} = \frac{\sum_{q=1}^Q (t_q - \bar{t})(a_q - \bar{a})}{\sum_{q=1}^Q (t_q - \bar{t})^2}$$

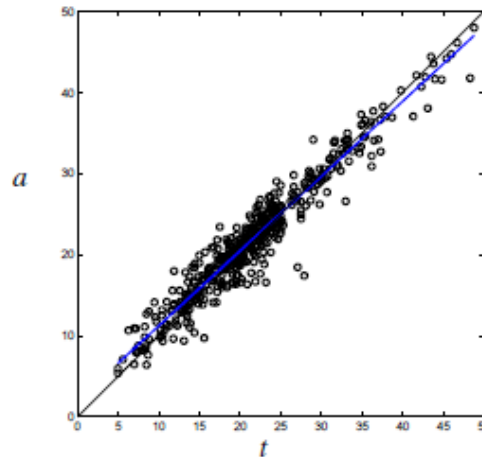
$$\hat{c} = \bar{a} - \hat{m}\bar{t}$$

dove:

$$\bar{a} = \frac{1}{Q} \sum_{q=1}^Q a_q$$

$$\bar{t} = \frac{1}{Q} \sum_{q=1}^Q t_q$$

Un esempio di regressione lineare è evidenziato dalla figura seguente³⁶, nella quale la retta blu rappresenta appunto la regressione lineare, la linea nera rappresenta il luogo dei punti in cui $a_q = t_q$ e i cerchi rappresentano i dati. Come si può facilmente notare la corrispondenza in questo caso è piuttosto buona, anche se non



perfetta. Inoltre, un grafico di questo tipo permette in modo molto immediato di andare a ricercare i possibili dati anomali: se infatti tutti i dati cadono nei pressi della retta di regressione e alcuni invece cadono molto lontani dalla stessa, essi potrebbero essere anomali, ed è quindi opportuno controllarli più attentamente. Un

³⁶ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 20

altro strumento che viene spesso utilizzato in questo tipo di problemi è il coefficiente R, un coefficiente di correlazione tra t_q e a_q calcolato come:

$$R = \frac{\sum_{q=1}^Q (t_q - \bar{t})(a_q - \bar{a})}{(Q - 1)s_t s_a}$$

dove

$$s_t = \sqrt{\frac{1}{Q - 1} \sum_{q=1}^Q (t_q - \bar{t})^2}$$

$$s_a = \sqrt{\frac{1}{Q - 1} \sum_{q=1}^Q (a_q - \bar{a})^2}$$

Questo coefficiente varia tra -1 e 1 e affinché la rete sia ben allenata è necessario che sia il più possibile vicino al valore 1 (caso in cui tutti i punti che rappresentano i dati cadono esattamente sulla retta di regressione lineare). In ogni caso l'analisi di regressione può essere effettuata sui tre insiemi di dati (training, validation, test) separatamente e sul totale dei dati: in tal modo per esempio è possibile avere un'indicazione di overfitting se la rete dà buoni risultati su training set e scarsi risultati sul validation set (in tal caso può risultare utile ridurre il numero dei neuroni e/o degli strati della rete e far ricominciare l'allenamento); analogamente è possibile avere un'indicazione circa l'estrapolazione della rete se genera buoni risultati sul training set e sul validation set e scarsi risultati sul test set (in tal caso, come è già stato affermato in precedenza, l'unico modo per risolvere il problema è quello di utilizzare più dati nell'allenamento della rete).

Nei problemi di classificazione invece si utilizza spesso uno strumento chiamato confusion matrix, una tabella nella quale le colonne rappresentano le classi target e le righe le classi di output. Un esempio di questo tipo di matrice è il seguente³⁷:

³⁷ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 22

Confusion Matrix			
Output Class	1	2	
	47 22.0%	1 0.5%	97.9% 2.1%
	4 1.9%	162 75.7%	97.6% 2.4%
	92.2% 7.8%	99.4% 0.6%	97.7% 2.3%
	1	2	
	Target Class		

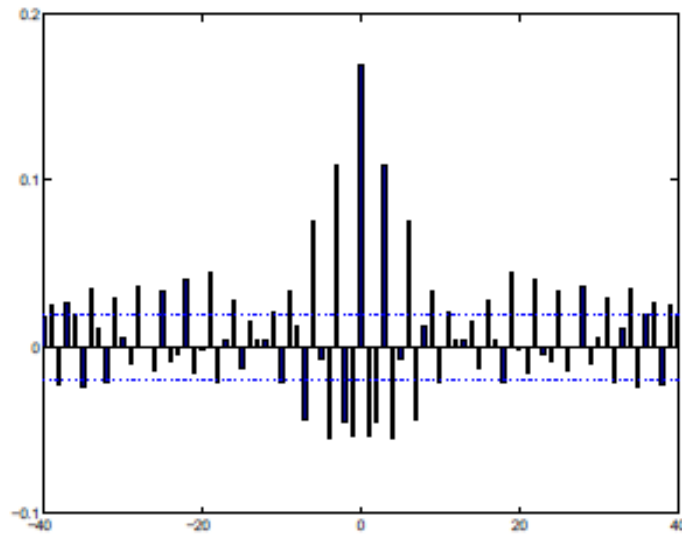
Essa mostra come dei 214 dati 47 sono stati classificati correttamente in classe 1 e 162 in classe 2 mentre quattro sono stati classificati in modo errato in classe 2 e uno in classe 1. In questo modo, tenendo conto anche dei valori percentuali è possibile avere un'idea esatta di quella che è stata la performance della rete. Nei problemi di previsione invece, una volta che la rete è stata allenata, vi sono essenzialmente due tipi di correlazione da controllare: quella tra errori di previsione nel tempo e quella tra errori di previsione e sequenza di input. Gli errori di previsione infatti non dovrebbero essere correlati nel tempo e non dovrebbero essere correlati con la sequenza degli input. Al fine di controllare il primo tipo di correlazione si utilizza la seguente funzione di autocorrelazione:

$$R_e(\tau) = \frac{1}{Q - \tau} \sum_{t=1}^{Q-\tau} e(t)e(t + \tau)$$

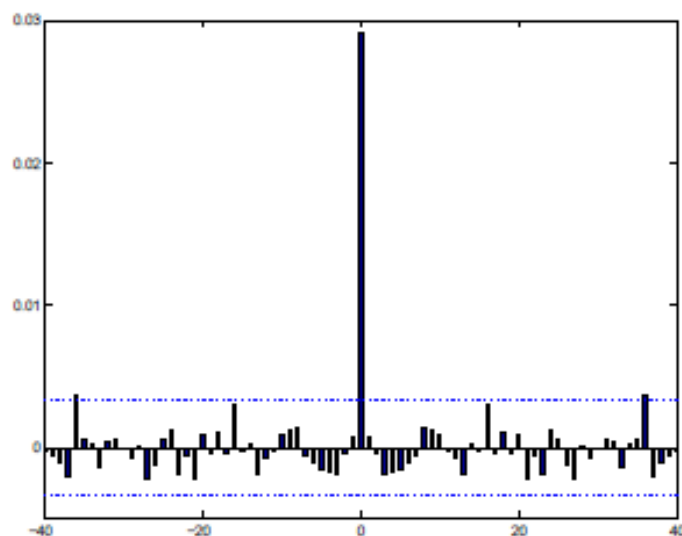
Se gli errori di previsione non sono correlati, ovvero sono di tipo white noise, il valore di $R_e(\tau)$ risulterà essere molto vicino allo zero eccetto per $\tau = 0$. Per stabilire se questo valore è vicino allo zero solitamente si sceglie un intervallo di confidenza del 95% usando il range:

$$\frac{-2R_e(0)}{\sqrt{Q}} < R_e(\tau) < \frac{2R_e(0)}{\sqrt{Q}}$$

Se quindi il valore di $R_e(\tau)$ soddisfa quest'ultima equazione è possibile affermare che l'errore $e(t)$ è di tipo white noise. Se per esempio una rete non è stata adeguatamente allenata, il grafico di $R_e(\tau)$ sarà del tipo mostrato in figura³⁸, dove si nota come la funzione non rimanga completamente all'interno del range dato dall'equazione precedente e rappresentato nel grafico dalle linee orizzontali



tratteggiate. Una rete ben allenata quindi dovrebbe mostrare un grafico in cui la funzione ricade completamente all'interno del range di riferimento, eccetto per $\tau = 0$, come accade nel caso seguente³⁹:



³⁸ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 25

³⁹ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 25

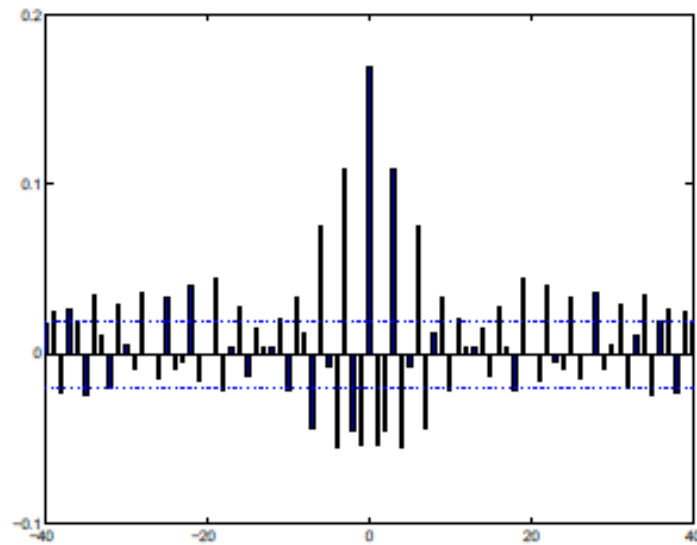
Per quanto riguarda invece il secondo tipo di correlazione, si utilizza una funzione di correlazione incrociata:

$$R_{pe}(\tau) = \frac{1}{Q - \tau} \sum_{t=1}^{Q-\tau} p(t)e(t + \tau)$$

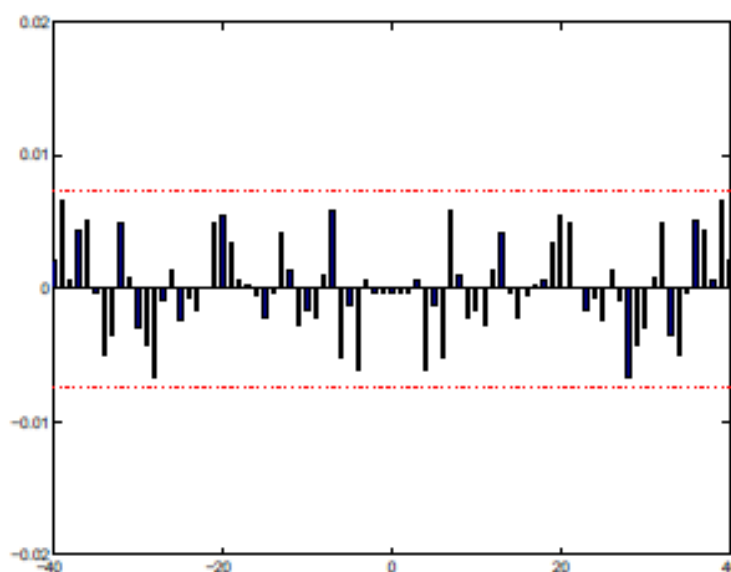
Analogamente a quanto avveniva in precedenza, se gli errori di previsione non sono correlati con la sequenza degli input il valore di $R_{pe}(\tau)$ tenderà ad essere vicino allo zero per ogni valore di τ . Altrettanto analogamente, per determinare se questo valore è vicino allo zero si sceglie un intervallo di confidenza del 95% usando il range:

$$-\frac{2\sqrt{R_e(0)}\sqrt{R_p(0)}}{\sqrt{Q}} < R_{pe}(\tau) < \frac{2\sqrt{R_e(0)}\sqrt{R_p(0)}}{\sqrt{Q}}$$

Come fatto in precedenza quindi si riportano i grafici della funzione di due reti neurali: la prima, non adeguatamente allenata, ha una funzione che non ricade completamente nel range determinato dall'equazione precedente; la seconda, adeguatamente allenata, ha invece una funzione che ricade completamente all'interno del range, per ogni valore di τ ⁴⁰:



⁴⁰ Hagan M. T., Debuth H. B., Beale M.H., De Jeus O., 2014. Neural Network Design, 2nd edition, cap. 22, pp. 26-27



Se si utilizzano reti dinamiche di tipo NARX, una correlazione tra errori di previsione e sequenza degli input potrebbe significare semplicemente che la lunghezza delle tapped delay line è troppo ridotta, e quindi un possibile modo per incrementare la performance della rete potrebbe essere quello di aumentare la lunghezza della stessa.

Un ultimo importante punto della fase post-allenamento è l'analisi del test set. Esso infatti, com'è noto, dà indicazioni circa la performance che la rete avrà una volta che verrà utilizzata. Se dopo che la rete è stata allenata la performance sul test set non risulta adeguata, i motivi vanno ricercati tra i seguenti: la rete ha raggiunto un minimo locale (anziché globale); la rete non ha abbastanza neuroni; la rete presenta un problema di overfitting; la rete sta estrapolando. Il primo problema è quasi sempre risolvibile allenando nuovamente la rete con diversi pesi iniziali per un numero di volte che può variare da cinque a dieci: in questo modo, come già detto in precedenza, la rete con il minimo errore rappresenterà un minimo globale. Gli altri tre problemi generalmente si individuano analizzando i tre insiemi di dati. Se per esempio l'errore sul validation set è molto più alto di quello sul training set significa che la rete ha un problema di overfitting: in questo caso, oltre a quanto detto in precedenza, è consigliabile utilizzare un algoritmo di allenamento più lento. Se invece gli errori su tutti gli insiemi di dati sono simili ma troppo elevati

significa che la rete non è abbastanza potente e in questo caso è necessario aumentare il numero di neuroni negli strati nascosti. Se infine l'errore sul training set è simile a quello del validation set, ma entrambi sono molto inferiori a quello sul test set, significa che la rete sta estrapolando e che quindi è stata usata per analizzare dati al di fuori del training e del validation set: in questo caso, come ricordato più volte, l'unico modo per incrementare la performance della rete è quello di utilizzare un maggior numero di dati per allenare la rete.

Per concludere è opportuno affermare che tutti gli strumenti presentati in quest'ultimo paragrafo, sebbene risultino molto utili per affrontare i problemi più classici che si possono presentare nella creazione e nell'utilizzo di una rete neurale, non possono essere considerati come strumenti universali: utilizzando infatti le reti neurali si incontrano problemi che devono essere risolti caso per caso, tali da non poter essere inquadrati o ricondotti totalmente a uno schema generale.

Capitolo 3: Caso Pratico

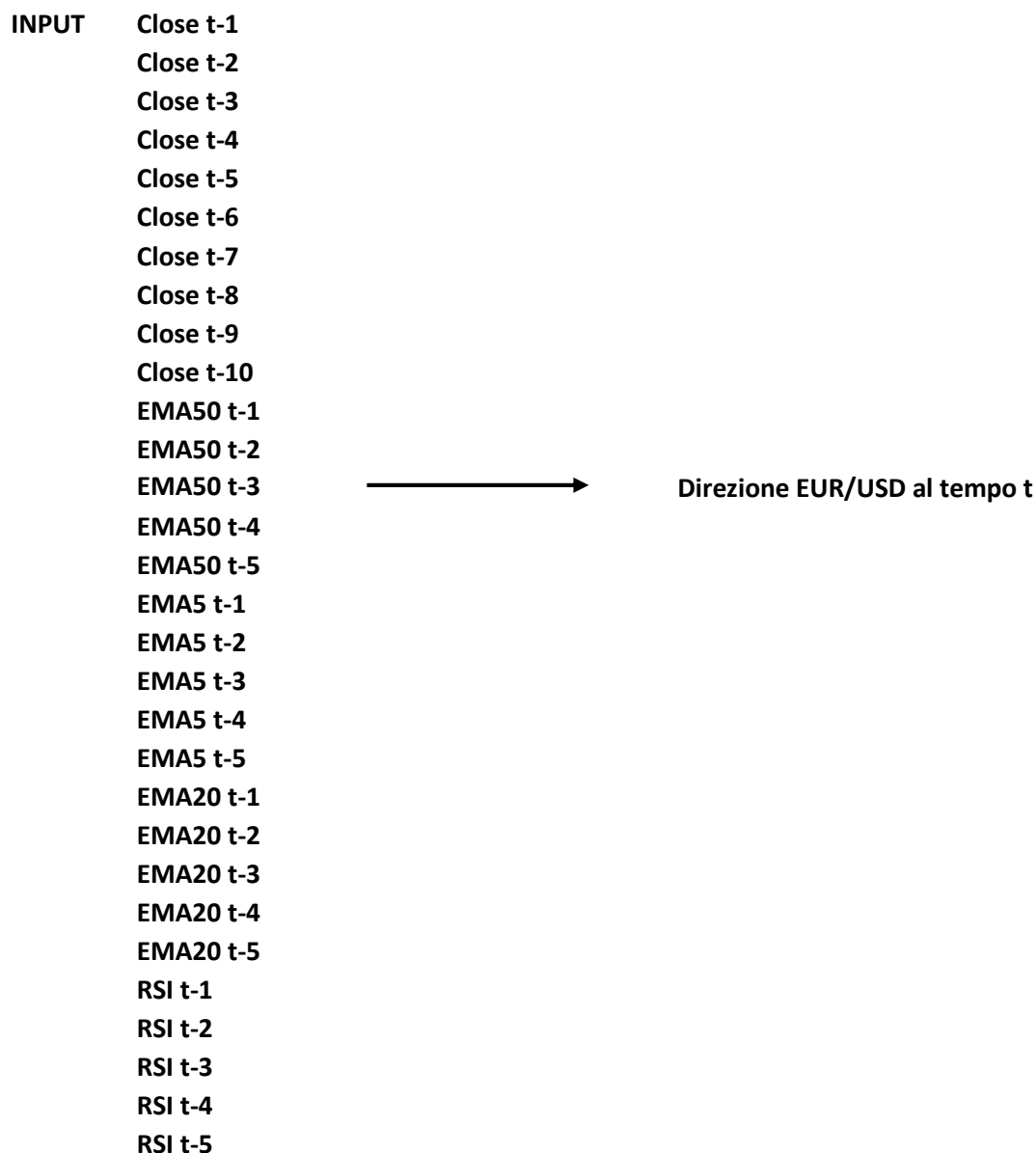
3.1 Introduzione

In quest'ultimo capitolo verrà presentata una metodologia di trading creata attraverso alcune delle reti neurali viste in precedenza, che sarà poi applicata al tasso di cambio EUR/USD per cercare di capire se un trading system di questo tipo possa generare buoni risultati. Il periodo di riferimento preso in considerazione per allenare le reti neurali copre nel complesso circa 3500 giornate borsistiche e quindi va dal 08/10/2003 al 7/03/2017. Il periodo nel quale invece il sistema di trading è stato utilizzato in maniera simulata sul mercato va dal 08/03/2017 al 18/09/2017. Per quanto riguarda i dati utilizzati essi sono stati estrapolati dal database Datastream di Thomson Reuters fornito gratuitamente agli studenti dell'Università di Pisa per la fase di apprendimento delle reti mentre sono stati scaricati manualmente dal software ProRealTime per la fase di simulazione sul mercato. Per la creazione e l'allenamento delle reti infine, così come per la realizzazione degli strumenti di supporto grafico per l'analisi dei risultati, sono stati utilizzati i software Matlab (Neural Network Toolbox) ed Excel.

3.2 Tipo di reti neurali utilizzate nel trading system

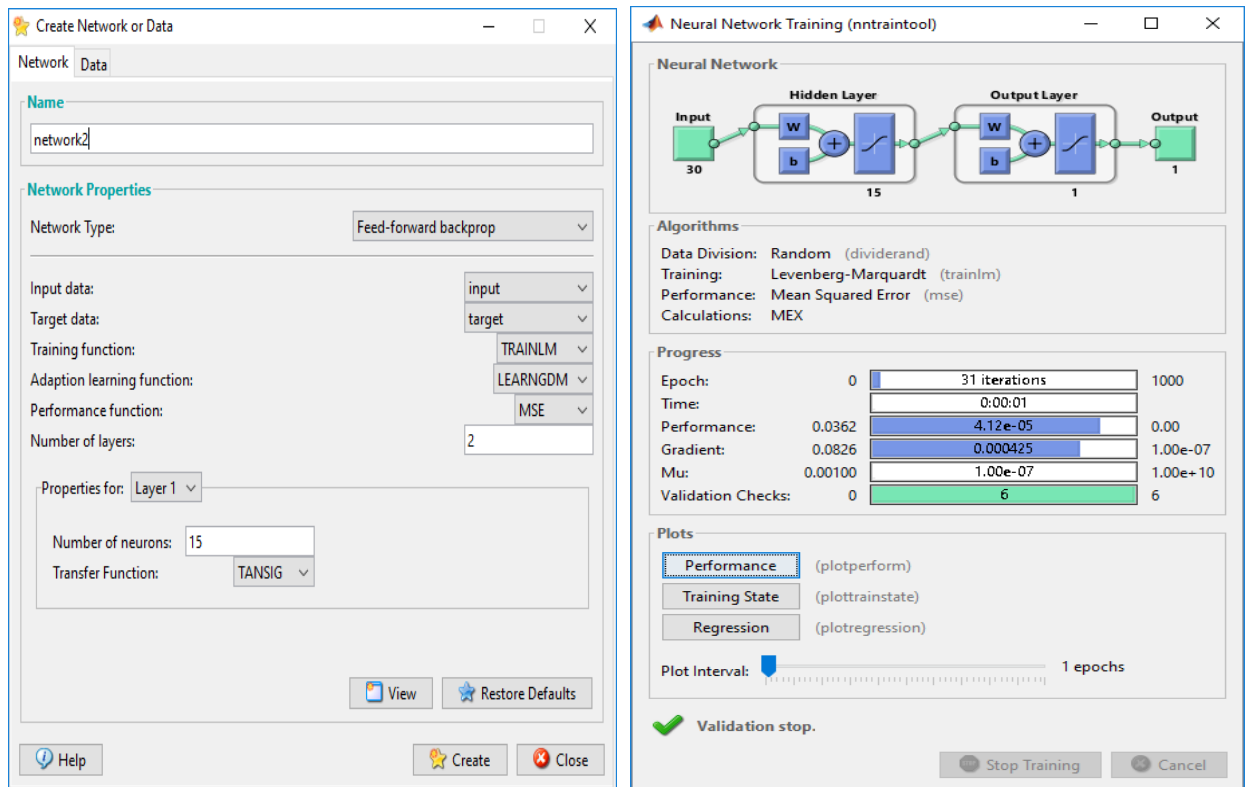
Andando ora ad analizzare nel dettaglio le caratteristiche delle reti neurali utilizzate per creare il sistema di trading è necessario per prima cosa specificare come siano state utilizzate tre diverse reti neurali, denominate rispettivamente net2, net3 e net4, che si differenziano esclusivamente per il periodo di riferimento dei dati con i quali sono state allenate. La rete net2 infatti ha un periodo di riferimento più breve (1500 giornate borsistiche), la rete net3 ha un periodo di riferimento medio (2500 giornate borsistiche) e la rete net4 ha un periodo di riferimento più lungo (3500 giornate borsistiche). Ogni rete ha 30 input esclusivamente di natura tecnica, dati dai prezzi di chiusura che vanno da quello del giorno precedente fino a quello di dieci giorni precedenti, dalle medie mobili esponenziali a 50, 20 e 5 periodi che vanno da quella del giorno precedente fino a quella di cinque giorni precedenti e dall'RSI che va da quello del giorno precedente

fino a quello di cinque giorni precedenti. Si è ipotizzato quindi che questi 30 parametri possano dare un'indicazione quanto più precisa possibile non tanto sull'esatto prezzo di chiusura del giorno successivo quanto piuttosto sulla direzione che il tasso di cambio EUR/USD assumerà il giorno successivo. Quanto detto finora è schematizzato di seguito.



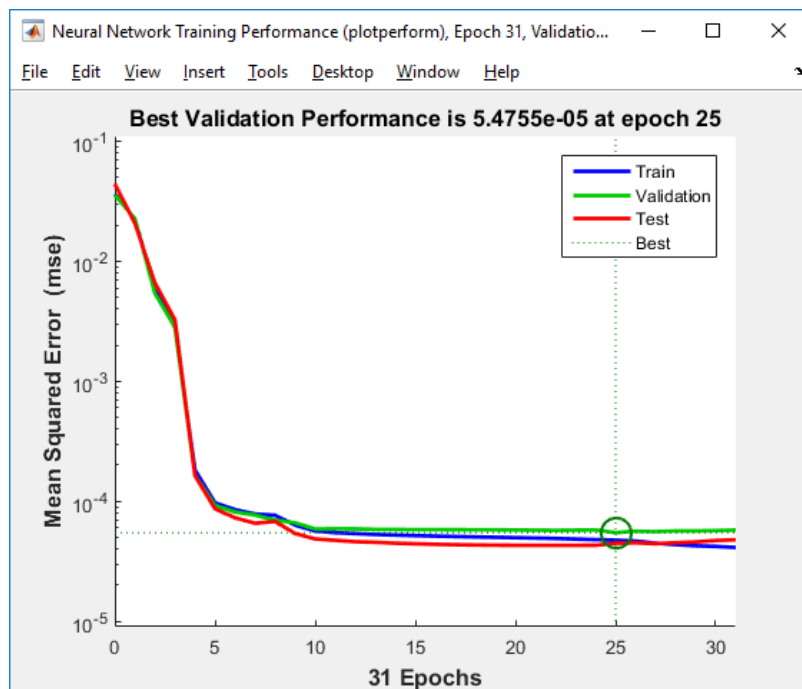
3.2.1 Rete neurale net2

La prima rete neurale utilizzata, net2 appunto, ha le caratteristiche sotto riportate:

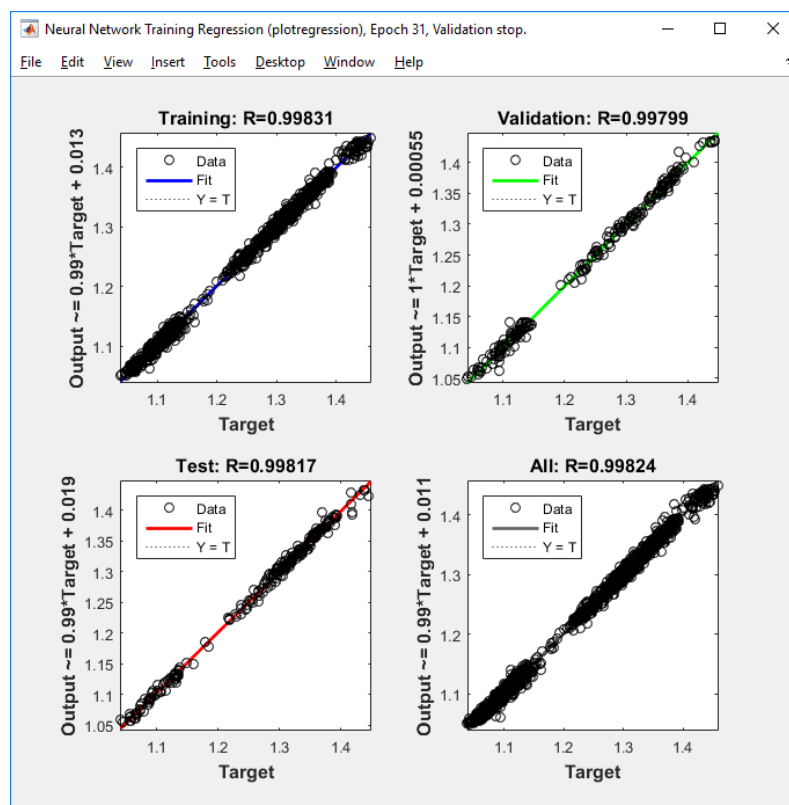


è quindi una rete di tipo feed-forward backpropagation con un solo strato nascosto in cui sono presenti quindici neuroni e la transfer function è la funzione tansig. La rete è allenata con l'algoritmo Levenberg-Marquardt statico, ma a ben vedere è un ibrido tra una rete statica e una rete dinamica per la natura degli input, dati essenzialmente da serie storiche ritardate. L'indice di performance è il classico errore quadratico medio. Per quanto riguarda il primo allenamento della rete, una volta inseriti input e target, esso ha generato i risultati riportati nella figura precedente e nelle figure successive per quanto riguarda la performance e la regressione. Come si può facilmente notare l'allenamento è stato completato in 31 iterazioni, quando il gradiente ha raggiunto un valore di 0.000425. Nonostante questo, poiché è stato utilizzato il metodo dell'early stopping, i parametri finali utilizzati dalla rete sono stati quelli in corrispondenza della venticinquesima iterazione, poiché da questo punto in poi l'errore sul validation set ha subito un incremento per sei volte consecutive, condizione sufficiente a far terminare l'allenamento della rete. Dal grafico della performance si evince inoltre come

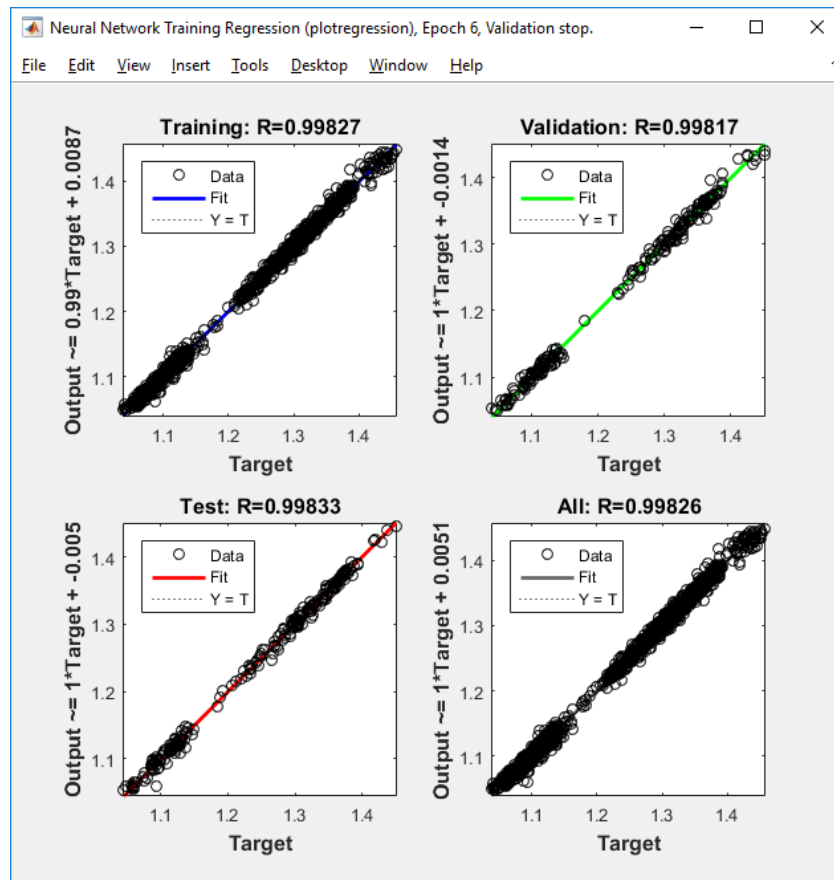
l'errore quadratico medio abbia raggiunto, su tutti i set e in corrispondenza del punto in cui è stato fermato l'allenamento, un indice di grandezza di oltre 10^{-4} .



Dal grafico della regressione emerge infine come il coefficiente R sia molto elevato per tutti i set di dati. Tuttavia al fine di aumentare questo coefficiente la

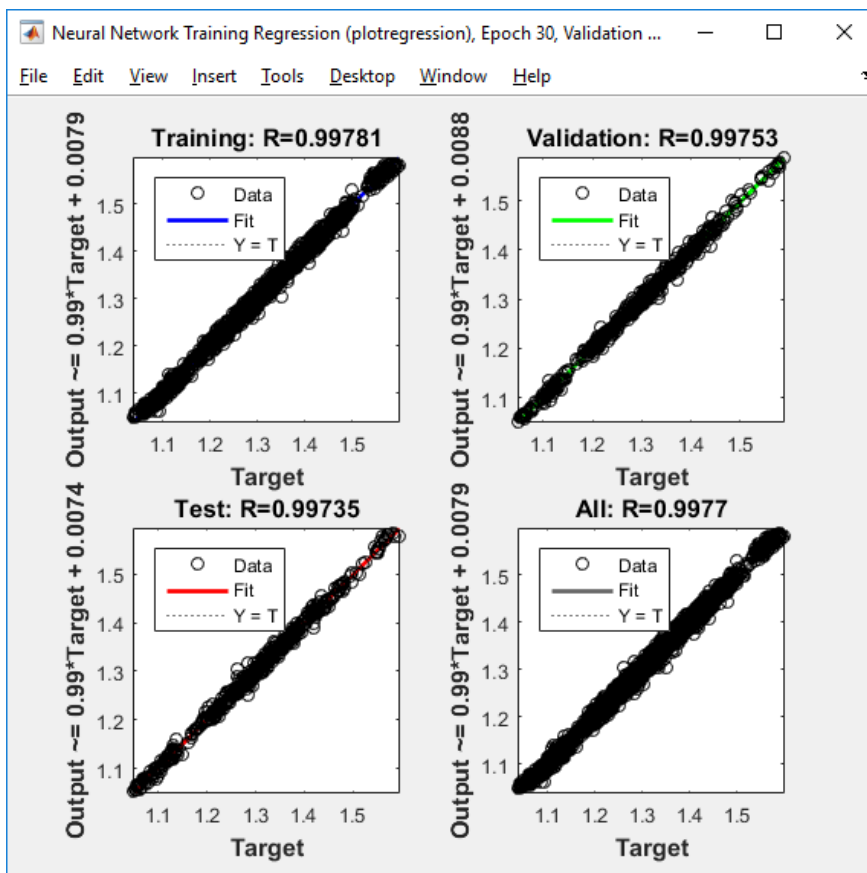
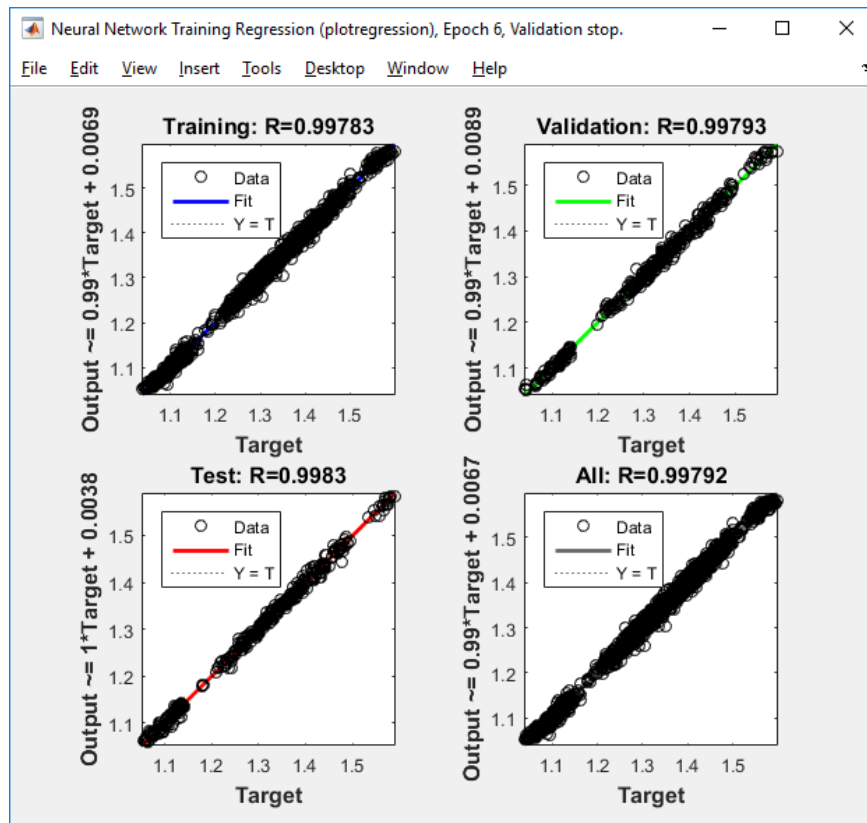


rete è stata allenata più di una volta e questo procedimento ha garantito il seguente risultato finale di regressione:



3.2.2 Reti neurali net3 e net4

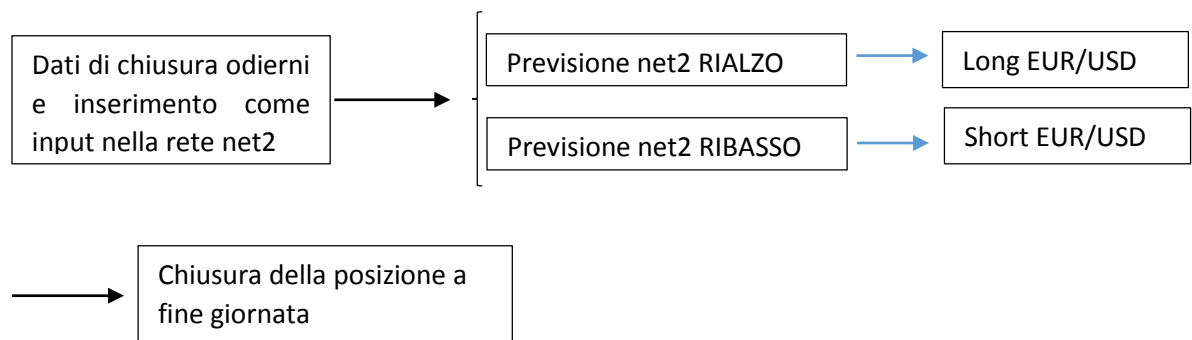
Come già ricordato in precedenza, le reti net3 e net4 si differenziano dalla rete net2 esclusivamente per l'ammontare dei dati utilizzati nell'allenamento. Infatti, per quanto riguarda gli input, il metodo di allenamento, le funzioni di trasferimento e tutte le altre caratteristiche le tre reti sono essenzialmente identiche. Proprio per questo motivo, per non appesantire troppo la trattazione, si è deciso di riportare di seguito solamente i risultati che quest'ultime due reti hanno generato alla fine dell'ultimo allenamento in termini di regressione. In particolare, anche le reti neurali denominate net3 e net4 hanno mostrato un coefficiente R sufficientemente elevato in tutti i set di dati, come da figure seguenti (la prima si riferisce alla rete net3 e la seconda alla rete net4):



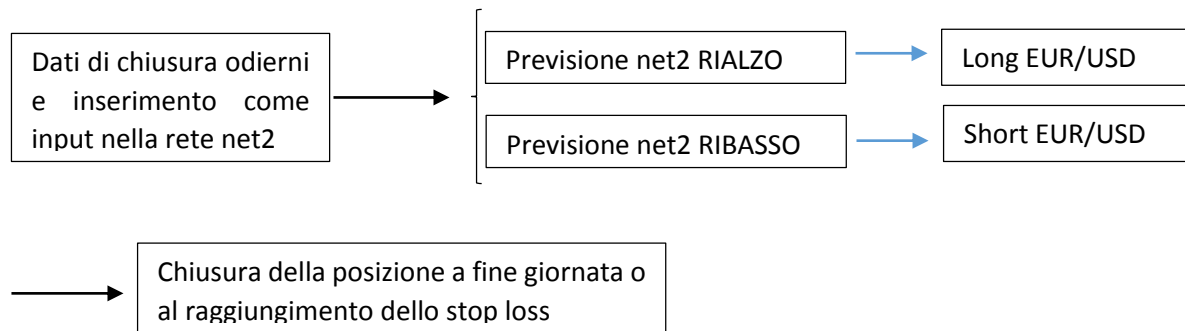
3.3 Trading system utilizzati

I trading system utilizzati sono stati fondamentalmente due, e sono stati denominati rispettivamente m1 e m2. Il sistema di trading m1 utilizza solamente la rete net2, mentre il sistema m2 utilizza le reti net2, net3 e net4. In particolare entrambi i sistemi di trading sono estremamente semplici, e hanno come punto di partenza i dati di chiusura della giornata odierna; una volta presi questi dati, si inseriscono come input nelle reti neurali, che calcolano la direzione prevista dal tasso di cambio EUR/USD durante la giornata borsistica successiva. Successivamente, il sistema m1 prevede l'apertura di una posizione long su EUR/USD se la previsione della rete net2 è un rialzo, mentre prevede l'apertura di una posizione short su EUR/USD se la previsione della rete net2 è un ribasso; il sistema m2 invece prevede l'apertura di una posizione long su EUR/USD solo se le previsioni di tutte le reti net2, net3 e net4 sono un rialzo, mentre prevede l'apertura di una posizione short su EUR/USD solo se le previsioni di tutte le reti net2, net3, net4 sono un ribasso. Nel caso in cui, per esempio, due delle tre reti prevedano un rialzo e una un ribasso, non si effettua nessuna operazione sul mercato. Per quanto riguarda invece la chiusura delle posizioni, è necessario fare una precisazione, poiché i due sistemi sono stati testati sia con l'utilizzo di uno stop loss sia senza. Nel caso in cui non si utilizzi lo stop loss, la posizione sul mercato verrà mantenuta fino alla chiusura della giornata in cui è stata aperta; nel caso invece in cui si utilizzi lo stop loss (che è stato impostato a 50 ticks, ovvero al 5% del capitale iniziale, ipotizzato in 1000 Euro), la posizione verrà mantenuta aperta fino al termine della giornata in cui è stata aperta o fino al raggiungimento dello stop loss. In ogni caso, per migliore chiarezza, i due trading system sono schematizzati nelle figure seguenti:

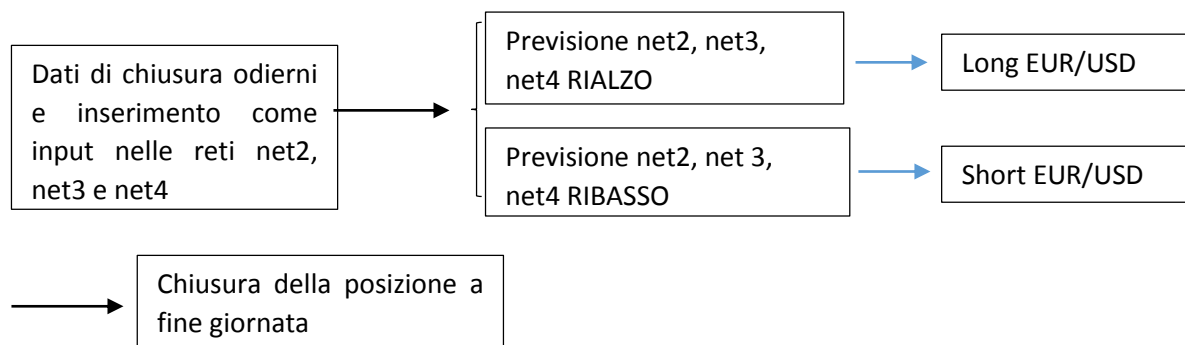
m1:



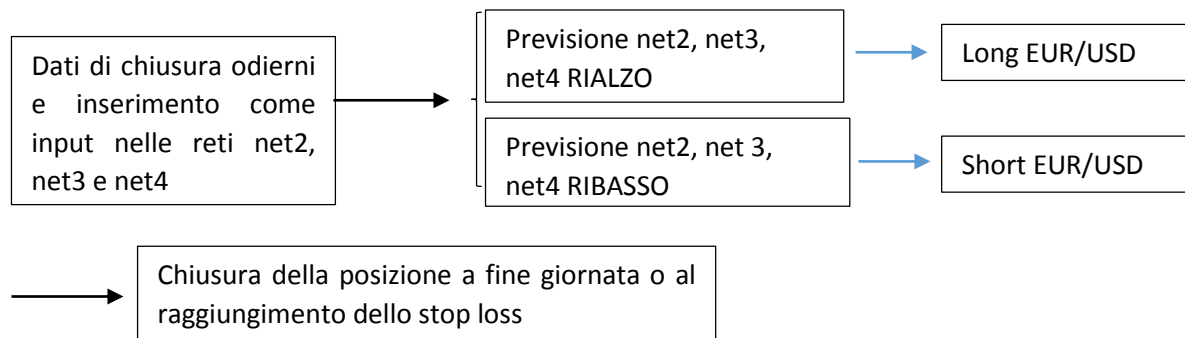
m1SL (sistema m1 con utilizzo di stop loss):



m2:



m2SL (sistema 2 con utilizzo di stop loss):



3.4 Descrizione dei risultati

Come già accennato all'inizio del capitolo tutti i sistemi di trading proposti sono stati testati sul mercato dal 08/03/2017 al 18/09/2017, quindi per un periodo complessivo di circa sei mesi. In generale i risultati sono stati positivi per tutti i sistemi analizzati ma, almeno fino alla data in cui si è interrotta la fase di test, i sistemi con l'utilizzo di stop loss hanno avuto performance superiori a quelli senza

l'utilizzo di stop loss. Inoltre il trading system denominato m1 ha ottenuto risultati migliori del sistema m2 in tutto il periodo analizzato, anche se ovviamente questi sono stati raggiunti con una volatilità più elevata. Infine, sempre parlando dal punto di vista generale, tutti i sistemi hanno avuto performance straordinarie nei primi due mesi arrivando a guadagnare oltre l'80%, per poi avere una fase contraddistinta da guadagni e perdite più o meno costanti fino ad arrivare ad un periodo molto negativo iniziato alla fine di Luglio e durato fino alla fine del mese di Agosto che ha inciso pesantemente sulla performance. Passando ora dal generale al particolare viene sotto riproposta la semplicissima function Matlab che è stata utilizzata per ottenere le previsioni delle tre reti e i grafici dei risultati dei quattro trading system (una volta create le reti tramite la procedura guidata del Neural Network Toolbox):

```
function [ previsione_rete2,previsione_rete3,previsione_rete4 ] =
reteEURUSDsubplot( vettoreinput)

load('reti234.mat');
load('DatiGraficoRisultati');
load('DatiGraficoRisultatiIstogramma');

% calcolo il valore previsto dalle reti 2, 3, 4
previsione_rete2=network2(vettoreinput);
previsione_rete3=network3(vettoreinput);
previsione_rete4=network4(vettoreinput);

%grafico risultati senza S.L.
figure (1);
subplot(2,1,1);
plot(t,m1,'b.-',t,m2,'k.-');
legend('m1','m2','location','SouthEast');
title('Risultati m1 e m2 senza Stop Loss');
grid minor;
xlabel('Tempo');
ylabel('Equity line');

%grafico risultati con S.L.
subplot(2,1,2);
plot(t,m1SL,'b.-',t,m2SL,'k.-');
legend('m1SL','m2SL','location','SouthEast');
title('Risultati m1 e m2 con Stop Loss');
grid minor;
xlabel('Tempo');
ylabel('Equity line');

%Istogramma
figure(2);
subplot(2,2,1);
bar(m1_senzaSL);
```

```

set(gca, 'XTickLabel', {'Mar', 'Apr', 'Mag', 'Giu', 'Lug', 'Aug'});
ylabel('Risultati');
xlabel('Mesi');
title('Risultati m1 senza S.L.');
```

```

grid minor;

subplot(2,2,2);
bar(m2_senzaSL);
set(gca, 'XTickLabel', {'Mar', 'Apr', 'Mag', 'Giu', 'Lug', 'Aug'});
ylabel('Risultati');
xlabel('Mesi');
title('Risultati m2 senza S.L.');
```

```

grid minor;

subplot(2,2,3);
bar(m1_conSL);
set(gca, 'XTickLabel', {'Mar', 'Apr', 'Mag', 'Giu', 'Lug', 'Aug'});
ylabel('Risultati');
xlabel('Mesi');
title('Risultati m1 con S.L.');
```

```

grid minor;

subplot(2,2,4);
bar(m2_conSL);
set(gca, 'XTickLabel', {'Mar', 'Apr', 'Mag', 'Giu', 'Lug', 'Aug'});
ylabel('Risultati');
xlabel('Mesi');
title('Risultati m2 con S.L.');
```

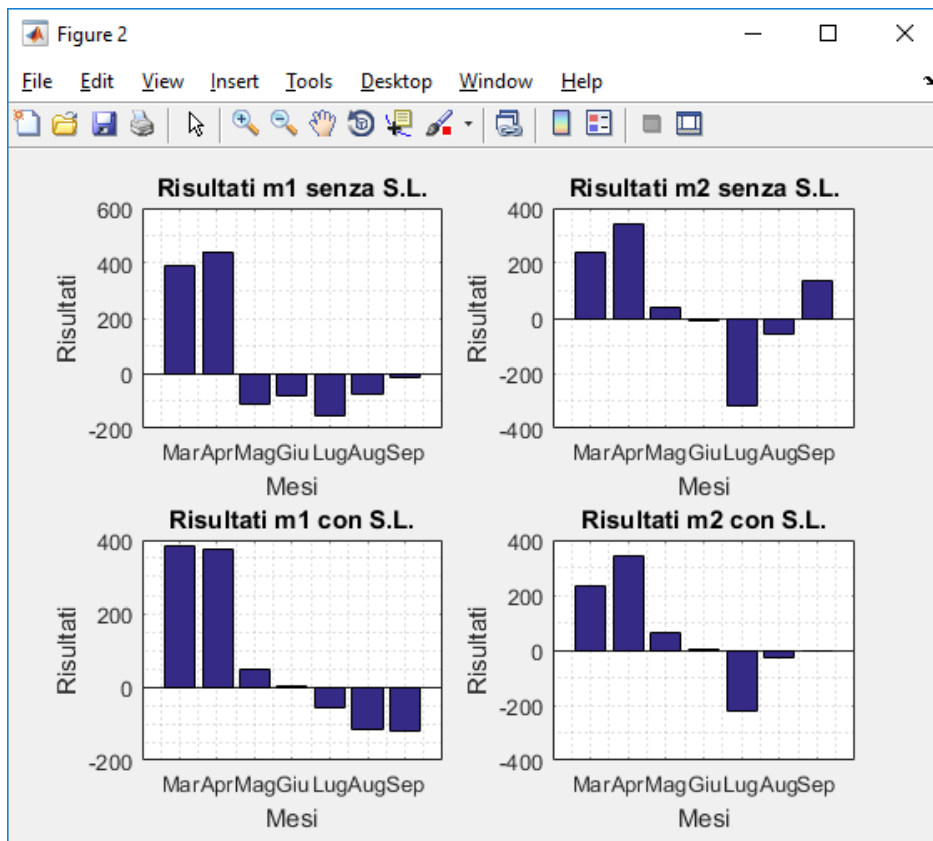
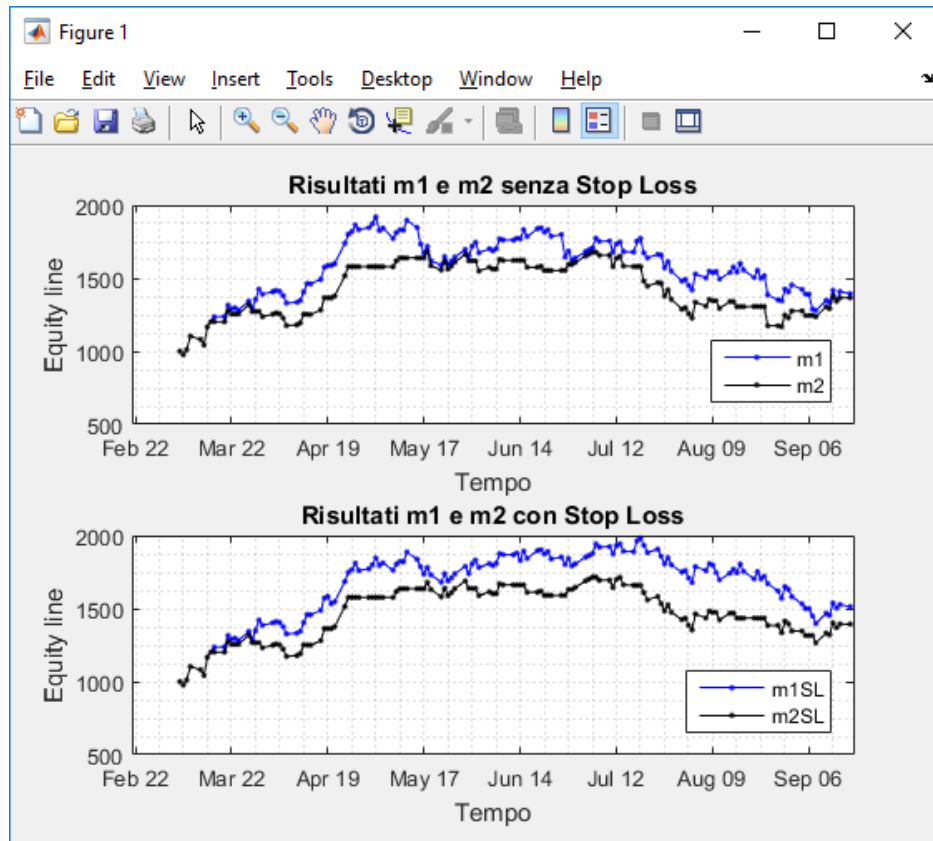
```

grid minor;

end
```

La function, dopo aver inserito giornalmente il vettore di input, calcola le previsioni della rete per il giorno successivo necessarie per operare sul mercato ed elabora i grafici dei risultati ottenuti dai vari trading system fino a quel momento. Al termine del periodo di test sul mercato, i grafici dei sistemi di trading erano dati dalle figure seguenti. Come si nota facilmente la performance del primo sistema di trading è stata del 39.4% nel caso in cui non è stato utilizzato lo stop loss e del 51.3% nel caso in cui quest'ultimo è stato utilizzato. Per quanto riguarda il secondo sistema di trading invece la performance è stata leggermente minore, con un guadagno del 39.4% in caso di utilizzo di stop loss e del 36.6% nel caso di non utilizzo dello stop. L'istogramma dei risultati mensili mostra ancora più chiaramente quanto già ricordato, ovvero il fatto che i sistemi di trading utilizzati non hanno saputo ripetere, negli ultimi mesi del test, le elevate performance fatte registrare nei due mesi iniziali. In particolare, i mesi più negativi in termini di perdita sono stati senza dubbio quelli di Luglio e Agosto. Nonostante ciò, il

capitale iniziale ha subito una perdita solo all'inizio del periodo di test, quando è sceso a 976 euro.



Scendendo ancora di più nel particolare, le seguenti tabelle mostrano i principali parametri relativi al backtest delle strategie:

M1 senza stop loss	Numero trade vincenti	76	Guadagno tot. Trade vincenti	3055	Media trade vincenti	40.20	Massimo valore del portafoglio raggiunto	1919	Massimo Runup	919
	Numero trade perdenti	63	Perdita tot. Trade perdenti	-2661	Media trade perdenti	-42.24	Minimo valore del portafoglio raggiunto	976	Massimo Drowdown	-353
					Rapporto tra operazioni vincenti e perdenti	0,95			GUADAGNO /PERDITA media per singolo trade	2,83
M2 senza stop loss	Numero trade vincenti	44	Guadagno tot. Trade vincenti	1991	Media trade vincenti	45.25	Massimo valore del portafoglio raggiunto	1683	Massimo Runup	683
	Numero trade perdenti	42	Perdita tot. Trade perdenti	-1625	Media trade perdenti	-38.69	Minimo valore del portafoglio raggiunto	976	Massimo Drowdown	-451
					Rapporto tra operazioni vincenti e perdenti	1,17			GUADAGNO /PERDITA media per singolo trade	4.26

I dati confermano come non sia possibile affermare che uno dei trading system sia in assoluto migliore dell'altro. Se infatti il sistema di trading m1 con l'utilizzo dello stop loss prevaleva nettamente sugli altri fino alla metà del mese di luglio, l'ultimo periodo di test ha fatto sì che questo sistema registrasse un netto calo nella

M1 con stop loss	Numero trade vincenti	73	Guadagno tot. Trade vincenti	3000	Media trade vincenti	41.10	Massimo valore del portafoglio raggiunto	1983	Massimo Runup	887
	Numero trade perdenti	66	Perdita tot. Trade perdenti	-2487	Media trade perdenti	-37.68	Minimo valore del portafoglio raggiunto	976	Massimo Drowdown	-305
					Rapporto tra operazioni vincenti e perdenti	1,09			GUADAGNO/PERDITA media per singolo trade	3,69
M2 con stop loss	Numero trade vincenti	43	Guadagno tot. Trade vincenti	1944	Media trade vincenti	45.21	Massimo valore del portafoglio raggiunto	1715	Massimo Runup	680
	Numero trade perdenti	43	Perdita tot. Trade perdenti	-1550	Media trade perdenti	-36,05	Minimo valore del portafoglio raggiunto	976	Massimo Drowdown	-361
					Rapporto tra operazioni vincenti e perdenti	1,25			GUADAGNO/PERDITA media per singolo trade	4.58

performance, tale da farlo riallineare, in termini generali, altri. In ogni caso, il sistema m1 con l'utilizzo dello stop loss rimane quello che ha fatto registrare la performance più alta (51.3%). Per quanto riguarda invece la percentuale di trade vincenti rispetto al totale, il sistema migliore risulta essere m1 senza l'utilizzo di stop loss, con 76 operazioni corrette su 133. Un altro parametro importante è il rapporto tra operazioni vincenti e perdenti: un valore maggiore di 1 infatti sta a significare che per ogni euro perso il sistema ne guadagna più di uno. In tal senso tutti i sistemi tranne m1 senza utilizzo di stop loss mostrano un valore di questo specifico rapporto maggiore di uno, ma il migliore rispetto agli altri risulta m2 con l'utilizzo di stop loss (1.25). Quest'ultimo sistema risulta migliore degli altri anche per guadagno medio per singolo trade, con un valore che si attesta al livello di 4.58

(è necessario tuttavia specificare come questo parametro sia influenzato dal numero di trade, che nel sistema m2 sono minori rispetto al sistema m1). Infine, andando ad analizzare i valori del massimo runup e del massimo drowdown si nota come, almeno sotto questo aspetto, il sistema m1 sia migliore del sistema m2, in quanto ha fatto registrare un massimo runup più alto e un massimo drowdown più basso.

Riassumendo, i parametri risultanti dal backtest non permettono di dare un giudizio assoluto sul fatto che un sistema sia migliore rispetto ad un altro, sebbene tutti i trading system abbiano fatto registrare buone performance positive (anche se con un trend molto negativo nei mesi di Luglio e Agosto che tuttavia appare in ripresa a partire dall'ultima parte del periodo di test).

3.5 Conclusione e possibili sviluppi della metodologia

Dall'analisi dei risultati sopra mostrati è possibile affermare come con queste semplici strategie di trading basate sullo sviluppo di reti neurali, che rappresentano un "ibrido" tra modelli statici e dinamici, sia stato possibile ottenere risultati positivi sul mercato in un intervallo di tempo sufficientemente ampio. Tuttavia appare doveroso non soffermarsi ai semplici risultati, ma cercare di andare oltre, per capire quali potrebbero essere le implementazioni da compiere al fine di migliorare questi sistemi di trading incrementandone i guadagni e soprattutto limitando al massimo i drowdown. Ovviamente questo tipo di ricerca andrebbe ben oltre gli orizzonti di questa trattazione, ma volendo citare solo alcune possibili migliorie che potrebbero essere apportate, magari in uno sviluppo futuro del presente lavoro, sarebbe senza dubbio necessario provare a realizzare nuove reti con diversi input iniziali, quali ad esempio altri indicatori tecnici come momentum, stocastico, altri tipi di medie mobili ecc., oppure inserire come input non solo variabili di natura tecnica ma anche variabili di natura fondamentale come differenziali tra tassi di interesse, Pil, tassi di inflazione, saldi delle bilance commerciali e tutte le altre variabili che possono influenzare un determinato tasso di cambio; un ulteriore sviluppo sarebbe quello di testare la rete non solo su un

tasso di cambio ma su più tassi di cambio, in modo da cercare di capire quale sia quello sul quale essa produce i risultati migliori; ancora, sarebbe necessario andare ad ottimizzare tramite appositi software alcuni importanti parametri come lo stop loss al fine di impostarlo alla distanza ottimale. Infine, l'ultimo passo sarebbe quello di automatizzare tutto il trading system rendendolo totalmente computerizzato e soprattutto autonomo, in modo che apra e chiuda le posizioni sul mercato di propria iniziativa e facendo sì che possa essere utilizzato continuamente sul mercato reale senza la necessità da parte del trader di compiere tutte le operazioni manualmente, cosa che non lascerebbe spazio all'emotività, causa più frequente di fallimento per tutti i trader.

Conclusione

In questa trattazione si è partiti dalla descrizione del mercato del Forex per poi introdurre questa nuova classe di modelli non lineari e arrivare, nell'ultimo capitolo, a creare alcuni semplici trading system da poter applicare al tasso di cambio EUR/USD. Come mostrato i risultati sono stati abbastanza soddisfacenti, anche se non eccezionali e questo, a mio avviso, dimostra la capacità e la forza che le reti neurali hanno nel risolvere problemi di questo tipo. I risultati infatti sono stati buoni anche se le reti con le quali sono stati costruiti i sistemi di trading erano molto semplici. Per questo motivo, sarebbe auspicabile avere altro tempo a disposizione per sviluppare ulteriormente questi sistemi in modo da farli diventare realmente redditizi nel lungo termine. In altre parole, questo breve elaborato deve essere senza dubbio considerato come un punto di partenza e non come un punto di arrivo.

Bibliografia

Hagan M. T. et al., Neural Network Design, 2nd edition, Martin Hagan, 2014.

Lai K. K., Wang S., Yu L., Foreign-exchange-rate forecasting with artificial neural networks, Springer, 2014.

Gurney K., An Introduction to Neural Networks, 1st edition, CRC Press, 1997-

Fausett L., Fundamentals of Neural Networks: Architectures, Algorithms, and Applications, 1st edition, Prentice Hall, 1993.

Bank for International Settlement (BIS), Triennial Central Bank Survey, Foreign exchange turnover in April 2016. www.bis.org/publ/rpfx16.htm.

Bank for International Settlement (BIS), Triennial Central Bank Survey, Foreign exchange turnover in April 2013: preliminary global results. www.bis.org.

Bank for International Settlement (BIS), Triennial Central Bank Survey, Foreign exchange and derivatives market activity in April 2010: preliminary results. www.bis.org.

Choudhry et al., High-frequency exchange-rate prediction with an artificial neural network. Intelligent systems in accounting, finance and management, 19 (2012): 170-178.

Chandrasekara V., Tilakaratne C., Forecasting Exchange Rates using Artificial Neural Networks. Sri Lankan Journal of Applied Statistics, 10 (2009): 187-201.

Dunis C. L., Laws J., Sermipinis G., The robustness of neural networks for modelling and trading the EUR/USD exchange rate at the ECB fixing. Journal of Derivatives & Hedge Funds, 15, 3 (2009): 186-205.

Lai K. K., Wang S., et al., Foreign-exchange-rate forecasting with artificial neural networks: a review. *International Journal of Information Technology and Decision Making*, 3, 1 (2004): 145-165.

Kamruzzaman M., Saker R., Forecasting of currency exchange rates using ANN: a case study. *Conference: Neural Networks and Signal Processing*, 2003. *Proceedings of the 2003 International Conference on*, Volume: 1 (2004): 793-797.

Dunis C. L., Williams M., Modelling and trading the euro/US dollar exchange rate: Do neural network models perform better? *Derivatives Use, Trading & Regulation*, 8, 3 (2002): 211–239.

Nasution B. P., Agah A., Currency exchange rate forecasting with neural networks. *Journal of Intelligent Systems*, 10, 3 (2000): 219-254.

Simon E., Forecasting foreign exchange rate with neural networks. Diploma project, Computer Science Institute, University of Neuchatel, 2002.

Foreign exchange market, Wikipedia. Ultimo aggiornamento 19 Agosto 2017.
https://en.wikipedia.org/wiki/Foreign_exchange_market (consultato il 30/04/2017).

Il mercato dei cambi, Borsa Italiana. Ultimo aggiornamento 20/09/2012.
<http://www.borsaitaliana.it/notizie/speciali/guerra-valute/mercato-valutario/mercato-cambi/mercato-cambi.htm> (consultato il 22/06/2017).

Borsa Italiana, <http://www.borsaitaliana.it>.

Thomson Reuters, <https://it.reuters.com/investing/markets>.

Investing, <https://it.investing.com/>.