

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea in Informatica



**NeuroEvolution of
Augmenting Topologies:
Studio, Implementazione
ed Estensione**

Supervisori:

Giuseppe Cuccu

Dr. Leonardo Vanneschi

**Relazione della prova
finale di:**

Marko Bojovic

Matr. 056197

**Neuroevolution of augmenting
topologies:**

Studio, Implementazione ed Estensione

Marko Bojovic

Ottobre 2009 - Marzo 2010

Indice

I	Background	8
1	Reti Neurali	9
1.1	Introduzione	9
1.2	Connessioni e pesi	9
1.3	Input e output	10
1.4	Funzioni di composizione	10
1.5	Funzioni di attivazione	11
1.6	Unità nascoste	12
1.7	Lineare vs Non-lineare	12
1.8	Perché Non-lineare	14
2	Algoritmi genetici	16
2.1	Storia	16
2.2	Definizione di individuo	17
2.3	Concetto di fitness	17
2.4	Selezione	17
2.5	Riproduzione e crossover	18
2.6	Mutazione	19
2.7	Elitismo	19
3	Neuroevolution	20
3.1	Apprendimento rete	20
3.2	Tecniche usuali	21
3.3	Limitazioni	22
3.4	GA: individui come pesi	23

II	Studio e Implementazione	24
4	NEAT	25
4.1	Che cosa è NEAT	25
4.2	Principi di funzionamento	25
4.3	Problemi affrontati nell'ideazione	26
4.4	Innovation numbers	26
4.5	Speciazione	27
4.6	Funzione di distanza	27
4.7	Composizione della popolazione	28
4.8	Fitness e shared fitness	28
4.9	Come avviene la selezione	29
4.10	Crossover tra individui	29
4.11	Applicazione della mutazione	30
4.11.1	Aggiunta di nuovi nodi	30
4.11.2	Nuove connessioni	31
4.11.3	Perturbazione dei pesi	31
5	NCD	32
5.1	Introduzione	32
5.2	Complessità di Kolmogorov	32
5.3	Approssimazione con compressione	33
6	Riprogettazione	35
6.1	Come è strutturato il codice	35
6.2	Struttura di un individuo	35
6.3	Struttura della popolazione	37
6.4	Estensione dell'insieme dei geni	38
6.5	Innovation numbers	38
6.6	Parametri	39
7	Le fasi dell'algoritmo	41
7.1	Inizializzazione della popolazione	41
7.2	Selezione	41

<i>INDICE</i>	5
7.3 Elitismo	42
7.4 Crossover	42
7.5 Mutazione	42
7.6 Calcolo della Fitness	45
7.7 Speciazione	45
7.8 Calcolo della Fitness condivisa	46
7.9 Passaggio da genotipo a fenotipo	47
7.10 Attivazione della rete	48
III Risultati	51
8 Problema dello XOR	55
8.1 Descrizione	55
8.2 Funzione di fitness	56
8.3 Come viene eseguito il test	57
8.3.1 Setup	58
8.3.2 Funzioni di supporto	58
8.3.3 Simulazione	59
8.4 Risultati del test	60
9 Maze navigator	61
9.1 Descrizione	61
9.2 Funzione di fitness	62
9.3 Simulazione	62
9.4 Parametri	63
9.5 Risultati del test	64
10 Tartarus	66
10.1 Descrizione	66
10.2 Simulazione	67
10.3 Parametri	68
10.4 Risultati	68

<i>INDICE</i>	6
IV Conclusioni	71

Premessa

L'obiettivo della tesi

L'obiettivo di questa tesi è studiare e implementare un algoritmo di neuroevolution ideato nel 2002, identificato come NEAT (Neuroevolution of Augmenting Topologies). Quest'algoritmo è particolare perché permette di evolvere differenti strutture di reti neurali contemporaneamente. Il linguaggio sfruttato per sviluppare l'algoritmo è Mathematica.

Oltre allo studio e implementazione, l'idea è di estendere il lavoro aggiungendo nuove funzionalità. Il centro di quest'estensione è basato sul confronto tra la funzione di distanza originale usata per la speciazione e la Normalized Compression Distance (NCD). L'ipotesi è che la NCD possa definire in modo più dettagliato la compatibilità tra due individui rispetto a quella utilizzata attualmente e perciò gli studi e i test si concentreranno sull'uso di questa funzione.

Parte I

Background

Capitolo 1

Reti Neurali

1.1 Introduzione

Le reti neurali, insieme con gli algoritmi genetici, rappresentano la base di questo progetto.

Una rete neurale è un'approssimatore di funzione generica, biologicamente ispirato dal cervello umano. Vengono usate nell'intelligenza artificiale per risolvere un ampio range di problemi. Come in un cervello, i neuroni vengono stimolati producendo effetti a catena fino ad ottenere un risultato.

Durante un'attività cerebrale, gli impulsi elettrici viaggiano da un neurone all'altro tramite collegamenti chiamati sinapsi. Questi impulsi possono poi giungere al sistema nervoso del corpo umano e permettere ad esempio l'attivazione di un arto.

In informatica questi impulsi sono interpretati da valori numerici e i neuroni sono delle funzioni che a loro volta producono numeri. Viene anche chiamata rete perché normalmente composta da più neuroni connessi tra loro, mentre i sistemi a neurone singolo vengono chiamati perceptron.

1.2 Connessioni e pesi

Durante il "viaggio" da un nodo all'altro, il valore attraversa una connessione, avente un peso numerico. I pesi vanno a modificare l'importanza

relativa di questo valore. Vengono modificati nel corso del processo di machine learning, fase nella quale una rete viene “allenata” e dove si cerca di ottenere una configurazione ottimale.

1.3 Input e output

Gli input sono un punto d’ingresso alla rete. Normalmente per ogni rete neurale viene definito il numero di input a priori e tale definizione dipende dalle osservazioni che l’ambiente fornisce. Lo scopo di una rete è solitamente delineato da un problema che essa dovrà risolvere. Oltre che essere un punto di ingresso di una rete neurale, il termine input viene usato anche per definire gli ingressi di ogni singolo neurone presente in tale rete.

In uscita, la rete ha uno o più nodi di output definiti a priori. Come il numero di nodi di input dipende dalle osservazioni, il numero di output dipenderà dalle azioni che verranno interpretate. I valori in uscita di questi nodi dipenderanno dai valori in input, dai valori ottenuti in output di altri neuroni e soprattutto da cosa restituirà il neurone stesso, cioè dalla funzione che lo compone. Un output può essere connesso con un altro neurone, con se stesso (in questo caso si parla di reti neurali ricorsive) oppure può essere l’output finale della rete.

Concludendo, per avere il valore di output della rete si ha il bisogno di calcolare prima l’output dei nodi intermedi partendo dai valori di input.

1.4 Funzioni di composizione

Ogni neurone è definito da due funzioni: la funzione di composizione e la funzione di attivazione. La funzione di composizione è di solito la somma pesata degli input del neurone. Vengono utilizzati i pesi sulle connessioni degli ingressi del neurone per la fase di composizione e il calcolo avviene nel seguente modo:

$$f_c(\bar{x}) = \sum_{i=1}^n w_i x_i \quad (1.1)$$

dove x è il vettore degli input e w è il vettore dei pesi. La composizione si riduce quindi al prodotto scalare tra i due vettori.

1.5 Funzioni di attivazione

La seconda funzione che descrive un neurone è la funzione di attivazione, la quale preso ciò che si ottiene componendo gli input restituisce un valore, chiamato appunto output. Alcune funzioni di attivazione sono:

- lineare a gradino
- a gradino
- gaussiana
- sigmoide

Di solito viene usata la sigmoide $\sigma(x)$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (1.2)$$

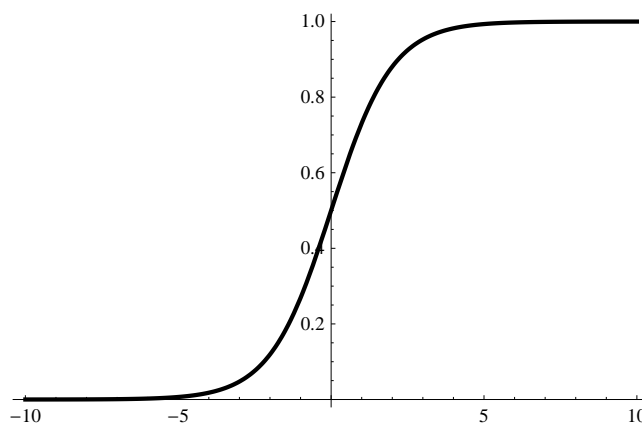


Figura 1.1: Il grafico della sigmoide

1.6 Unità nascoste

I nodi intermedi, cioè quelli che si trovano tra l'input e i neuroni di output, vengono chiamati strati nascosti o unità nascoste. Questi nodi sono anch'essi neuroni e vengono aggiunti o a priori o dinamicamente a seconda dell'algoritmo. La necessità di aggiungere strati a una rete neurale nasce dal fatto che un solo strato non è sufficiente per risolvere problemi non-linearmente separabili (es. XOR).

1.7 Lineare vs Non-lineare

Se si volessero classificare dei punti come quelli mostrati in figura 1.2, basterebbe una funzione lineare che rappresentasse la retta che separa le due classi.

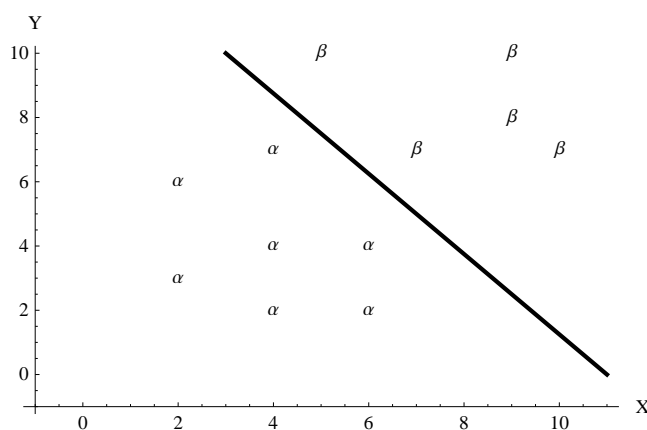


Figura 1.2: La separazione lineare dei punti

Come si potrebbe applicare tale funzione a una rete neurale? Si potrebbe pensare di avere un neurone singolo con 2 input chiamati x e y , un bias b e un'output chiamato z . Con w_x , w_y e w_b i pesi delle rispettive connessioni. La definizione analitica della retta è:

$$y = mx + q \tag{1.3}$$

Se si sostituiscono m con w_x e q con w_b e se si tiene conto che il valore di bias b è sempre a 1, si ottiene:

$$w_y y = w_x x + w_b \quad (1.4)$$

Quindi ponendo l'equazione a zero si ottiene:

$$w_x x + w_b - w_y y = 0 \quad (1.5)$$

Modificando il peso w_x si altera il coefficiente angolare della retta, mentre modificando w_b si ottiene la traslazione della retta. Quello che un metodo di machine learning dovrebbe fare è modificare i pesi w_x , w_y e w_b fino ad ottenere una retta che separa i punti di un training set.

Se invece la situazione diventa più complicata, dove un insieme di punti non può essere separato da una retta come mostrato in figura 1.3 allora diviene necessario applicare un metodo che separi i punti in modo non lineare. Questo diventa possibile con l'aggiunta di altri strati di neuroni alla rete neurale con l'utilizzo di funzioni di attivazione non-lineari.



Figura 1.3: La separazione non-lineare dei punti

1.8 Perché Non-lineare

Il problema principale nell'utilizzo di una funzione lineare è che l'aggiunta di strati di neuroni diverrebbe inutile perché si comporterebbe come una rete a uno strato.

Se si prende la funzione di composizione come funzione lineare e come esempio un neurone con un solo input x con il proprio peso w_1 , in uscita a questo neurone si avrebbe un qualcosa di questo tipo:

$$out_1 = w_1x \quad (1.6)$$

Se questo neurone fosse collegato a un secondo neurone con il rispettivo peso w_2 , il suo output diverrebbe:

$$out_2 = w_2out_1 \quad (1.7)$$

Quindi diventa:

$$out_2 = w_2w_1x \quad (1.8)$$

Continuando si avrebbe:

$$out_{finale} = (w_N \dots w_2w_1)x \quad (1.9)$$

Definendo $W = (w_N \dots w_2w_1)$ si avrebbe:

$$out_{finale} = Wx \quad (1.10)$$

Quindi non avrebbe senso avere strati intermedi nella rete perché lo stesso risultato si otterrebbe con una rete neurale a perceptron singolo. È questo il motivo del perché viene usata una funzione di attivazione non-lineare. Tornando all'esempio precedente e usando la funzione non-lineare sigmoide si ha che:

$$out_1 = \sigma(w_1x) \quad (1.11)$$

$$out_2 = \sigma(w_2 out_1) \tag{1.12}$$

$$out_2 = \sigma(w_2 \sigma(w_1 x)) \tag{1.13}$$

Giustificando quindi l'introduzione di strati nascosti.

Capitolo 2

Algoritmi genetici

2.1 Storia

In natura un essere vivente nasce, cresce, si riproduce e infine muore. A ogni nuova generazione qualche caratteristica della generazione precedente viene tramandata, che sia un aspetto fisico, che sia una disfunzione. L'estinzione di un certo tipo di organismi dipenderà dalla selezione naturale, da chi è idoneo all'ambiente che lo circonda e da chi non lo è.

Gli algoritmi genetici simulano proprio questo comportamento, utilizzando i concetti della genetica per rappresentare un organismo e i principi darwiniani per simularne gli effetti. Un organismo o individuo viene codificato come una sequenza di geni, questa tecnica venne introdotta da Holland nel 1975. Un insieme di individui forma una popolazione ed è l'insieme sul quale l'algoritmo lavorerà per cercare soluzioni a un certo problema dato.

La sopravvivenza e l'evoluzione di certi individui dipenderà dai seguenti principi:

- Riproduzione
- Capacità di adattamento all'ambiente
- Ereditarietà
- Variazione

- Competizione

Negli algoritmi genetici viene presa una popolazione di individui e su questi vengono applicati i sopra citati principi passando da generazione a generazione fino ad ottenere una o più soluzioni accettabili e appropriate al problema.

2.2 Definizione di individuo

Negli algoritmi genetici in genere un individuo è visto come una sequenza binaria. Questa sequenza rappresenta le sue caratteristiche ed è di pari lunghezza per ogni individuo di una popolazione.

Ciò che è scritto nella sequenza genomica, ciò che l'individuo contiene come istruzioni ereditate viene anche chiamato genotipo. Queste istruzioni possono essere espresse e tale manifestazione viene chiamata fenotipo. Quindi il fenotipo è un insieme delle caratteristiche influenzate dal genotipo, che è quindi l'insieme dei geni dell'individuo, ma è anche influenzato dall'ambiente che lo circonda.

2.3 Concetto di fitness

Per fare in modo che solo i migliori vengano selezionati per creare la prole della successiva generazione, viene introdotta la fitness, un valore numerico per dare una misura di qualità di un individuo.

Per esempio se il problema è massimizzare il numero di 1 in una stringa binaria, un individuo potrebbe essere definito come la sequenza che rappresenta questa stringa. È naturale quindi che la funzione di fitness sia definita come il conteggio o la somma di 1.

2.4 Selezione

La selezione è la fase di un algoritmo genetico dove vengono selezionati gli individui che faranno da genitori per la nuova prole. È proprio in quest'op-

erazione che entra in gioco il concetto di fitness. Solitamente tra gli individui dell'intera popolazione vengono selezionati i migliori secondo il valore di fitness. Tre sono le tecniche di selezione più comunemente usate:

- Roulette wheel
- Ranking
- Tournament

Il primo modo di selezione è proporzionale alla fitness, dove nessun elemento ha probabilità pari a zero di essere selezionato, come nella roulette. Il secondo dipende dall'ordinamento degli individui, che comunque avviene tramite fitness. Nel terzo metodo, quello più usato, vengono scelti casualmente k individui dalla popolazione e di questi k elementi viene preso il migliore.

2.5 Riproduzione e crossover

Nella successiva fase entra in gioco la riproduzione o il crossover. Il crossover è la fase di incrocio tra 2 o più organismi. In quest'operazione vengono scelti dall'insieme degli organismi selezionati, tipicamente due individui alla volta. Questi due individui scambiano tra loro le parti di sequenze. Il numero di punti di taglio può variare. Le sequenze vengono allineate e nel primo caso viene scelto un unico punto di taglio dopo il quale vengono scambiate le seconde parti tra i due organismi. Nel secondo caso, quello del multi-point, si hanno più punti di taglio e quindi più parti di sequenze scambiate.

La riproduzione è la ricopiatura degli individui genitore dalla vecchia alla nuova generazione senza che tra questi avvenga l'operazione di crossover.

Data la probabilità di crossover P_{xo} , la riproduzione avviene secondo la probabilità $1 - P_{xo}$.

2.6 Mutazione

Generati i figli, avviene l'aggiunta di innovazione tramite la mutazione. La mutazione non è altro che una perturbazione dei valori contenuti nei geni. Con una certa probabilità, ogni valore della sequenza può essere modificato; nel caso di stringhe binarie, il valore viene modificato da 0 a 1 e viceversa.

In questo modo si ottengono elementi con sequenze nuove e quindi con cambiamenti nella fitness introducendo così nuovo materiale genetico.

2.7 Elitismo

L'operazione di elitismo è la semplice copia di elementi migliori dalla vecchia generazione alla nuova. Questo significa che, qualsiasi tipo di selezione venga effettuata, l'elitismo garantisce che gli individui che meglio soddisfano le richieste del problema alla generazione N , saranno presenti anche nella nuova generazione $N+1$. La scelta degli elementi avviene tramite l'ordinamento per valore di fitness e la selezione dei primi K individui.

Capitolo 3

Neuroevolution

3.1 Apprendimento rete

Neuroevolution è il termine utilizzato per descrivere la fase dell'apprendimento di una o più reti neurali tramite l'uso degli algoritmi genetici. L'idea è quella di evolvere una rete neurale apprendendo pesi delle connessioni. Si tratta di continuo testing tramite il test set sulla rete e quindi poi di continue generazioni dei valori che compongono i pesi delle connessioni fino a raggiungere il setup ottimale, il che porterà ad avere l'output desiderato per ogni valore del test set dato.

Questo modo di agire avviene identicamente anche nella realtà. Se si prende ad esempio una persona che vuole imparare una qualche tecnica di un'arte marziale, al momento di iniziare gli verranno dette le regole, gli obiettivi e l'esercizio da fare per apprendere. Inizialmente questa persona avrà movimenti lenti, si dimenticherà dei passi, subirà dei colpi da parte dell'avversario ottenendo così una valutazione bassa dalla parte del suo maestro. Continuando a ripetere la tecnica per giorni correggendo ogni volta i movimenti, la persona in questione si troverà col conoscere quella tecnica, ma non solo, si troverà in grado di apprendere più facilmente tecniche simili.

3.2 Tecniche usuali

Le tecniche solitamente utilizzate nell'apprendimento dei pesi si basano sul gradiente dell'errore. La più nota è sicuramente la Δ -rule. I pesi delle connessioni entranti negli output vengono aggiornati direttamente secondo l'errore sull'output. Le connessioni entranti nelle unità nascoste, vengono aggiornate retropropagando l'errore degli output.

La prima fase in questo processo è il calcolo dell'errore tra l'output della rete e il target da raggiungere. Se d è il target e net è l'output della rete, allora l'errore sarà così definito:

$$E = \frac{1}{2}(d - net)^2 = \frac{1}{2}\left(d - \sum_{i=1}^n w_i x_i\right)^2 \quad (3.1)$$

La successiva fase è quella di calcolare la derivata parziale dell'errore rispetto a ogni peso

$$\frac{\partial E}{\partial w_i} = (net - d) \frac{\partial net}{\partial w_i} = (net - d)x_i \quad (3.2)$$

e di usare quel valore per aggiornare ogni peso usando:

$$\Delta w_i = \eta \left(d - \sum_{i=1}^n w_i x_i\right) x_i = \eta (d - net) x_i \quad (3.3)$$

dove η è la learning rate e la modifica dei pesi avviene in direzione opposta rispetto a $\frac{\partial E}{\partial w_i}$.

Nel caso non-lineare, si avrà y al posto di net . Tale valore rappresenta il risultato della funzione non lineare:

$$y = \sigma(net) \quad (3.4)$$

Ricordando che la funzione non-lineare chiamata sigmoide è così definita:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.5)$$

Quindi invece di

$$\frac{\partial E}{\partial w_i} = (net - d) \frac{\partial net}{\partial w_i} \quad (3.6)$$

si avrà

$$\frac{\partial E}{\partial w_i} = (y - d) \frac{\partial y}{\partial net} \frac{\partial net}{\partial w_i} \quad (3.7)$$

La derivata $\frac{\partial y}{\partial net}$ è data da:

$$\frac{\partial y}{\partial net} = \frac{\partial \sigma(net)}{\partial net} = \sigma(net)(1 - \sigma(net)) \quad (3.8)$$

Infine il gradiente sarà:

$$\frac{\partial E}{\partial w_i} = (y - d) \sigma(net)(1 - \sigma(net)) x_i \quad (3.9)$$

In questo progetto, per aggiornare i pesi, vengono utilizzati gli algoritmi genetici. Questa tecnica, come abbiamo visto, ha un approccio totalmente diverso.

3.3 Limitazioni

Alcune limitazioni delle tecniche descritte sono date dai minimi locali, overfitting e/o underfitting.

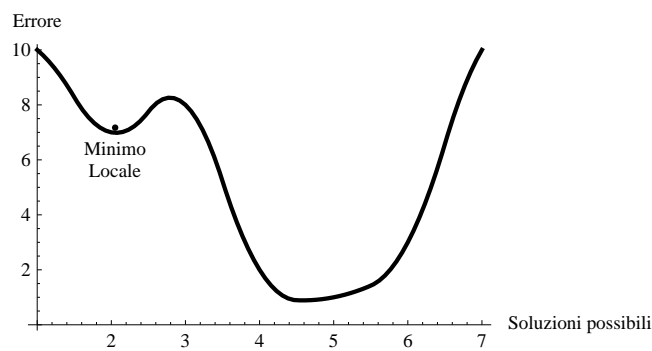


Figura 3.1: Grafico delle soluzioni possibili ed errore corrispondente

I minimi locali fanno sì che l'errore si stabilizzi su un certo valore che non è ottimale. Prendendo in considerazione il grafico che rappresenta le soluzioni

possibili e l'errore corrispondente, si nota che dopo N iterazioni si potrebbe arrivare in un minimo locale senza giungere quindi al minimo globale, detto anche ottimo.

L'underfitting è una situazione nella quale la rete non è riuscita a dare i risultati voluti, in quanto la rete non è allenata a sufficienza. In termini pratici significa che si ottengono errori troppo alti. L'overfitting descrive la situazione opposta dove una rete si adatta alle caratteristiche del training set ma non potrà prevedere un comportamento generico, quindi avviene quando la rete è stata allenata troppo su un specifico training set.

3.4 GA: individui come pesi

L'utilizzo di una popolazione di individui definiti come insieme dei pesi di una rete neurale rende il GA meno sensibile ai minimi locali.

Invece di modificare/aggiornare una rete singola vengono presi in considerazione N individui, ognuno con la propria configurazione dei pesi. In questo modo vengono portate avanti più soluzioni parallelamente.

Parte II

Studio e Implementazione

Capitolo 4

NEAT

4.1 Che cosa è NEAT

NEAT (NeuroEvolution of Augmenting Topologies) è un algoritmo che viene utilizzato per evolvere una rete neurale modificando, oltre che i pesi, anche la struttura topologica della rete.

È un'idea che nasce nel 2002 da Ken Stanley e Risto Miikkulainen. Tale idea introduce nuove nozioni per risolvere problemi complessi.

4.2 Principi di funzionamento

L'algoritmo si basa sulla neuroevolution tradizionale dove una rete neurale viene evoluta tramite gli algoritmi genetici. Le differenze stanno nel fatto che viene evoluta anche la struttura della rete e questo fattore permette di trovare strutture adatte senza doverle definire a priori. Normalmente in un algoritmo di neuroevolution viene scelta a priori la struttura della rete in base al problema da risolvere, nel caso di NEAT, la struttura viene scelta minima e man mano viene ampliata in automatico secondo le necessità.

4.3 Problemi affrontati nell'ideazione

Diversamente dalla neuroevolution standard, in NEAT gli individui hanno lunghezza variabile durante tutto il processo di evoluzione. Questa dinamicità permette di trovare soluzioni per un problema di cui sia possibile descrivere la soluzione senza conoscere la struttura a priori.

Il principale problema nell'applicare questa tecnica è che gli individui tendono a crescere di dimensione al di là del necessario. Questo problema è conosciuto come bloat. NEAT cerca di limitarlo partendo con strutture semplici e facendo crescere le strutture degli individui molto lentamente.

4.4 Innovation numbers

Per ovviare ai problemi sopra descritti, gli ideatori hanno aggiunto un parametro identificativo per ogni gene, tale parametro viene chiamato *innovation number*. Questo ID descrive un gene, un carattere o ancor meglio una certa struttura della rete. Se questa struttura compare in più individui, essa avrà lo stesso innovation number.

È stato introdotto per permettere il crossover tra soli geni omologhi, cioè tra strutture caratteriali identiche.

Per far capire meglio al lettore il concetto di geni omologhi, e per introdurre il significato di geni disgiunti e geni in eccesso, si noti la figura 4.1 dove ogni riquadro è un gene e dove il numero in ogni riquadro rappresenta l'innovation number di quel specifico gene.

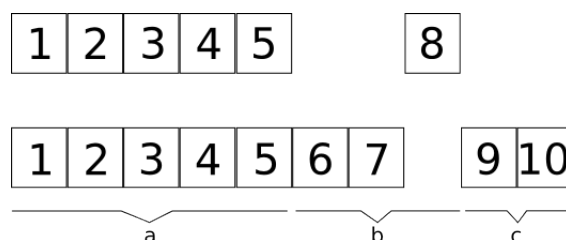


Figura 4.1: Due individui allineati secondo innovation number

La sezione *a* mostra le sottosequenze in comune, questo è chiaro vista la

presenza degli stessi innovation numbers. La sezione b è l'insieme di geni disgiunti e c la sequenza di geni in eccesso. La differenza tra questi due insiemi sta nel fatto che i geni disgiunti sono tutti quelli che non compaiono in entrambi individui fino alla lunghezza della più corta sequenza, il resto viene appunto chiamato l'insieme di geni in eccesso.

4.5 Speciazione

Un'altra caratteristica di questo algoritmo è la suddivisione degli individui in specie. Questa soluzione permette la ricerca di soluzioni in molte direzioni. Dato che si è detto che l'algoritmo evolve anche la struttura della rete, questo significa che dopo N generazioni evolutive potrebbe comparire una struttura topologica molto diversa dall'iniziale, e in un caso del genere questo nuovo organismo si troverà in una nuova specie.

4.6 Funzione di distanza

Per avere una misura di quale individuo fa parte di quale specie, si utilizza una funzione di distanza. Sapendo che ogni gene contiene memorizzato il proprio innovation number, la funzione di distanza viene definita come:

$$\delta = \frac{c_1 \cdot e}{n} + \frac{c_2 \cdot d}{n} + c_3 \cdot \bar{w} \quad (4.1)$$

Dove

- c_1, c_2, c_3 sono delle costanti
- e , il numero di geni in eccesso, cioè la differenza algebrica tra i massimi valori di innovation number di due individui
- d , il numero di geni disgiunti che equivale al numero di geni che compare in un individuo e non nell'altro

- \bar{w} è la media delle differenze dei pesi tra geni omologhi, cioè tra i geni che hanno lo stesso innovation number in entrambi gli individui.

4.7 Composizione della popolazione

La popolazione è l'insieme di individui suddivisi in diverse specie. La struttura è minima all'inizio, composta da tutti i nodi di input connessi a tutti i nodi di output quindi al momento dell'inizializzazione, l'insieme è composto da un'unica specie. Questo perché inizialmente la popolazione viene creata con tutti individui di struttura identica. Durante l'evoluzione la struttura di un individuo crescerà in modo diverso dagli altri e questo definirà la differenza tra varie specie. Il numero della specie di appartenenza viene tenuto memorizzato nel genoma.

4.8 Fitness e shared fitness

Tra le caratteristiche inserite in un individuo, ci sono anche i valori di fitness e di fitness condivisa. Il primo è stato descritto precedentemente, la qualità dell'individuo, il secondo è un valore che varia al variare della dimensione della specie oltre che al variare della fitness. Il valore di fitness condivisa è seguentemente definito:

$$f'_i = \frac{f_i}{\sum_{j=1}^n sh(\delta(i, j))} \quad (4.2)$$

dove

- f_i è la fitness dell'individuo,
- mentre la funzione di sharing sh vale 0 se la distanza $\delta(i, j)$ si trova sopra la soglia δ_t , altrimenti $sh(\delta(i, j))$ vale 1

In questo caso $\sum_{j=1}^n sh(\delta(i, j))$ si riduce al numero di individui della stessa specie.

La shared fitness è stata introdotta per mantenere la diversità nella popolazione e per permettere alle nuove specie di evolversi.

4.9 Come avviene la selezione

Per generare la nuova prole, dalla popolazione iniziale avviene una selezione degli individui migliori. Questa selezione è di tipo Ranked, vengono scelti il 20% dei migliori elementi nella popolazione. Come descritto in precedenza, tipicamente la misura utilizzata per definire quali elementi sono migliori di altri è la fitness.

In caso di questo algoritmo viene utilizzata la shared fitness nel definire l'ordine di selezione. L'intera popolazione viene unita in un unico insieme e riordinata secondo il valore di shared fitness e infine vengono scelti i primi 20% di questo insieme riordinato.

Questa misura è stata introdotta per salvaguardare le nuove specie. Alla prima apparizione di una specie, l'elemento o gli elementi facenti parte potrebbero avere un valore di fitness basso e quindi rischierebbero di non riapparire nella nuova generazione se la misura di selezione fosse data soltanto dalla fitness. Per ovviare al problema si usa quindi la shared fitness che dipende sia dalla fitness sia dal numero di elementi della specie. In questo modo viene incoraggiata l'evoluzione di specie anche se poco numerose e quindi vengono ricercate più soluzioni contemporaneamente mantenendo alta la diversità nella popolazione.

L'insieme di elementi selezionati verrà poi utilizzato per le successive fasi di generazione della nuova popolazione spiegate nelle sezioni successive.

4.10 Crossover tra individui

Normalmente il crossover avviene tra individui di pari lunghezza, ma nel caso di NEAT, gli individui potrebbero avere lunghezza differente per via della diversa per struttura.

Questo problema viene risolto sempre tramite innovation numbers, che rendono facile l'identificazione univoca dei geni.

Per l'incrocio tra due individui vengono presi in considerazione solamente gli innovation numbers che compaiono in entrambi, rendendo così possibile il crossover solamente tra geni omologhi. Una volta identificati, partendo dal

primo vengono scambiati i geni con una definita probabilità fino a generare i due nuovi individui. I geni restanti (disgiunti e in eccesso) dei due individui vengono scelti o dal genitore con miglior valore di fitness oppure vengono selezionati tutti e uniti in un'unica sequenza.

Alcune versioni di NEAT non applicano il crossover ritenendo che senza questo operatore, si ottengono comunque ottimi risultati con miglioramenti nelle performance. Il problema principale riguardante la performance è dovuto alle lunghezze differenti delle sequenze. Per eseguire l'operatore di crossover, bisogna prima identificare quali sono i geni omologhi, quali invece quelli disgiunti e quali in eccesso, ed eseguire infine lo scambio di geni. Con la presenza di innovation numbers viene abbastanza facile come operazione, ma richiede comunque uno sforzo computativo non trascurabile per ogni operazione di crossover.

4.11 Applicazione della mutazione

Una volta ottenuto un nuovo individuo, a questo viene applicato l'operatore di mutazione. Vi sono 3 operatori di mutazione diversi e vengono applicati secondo il seguente ordine.

4.11.1 Aggiunta di nuovi nodi

Un primo fattore di mutazione è l'aggiunta di nuovi nodi. Viene prima scelta una connessione da spezzare e viene inserito poi un nuovo nodo/neurone tra di essa. La connessione spezzata viene disabilitata tramite un flag apposito. Il peso della vecchia connessione viene riutilizzato nella connessione tra il primo nodo e il nuovo nodo, mentre il peso di valore 1 viene usato nel collegamento tra il nuovo nodo e il secondo nodo. Questa scelta dei pesi viene effettuata per non modificare troppo la linearità della rete neurale e quindi per non peggiorare drasticamente la fitness. Se si prendono un nodo di input e il neurone di output, il valore in uscita corrispondente sarà dato da:

$$y_f = \sigma(x_{in}w_{in}) \quad (4.3)$$

dove x_{in} è il valore dato in input, mentre w_{in} è il peso della connessione tra il nodo di input e il neurone di output. Se si aggiunge un neurone chiamato h in mezzo ai due appena descritti, tenendo conto che il peso della connessione tra il nodo di input e il nodo h vale w_{in} e il peso tra il nodo h e il nodo di output vale 1, si ottiene:

$$y_h = \sigma(x_{in}w_{in}) \quad (4.4)$$

e quindi si avrà l'uscita finale della rete:

$$y_f = \sigma(y_h 1) = \sigma(y_h) \quad (4.5)$$

Dato che la funzione della sigmoide restituisce un valore compreso tra 0 e 1 e specificatamente se passato alla funzione un valore compreso in questo intervallo, l'output restituirà un valore prossimo a 0.5.

$$y_f = \sigma(y_h) = \sigma([0, 1]) \approx 0.5 \quad (4.6)$$

4.11.2 Nuove connessioni

Una seconda fase della mutazione è l'aggiunta di una nuova connessione. Data la struttura della rete, vengono selezionati due nodi non attualmente connessi tra loro e quindi poi collegati aggiungendo un peso generato casualmente.

Tra le possibili connessioni, si considerano anche quelle che sono state disabilitate in precedenza, ad esempio nella fase di aggiunta di un nuovo nodo. In tal caso viene cambiato il flag della connessione mentre il suo peso rimane quello memorizzato.

4.11.3 Perturbazione dei pesi

Come nei sistemi standard, la perturbazione dei pesi avviene modificando alcuni pesi delle connessioni secondo la probabilità $P_{wMutate}$. La perturbazione avviene sommando un valore random generato secondo la distribuzione normale $N[0, 1]$.

Capitolo 5

NCD

5.1 Introduzione

La Normal Compression Distance (NCD), introdotta da Faustino Gomez nel 2009, è una tecnica utilizzata per misurare la distanza tra due oggetti tramite la compressione. Il lavoro di Gomez si è basato sulla Normalized Information Distance (NID) che comprende anche lo studio sulla complessità di Kolmogorov.

5.2 Complessità di Kolmogorov

Lo studio del matematico russo Kolmogorov portò nel '65 a definire la complessità di un oggetto come misura del suo contenuto informativo e della sua casualità. In questa ricerca ha anche mostrato come misurare tale contenuto in rapporto a un altro oggetto.

Dato che esistono oggetti semplici e oggetti complessi, il problema è la molteplicità di modi in cui è possibile descrivere un oggetto. Questo perché un oggetto può avere una descrizione semplice in un linguaggio, ma non in un altro.

Con la teoria degli algoritmi, Kolmogorov ha scoperto che è possibile restringere l'intervallo delle possibili descrizioni ed è riuscito a definire la complessità come un concetto unico.

Dati due oggetti x e y , la complessità di Kolmogorov di x dato y è la lunghezza del più corto programma binario, eseguito su una macchina di Turing universale, che darà in output x dato l'input y .

5.3 Approssimazione con compressione

Una delle tecniche usuali per misurare la similarità tra due oggetti è quindi la Normalized Information Distance, dati due oggetti x e y sotto forma di sequenza binaria, si ha:

$$NID = \frac{\max(K(x|y), K(y|x))}{\max(K(x), K(y))} \quad (5.1)$$

dove $K(x|y)$, la complessità di Kolmogorov di x dato y , e $K(x)$ è la complessità di Kolmogorov di x dato nessun input. Utilizzando NID come misura di distanza tra due oggetti, il problema principale diventa la complessità di computazione perché $K(x)$ non è Turing-computabile.

Questa difficoltà viene affrontata quindi con la Normalized Compression Distance:

$$NCD(x, y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))} \quad (5.2)$$

dove $C(x)$ è la lunghezza di x compresso, e $C(xy)$ è la lunghezza della compressione di x e y concatenati.

Se si suppone che x e y siano lo stesso oggetto, $x = y$, dove un oggetto è un insieme di caratteri, la compressione $C(x)$ e la compressione $C(xy)$ sarà la stessa, quindi pure la loro lunghezza, perché ogni buon compressore tende a codificare una volta sola più copie della stessa sequenza. Questo comporterà ad avere:

$$NCD(x, y) = \frac{C(x) - C(x)}{C(x)} = \frac{0}{C(x)} = 0 \quad (5.3)$$

Se invece si suppone il caso contrario, cioè con due oggetti completamente disgiunti $x \cap y = \emptyset$, si avrà $C(xy)$ pari alla somma $C(x) + C(y)$, perché la concatenazione non porterà a nessuna compressione vista la diversità dei due oggetti.

Quindi si supponga per il momento che $C(x) > C(y)$, si ottiene:

$$NCD(x, y) = \frac{C(xy) - C(y)}{C(x)} \quad (5.4)$$

con la differenza $C(xy) - C(y) = C(x)$ si avrà:

$$NCD(x, y) = \frac{C(x)}{C(x)} = 1 \quad (5.5)$$

Si noti che l'intervallo operativo di questa funzione di distanza è compreso tra 0 e 1. La comodità di NCD sta nella facilità di applicazione dell'algoritmo a qualsiasi tipo e forma di oggetti.

La nostra ipotesi sarà quindi che NCD sia una misura di diversità migliore di quella utilizzata nella versione originale di NEAT per giudicare la diversità di due specie. Quindi l'idea principale di questo progetto è sostituire questa metrica con quella attualmente utilizzata dall'algoritmo, definita nell'equazione 4.1.

Uno dei vantaggi è la facilità di implementazione, perché è sufficiente prendere gli individui come stringhe e comprimerli tramite un compressore canonico, per esempio *Compress[]* presente nella libreria di Mathematica.

Dopodiché basterà utilizzare le lunghezze delle stringhe compresse nell'equazione 5.2. Dopo l'applicazione verranno studiati e confrontati i risultati di tale modifica con quella originale.

Capitolo 6

Riprogettazione

6.1 Come è strutturato il codice

Il codice è scritto in linguaggio Mathematica ed è organizzato in sezioni facilmente individuabili. È scritto e organizzato secondo la logica di esecuzione, partendo dalla definizione dei parametri, delle funzioni di supporto fino alla sezionata definizione di ogni fase dell'algoritmo come descritto nel capitolo 7.

Le sezioni presenti in questo capitolo rappresentano le modifiche organizzative di questa versione dell'algoritmo rispetto a NEAT originale descritto nel capitolo 4.

6.2 Struttura di un individuo

Un individuo è definito come una lista di 4 elementi, a loro volta liste. **La prima sottolista** è l'insieme di parametri che caratterizzano la qualità dell'organismo. Contiene il valore di fitness e il valore di shared fitness. La lista è così composta:

$$\{\text{fit}, \text{sh_fit}\}$$

esempio: {3.62, 0.11}

Questa lista verrà chiamata *info* d'ora in poi. Nell'algoritmo originale questo insieme di parametri contiene anche il numero di specie e il valore di compatibilità con la specie. In questa versione non vengono utilizzati.

Il numero di specie non è più necessario perché definito dalla posizione della specie nella popolazione, mentre la compatibilità viene calcolata ed utilizzata una sola volta e quindi diventa futile tenerla memorizzata.

La seconda sottolista contiene tre valori: il numero di nodi di input, il numero di nodi di output e il numero di unità nascoste. Questi valori vengono usati sia per avere in chiaro il numero preciso di ogni tipo di nodo, sia per la loro numerazione. La lista è così implementata:

$$\{\text{in, out, hid}\}$$

$$\text{esempio: } \{3, 1, 7\}$$

Nel caso dell'esempio si hanno tre ingressi, un'uscita e sette unità nascoste. I nodi di input saranno numerati da 1 a 3, l'output avrà il numero 4 e le unità nascoste saranno composte dai nodi 5, 6, 7, 8, 9, 10 e 11.

Durante l'inizializzazione, ogni individuo viene creato con struttura minima, dove ogni nodo di input è collegato con ogni nodo di output. Nel caso dell'esempio sopra, la lista sarà così definita:

$$\{3, 1, 0\}$$

La terza lista contiene la sequenza di geni, dove ogni singolo gene rappresenta un collegamento della rete. Ogni collegamento o link è costituito dal peso e dai nodi di arrivo e di partenza. Questa sequenza permetterà poi la costruzione della rete neurale. La struttura è così descritta:

$$\{\{\text{orig, dest}\}, w\}$$

Una sequenza tipo potrebbe essere la seguente:

$$\{\{\{4, 1\}, 3.14\}, \{\{4, 2\}, 2.71\}, \{\{4, 3\}, 1.11\}\}$$

L'ultima lista dell'individuo contiene la lista di esecuzione, composta da una sequenza numerica dove ogni numero è un nodo della rete.

Dato che la topologia cambia per ogni individuo, è necessario mantenere esternamente l'ordine di esecuzione per ogni nodo della rete e giungere all'output. Il problema principale deriva dalla dinamicità della struttura. Per capire meglio a che cosa serve e come viene utilizzata si può immaginare una rete con unità nascoste ottenuta dopo N generazioni. Estruendo la lista di esecuzione si avrà la sequenza corretta per attivare i nodi della rete. Prendendo ad esempio la seguente lista di esecuzione:

esempio: {5, 6, 8, 7, 4}

verranno calcolati in ordine gli output dei nodi 5, 6, 8, 7 e 4, considerando l'ultimo come output della rete.

Ecco un esempio di un individuo completo:

```

{
  {1000.94, 100.094}, (*info*)
  {3, 1, 0}, (*nodi*)
  {
    {{{4, 1}, -0.533095}, (*link tra 1 e 4*)
    {{4, 2}, 0.989597}, (*link tra 2 e 4*)
    {{4, 3}, 0.478845}}, (*link tra 3 e 4*)
    {}}, (*lista di link disabilitati*)
  {4} (*lista di esecuzione*)
}

```

6.3 Struttura della popolazione

La popolazione, come detto in precedenza, è l'insieme dei genomi organizzati in specie. La struttura dati utilizzata è la lista. La popolazione quindi è una lista di specie. Ogni specie a sua volta è una lista di individui.

Durante l'inizializzazione tutti gli individui fanno parte di un'unica specie dato che sono strutturalmente uguali. Inizialmente la popolazione è dunque

così strutturata:

$$\{\{\text{individuo1}\}, \{\text{individuo2}\}, \{\text{individuo3}\}, \dots\}$$

Dopo N generazioni si ottiene una popolazione con K specie:

$$\{\{\{\text{individuo1}\}, \{\text{individuo2}\}\}, \{\{\text{individuo3}\}\}, \{\dots\}\}$$

Le parentesi evidenziate rappresentano i delimitatori di specie.

6.4 Estensione dell'insieme dei geni

Nella versione originale un gene conteneva anche il valore di innovation number e un valore booleano che indicava se il gene era disabilitato o no. In questa versione questi due parametri vengono definiti più semplicemente.

La lista che contiene la sequenza genomica viene suddivisa in 2 sottoliste, dove la prima contiene i geni abilitati, la seconda quelli disabilitati. In questo modo un gene viene spostato da una all'altra a seconda della scomparsa/ricomparsa di quel gene.

6.5 Innovation numbers

Gli innovation number venivano inseriti come valore in ogni gene per identificare quel specifico tipo di gene, quella specifica struttura. La differenza di struttura dipendeva solamente dalla connessione e quindi dalla sottolista che indicava i nodi di arrivo e partenza presente in ogni gene. Per non avere copie multiple di valori che identificavano la stessa struttura, si è pensato di rendere questa operazione globale.

Alla comparsa di ogni nuova struttura, l'innovation number per quel specifico tipo di collegamento viene prima incrementato e poi memorizzato globalmente, e alla ricomparsa di quella struttura viene fatto solamente il controllo se essa è presente o no. Se si vuole scoprire quale innovation

number ha una certa connessione, è sufficiente chiamare la funzione apposita passando la lista del link che si vuole controllare.

6.6 Parametri

Prima dell'utilizzo dell'algoritmo, vanno impostati alcuni parametri statistici e non. Tutti i parametri utilizzati hanno un valore di default:

- POPSIZE (default: 150): viene usato per definire la dimensione della popolazione;
- MAXW (default: 4): è il valore massimo assoluto utilizzato nella funzione di assegnazione del peso iniziale di una connessione, l'intervallo dei valori possibili sarà quindi dato da $[-MAXW, +MAXW]$;
- PXO (default: 0.5): è la probabilità con la quale avviene lo scambio di geni durante l'operazione di crossover, mentre $1-PXO$ è la probabilità di riproduzione.;
- PWMUTATE (default: 0.8): è la probabilità che indica se la mutazione del peso deve avvenire oppure no;
- PWPERTURB (default: 0.9): viene usato per definire con quale probabilità al peso attuale di una connessione viene sommato un valore random dato dalla distribuzione normale $N[0, 1]$;
- PNMUT (default: 0.4): rappresenta la probabilità di aggiunta di un nuovo nodo;
- PCMUT (default: 0.3): è la probabilità di aggiunta di una nuova connessione;
- C1 (default: 1), C2 (default: 1), C3 (default: 0.4) e N (default: numero arbitrariamente grande): sono le costanti utilizzate nella calcolo della distanza descritta nella sezione 4.1.

- δ_t (default: 0.05 o 3): è la soglia di separazione tra specie utilizzata nella fase di speciazione. Il primo valore di default è usato per la NCD, il secondo per la metrica standard.
- PSURVT (default: 0.2): è la percentuale utilizzata nella fase di selezione e indica la percentuale di individui da estrarre per le successive fasi.
- INNUM (default: 0) e Gen (default: 0): sono dei contatori, il primo per gli innovation numbers, il secondo per il conteggio delle generazioni.

I valori di default sono presi dal paper, quindi già ottimizzati. Durante il setup delle variabili, è necessario modificare la soglia δ_t se si cambia la metrica. È importante notare che la NCD non necessita di parametri altri che il threshold, contro i 4 utilizzati per il tuning della distanza standard (C1,C2,C3 e N).

Capitolo 7

Le fasi dell'algoritmo

7.1 Inizializzazione della popolazione

La popolazione viene generata tramite la funzione:

```
popGen[in_ , out_ , size_ , fitness_]
```

con in ingresso il numero di nodi di input, di output, dimensione della popolazione e la funzione di fitness. Restituisce una popolazione formata da un'unica specie dove ogni individuo è costituito da una struttura topologica minima. Viene calcolata la fitness e la shared fitness per ogni individuo. Il numero di hidden nodes sarà a zero e la lista di esecuzione sarà composta dai soli nodi di output.

7.2 Selezione

Durante questa fase tutti gli individui vengono globalmente ordinati secondo il valore di fitness condivisa. Di questo insieme vengono presi i primi $PSurvT$, percentuale definita nella sezione 6.6. Questo tipo di selezione viene chiamato *Truncated*.

La funzione di selezione è quindi impostata sul tipo di selezione truncated nel seguente modo:

```

selection := truncated
truncated [pop_, survTHold_ : PSURVT] :=
  Take[
    SortBy[
      Flatten [pop, 1], #1[[1, 2]] &],
    -Floor [survTHold POPSIZE]]

```

7.3 Elitismo

Come è stato descritto nel capitolo sugli algoritmi genetici, quest'operazione seleziona e copia i migliori individui secondo fitness.

L'elitismo è necessario per assicurarci di non perdere gli elementi migliori della precedente generazione. Questa perdita può avvenire durante la selezione in quanto basata sulla shared fitness.

7.4 Crossover

Il crossover è una fase delicata e allo stesso momento complessa. Prima di eseguire l'incrocio, bisogna selezionare due organismi della stessa specie, dopodiché viene selezionato soltanto l'insieme di geni in comune e quindi viene effettuato il crossover. Per individuare le strutture in comune, si utilizza l'innovation number. Il resto dei geni, non facenti parte dell'insieme comune, vengono selezionati dall'individuo con fitness maggiore.

Quest'operazione è disattivabile per ottenere simili risultati con maggiori performance. È sufficiente impostare $PXO = 0$.

7.5 Mutazione

Quest'operazione è suddivisa in 3 fasi con 3 diverse probabilità di essere eseguite.

La prima fase è la possibilità di aggiungere un nuovo nodo. Per fare ciò viene selezionata una connessione esistente e viene resa inattiva, cioè il gene viene spostato nella lista di link disattivati. Si supponga di voler disattivare

il seguente link, che verrà chiamato L1:

$$l1: \{\{\text{dest}, \text{orig}\}, w\}$$

dando uno sguardo alla completa lista dei geni suddivisa in geni attivi e in geni inattivi si ha:

$\{\{L1, L3, L4\}, \{L2\}\}$ prima della disattivazione di L2
 $\{\{L3, L4\}, \{L1, L2\}\}$ dopo la disattivazione di L2

Dopo l'operazione di disattivazione, vengono creati un nuovo nodo e due nuove connessioni. Il vecchio valore peso di l1 viene estratto e inserito nel primo link chiamato nL1, creato tra il nodo *orig* e il nuovo, mentre per il secondo link, denominato nL2, viene usato il peso 1 ottenendo la seguente configurazione:

$$nl1: \{\{\text{new}, \text{orig}\}, w\}$$

$$nl2: \{\{\text{dest}, \text{new}\}, 1\}$$

Le due connessioni appena create vengono inserite nella lista di geni attivi ottenendo:

$\{\{L1, L3, L4\}, \{L2\}\}$ prima della disattivazione di L2
 $\{\{L3, L4\}, \{L1, L2\}\}$ dopo la disattivazione di L2
 $\{\{L3, L4, nL1, nL2\}, \{L1, L2\}\}$ dopo l'inserimento dei nuovi geni

Infine viene aggiornata la lista di esecuzione aggiungendo in posizione antecedente al secondo nodo della connessione da spezzare.

La funzione che ne deriva è:

```
addNodeMutate[genome_] :=
With{newNodes = addNode[genome][[2]],
  chosenOne = (RandomChoice@genome[[3, 1]])[[1]]},
Join[
  {genome[[1]], newNodes},
  {spanLink[Total@newNodes, chosenOne, genome[[3]]]},
  {Insert[genome[[4]], Total@newNodes,
    Position[genome[[4]], chosenOne[[1]]]}]}
```

La seconda operazione di innovazione è l'aggiunta di una nuova connessione.

```
addConnMutate[genome_] := With{newConn = selectOne[genome]},
If[Length@newConn != 0,
  {genome[[1]], genome[[2]],
  If[Length@Select[genome[[3, 2]], #[[1]] == newConn &] != 0,
    enableLink[genome[[3]], newConn],
    addLink[genome[[3]], newConn]
  ], renewExecList[genome[[4]], newConn]},
genome]]
```

Preso l'insieme di connessioni possibili, viene selezionata arbitrariamente una connessione da ristabilire. Questo link avrà un peso generato casualmente. Prima dell'aggiunta di una nuova connessione, viene effettuato il controllo se essa già esiste tra le connessioni disabilitate, in tal caso viene ripescata dalla lista di link disabilitati e quindi riinserita nella lista di geni abilitati. Anche dopo questa mutazione viene aggiornata la lista di esecuzione.

La terza ed ultima fase è la mutazione dei pesi che avviene tramite le seguenti funzioni:

```
mutateProcess := RandomReal[NormalDistribution[0, 1]]
mutateWeights[links_] :=
  RandomChoice[{PWPERTURB, 1 - PWPERTURB} ->
    {#[[1]] -> (#[[2]] + mutateProcess),
    #[[1]] -> #[[2]]}] & /@ links
wMutate[genome_] :=
  {genome[[1]],
  genome[[2]], {mutateWeights[genome[[3, 1]], genome[[3, 2]]},
  genome[[4]]}
```

- *mutateProcess* restituisce un valore casuale pescato dall'insieme restituito dalla distribuzione gaussiana (o normale).
- *mutateWeights* effettua la mutazione, cioè la chiamata a *mutateProcess* su ogni gene (o connessione) con probabilità *PWPERTURB*.
- Infine *wMutate* è la funzione principale applicata sull'intero individuo, funzione che andrà a chiamare *mutateWeights* solamente sulla parte

contenente le connessioni, la terza lista dell'individuo. Al termine verrà restituito l'individuo con i pesi perturbati.

7.6 Calcolo della Fitness

Dopo il processo di mutazione si avrà un nuovo organismo da inserire nella popolazione, ma prima di quest'operazione avviene il calcolo della fitness. La funzione di fitness dipende dal problema dato, quindi va definita ed assegnata solamente prima di una simulazione e non prima dell'utilizzo dell'algoritmo NEAT.

Nella *Parte III* contenente i risultati dei test, sono definite le funzioni di fitness utilizzate.

7.7 Speciazione

Nel questo progetto, viene messa a confronto la metrica standard di NEAT (sez. 4.1) con la NCD. Vengono presi due individui e resi stringhe, vengono poi compressi separatamente da una parte e prima concatenati e poi compressi dall'altra. Vengono quindi confrontate le differenze.

La funzione di distanza è quindi così definita:

```
distance[ind1_, ind2_] :=
With[{a = Compress[ToString@ind1],
      b = Compress[ToString@ind2],
      c = Compress[ToString@ind1 <> ToString@ind2]},
  Abs@N@((Length@ToCharacterCode@a - Length@ToCharacterCode@b)/
    Length@ToCharacterCode@c)]
```

Per fare un esempio d'uso, si supponga di estrarre due individui da una popolazione e di valutarle come stringhe:

```
a = {{1002.09, ..., -0.949337}}, {3}};
b = {{1003.53, ..., 1.812211}}, {3}};
c = a<>b; (*a concatenato a b*)
```

Le stringhe una volta compresse avranno le seguenti lunghezze:

- `SizeOf@Compress@a` = 118
- `SizeOf@Compress@b` = 110
- `SizeOf@Compress@c` = 154

La funzione di distanza NCD (equaz. 5.2) restituirà 0.0519481. Se per esempio la soglia δ_t dovesse essere pari a 0.1, il valore ottenuto in quest'esempio indicherebbe che i due individui fanno parte della stessa specie.

Durante la speciazione di un individuo, viene effettuato un confronto tra questo e un elemento preso in modo casuale da ogni specie. Quindi ad ogni nuovo elemento generato vengono eseguite N chiamate alla funzione di distanza utilizzata quante sono le specie. Se la distanza calcolata è minore di una certa soglia predefinita δ_t allora i due individui fanno parte della stessa specie. In caso contrario il processo continua. Se non viene trovata la specie di appartenenza, ne viene creata una nuova, contenente solo l'individuo.

7.8 Calcolo della Fitness condivisa

Il calcolo della fitness condivisa avviene tramite le seguenti funzioni:

```
setShFitness [pop_] :=
    speciesShFit /@ pop
speciesShFit [species_] :=
    ReplacePart [# , {1, 2} -> (#[[1, 1]] / Length@species)] & /
    @ species
```

La *setShFitness* applica la funzione *speciesShFit* su ogni specie della popolazione *pop*. È necessario chiamare questa funzione solo al termine della generazione di una popolazione perché la formula della shared fitness dipende dal numero di elementi della specie dell'individuo sul quale eseguire il calcolo.

La *speciesShFit* calcola il valore di shared fitness per ogni individuo presente nell'insieme *species* sfruttando il valore di fitness memorizzato nell'individuo e la dimensione della specie data dalla lunghezza della lista *species*.

7.9 Passaggio da genotipo a fenotipo

Si è parlato del processo evolutivo e del calcolo della fitness, ma nel dettaglio in questo algoritmo una qualsiasi funzione di fitness sfrutterà l'output della rete neurale. Essendo un'organismo in forma genomica si ha la necessità di trasformare questa sequenza di caratteri in una struttura che corrisponda a una rete neurale. Per fare ciò si ha una funzione chiamata *gen2phen*:

```
gen2phen [ genome_ ] :=
  { genome [[ 4 ] ] ,
    rules2sparse [ genome [[ 3 , 1 ] ] ] // Normal, genome [[ 2 ] ] }
```

Passato come parametro l'intero genoma, quindi l'intera lista di dati di un individuo, viene generata una rete neurale pronta all'esecuzione. Il fenotipo viene costruito dal genotipo permettendo la facile interpretazione in un calcolo dell'output di una rete neurale.

Il fenotipo ottenuto è una lista che contiene a sua volta 3 liste. La prima è la copia identica della lista di esecuzione data dal genotipo. La seconda sottolista è la matrice quadrata delle connessioni trasformata dalla lista dei geni, con valori dei pesi in caso di connessione esistente, con valore 0 se la connessione non è presente.

Nella tabella di seguito si può notare un esempio di come è composta una matrice delle connessioni. La prima riga della tabella raffigura i nodi di origine, mentre la prima colonna rappresenta i nodi di destinazione.

Esempio di matrice delle connessioni:

	In1	In2	Out3	Hid4	Hid5	Hid6
In1	0	0	0	0	0	0
In2	0	0	0	0	0	0
Out3	-0.374697	0.854101	0	1.23248	-1.92226	0.0846516
Hid4	1.04869	0.931845	0	0	0	0
Hid5	-0.565216	2.68438	0	0	0	0
Hid6	0	0.576504	0	0	0	0

La terza sottolista è la copia identica della lista delle quantità dei nodi proveniente dal genoma, cioè la seconda sottolista di un individuo. Queste tre componenti permetteranno il calcolo della rete in modo vettoriale, metodo che viene descritto a fondo nella sezione successiva.

7.10 Attivazione della rete

Per far attivare la rete si utilizza la funzione

```
netSol[phenotype_, data_]
```

Tale funzione richiede come parametri il fenotipo e il vettore di input. La prima operazione eseguita è la normalizzazione del vettore di input che d'ora in poi verrà chiamato vettore degli stati. Il vettore è lungo quanti sono gli input, per le operazioni successive è necessario renderlo di lunghezza uguale alla cardinalità della matrice del fenotipo. Con un'operazione di padding si aggiungono zeri in fondo alla sequenza e si passa da un vettore iniziale

$$\{In_1, In_2, \dots, In_k\}$$

a un vettore di dimensione identica alla cardinalità della matrice dell'individuo.

$$\{In_1, In_2, \dots, In_k, 0, 0, \dots, 0\}$$

Continuando l'esempio della sezione precedente, il vettore $\{1, 1\}$ diventa $\{1, 1, 0, 0, 0, 0\}$

Data la lista di esecuzione, si utilizzano quindi i suoi valori come indice. Questo indice, d'ora in poi chiamato I , rappresenta la riga della matrice delle connessioni W e la posizione dove salvare il valore ottenuto dalla rete.

La riga I della matrice W moltiplica quindi il vettore degli stati e ne memorizza il risultato nel vettore stesso in posizione I . In questo modo il vettore degli stati viene aggiornato man mano che viene eseguito il prodotto

scalare aggiungendo gli output dei neuroni intermedi. L'operazione termina all'esaurimento della lista di esecuzione.

Infine usando la terza componente del fenotipo, cioè la lista delle quantità dei nodi, si estraggono gli elementi dal vettore ottenuto. Questi elementi saranno dati dalla posizione $In + 1$ alla posizione $In + Out$ dove In e Out sono il numero totale degli input e degli output della rete.

Tornando all'esempio precedente, si hanno:

- $\{1, 1\}$ uno dei possibili vettori di input
- $\{1, 1, 0, 0, 0, 0\}$ il vettore degli stati
- La matrice delle connessioni W

	In1	In2	Out3	Hid4	Hid5	Hid6
In1	0	0	0	0	0	0
In2	0	0	0	0	0	0
Out3	-0.374697	0.854101	0	1.23248	-1.92226	0.0846516
Hid4	1.04869	0.931845	0	0	0	0
Hid5	-0.565216	2.68438	0	0	0	0
Hid6	0	0.576504	0	0	0	0

- $\{4, 5, 6, 3\}$ la lista ordinata di esecuzione.
- $\{2, 1, 3\}$ le quantità degli input, degli output e delle unità nascoste.

L'operazione eseguita è la moltiplicazione del vettore di input normalizzato per la riga I della matrice delle connessioni. Il risultato del prodotto scalare viene inserito nel vettore di input in posizione I , dove $I \in \{4, 5, 6, 3\}$. Al termine dell'esecuzione si avrà il seguente stato finale:

$$\{1, 1, 0.345393, 0.999939, 0.999969, 0.944005\}$$

Dal quale verrà estratto il vettore finale rappresentante gli output, in questo caso solo il terzo elemento:

$$Output = \{0.345393\}$$

Per definire quali elementi del vettore degli stati fanno parte dell'output, si utilizza la lista delle quantità dei nodi che in questo caso sarà $\{2, 1, 3\}$. I nodi di output saranno dati da $In + 1$ fino a $In + Out$ che in questo caso corrisponderà all'intervallo $[2 + 1, 2 + 1]$, cioè al terzo elemento.

Parte III

Risultati

Organizzazione

Il rapporto sui test è suddiviso in capitoli, ognuno dei quali descrive a fondo il problema, la propria funzione di fitness, la simulazione e i risultati ottenuti. A livello pratico, ogni test carica l'algoritmo NEAT separatamente, mantenendo alta la modularità e flessibilità del progetto. Ogni simulazione si interfaccia con NEAT tramite:

```
popGen[in_ , out_ , size_ , fitFn_]  
generation [pop_]
```

La prima per generare la popolazione definendo il numero di input e di output, la dimensione della popolazione e la funzione di fitness per il problema dato. La seconda funzione viene usata per creare la nuova generazione $N+1$, ove il parametro di input è la popolazione alla generazione N .

A ogni generazione vengono salvati i dati necessari per la generazioni di grafici e statistiche.

Per ogni problema, abbiamo generato statistiche relative all'uso della distanza standard nella speciazione, o della NCD al suo posto. A ogni individuo sono stati garantiti un numero limitato di timesteps per risolvere il problema. Si è lavorato con popolazioni di 250 individui, con ogni run della durata di 150 generazioni. È da notare come quest'ultimo valore sia troppo basso per studiare approfonditamente le dinamiche dell'algoritmo, ma il nostro obiettivo è stato quello di verificare la plausibilità della nostra ipotesi per favorire e indirizzare successivi studi in merito. Vedremo comunque che le differenze diventano evidenti già da questo numero di generazioni. Per ogni test, abbiamo fatto girare l'algoritmo su un problema per 40 run per ogni distanza di speciazione. Le successive statistiche ricavate sono le medie sui

run dei valori di media e deviazione standard della caratteristica studiata sulla popolazione, in questo caso individuo migliore (best), fitness, numero di nodi intermedi e numero di link. Nel caso del best le deviazioni standard sono calcolate rispetto ai run. In tutti i grafici le deviazioni standard sono plottate come error bars sui valori medi corrispondenti.

Considerazioni generali

Per raggiungere il totale di 76x6 ore macchina necessarie al completamento di tutti i test, abbiamo utilizzato 6 macchine con diverse prestazioni sincronizzate sia a livello di codice che di risultati.

Il tempo di computazione era inizialmente 3 ordini di grandezza superiore, per cui è stata necessaria una fase di redesign e ottimizzazione soprattutto a livello di funzioni pesantemente matematiche, e di passaggi di strutture dati dovuti al linguaggio funzionale, per un totale di 32 ore uomo.

Al termine dell'esecuzione, i risultati sono stati riuniti ed esaminati separatamente da un programma apposito di analisi statistica, scritto sempre in mathematica. I dati esaminati trattano best element, fitness, numero di unità nascoste e connessioni, relativi a ogni generazione.

Capitolo 8

Problema dello XOR

8.1 Descrizione

Lo xor è un'operazione logica binaria con tavola di verità:

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

quindi rappresentando graficamente l'input in 2D come mostrato in figura 8.1, questo corrisponde a un problema di classificazione non linearmente separabile.

Data la semplicità di questo problema, lo useremo per introdurre i concetti che vedremo riproposti nei test principali nei prossimi capitoli.

Si vuole quindi classificare con una rete neurale l'input dello xor. L'uscita della rete avrà un valore che apparterrà all'intervallo $(0, 1)$ per ogni punto in input. L'obiettivo è quindi ottenere un output corrispondente alla tabella di verità dello xor.

Ovviamente in output non sarà possibile ottenere esattamente lo zero oppure l'uno, ma il target sarà quello di classificare l'output separando le classi tramite una soglia pari a 0.5, cioè esattamente a metà tra 0 e 1.

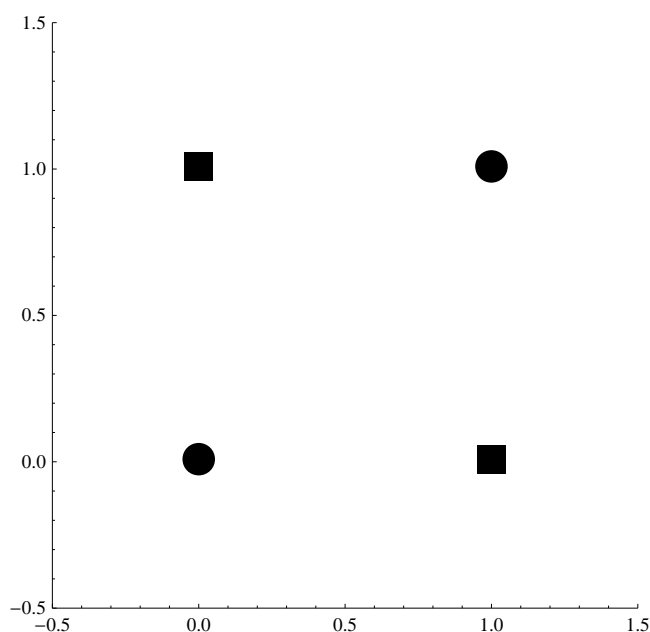


Figura 8.1: Il grafico dei punti in input dello xor

Quindi date le classi C_1 e C_2 e dato un input x , se l'output prodotto dalla rete neurale restituisce un valore ≤ 0.5 allora $x \in C_1$, altrimenti $x \in C_2$.

8.2 Funzione di fitness

Per ottenere una rete che risolva il problema verrà quindi evoluta la popolazione per N generazioni finché non si avrà un individuo il cui fenotipo classifichi correttamente tutti i punti.

Per fare ciò bisognerà separare i quattro punti, secondo il valore xor tra le due coordinate, in due classi che saranno identificate come classe 1 e classe 2. Alla classe 1 apparterranno gli input $(0, 0)$ e $(1, 1)$, mentre alla classe 2 gli input $(0, 1)$ e $(1, 0)$. Si definisce quindi che se l'output prodotto dalla rete neurale è minore di 0.5 allora l'input associato a quell'output appartiene alla classe 1, altrimenti esso appartiene alla classe 2.

Quindi maggiore è il numero di punti classificati correttamente, maggiore è la qualità della rete.

Per avere una misura più precisa della qualità di un individuo si userà la RMSE (Root Mean Squared Error), l'errore quadratico medio.

$$RMSE = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^m (y_i - d_i)^2} \quad (8.1)$$

dove

- m è il numero degli input del test set,
- y_i è l'output corrispondente all'input i ,
- d_i è il target dell'input i .

I target sono o lo zero o l'uno e in questo modo per avere l'errore che tende a zero, y_i dovrà tendere a d_i , $\forall i \in [1, m]$.

Per massimizzare si utilizza 1-RMSE e la formula di fitness diventa:

$$f = 1 - RMSE = 1 - \sqrt{\frac{1}{m} \cdot \sum_{i=1}^m (y_i - d_i)^2} \quad (8.2)$$

Abbinato alla soluzione precedente, il risultato ottenuto da f viene moltiplicato per il numero di punti classificati correttamente.

$$f_{xor} = (1 - RMSE) \cdot N_{cc} = \left(1 - \sqrt{\frac{1}{m} \cdot \sum_{i=1}^m (y_i - d_i)^2}\right) \cdot N_{cc} \quad (8.3)$$

Questo per scoraggiare gli individui con RMSE bassa ma con 3 punti su 4 classificati correttamente e per incoraggiare le reti che producono sia RMSE tendente a zero sia che classificano correttamente più punti possibile.

8.3 Come viene eseguito il test

Il codice è separato in 3 parti, il setup, le funzioni di supporto comprendenti la fitness e la simulazione.

8.3.1 Setup

Il setup comprende:

- l'impostazione della directory di lavoro tramite il comando `SetDirectory[NotebookDirectory[]];`,
- il caricamento dell'algoritmo NEAT tramite pacchetto `neat-pack.m`,
- il reset delle variabili globali quali le probabilità e la soglia δ_t .

Questa sezione contiene infine il test set, una lista che contiene gli input da verificare per il calcolo della fitness. Per rendere il tutto automatico, la lista del test set contiene 2 sottoliste. Le due sottoliste rappresentano le 2 classi e quindi per quanto riguarda lo xor, conterrà due punti per classe.

```
tset = {{{0, 0}, {1, 1}}, {{0, 1}, {1, 0}}};
```

Questo in realtà funziona per ogni test set.

8.3.2 Funzioni di supporto

Questa sezione contiene tutte le funzioni di supporto alla simulazione e alla fitness. La funzione di fitness è scritta secondo le direttive definite nella sezione precedente.

```
xorFitness[genome_] :=
  N@((1 - Sqrt@(1/Length@Flatten[tset, 1]
    (First@
      Total@((class[#, tset] - netSol[gen2phen[genome], #] &
        /@ Flatten[tset, 1])^2))
    )
  ) xorFitnessClass[genome])

xorFitnessClass[genome_] :=
  N@(Count[classOrder[tset] - classification[genome, tset], 0])
```

La funzione `xorFitnessClass` calcola conta i punti classificati correttamente. Tale funzione viene richiamata poi nella `xorFitness` durante la moltiplicazione con $1 - RMSE$.

La sezione contiene un'altra importante e utile funzione.

```
classification [organism_ , tset_]
```

Quest'ultima restituisce una lista con le classi di appartenenza secondo l'ordine degli input. Quindi passando in input la seguente lista:

```
{{{0, 0}, {1, 1}}, {{0, 1}, {1, 0}}};
```

se la classificazione è corretta, si otterrà:

```
{1, 1, 2, 2}
```

8.3.3 Simulazione

Nella simulazione si assegna a NEAT la funzione di fitness da utilizzare:

```
fitness = xorFitness
```

Si genera la popolazione definita da 2 input, 1 output e dalla funzione di fitness appena assegnata:

```
pop = popGen[2, 1, 250, fitness];
```

Infine si esegue la simulazione:

```
tmpPop = pop;
Gen = 0;
While[xorFitnessOld@bestSol@tmpPop < 4,
  tmpPop = generation [tmpPop];
]
tmpPop;

Export["results/xor-data-" <>
  ToString@Gen <> "-gens-" <>
  ToString@Hash@tmpPop <> ".m", tmpPop];

genomeShow2@bestSol@tmpPop
```

La simulazione termina non appena appare un individuo che colloca correttamente nella propria classe ogni punto del test-set. Ad ogni generazione

viene stampato a video e salvato il numero di generazione e il miglior valore di fitness di quella specifica generazione. Con il comando *Export* viene salvata la popolazione del risultato ottenuto per le successive statistiche.

8.4 Risultati del test

Eseguendo una serie di run del test con il crossover disabilitato, si nota che il problema dello xor viene risolto dopo 50 generazioni in media, con picchi minimi di 12 e massimi di 153 generazioni. Abilitando il crossover, la media scende a 30 generazioni con il risultato che i tempi di esecuzione raddoppiano.

Ora andiamo a confrontare la metrica standard contro la nostra NCD. Come metrica, andremo a studiare la fitness massima per generazione, mediata su 26 runs, come da figura 8.4

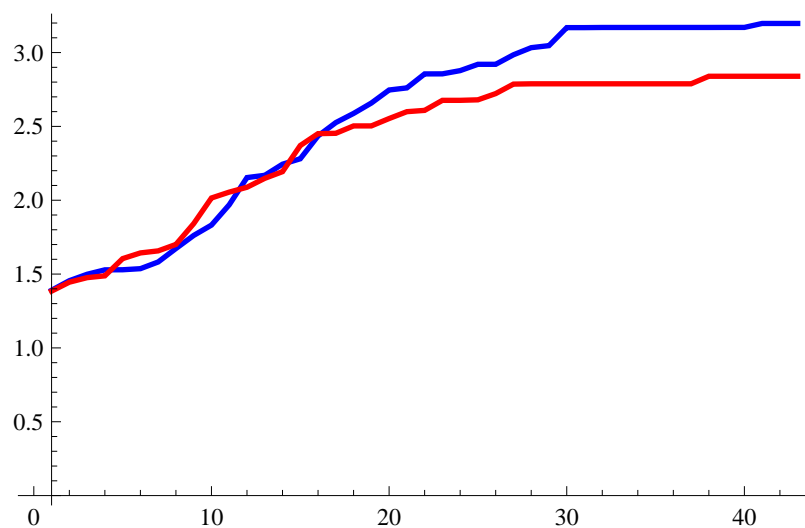


Figura 8.2: La linea blu indica il caso di NCD, la rossa indica l'andamento della metrica standard

Capitolo 9

Maze navigator

9.1 Descrizione

Il problema del maze-navigator è la navigazione di un agente attraverso un labirinto. L'ambiente è composto da un maze che a sua volta è costituito da un *start*, da un *goal* e dai muri. Questi oggetti sono definiti tramite le semplici coordinate in due dimensioni. Lo *start* e il *goal* hanno ciascuno due coordinate (x, y) , mentre un muro è definito da due punti estremi di una linea. Il maze ha anche un agente che potrebbe essere visto come un robot. Tale agente ha come proprietà la posizione, l'orientamento, la velocità lineare e la velocità angolare.

La posizione è espressa in coordinate cartesiane ed è quindi definita come lista di due valori:

$$pos = \{x, y\}$$

L'orientamento è espresso in radianti ed è un valore:

$$orient \in [0, 2\pi]$$

La velocità lineare è un valore reale:

$$lin.vel \in [0, 3]$$

La velocità angolare è la velocità con la quale l'agente ruota a ogni step:

$$ang.vel \in [-3, +3]$$

L'agente ha a disposizione 2 tipi di sensori: range-finders e radar. I range-finders sono dei sensori che misurano la distanza dall'ostacolo più vicino che si trova nella propria direzione. In caso di nessun ostacolo, il valore di un rangefinder sarà il raggio massimo. Questo modello possiede 6 rangefinders, a 0° , $\pm 45^\circ$, $\pm 90^\circ$ e a 180° rispetto all'orientamento dell'agente.

Il radar invece è composto da 4 spicchi sfalsati di 45° rispetto all'orientamento. Si attiva solamente lo "spicchio" in direzione del goal.

Il controller è definito dalla rete neurale. La rete ha 11 inputs e 2 outputs. Gli undici input sono dati dai 6 range-finders, 4 spicchi radar e da 1 bias. Gli output sono il delta sulla velocità lineare e il delta sulla velocità angolare da applicare all'agente.

9.2 Funzione di fitness

Essendo l'obiettivo quello di avvicinarsi al goal finale, la funzione di fitness dipenderà dalla distanza dell'agente pos dal target $goal$. Si usa una funzione di distanza euclidea dal goal:

$$d(pos, goal) = \sqrt{\sum_{i=1}^n (pos_i - goal_i)^2}$$

Per poter ottenere un problema di massimizzazione, si introduce un bias:

$$f = 300 - d(pos, goal)$$

9.3 Simulazione

Come nel caso del tartarus problem, il codice del maze navigator viene interfacciato con NEAT in modo simile.

Durante l'inizializzazione del test, viene scelto il maze da caricare. Il maze è definito in un file con le coordinate del start, del goal e delle linee che compongono i muri.

Dopo il caricamento del maze, vengono interfacciate le funzioni necessarie allo svolgimento del test. Il funzionamento è molto simile al tartarus problem. Vengono estratte le osservazioni da passare all'attivazione della rete, la quale restituirà un output che determinerà l'azione da intraprendere. Gli input sono composti dalle funzioni *pollrfinders[]* e *pollradar[]* e dal bias pari a 1. La prima restituisce le osservazioni date dai rangefinders, *pollradar[]* le osservazioni del radar. Gli output ottenuti sono le differenze da sommare alla velocità lineare e alla velocità angolare dell'agente.

```
trymove [ ] ,
  Mod[hero [[ dang ]] + hero [[ ang ]], 360],
  Max[Min[1 (c2 - 0.5) + hero [[ dpos ]], MAXSPEED], -MAXSPEED],
  Max[Min[1 (c1 - 0.5) + hero [[ dang ]], MAXANGVEL], -MAXANGVEL]}
```

La funzione *trymove[]* prova a muovere l'agente utilizzando gli output della rete c1 e c2.

La funzione *test[nruns, ngens, dist]* permette la chiamata al test dove *nruns* è il numero di run, *ngens* il numero massimo di generazioni e *dist* la metrica utilizzata.

9.4 Parametri

Il maze utilizzato nell'esperimento è un labirinto standard a pettine con un corridoio centrale e un muro di separazione dal goal.

Il lato lungo del maze misura 200X100 unità, la velocità massima (angolare e lineare) dell'agente è di 3 unità per timestep, e all'agente sono garantiti 1000 timesteps di valutazione. Lo starting point è rappresentato dal pallino a sinistra, il goal da quello a destra. Il bias della fitness è impostato a 300, per trasformare la minimizzazione della distanza in un problema di massimizzazione.

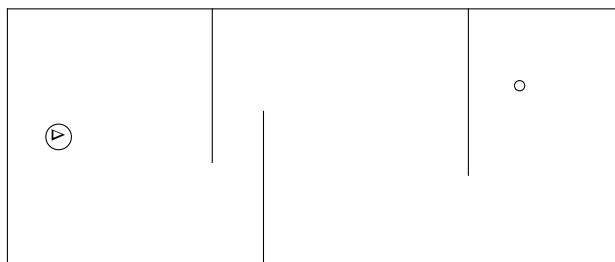


Figura 9.1: Maze usato nei test

9.5 Risultati del test

Al termine dei nostri esperimenti abbiamo constatato che le prestazioni dell'algoritmo, dal punto di vista della fitness, sono comparabili. Siamo quindi andati a tracciare dei grafici per visualizzare tale corrispondenza.

Si noti che in tutti i grafici la linea rossa continua corrisponde all'uso della metrica standard per la speciazione, mentre la linea blu tratteggiata corrisponde all'uso di NCD.

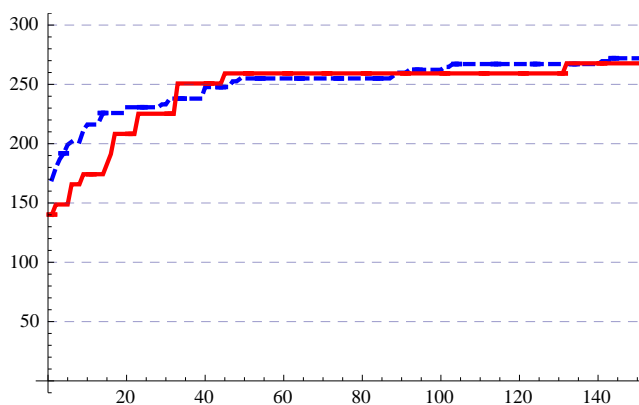


Figura 9.2: Best fitness della popolazione per ogni generazione. Si noti che le errorbar sono molto piccole rispetto alla scala della fitness.

A primo impatto sembrerebbe che l'uso di una delle due distanze non incida sul risultato rispetto all'altro. Siamo quindi andati ad effettuare un test t di Student sui due campioni. Il risultato è che l'ipotesi di uguaglianza non può essere rifiutata, a un livello di confidenza del 0.99, in quanto il p-value è dell'ordine dello 0.124.

Fin qui non sembrerebbero esserci vantaggi dell'uso di una distanza rispetto all'altra. Andando invece a confrontare le dimensioni dei *best* che hanno generato questi dati, vediamo che nel caso della NCD le dimensioni sono più contenute del 50% rispetto sia al numero dei nodi che al numero dei link.

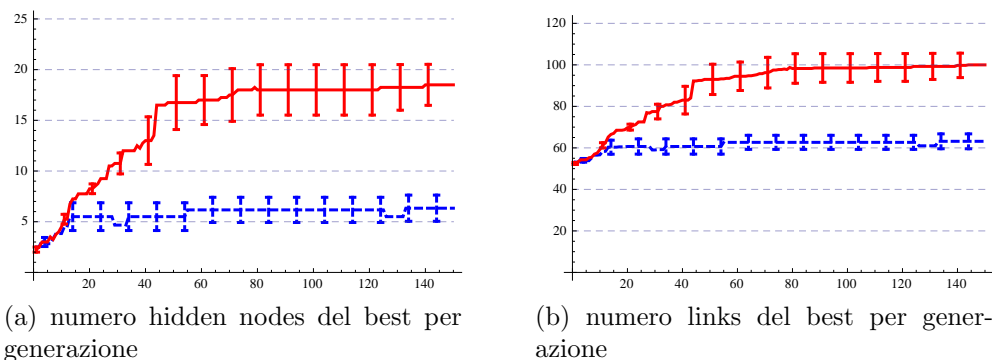


Figura 9.3: Numero di nodi e connessioni dei best della popolazione

Questa differenza di dimensioni è persino più accentuata tra i valori medi nella popolazione.

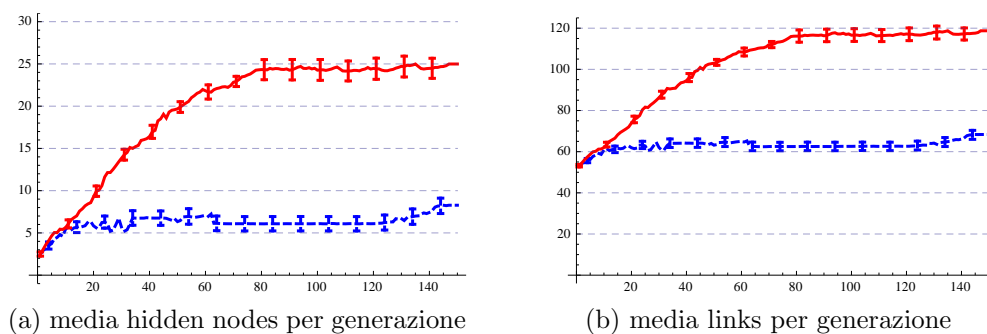


Figura 9.4: Numero di nodi e connessioni medi nella popolazione

Si tratta di una differenza sostanziale specie considerando che il nostro test è durato solo 150 generazioni. Questa differenza è stata percepita anche a livello di tempo macchina, in cui i run con NCD si sono dimostrati più brevi in media del 30%. I dati sembrano molto promettenti, e sicuramente richiedono test più approfonditi.

Capitolo 10

Tartarus

10.1 Descrizione

Il Tartarus problem, o Block Packer, è un utile benchmark usato nel campo dell'intelligenza artificiale. Esso è rappresentato da M campi (mazes) di 6×6 celle, e in ogni maze sono presenti 6 casse sparse e un robot posizionato e direzionato casualmente. Lo scopo del robot è quello di spingere più casse possibili contro il muro in un numero limitato di steps.

Il robot può osservare solamente le caselle adiacenti, quindi 8 in totale. Ogni casella viene interpretata con un valore, 0 se vuota, 1 se è occupata da una cassa e 2 se è un muro. Questi otto valori, più un bias, vengono utilizzati come input nella rete neurale la quale avrà tre output, la probabilità di girare a destra, la probabilità di girare a sinistra e la probabilità di andare in avanti.

In base a questo output, il robot cercherà di muoversi (a meno di impedimenti: muro o doppia cassa) con l'obiettivo di spingere più casse possibili contro i muri. Dopo N iterazioni vengono contati il numero di pareti toccate dalle casse che rappresenterà lo score del robot per quel particolare maze.

La fitness finale sarà data dalla media dello score di tutti i maze.

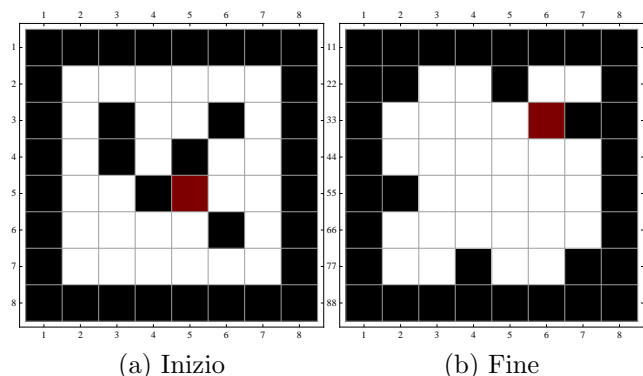


Figura 10.1: Rappresentazione della situazione iniziale e dello stato finale del maze, in uno dei nostri run. La casella rossa è occupata dal dozer.

10.2 Simulazione

Il codice del tartarus problem è composto da funzioni necessarie per estrarre le osservazioni ed eseguire le azioni del robot. Oltre alle funzioni di supporto, contiene anche quelle di inizializzazione dei mazes e del calcolo dello score.

Il test interagisce con NEAT con il passaggio delle osservazioni alla rete neurale, la quale a sua volta restituirà gli output. Gli output ottenuti verranno utilizzati per determinare l'azione e il processo di (lettura osservazioni \rightarrow attivazione rete \rightarrow esecuzione azione) si ripete per ogni step della simulazione. Quindi per il calcolo della fitness di un individuo, vengono eseguiti 80 step. Un passo dell'azione è definito dalla funzione:

```

movedozer [
  moves [[ maxPos [
    netact [i, readsensors []]
  ]]]]

```

- *readsensors[]* restituisce le osservazioni dell'agente. Questi dati sono rappresentati da un vettore che viene passato alla funzione *netact[]*
- *netact[i, input]* è la funzione utilizzata per attivare la rete neurale, accetta in input l'individuo *i* e l'input. Restituisce l'output della rete sotto forma di vettore, in questo caso di lunghezza 3.

- *maxPos[outvect]*, preso il vettore *outvect*, restituisce la posizione dell'output più grande, questo valore sarà compreso tra 1 e 3 in questo caso.
- *moves[pos]*, utilizzando la posizione ottenuta con *maxPos*, fornisce l'azione da intraprendere, scelta tra Left, Right e Forward.
- Infine *movedozer[action]* ruota o sposta in avanti il robot

Alla fine di una simulazione, quindi dopo 80 step, viene eseguito lo scoring tramite la funzione *score[]*. Questa funzione conta in un maze quante pareti vengono a contatto con un blocco per ogni blocco.

Con la funzione *test[nruns, ngens, dist]* viene avviato l'esperimento e tramite i parametri vengono definiti il numero di run, il numero massimo di generazioni e la funzione di distanza utilizzata.

10.3 Parametri

La funzione di fitness utilizzata per la valutazione di un individuo è la media degli score su 20 mazes, per garantire l'abilità di generalizzazione dell'individuo. Su ogni maze vengono garantiti 80 timesteps per la soluzione del problema. I 20 mazes sono generati a random, con score iniziale 0 (niente blocchi contro i muri), garantendo la possibilità di risoluzione, mantenendo tali mazes gli stessi su tutti gli individui.

10.4 Risultati

Nonostante la diversità del problema, la variazione di funzione di speciazione si è dimostrata ininfluente dal punto di vista della fitness anche in questo caso.

Un risultato interessante è che riscontriamo qui lo stesso andamento già visto col maze navigator. Anche qui il t di Student fallisce nel negare l'ipotesi nulla al 0.99, con p-value dell'ordine del 0.115. I best generati con l'uso di NCD per la speciazione, a pari fitness, risultano nuovamente molto meno

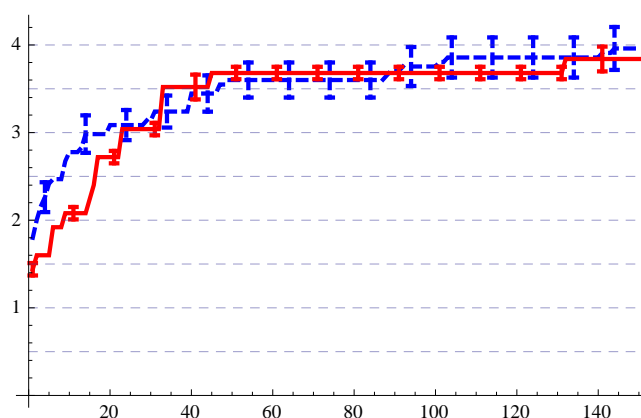
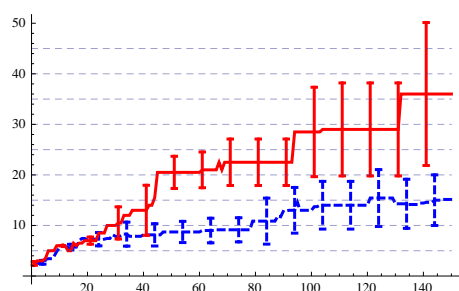
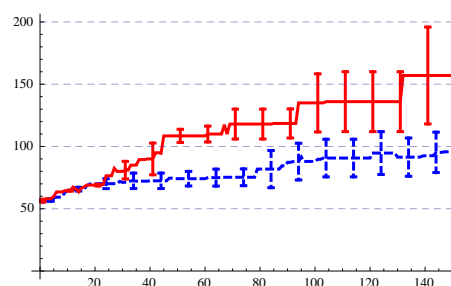


Figura 10.2: best fitness della popolazione per ogni generazione

complessi dei loro pari generati in speciazione standard. Si parla del 50% di nodi e link in meno in media.



(a) numero hidden nodes del best per generazione



(b) numero links del best per generazione

Figura 10.3: Numero di nodi e connessioni dei best della popolazione

Abbiamo osservato lo stesso comportamento anche in questo caso anche sulla media della popolazione.

A livello di tempo macchina, in questo caso NCD si è dimostrata più rapida in media del 20%, a fronte della minore complessità computativa ma maggior numero di esecuzioni richiesto per il calcolo della fitness.

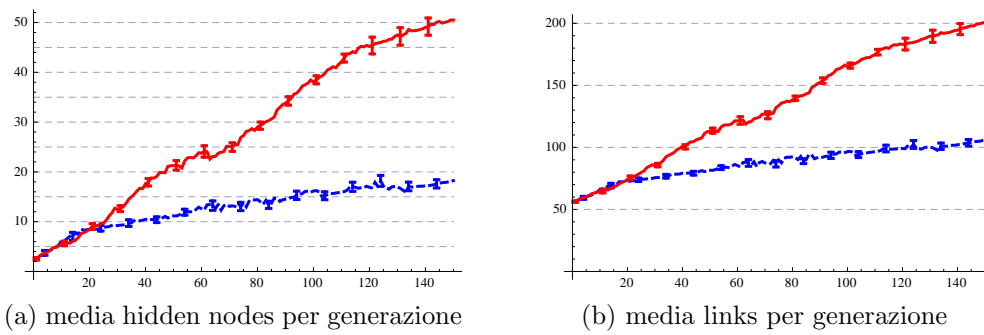


Figura 10.4: Numero di nodi e connessioni medi sulla popolazione

Parte IV

Conclusioni

Cosa si è fatto

Con questa tesi si è cercato di riprodurre e di testare un algoritmo allo stato dell'arte nel campo della neuroevolution. Oltre al semplice sviluppo del codice, si è cercato di aggiungere qualcosa di innovativo all'algoritmo, dei cambiamenti che porteranno a ulteriori studi in futuro.

La prima fase dello studio ha compreso l'apprendimento delle conoscenze sulle reti neurali, sugli algoritmi genetici e sulla neuroevolution per ottenere le basi in modo da procedere ulteriormente nel campo. Una seconda fase è stata dedicata allo studio dell'algoritmo originale, allo studio del linguaggio di programmazione Mathematica e alla lettura delle pubblicazioni necessarie.

Dopo lo studio, il lavoro è stato concentrato sull'implementazione dell'algoritmo seguendo le specifiche di base. In un secondo momento sono state modificate alcune parti come l'implementazione di innovation numbers e aggiunte altre funzionalità come la NCD. Bisogna sottolineare ancora una volta che l'uso della Normal Compression Distance come distanza di speciazione è un'innovazione di nostra concezione, e contributo scientifico originale rispetto al NEAT standard. Semplifica sia la visione del concetto di distanza tra due oggetti sia la sua implementazione, e ottiene risultati comparabili mantenendo più bassa la complessità degli individui e richiedendo l'ottimizzazione manuale di quattro variabili in meno rispetto alla distanza standard.

La fase finale è stata dedicata alle miglorie nel codice e allo svolgimento dei test oltre che alla scrittura di questa tesi.

Sviluppi futuri

Abbiamo individuato diversi possibili ampliamenti all'algoritmo di base. Una prima possibilità è la sostituzione del metodo di selezione, passando da truncated a tournament. Purtroppo il metodo originale non permette la selezione di individui con fitness bassa per una successiva riproduzione e mutazione. Il punto è che ogni individuo dovrebbe avere probabilità di essere selezionato diversa da zero, questo perché se per esempio vengono presi due individui con fitness bassa e se vengono incrociati, possono produrre un individuo con valore di fitness alta.

Un altro possibile campo di ampliamento è stato riconosciuto nella speciazione. Al momento abbiamo studiato la distanza standard che viene applicata sul fenotipo, e la NCD che viene applicata direttamente sul genotipo ed entrambe definiscono la differenza tra due individui. Come terza scelta si potrebbe pensare di applicare una misura basata sul comportamento, per esempio si potrebbero confrontare due individui secondo lo storico delle azioni prese (sempre con NCD).

In questo modo tutti gli individui che hanno lo stesso comportamento faranno parte della stessa specie. A questo punto si potrebbe modificare la shared fitness per avvantaggiare gli individui di una specie con struttura più piccola, limitando attivamente il bloat.

Quindi è possibile rimuovere la limitazione passiva (struttura minima a crescita lenta), e esplorare la possibilità di strutture più complesse.

Considerazioni personali

Dopo quest'esperienza ho sicuramente acquisito nuove conoscenze, dato che gli argomenti trattati facevano parte di un campo per me allora inesplorato. L'ambiente frequentato mi ha dato la possibilità di stare in contatto con rinomati ricercatori che mi hanno dato supporto sia durante la fase di studio che durante la fase di implementazione.

Ritengo che questo progetto abbia contribuito ad accendere in me il desiderio di esplorare più approfonditamente questo settore, e per la prima volta prendo seriamente in considerazione l'ipotesi di una carriera accademica scientifica in futuro.

Ringraziamenti

Vorrei ringraziare in primis Giuseppe per avermi totalmente supportato e sopportato;

Mio fratello e i miei genitori per avermi permesso di arrivare fin qui;

Roberta per essere stata paziente;

E tutti quelli che mi hanno aiutato in questo percorso...