Breve introduzione al Calcolo Evoluzionistico

Stefano Cagnoni

Dipartimento di Ingegneria dell'Informazione, Università di Parma
cagnoni@ce.unipr.it

1 Introduzione

Il mondo fisico ed i fenomeni naturali in generale sono spesso fonte di ispirazione per la definizione di teorie computazionali. Ciò si verifica in modo particolare, anche se non esclusivo, nel campo dell'intelligenza artificiale, settore il cui principale obiettivo è la creazione di sistemi automatici che manifestino comportamenti "intelligenti". Nel caso delle reti neurali il paradigma computazionale utilizzato è di fatto, seppure in forma estremamente semplificata, un modello del funzionamento del neurone biologico. In altri casi, i fenomeni che hanno ispirato teorie computazionali vanno oltre l'ambito dell'emulazione di semplici meccanismi fisici o fisiologici e spaziano da modelli di interazione sociale, come nel caso nella definizione di modelli computazionali distribuiti cooperativi o competitivi, alle teorie che sovraintendono la nascita e lo sviluppo di organismi viventi.

Fra queste ultime, le teorie darwiniane dell'evoluzione hanno ispirato un'intera famiglia di paradigmi computazionali, utilizzati nel campo dell'ottimizzazione, del "problem solving" e dell'apprendimento automatico. L'interesse suscitato, i risultati prodotti da questi metodi e la varietà degli approcci utilizzati consente di inquadrarli in un settore a sé, cui si fa riferimento col nome di calcolo evoluzionistico.

Le origini del calcolo evoluzionistico possono ritrovarsi in alcune intuizioni di Turing e von Neumann. Una prima definizione formale di metodi che possono essere definiti evoluzionistici si ha nei primi anni '60 coi lavori di Fogel, Rechenberg e Holland. Tuttavia, a causa principalmente dell'insufficiente potenza dei calcolatori disponibili all'epoca, è solo alla fine degli anni '80 che il calcolo evoluzionistico inizia a svilupparsi ed a trovare applicazione pratica.

Alla base dei metodi evoluzionistici è la teoria dell'evoluzione naturale e la fondamentale distinzione fra *fenotipo*, l'individuo così come ci appare nel suo esistere ed operare, e *genotipo*, cioè il codice genetico che ne determina le caratteristiche. Tale codice è contenuto nel DNA in forma di *geni*, a loro volta scomponibili in *cromosomi*, che costituiscono gli elementi "atomici" di cui è composta la codifica. Il processo evolutivo si innesca quando *mutazioni* casuali o combinazioni di geni derivate da processi riproduttivi danno origine ad individui migliori, cioè dotati di un miglior adattamento (*fitness*) all'ambiente in cui vivono. Tali individui tendono a sopravvivere più a lungo (*selezione*), a generare quindi più figli, i quali hanno una elevata probabilità di presentare quelle stesse caratteristiche positive, a causa del meccanismo di incrocio (*crossover*) dei caratteri genetici dei genitori. In questo modo, col passare delle generazioni, la probabilità che un individuo presenti tali qualità aumenta, fino a divenire una caratteristica della specie. A livello di popolazione, il risultato di questo processo è l'aumento della fitness degli individui che la compongono e quindi l'*evoluzione della specie*.

```
1. Genera una popolazione in modo random
2. Decodifica ogni cromosoma per ottenere un
   individuo
3. Valuta la fitness di ciascun individuo
4. Genera una nuova popolazione, in parte clonando
   gli individui con la fitness più elevata, in parte
   ricombinando e mutando i loro cromosomi
5. Ripeti 2,3,4 fino ad una condizione di stop
```

Figura1. Algoritmo alla base del calcolo evoluzionistico.

Questo modello dell'evoluzione si riflette nell'algoritmo generale (figura 1) su cui sono basati i diversi paradigmi in cui si articola il calcolo evoluzionistico. Dato un problema da risolvere (l'ambiente naturale), una codifica (genotipo) mediante la quale si possono rappresentare soluzioni del problema e una funzione, detta funzione di fitness, in grado di valutare la bontà (fitness) di ogni soluzione, si genera in modo casuale una popolazione $\mathcal{S}(t)$ di soluzioni (individui, che si manifestano attraverso i loro fenotipi) e si valuta quindi la fitness di ciascuna soluzione. Come detto, nel processo evolutivo, la fitness misura la capacità dell'individuo ad adattarsi all'ambiente in cui vive e quindi la sua "probabilità di sopravvivenza". Pertanto, negli algoritmi evolutivi, la fitness è il parametro che regola i meccanismi mediante i quali, a partire da $\mathcal{S}(t)$ si crea una nuovo insieme di soluzioni $\mathcal{S}(t+1)$, caratterizzato da un uguale numero di elementi. Per fare ciò, dall'insieme $\mathcal{S}(t)$ viene selezionato un sottoinsieme di soluzioni $\hat{\mathcal{S}}(t)$, con un processo probabilistico regolato dai valori di fitness di ciascuna soluzione. Il nuovo insieme $\mathcal{S}(t+1)$ è ottenuto quindi a partire da $\hat{\mathcal{S}}$ mediante 3 meccanismi:

- *clonazione*: alcune delle soluzioni, già presenti nell'iterazione precedente, "sopravvivono" e permangono invariate in $\mathcal{S}(t+1)$;
- *crossover*: vengono generate nuove soluzioni (figli) aventi codifiche (genotipi) derivate da quelle di due o più soluzioni (genitori) di $\mathcal{S}(t)$;
- *mutazione*: vengono generate nuove soluzioni mediante alterazioni (mutazioni) casuali della codifica di soluzioni presenti in $\mathcal{S}(t)$.

La ripetizione di questo processo di valutazione, selezione e generazione, in cui si assegna alle soluzioni con valori di fitness più elevati una maggiore probabilità di essere presenti anche nella nuova generazione e di contribuire alla generazione di nuovi individui, porta all'aumento della fitness media delle soluzioni. Il processo viene arrestato quando si raggiungono soluzioni con fitness sufficientemente elevata o quando la fitness media cresce al di sotto di una certa soglia, nel momento cioè in cui tutti gli individui tendono ad essere simili fra loro. Il risultato dell'algoritmo è convenzionalmente la soluzione avente la fitness più elevata presente nell'insieme $\mathcal{S}(T)$ ottenuto al termine di T generazioni, ovvero la soluzione con fitness più elevata ottenuta in assoluto in qualunque iterazione t.

In una interpretazione pratica del suo significato, l'algoritmo illustrato può essere visto come un metodo di ottimizzazione stocastica di una funzione definita dall'utente (la funzione di fitness). In modo più generale, se si considera la funzione di fitness come la misura della bontà della soluzione di un problema, l'algoritmo si propone come metodo di ricerca di soluzioni ottime di un problema definito dall'utente in uno spazio determinato dalla codifica scelta dall'utente per rappresentare le soluzioni.

Nel caso più diretto, il problema da affrontare è la ricerca del massimo di una funzione, attraverso la determinazione, ad esempio, di parametri numerici inclusi nella formula che la descrive. In questo caso, la fitness è direttamente la funzione da massimizzare. In generale, tuttavia, quando ad esempio il problema da risolvere non è di tipo analitico, il procedimento non è così diretto. In tal caso, infatti, si massimizza la fitness operando sulla codifica di una soluzione ad un problema di cui la fitness misura la bontà. E' quindi necessario definire una funzione di decodifica che trasformi il genotipo nel corrispondente fenotipo (soluzione), applicare la soluzione così ottenuta al problema che si intende risolvere e misurarne la bontà attraverso la funzione di fitness. L'algoritmo quindi può essere scomposto in una parte indipendente dal problema da risolvere, ma dipendente dalla rappresentazione utilizzata, costituita essenzialmente dall'implementazione dei meccanismi di selezione e di generazione di una nuova popolazione attraverso gli operatori genetici e in una parte che dipende dall'applicazione, costituita dalla definizione del genotipo, dalla funzione di decodifica della soluzione e dalla funzione che valuta la fitness. I principali pacchetti software che implementano algoritmi di tipo evoluzionistico, fissano la rappresentazione (e quindi il paradigma evoluzionistico) e forniscono la parte indipendente dall'applicazione e l'interfaccia verso la funzione di fitness e quella di decodifica che dovranno essere scritte dall'utente.

L'utente deve quindi mappare il problema da risolvere e le sue soluzioni sulla rappresentazione utilizzata e scrivere delle funzioni di fitness e di decodifica delle soluzioni che rispettino l'interfaccia definita dal pacchetto. L'utente inoltre deve fissare alcuni parametri che regolano l'applicazione degli operatori genetici. In particolare:

– *Dimensione della popolazione*, cioè il numero di individui da cui è composta $\mathcal{S}$. Come giá detto, questo valore resta invariato nel tempo.
– *Frequenza di crossover*, cioè la percentuale di membri di $\mathcal{S}(t+1)$ che devono essere generati attraverso l'operazione di crossover.
– *Frequenza di mutazione*, cioè la probabilità con cui un cromosoma (elemento della codifica) può subire una modifica casuale nel passaggio da una generazione alla successiva.

La diversa codifica delle soluzioni, unitamente ai metodi utilizzati per elaborare e ricombinare l'informazione contenuta nel genotipo per generare nuovi individui a partire da quelli presenti nella attuale popolazione, consente di suddividere i metodi di calcolo evoluzionistico in diversi paradigmi. Nel presente capitolo ci limiteremo ad introdurre i due paradigmi di uso più universale e comune, cui si fa di solito riferimento coi termini di *algoritmi genetici* e *programmazione genetica*.

1.1 Algoritmi genetici

Gli algoritmi genetici applicano l'algoritmo descritto in fig. 1 ad un insieme di soluzioni codificate mediante stringhe di bit di lunghezza fissa. Nel caso più diretto tali stringhe rappresentano la concatenazione delle rappresentazioni binarie dei parametri numerici di una funzione da minimizzare o di una funzione che codifica la soluzione ad un problema in termini numerici. In generale, comunque, data l'universalità della rappresentazione binaria, il genoma può rappresentare virtualmente qualunque tipo di informazione (segnali, immagini, forme, relazioni di tipo strutturale o topologico, ecc.). Per impostare un esperimento con un algoritmo genetico è quindi necessario definire in modo esatto la struttura del genoma, stabilendone quindi la lunghezza ed il significato dei singoli bit.

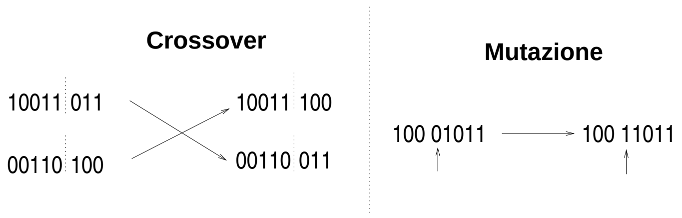

Figura2. Gli operatori di crossover e mutazione utilizzati negli algoritmi genetici

Le operazioni di crossover e di mutazione utilizzate con questo tipo di rappresentazione sono mostrate in figura 2. Durante il crossover si opera uno scambio di materiale genetico (sottostringhe corrispondenti della codifica) fra i due genitori, secondo diverse possibili modalità. Le due stringhe risultanti da tale operazione sono i "figli" che verranno inseriti nella nuova generazione della popolazione. Fra i diversi tipi di crossover i più comuni sono il *crossover singolo punto*, in cui le stringhe che codificano i due genitori vengono "tagliate" in uno stesso punto; si opera poi uno scambio della parte destra (o sinistra) delle stringhe per ottenere due figli in cui il genotipo del primo è costituito dalla concatenazione della parte destra del genotipo del primo genitore con la parte sinistra (destra) di quello del secondo, mentre il genotipo del secondo figlio è costituito dalla concatenazione della parte destra (sinistra) del genotipo del secondo genitore con la parte sinistra (destra) di quello del primo. Nel *crossover doppio-punto* la stringa è considerata "circolare" (l'ultimo bit si immagina concatenato al primo) e quindi il genotipo, "tagliato" in due punti, viene suddiviso in due parti che, nella rappresentazione lineare della stringa, corrispondono alla sottostringa interna ai due tagli e alle due sottostringhe esterne ad essi che, tuttavia, come detto, si immaginano concatenate fra loro in una unica stringa. Lo scambio avviene quindi in modo del tutto analogo al crossover singolo punto. Il *crossover uniforme*, invece, prevede che, per ogni posizione all'interno della stringa, i bit corrispondenti dei due genitori vengano assegnati uno a un figlio e l'altro all'altro figlio in modo casuale. La *mutazione* consiste semplicemente nell'inversione di un bit scelto a caso in una qualunque posizione del genoma.

Gli algoritmi genetici sono sicuramente il paradigma evoluzionistico piú comunemente utilizzato, sia per le caratteristiche della rappresentazione che consente di realizzare implementazioni particolarmente efficienti degli operatori genetici, sia per la quantità di problemi di ottimizzazione le cui soluzioni possono essere rappresentate in modo immediato ed efficace con stringhe binarie.

1.2 Programmazione genetica

Una delle possibili rappresentazioni utilizzabili per le funzioni è costituita dai cosiddetti *alberi sintattici*. Questi sono strutture ad albero in cui ogni nodo rappresenta uno degli operatori ammessi dalla grammatica da cui la funzione è generata. I figli di ciascun nodo rappresentano gli argomenti dell'operatore e sono in numero pari, ovviamente, alla sua cardinalità. Le foglie dell'albero sono costituite dai *simboli terminali* della grammatica, cioè da quegli elementi della grammatica che non richiedono argomenti per essere valutati. E' questo il caso delle costanti, dei puntatori, o delle variabili utilizzate dalla funzione. In figura 3 è mostrato un esempio di rappresentazione di una funzione attraverso un albero sintattico.

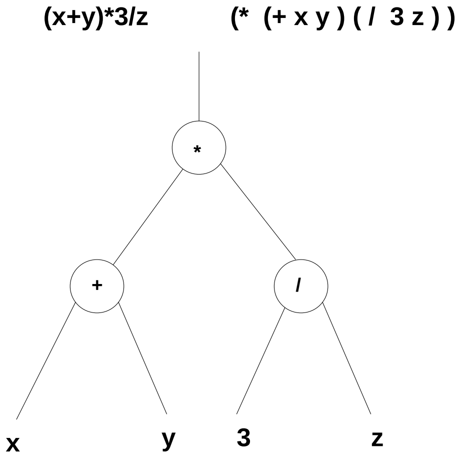

Figura3. Rappresentazione di una funzione, espressa sia in notazione algebrica che in notazione prefissa LISP-like, mediante un albero sintattico.

Si può notare come sia sufficiente operare un attraversamento simmetrico dell'albero (visitare cioè, nell'ordine, il sottoalbero sinistro, la radice e infine il sottoalbero destro in modo ricorsivo) per ottenere l'espressione algebrica di partenza. Un'altra rappresentazione equivalente è realizzabile mediante liste con sintassi simile al LISP, che consente una efficiente implementazione del paradigma in tale linguaggio. I primi esempi di programmazione genetica, proposti da Koza nel testo che di fatto ha introdotto questo paradigma evoluzionistico, utilizzavano quest'ultima rappresentazione. E' importante notare come i genomi abbiano in questo caso dimensione variabile: questo implica che non è possibile trovare una corrispondenza semantica fra nodi di individui differenti, come accade invece nel caso dei singoli bit del genoma utilizzato negli algoritmi genetici. Si noti anche che possono esistere diverse rappresentazioni per una stessa funzione: si pensi come semplice esempio allo scambio fra i figli di un operatore commutativo come l'*addizione*.

Definendo opportuni operatori genetici che operano su questo tipo di rappresentazione e applicando l'algoritmo di figura 1 è quindi possibile evolvere una popolazione di funzioni e quindi, in pratica, generare programmi. Da questa proprietà deriva il termine *programmazione genetica*, che si estende anche a quegli altri metodi evoluzionistici, caratterizzati da rappresentazioni diverse, in cui il genoma rappresenta il codice di funzioni o programmi e non soltanto la codifica dei parametri che ne determinano il valore. La differenza fondamentale fra algoritmi genetici e programmazione genetica risiede quindi nel livello al quale viene operata l'ottimizzazione. Un algoritmo genetico ottimizza una soluzione definita e parametrizzata dall'utente, e quindi l'ottimizzazione opera sulla rappresentazione dei parametri di una funzione la cui struttura è nota. Nella programmazione genetica si opera ad un livello superiore, in quanto l'utente definisce gli elementi di una grammatica (operatori e simboli terminali) che vengono utilizzati per generare le funzioni che si vogliono evolvere. L'ottimizzazione della fitness avviene quindi attraverso la manipolazione della rappresentazione del codice stesso delle funzioni e non solo dei loro parametri.

Nel caso della più comune rappresentazione mediante alberi sintattici (o liste LISP-like, che costituiscono una rappresentazione equivalente) gli operatori genetici operano su (sotto)alberi: l'operatore di crossover scambia due sottoalberi qualsiasi dei due genitori, mentre la mutazione sostituisce un albero con un nuovo albero generato in modo random (v. fig. 4).

A causa della maggiore estensione dello spazio di ricerca (lo spazio di tutte le funzioni che soddisfano la grammatica definita dall'utente) e della minore efficienza della mappatura in memoria della rappresentazione utilizzata, la programmazione genetica è un paradigma che richiede sia un elevato carico computazionale che un'elevata occupazione di memoria: i tempi di convergenza possono essere fino ad un ordine di grandezza più lunghi di quelli degli algoritmi genetici, a parità di dimensione della popolazione e di problema, anche se ovviamente un confronto di questo tipo non può essere proposto in termini rigorosi. Unitamente alla relativa "gioventù" del paradigma, questo è il motivo principale per cui la programmazione genetica ha sinora trovato applicazioni più limitate e minore diffusione rispetto agli algoritmi genetici. La potenza intrinseca del paradigma e la sua versatilità, confortate da alcuni risultati di rilievo comunque già ottenuti in diversi settori, alimentano notevoli attese per il prossimo futuro, in considerazione anche del continuo progresso degli strumenti di elaborazione, in termini di potenza computazionale e di capacità di memoria.

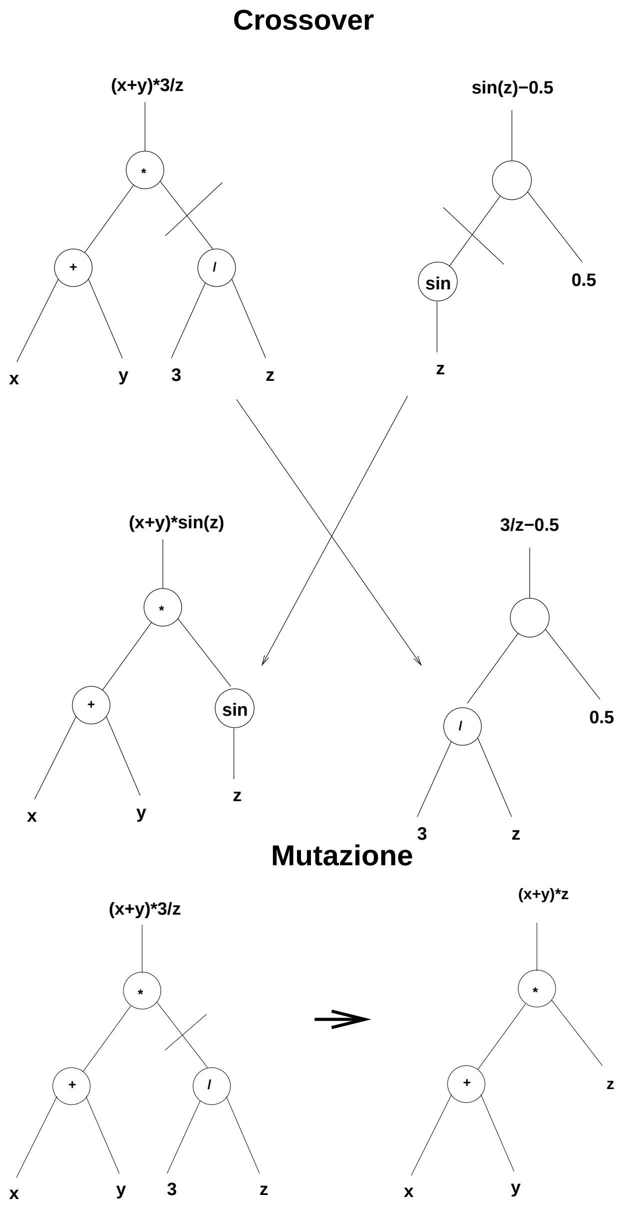

Figura4. Gli operatori di crossover e mutazione utilizzati nella programmazione genetica con rappresentazione mediante alberi sintattici