Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea in Fisica

Valutazione della complessità di reti neurali generate tramite algoritmi genetici

Relatore:
Dott. Enrico Giampieri

Correlatore:
Dott. Nico Curti

Presentata da:
Mattia Ceccarelli

Anno Accademico 2017/2018

Indice

1 Introduzione **4**
1.1 Algoritmi Genetici 4
1.1.1 Operatori 5
1.1.2 Struttura di un Algoritmo Genetico 5
1.2 Reti Neurali 6
1.2.1 Perceptron 6
1.2.2 Struttura *fully connected* 8

2 Metodologia **10**
2.1 Popolazione di reti neurali 10
2.2 Operatori 10
2.2.1 Crossover 10
2.2.2 Mutazione 12
2.2.3 Selezione 12
2.3 Funzione di Fitness 12
2.4 Datasets 14
2.5 Struttura dell'algoritmo 16
2.6 Analisi Dati 16

3 Risultati **17**
3.1 Classificazione dei dataset 17
3.1.1 Moons Dataset 17
3.1.2 Circles Dataset 20
3.1.3 Circles+ Dataset 22
3.2 Complessità della rete 25
3.2.1 Moons Dataset 25
3.2.2 Circles Dataset 29
3.2.3 Circles+ Dataset 32

4 Conclusioni **35**

Bibliografia **38**

Capitolo 1

Introduzione

Il progetto di tesi presentato ha come obiettivo verificare la possibilità di evolvere la struttura di una rete neurale attraverso un algoritmo genetico in modo da automatizzarne il processo di costruzione, che ad oggi consiste in un procedimento di "*trial and error*". Le reti verranno valutate nella separazione di due classi di punti in dataset artificiali semplici per verificare la bontà dell'algoritmo. Dopodiché lo studio si concentrerà sull'analisi del come e quando la rumorosità dei dati influenzi la complessità della rete ottenuta dall'algoritmo genetico.

In questo capitolo verranno presentati i principali concetti teorici essenziali alla comprensione delle Metodologie applicate, tra i quali naturalmente la base di un algoritmo genetico e di rete neurale.

Il secondo capitolo include la spiegazione di come questi concetti siano stati adattati nel caso presentato ed in particolare come vengano implementati algoritmi genetici e reti neurali, cosa si intenda per "individuo" e "popolazione", quali dataset siano stati utilizzati e come venga effettuata la presa e l'analisi dati.

Nel terzo capitolo vengono presentati i risultati separati in una parte di valutazione della capacità di classificazione dell'algoritmo e nella seconda parte, in cui vengono mostrati i risultati relativi alla struttura della rete in funzione della rumorosità dei dataset.

Infine vengono tratte le relative conclusione e vengono proposti sviluppi futuri sull'algoritmo.

1.1 Algoritmi Genetici

Gli algoritmi genetici sono software di ricerca ispirati dalla selezione naturale applicata ad una popolazione di individui, i quali vengono valutati nello svolgimento di un *task* (Ambiente). Il parametro che differenzia soluzioni migliori o peggiori è il *fitness*, misurato attraverso la *funzione di fitness* la quale dipende dal problema. L' evoluzione della

popolazione avviene attraverso la selezione dei migliori individui che passeranno il loro *cromosoma* alla generazione successiva.

Sono stati inventati da John Holland negli anni '60 e sono stati sviluppati dallo stesso Holland e dai suoi studenti e colleghi tra gli anni '60 e '70 [1]. L'idea iniziale non era quella di risolvere un particolare problema, in contrasto con l'approccio adottato dai ricercatori per algoritmi evolutivi e programmazione evolutiva, ma di studiare l'adattamento naturale delle specie e trovare un modo per importare questo meccanismo in un computer.

Nel suo libro "*Adaptation in Natural and Artificial Systems*" Holland presenta gli algoritmi genetici come un'astrazione dell'evoluzione biologica: gli algoritmi genetici sono un metodo per muovere una popolazione di *cromosomi* ad una nuova popolazione usando una specie di Selezione Naturale insieme ad operatori ispirati dalla biologia.

1.1.1 Operatori

I principali operatori che compongo un semplice algoritmo genetico sono[1]:

Selezione Questo operatore seleziona i migliori individui, più è alto è il fitness e più è probabile che un individuo venga scelto per creare la nuova generazione

Crossover L'operatore di Crossover produce un taglio nel genoma degli individui "genitori" per formare due individui "figli": per esempio prendendo le due stringhe 111000 e 000111, producendo un taglio alla terza posizione otterremo le stringhe 111111 e 000000.

Mutazione L'operatore di mutazione si occupa di cambiare casualmente uno o più caratteri di individui scelti a caso nella popolazione.

Il funzionamento di un tipico algoritmo genetico,come descritto da [1] una volta definito il problema, procede in questo modo:

1.1.2 Struttura di un Algoritmo Genetico

1. Creazione casuale di n elementi, che rappresentano la prima popolazione.
2. Calcolo del fitness $f(x)$ di ogni soluzione x della popolazione.
3. Fino a che non sono stati generati n discendenti ripetere:
    a. Selezione di due genitori dalla popolazione dove un individuo può anche essere scelto più volte.
    b. Con probabilità p_c (probabilità di crossover) applicare l'operatore di crossover sui due genitori. Nel caso non avvenisse alcun crossover, copiare i genitori.

c. Con probabilità p_m (probabilità di mutazione) applicare l'operatore di mutazione sui figli.

4. Sostituire la vecchia popolazione con la nuova generazione e ripetere dal secondo passaggio.

Ogni iterazione di questo processo è chiamata *generazione*.

Il classico esempio di utilizzo di un algoritmo genetico è la ricerca dei massimi di una funzione. In tal caso, un individuo è rappresentato da una stringa di bit, la *funzione di fitness* è la funzione stessa e il *fitness* delle soluzioni è il valore della funzione calcolato nel punto di cui l'individuo è la rappresentazione binaria. Oltre ad essere l'esempio più semplice risulta anche quello più significativo: di fatto lo scopo di un algoritmo genetico è l'ottimizzazione.

1.2 Reti Neurali

Una rete neurale è una struttura interconnessa di semplici unità procedurali, chiamate nodi. La loro funzionalità si ispira ai neuroni del regno animale. La capacità di elaborazione della rete neurale è contenuta nella "forza" delle connessioni tra nodi, espressa dai *pesi* dei collegamenti, ottenuti da processi di *addestramento* o *apprendimento*. [2]

L'idea di rete neurale proviene dalle ricerche sull'apprendimento di D.O. Hebb degli anni '40, citando dal suo libro *"Organization of Behavior"* [7]:

> *Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.[...] When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased*

Che è la spiegazione della plasticità neuronale (per *firing* si intende l'invio di un segnale, elettrico in questo caso). Dalla cosiddetta "*Hebbian Theory*" deriva il modello computazionale di rete neurale di Warren McCulloch e Walter Pitts chiamato "*Threshold logic*" che aprì la ricerca sulle reti neurali e portò in seguito al più moderno "*Perceptron*".

1.2.1 Perceptron

[3] Il perceptron è stato sviluppato negli anni '50 e '60 dal ricercatore Frank Rosenblatt ispirandosi ai lavori antecedenti di Warren McCulloch e Walter Pitts. È l'unità di base di una rete neurale e il suo funzionamento è il seguente: il perceptron riceve n valori in ingresso $x_1, x_2, ..., x_n$ e restituisce 1 o 0 a seconda che la somma pesata degli input superi

o no un valore di soglia, con pesi $w_1, w_2, ..., w_n$. Ad esempio nel perceptron mostrato in figura 1.1:

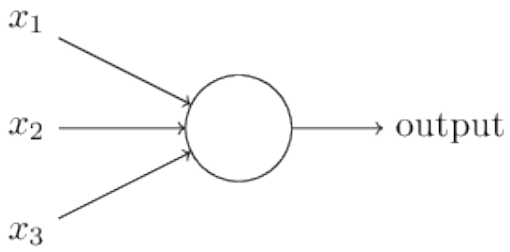

Figura 1.1: *Perceptron con 3 input ed un output*

l'output sarà determinato da:

$$\begin{cases} 0 \text{ se } \sum_i x_i w_i \leq \textit{valore di soglia} \\ 1 \text{ se } \sum_i x_i w_i > \textit{valore di soglia} \end{cases}$$

anche se è più comune trovare la scrittura:

$$\begin{cases} 0 \text{ se } \sum_i x_i w_i + b \leq 0 \\ 1 \text{ se } \sum_i x_i w_i + b > 0 \end{cases}$$

dove b è detto *bias* del perceptron. È attraverso *pesi* e *bias* che il perceptron può soppesare diverse prove e compiere decisioni.

Tuttavia se la rete contenesse perceptron, anche un piccolo cambiamento nei parametri interni potrebbe causare un cambiamento netto nel comportamento della rete [[3]], per questo è preferibile utilizzare una *funzione di attivazione* che rende continuo l'output di un nodo. Un esempio di funzione di attivazione è la sigmoide definita come:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.1}$$

e l'output di un nodo della rete diventa :

$$y = \sigma(\sum_i x_i w_i + b) \tag{1.2}$$

risultato che è continuo e compreso tra zero ed uno.

Un altro tipo di funzione di attivazione è la *Rectified Linear Units* o *ReLU* e si presenta come:

$$y(x) = max\{0, x\} \tag{1.3}$$

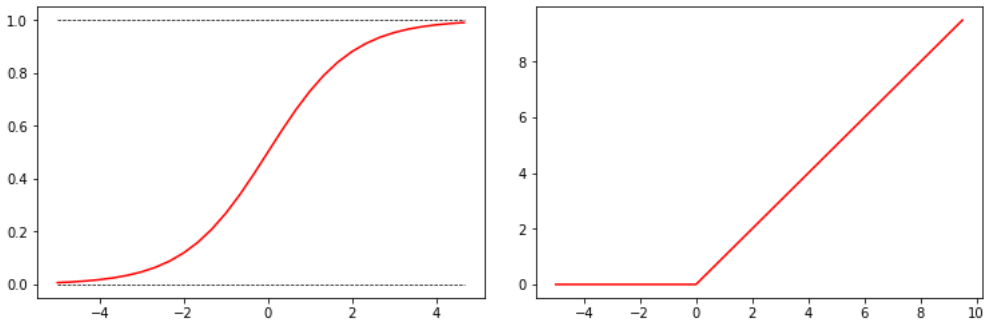

Figura 1.2: *confronto tra due funzioni di attivazione: a sinistra sigmoidale e a destra ReLU*

La scelta della migliore funzione di attivazione non è univoca e dipende dal problema che viene affrontato.

1.2.2 Struttura *fully connected*

La struttura di una rete neurale *fully connected* composta da molti *layer* di neuroni è come quella mostrata in figura 1.3:

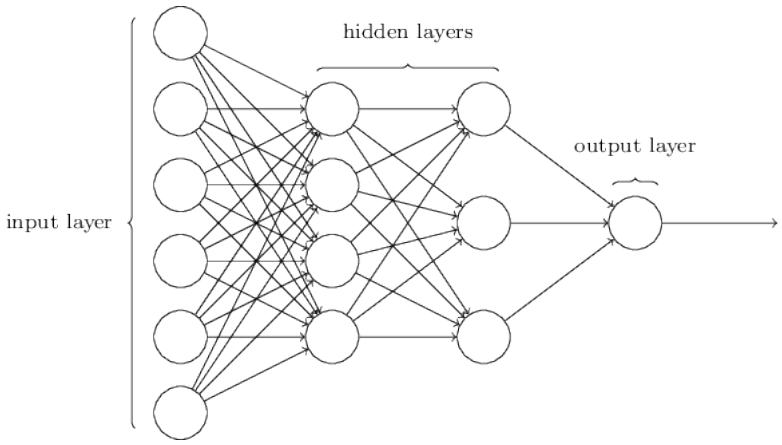

Figura 1.3: *Esempio di rete neurale fully connected in cui viene mostrata la distinzione tra input layer, hidden layer e output layer*

In una rete come questa ad ogni collegamento è associato un peso e ad ogni nodo è associato un *bias*: gli output dei *neuroni* del *layer* di input diventano a loro volta valori in ingresso del *layer* successivo in un procedimento a catena fino all'ultimo *layer*, che restituisce la risposta della rete. L'addestramento della rete consiste nel valutarne gli output in un determinato set di dati, chiamato *training dataset*, confrontarli con i valori

attesi, forniti dallo stesso *dataset*, e modificare pesi e *bias* in modo che la risposta si avvicini a ciò che ci si aspetta.

L'algoritmo di addestramento più utilizzato è la *Backpropagation*, che è la modalità utilizzata per i *Multy Layer Perceptron* in questa prova.

L'idea su cui si basa è di valutare l'errore commesso da ogni output della rete, dopo un' inizializzazione casuale dei pesi, in base all'output atteso e di retro propagare lungo la rete gli scarti per aggiornare il valore dei pesi [4]. Indicando con:

- t_k: risultato atteso al k-esimo output;
- y_k: risultato ottenuto al k-esimo output;
- w: peso generico;
- W: l'insieme dei pesi.

L'errore commesso dal k-esimo output è stimato da:

$$E_k = (t_k - y_k) \tag{1.4}$$

Da cui possiamo definire la funzione d'errore come:

$$J(W) = \sum_{k=1}^{N} \frac{1}{2} E_k^2 = \sum_{k=1}^{N} \frac{1}{2} (t_k - yk)^2 \tag{1.5}$$

Si procede poi nel correggere i pesi partendo dall'output fino all'input nel modo seguente:

$$w = w + \delta w = w + \eta \frac{\partial J}{\partial w} \tag{1.6}$$

dove η è una costante che garantisce un'opportuna convergenza ai pesi. Lo scopo è quello di trovare un minimo alla funzione d' errore. [4]

Capitolo 2

Metodologia

In questo capitolo verranno descritti i metodi utilizzati per l'ottimizzazione di una rete neurale *fully connected* attraverso un algoritmo genetico: l' obiettivo è quello di evolvere la struttura degli *hidden layer* della rete al fine di classificare al meglio un dataset separato in due classi di dati.

2.1 Popolazione di reti neurali

L'algoritmo genera la prima popolazione di oggetti *Random Network* casualmente, i parametri interni sono stati scelti in modo che questa fosse limitata e non raggiungesse subito un risultato ottimale. In particolare il numero di layer è compreso tra 1 e 5 mentre il numero massimo di neuroni per layer è 25. La popolazione è stata mantenuta al massimo di 20 individui per lo stesso motivo. Il *cromosoma* di un individuo è quindi una lista di numeri di lunghezza variabile e l'algoritmo genetico è stato adattato a questo problema: la ricerca di un minimo in uno spazio in cui il numero di variabili non è definito.

2.2 Operatori

La differenza nell'utilizzare cromosomi di lunghezza variabile si ritrova soprattutto negli operatori dell'algoritmo genetico scritto *ad hoc* per il progetto che devono gestire una variabile che normalmente non rappresenterebbe un problema. Per questo ogni operatore è stato adattato per fare in modo che sia possibile esplorare lo spazio delle soluzioni anche in tal senso.

2.2.1 Crossover

Il Crossover si differenzia da quello di un algoritmo genetico in cui la lunghezza di ogni cromosoma sia prestabilita. Infatti, è facile che i due genitori selezionati non abbiano lo

stesso numero di hidden layer: in questo caso non è possibile "tagliare" il cromosoma nello stesso punto (come mostrato nel capitolo 1.1). Inoltre è necessario che la lunghezza dei figli possa essere diversa da quella dei genitori in modo da non limitare le possibilità di esplorazione.

Seguendo questi propositi, vengono scelti casualmente due punti di taglio, uno per genitore, dalla mediana di tre interi casuali (questo per fare in modo che la lunghezza non cambi troppo tra una generazione e la successiva, come mostrato in figura 2.1). Dopodiché il crossover procede come in un classico algoritmo genetico, ossia restituendo due figli, combinazione delle parti tagliate.

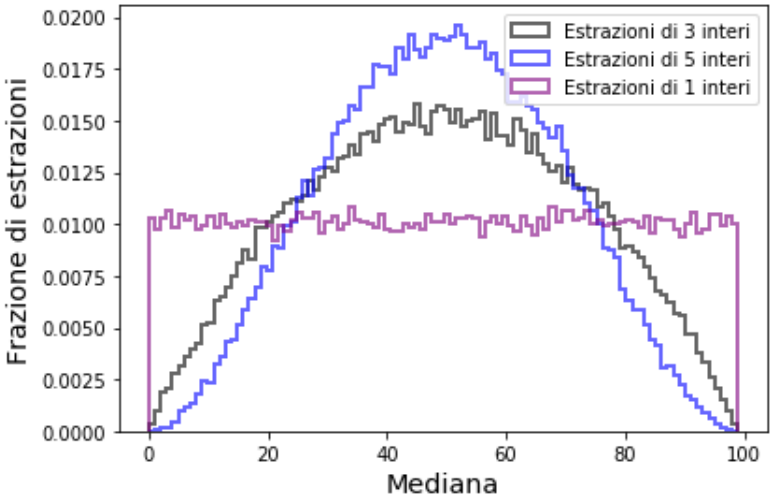

Figura 2.1: *In figura viene mostrato il risultato di 100000 estrazioni di serie da 3(in nero) e 5(in blu) interi casuali (da 1 a 100) dalle quali viene estratta la mediana. Aumentando il numero di numeri casuali da cui estrarre la mediana, il picco si stringe e si alza al centro. In viola un'estrazione uniforme.*

Un esempio di funzionamento, scelti i genitori:

$$\text{genitore 1} = [10,3,4,11];\ \text{genitore 2} = [7,23,1,2,19]$$

e prendendo due tagli rispettivamente alla 3^a e 2^a posizione si ottengono:

$$\text{figlio 1} = [10,3,4,1,2,19];\ \text{figlio 2} = [7,23,11]$$

i quali popoleranno la generazione successiva.

2.2.2 Mutazione

Anche il processo di mutazione ha subito qualche variazione, infatti, occorre che anche la lunghezza del cromosoma possa cambiare a seguito dell'azione di questo operatore. Per fare ciò sono state introdotte due probabilità indipendenti: $p_m = 5\%$ rappresenta la probabilità che un layer qualsiasi del cromosoma venga modificato aggiungendo o togliendo un neurone, $p_l = 5\%$ rappresenta invece la probabilità che venga aggiunto o rimosso un layer dalla rete.

2.2.3 Selezione

Nella varietà di operatori di selezione, verranno comparati i risultati di due algoritmi, scelti per la semplicità di implementazione che nulla toglie alla capacità di questi di evolvere il sistema.

1. Il primo, denominato *random mate*, è il più rapido: partendo da una popolazione ordinata per miglior *fitness* passa alla nuova generazione un' *élite* di individui pari al 20%. Poi scorre dall'inizio la popolazione per il primo genitore, mentre il secondo genitore viene scelto a caso. Questo avviene fino a che la generazione successiva non è riempita.

2. Il secondo, denominato *all sons*, è concettualmente più semplice ma richiede molto più tempo per completare una run: continuando a mantenere un 20%di *élite*, in una popolazione viene fatto il crossover di ogni coppia possibile di individui. Di questi vengono scelti i migliori per riempire la generazione successiva. È facile capire come questo sia il metodo più dispendioso in termini di risorse, infatti ogni rete va addestrata per determinarne il *fitness* e in una popolazioni di soli 10 individui il numero totale di reti da valutare per ogni generazione sale a $200 = (10 * 10 * 2)$

Mantenere un'*élite* di individui è utile per fare in modo che la popolazione evolva ma allo stesso tempo mantenga intatti i caratteri che nella precedente generazione hanno conseguito il miglior score.

2.3 Funzione di Fitness

La funzione di fitness dell'algoritmo riceve il cromosoma di ogni individuo che come mostrato in sezioni precedenti ha la forma di una lista di interi, che rappresentano gli *hidden layer* della rete. È a questo punto che viene costruita la rete vera e propria, che altro non è che un'istanza della classe *MLPClassifier* (che sta per *Multi-layer Perceptron classifier*) della libreria *scikit learn* ([5]).

L'addestramento della rete è affidato alla funzione *fit*, propria di ogni Classificatore di *scikit learn*, che restituisce un oggetto *MLP* addestrato sul *training set*.

A questo punto è possibile misurare la risposta di un *MLP* addestrato agli input: l'*input layer* è composto da due nodi, ossia le due coordinate di un punto, l'*output layer* da un solo nodo, con output compreso tra 0 (classe 1) e 1(classe 2) come mostrato in figura 2.2:

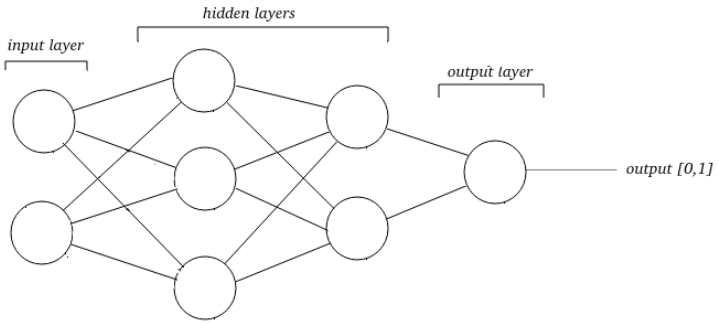

Figura 2.2: *Esempio di MLPClassier generato da un individuo con cromosoma [3,2]*

Infine si sfrutta il *test set* per attribuire un *fitness*, o *score*, all'individuo. Ciò viene fatto utilizzando due metri di valutazione differenti, valutati in parallelo:

Accuracy score questo metodo è il più semplice e diretto: attribuisce un fitness pari alla percentuale di risposte corrette della rete. Il problema di questo procedimento è che equipara una risposta di 0.51 ad 1 ed una di 0.49 a 0 nonostante in entrambe le risposte l'incertezza sia alta. L'*accuracy score* viene massimizzato nell'algoritmo.

Logarithmic loss function o *log loss*, è una misura di quanta incertezza pone la rete nelle sue risposte. Dalla documentazione di *scikit learn* la *log loss* è definita come "*the negative log-likehood of the true labels given a probabilistic classifier's prediction*" o :

$$-logP(\frac{y_t}{y_p}) = -(y_t log(y_p) + (1 - y_t) log(1 - y_p)) \tag{2.1}$$

Dove y_t è la classe dell'oggetto (o 0 o 1), mentre y_p è la probabilità stimata che l'oggetto sia di quella classe.

Non è stata attribuita nessuna forma di penalità alla profondità delle reti o ,più in generale, a quanti collegamenti essa forma, che potrebbero quindi diventare sempre più grandi e computazionalmente pesanti da gestire, ma verrà studiato in seguito come si comportano queste grandezze (numero di collegamenti e lunghezza).

Ogni MLP viene addestrato e valutato su due sezioni diverse del dataset: *train set* e *test set*. Entrambe rappresentano una frazione dell'intero dataset scelta casualmente, in particolare sono rispettivamente l'80% e il %del dataset. Nel *train set* l'MLP viene addestrato mentre nel *test set* viene valutato e gli viene attribuito uno score sulla base di questo risultato. Questa separazione serve per evitare l'*overfitting* delle reti e in seguito verrà discusso come l'unico tipo di penalità che una rete molto complessa possa subire.

2.4 Datasets

I set di dati sono artificiali e forniti da *scikit learn*. Sono stati usati per le prove 3 tipi di dataset:

moons I punti rossi e blu vengono generati a forma di "semicerchi", come mostrato in figura 2.3

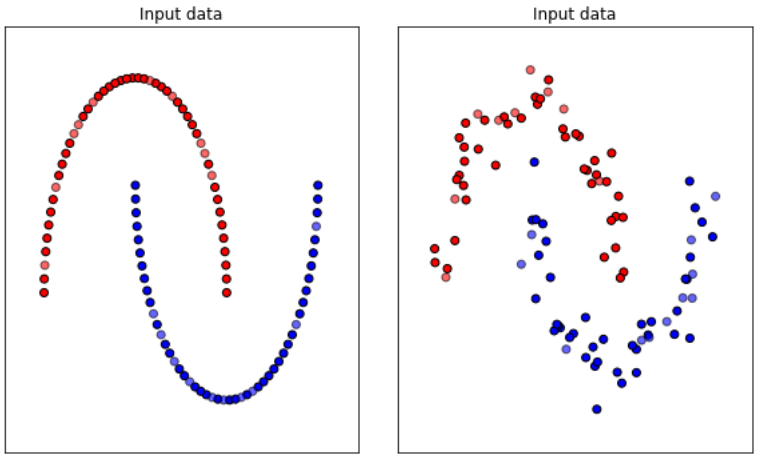

Figura 2.3: *Nell'immagine sono messe a confronto due versioni del dataset moons: a sinistra priva di noise e a destra con noise = 0.1*

circles I punti vengono generati a forma di cerchi concentrici, è possibile fornire diversi gradi di separazione ai due cerchi attraverso il parametro *factor*. Il dataset è mostrato in figura 2.4

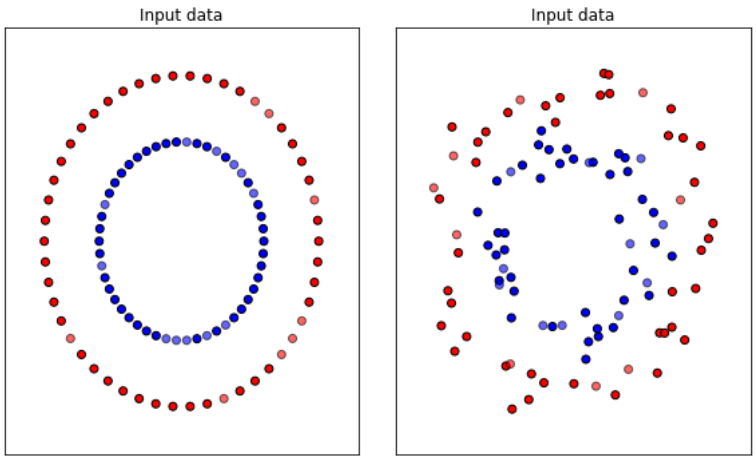

Figura 2.4: *Nell'immagine a confronto due versioni del dataset circles: a sinistra senza noise e a destra con noise = 0.1. Entrambe con lo stesso fattore tra i cerchi*

circles+ È un dataset personalizzato creato dall'unione di due *circles*, come mostrato in figura 2.5

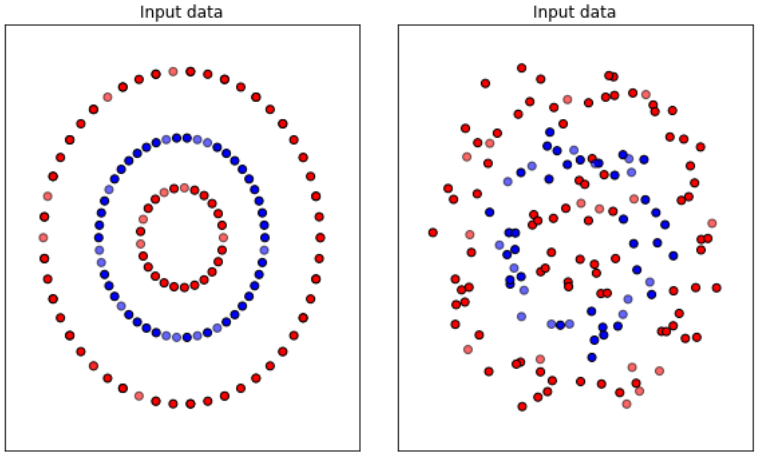

Figura 2.5: *Nell'immagine a confronto due versioni del dataset circles+: a sinistra con noise = 0 e a destra con noise = 0.1*

Per tutti i datasets è possibile attribuire un *noise* in modo che sia più difficile separare le due classi di punti. Nel corso delle prove sono state valutate le prestazioni dell'algoritmo su tutti i datasets. Sono stati appositamente evitati dataset linearmente separabili.

I dataset sono generati a partire dalle due funzioni *make moons* e *make circles* di *scikit learn*.

2.5 Struttura dell'algoritmo

Per chiarezza viene riportata la struttura dell'algoritmo, che ne mostra i vari passaggi:

1. creazione della prima generazione.
2. generazione del dataset, separazione di questo in *train set* e *test set*, valutazione del *fitness* e ordinamento della popolazione.
3. finché l'individuo con il *best fitness* non rimane lo stesso per 5 generazioni consecutive:
    a. separazione del dataset in *train set* e *test set* con *random state* differenti ad ogni ciclo.
    b. Finchè la nuova generazione non è riempita ripetere:
        - Selezione del primo e secondo genitore.
        - Crossover e creazione dei figli.
        - Mutazione dei figli.
    c. Valutazione del fitness e ordinamento della nuova generazione
    d. Risultati del migliore individuo sul dataset scelto.
4. Termine dell'algoritmo.

Ogni iterazione di questo processo viene chiamata *run*

2.6 Analisi Dati

L'analisi dei dati può essere suddivisa in due parti: nella prima lo studio si concentra sulla capacità dell'algoritmo di separare le due classi di punti in ogni set di dati e con ognuna delle modalità presentate in precedenza, mostrando in ogni caso l'evoluzione qualitativa della migliore rete per ogni generazione.

Nella seconda parte ci si concentra invece su un'analisi quantitativa di alcune grandezze che caratterizzano la rete neurale: il numero di *hidden layer*, la grandezza del layer con minore numero di neuroni e il numero di collegamenti che la rete produce. Ognuna di queste è stata studiata in funzione del *noise* del dataset come misura di quanto questo risulti separabile: un dataset più rumoroso risulta inevitabilmente più difficile da classificare, sopratutto nel momento in cui punti blu e rossi siano molto vicini. In questo secondo caso vengono effettuate numerose simulazioni e da ognuna vengono estratti i dati citati sopra dalla rete di convergenza. Dopodichè viene mostrata la media punto per punto in funzione del noise ed in funzione dello score ottenuto a quel livello di noise.

Capitolo 3

Risultati

In questo capitolo verranno presentati i dati ottenuti confrontando le metodologie discusse nel capitolo precedente.

3.1 Classificazione dei dataset

La prima parte dei risultati si concentra nell'analisi delle prestazioni dell'algoritmo nell'evolvere una popolazione di reti in grado di separare due classi di punti nei tre dataset presentati in precedenza. Verranno analizzate le prestazioni dei due algoritmi di selezione insieme ai due tipi di funzione di fitness.

3.1.1 Moons Dataset

Il primo dataset su cui è stato testato l'algoritmo è *moons*, nelle immagini successive vengono riportati alcuni esempi di run effettute, differenziando *Accuracy score* e *log loss*, cioè i due diversi tipi di *fitness function*, e i due algoritmi di selezione, che sono *random mate* e *all sons*.

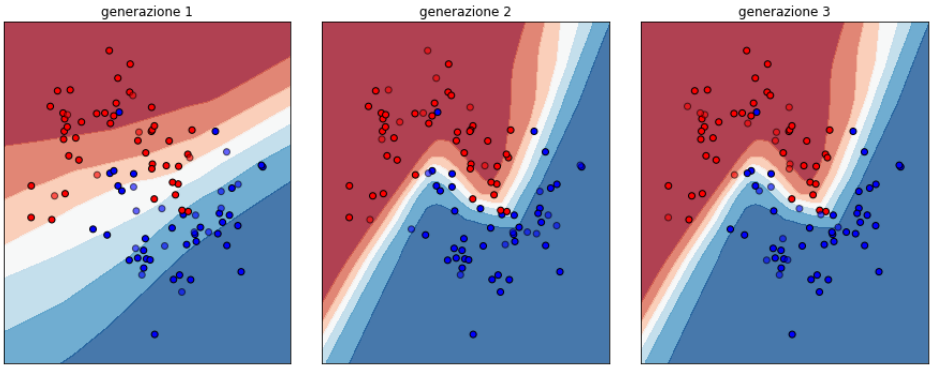

Figura 3.1: *L'immagine mostra una run tipica usando l'accuracy score e la random mate selection e noise = 0.25*

Accuracy score Nella figura 3.1 viene mostrata una run rappresentativa effettuata utilizzando come algoritmo di Selezione *random mate*: da notare come queste classificazioni raggiungano uno score molto alto (generalmente 100%) nonostante la "sicurezza" della rete non sia molto alta nella classicazione di alcuni punti, specialmente se questi si trovano vicini a punti della classe opposta. Questo accade perché lo score della rete viene valutato come se questa restituisse una risposta binaria per ogni input (0/1 o rosso/blu) e ciò è indipendente dall'incertezza con la quale viene determinata la classe di un punto. Ciò significa che per l'*accuracy score* se un risultato è 0.01 o 0.49 questo sarà sempre o corretto o sbagliato a seconda che il valore vero sia 0 o 1, nonostante il secondo risultato sia pieno di incertezza. La "sicurezza" della rete è rappresentata sui grafici dal colore: tanto più è intenso e tanto più il *Multi-layer perceptron* è sicuro della sua scelta. Il colore viene quantificato dalla funzione *predict proba* di *scikit learn*, che restituisce la probabilità calcolata dalla rete che un punto sia rosso o blu.

Alcune generazioni (le ultime) vengono escluse dalle immagini perchè, come detto nel capitolo precedente, sono tutte uguali.

In figura 3.2 viene invece mostrato il grafico del fitness del miglior individuo in funzione della generazione:

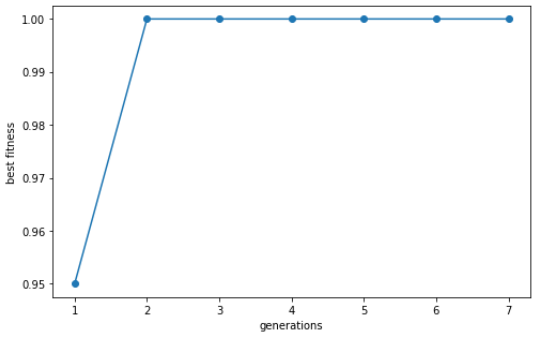

Figura 3.2: *Best fitness di ogni generazione della popolazione con accuracy score e random mate*

Da notare come su questo dataset il raggiungimento di un individuo con fitness molto alto (di fatto insuperabile) sia molto veloce (in questo caso di una sola generazione), nonostante la separazione sembri molto incerta, come specificato sopra.

Risultati simili vengono raggiunti dal secondo algoritmo di selezione, mostrati in figura 3.3:

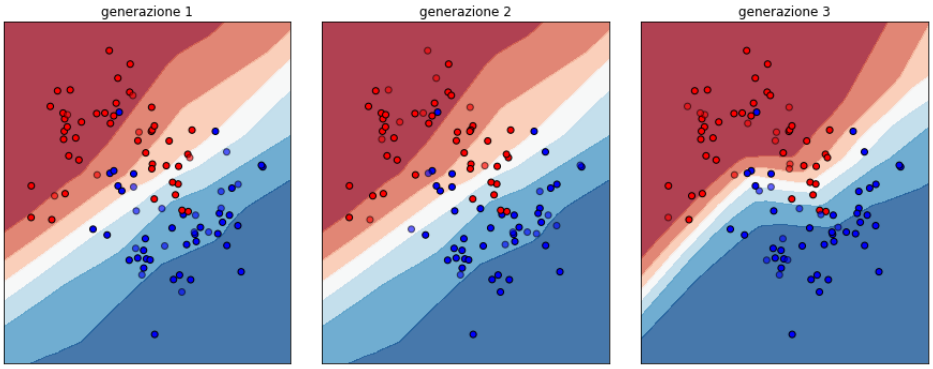

Figura 3.3: *In figura viene mostrata una run d'esempio utilizzando lo stesso noise e stessa funzione di fitness ma diverso algoritmo di selezione*

Anche in questo l'evoluzione è rapida e il risultato finale incerto, nonostante l'ottimo risultato raggiunto dalle reti nell'*accuracy test*. L'andamento del *best fitness* è identico a quello precedente (figura 3.4):

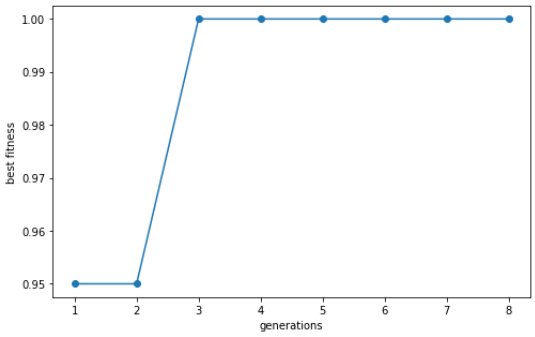

Figura 3.4: *In figura viene mostrato l'andamento del best fitness in funzione della generazione*

Data l'inefficienza sia computazionale sia in termini di performance, è stato deciso di abbandonare questo algoritmo di selezione per i prossimi datasets.

logarithmic loss Significative differenze su questo dataset si hanno cambiando il tipo di funzione di fitness e passando alla *logaritmic loss*, come mostrato in figura 3.5

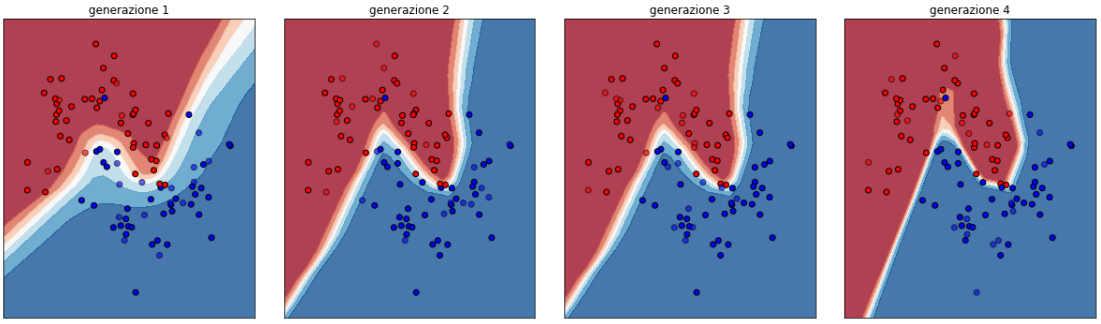

Figura 3.5: *Evoluzione di una rete neurale seguendo la logarithmic loss function sul dataset moons*

3.1.2 Circles Dataset

Vengono presentati in forma parallela alla sezione precedente i risultati ottenuti dall'algoritmo nel dataset *circles*, con noise = 0.2

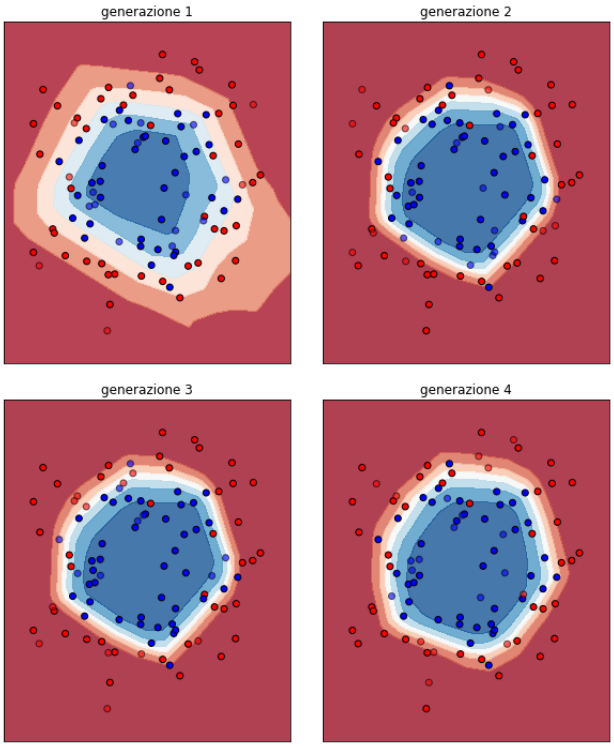

Figura 3.6: *L'immagine mostra una run d'esempio nel dataset circles usando l'accuracy score*

Anche in questo caso l'accuracy score mostra una convergenza dell'algoritmo molto rapida e approssimativa nella classificazione.

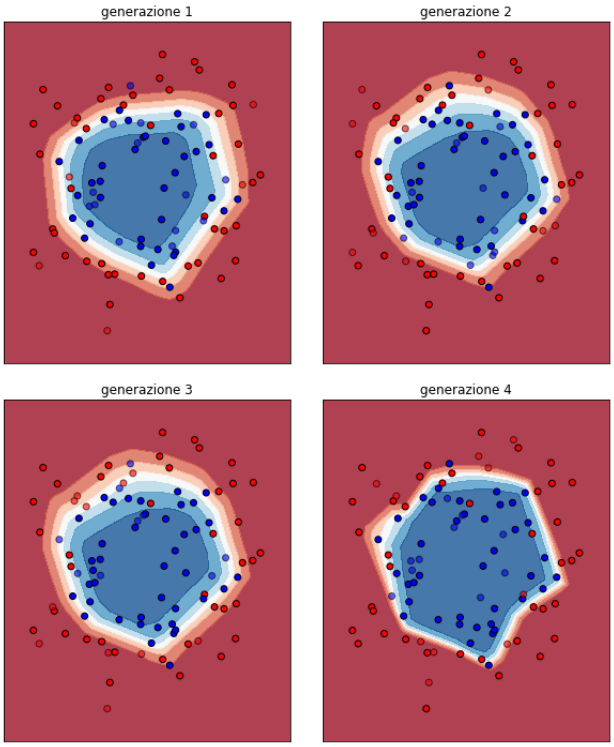

Figura 3.7: *L'immagine mostra una run d'esempio nel dataset circles usando la log loss*

Come per il precedente dataset, questa funzione di fitness sembra riportare una classificazione più precisa, come mostrato in figura 3.7.

3.1.3 Circles+ Dataset

I risultati più interessanti in termini di evoluzione della popolazione provengono dal dataset *Circles+*, ideato come "stress test" per l'algoritmo. Nelle figure 3.8 e 3.9 vengono riportati i dati di classificazione.

In figura 3.8 è riportata una classificazione d'esempio utilizzando l'*accuracy score functions*:

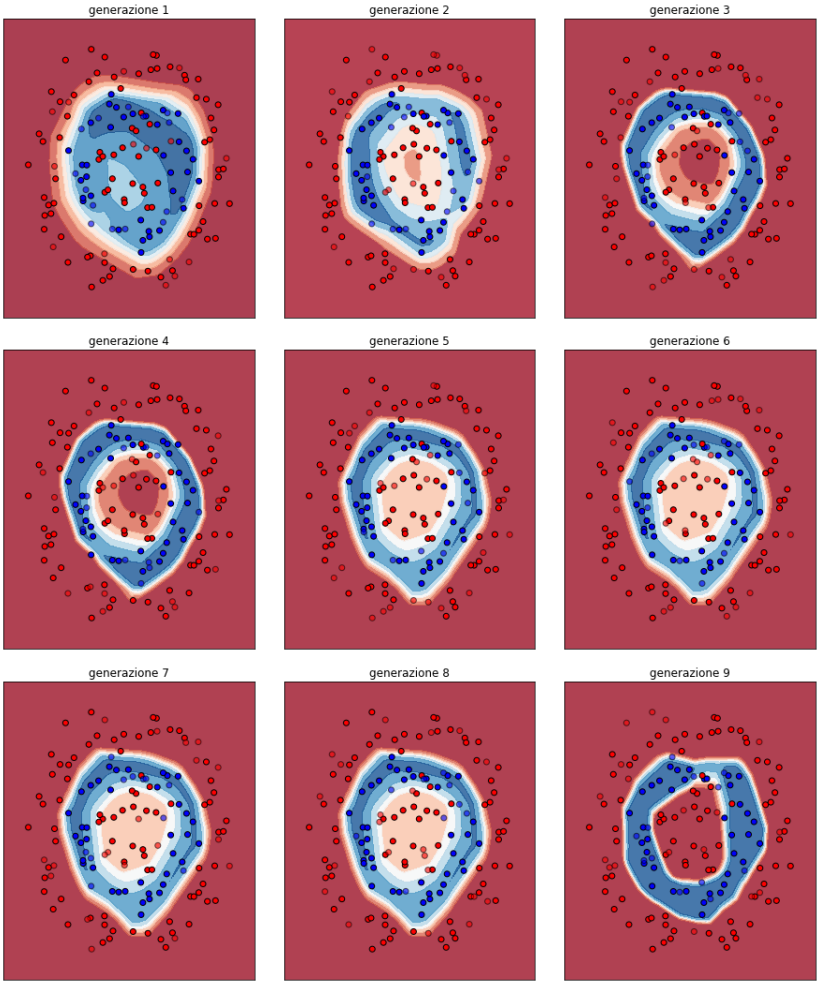

Figura 3.8: *L'immagine mostra una run d'esempio nel dataset circles+ usando l'accuracy score*

Infine viene mostrata una run d'esempio utilizzando la *logarithmic loss function*.

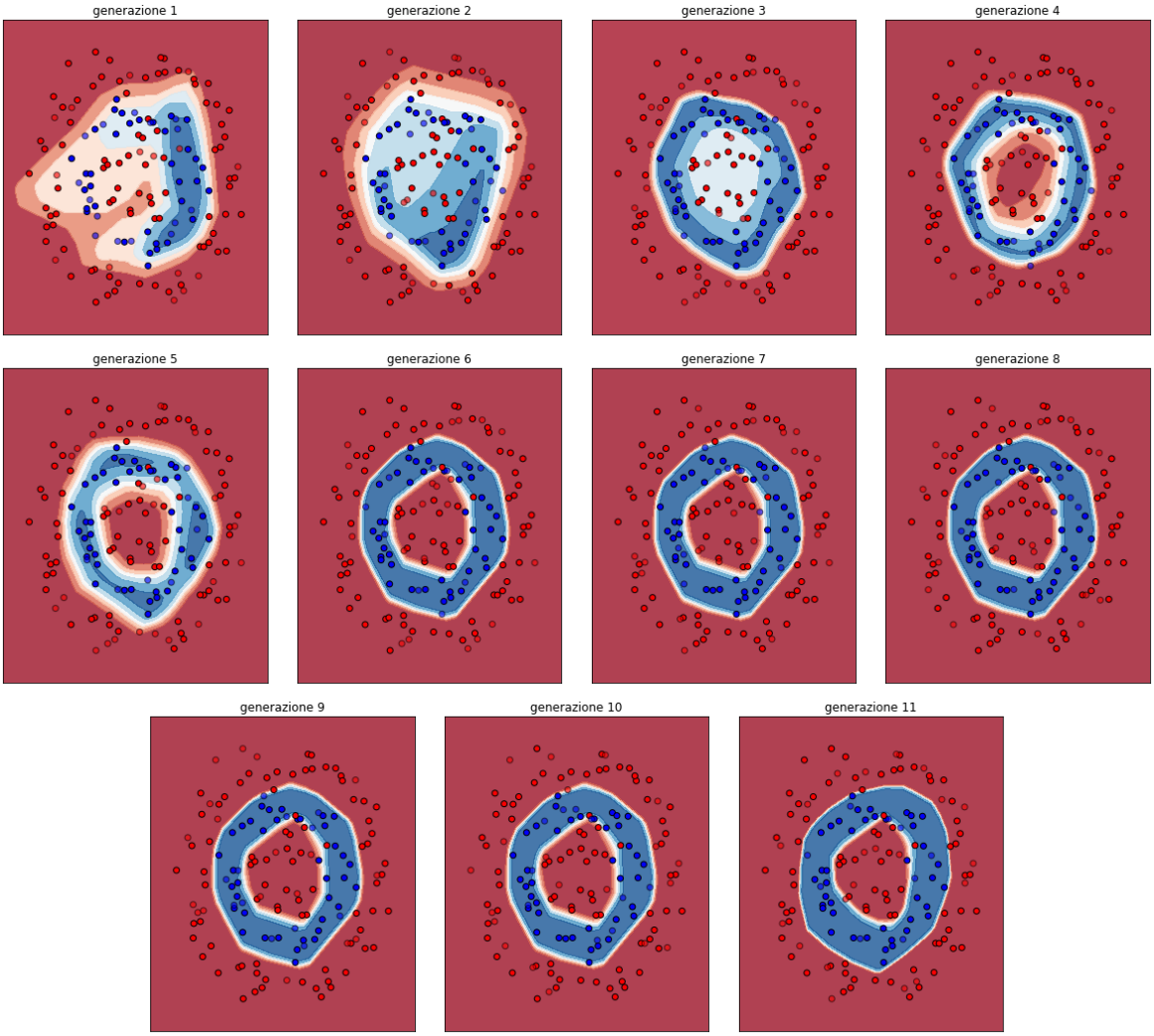

Figura 3.9: *L'immagine mostra una run d'esempio nel dataset circles+ usando la log loss con noise = 0.1*

In questo caso la convergenza è molto simile nelle due funzioni di fitness. In entrambe le simulazioni è possibile osservare un'evoluzione più lenta rispetto ai primi due dataset. In particolare questa evoluzione inizia da una popolazione casuale che non riconosce (per nulla o in parte) il cerchio interno di punti di colore rosso ma avanzando nelle generazioni questo gruppo interno viene correttamente classificato. Dopodichè in entrambi i casi l'evoluzione porta ad un miglioramento della precisione della separazione fino alla convergenza, la quale risulta

3.2 Complessità della rete

I parametri importanti da considerare nel valutare la complessità di una rete sono:

- Collegamenti
- Numero di neuroni del layer più piccolo
- Numero di *Hidden Layer*

Questi ci danno una misura di quanta informazione può contenere e quanto è possibile comprimerla. Nell'algoritmo genetico non è stata impostata alcuna penalità in relazione alla complessità della rete,l'obiettivo infatti non è stato quello di trovare la rete più performante, quindi esiste la possibilità che non converga mai ad un risultato finale, costruendo reti sempre più profonde e precise nel separare i dati. In realtà ciò non è osservato nei casi visionati fino ad ora, ossia quando le due classi non sono perfettamente separabili. Inoltre, come già accennato, per la funzione di fitness *accuracy score* non può succedere perchè prima o poi l'algoritmo convergerà ad una soluzione con score pari a 100%, di fatto insuperabile.

La situazione è diversa per la *log loss* che invece ha un limite inferiore asintotico. Per questo motivo è stato studiato il comportamento dell'algoritmo con questa funzione di fitness al variare della separabilità dei dati, quantificata dal *noise* dei rispettivi dataset.

L'obiettivo è quindi quello di studiare l'andamento di questi parametri in funzione del *noise* dei dataset.

3.2.1 Moons Dataset

Iniziamo con il presentare i dati relativi all'andamento del fitness delle reti in funzione del noise introdotto nel dataset *moons* in figura 3.10.

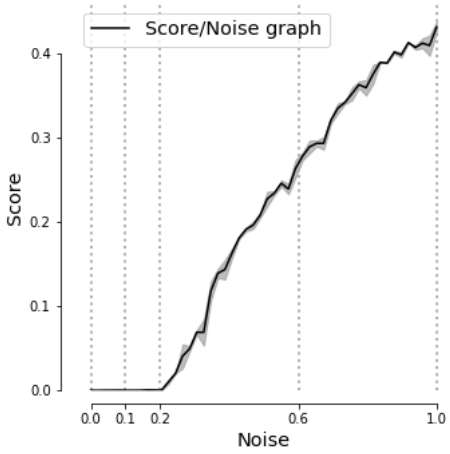

Figura 3.10: *Nel grafico viene mostrato l'andamento dello score in funzione del noise del dataset. In evidenza alcuni punti interessanti dell'andamento*

Nel dataset *moons* lo score non risulta sensibile ai cambiamenti del noise e rimane prossimo a 0 (per la *logloss* uno score migliore rappresenta un fitness maggiore) fino a che la run raggiunge noise = 0.2: probabilmente questo è il valore in cui i punti non sono più perfettamente separabili. Il grafico non sembra mostrare un andamento di saturazione per alti livelli di noise, ciò significa che nonostante le reti ancora riescono a distinguere alcuni punti rossi e blu.

Di seguito verranno mostrati gli andamenti delle grandezze scelte come rappresentative della complessità della rete.

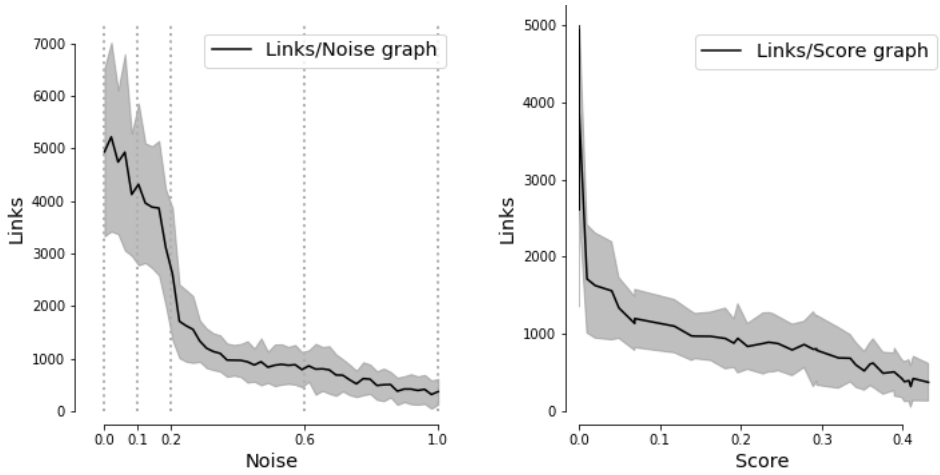

Figura 3.11: *A confronto gli andamenti del numero di collegamenti in funzione del noise ed in funzione dello score*

In figura 3.11 viene mostrato l'andamento dei collegamenti della rete in funzione del noise e del fitness.

Nel primo grafico si nota un andamento fortemente decrescente anche nel range di valori di noise in cui lo score del dataset è costante a 0. Aumentando ancora il noise l'andamento si stabilizza rimanendo comunque decrescente. Se per *noise* prossimi allo zero la rete raggiunge un numero di collegamenti elevato (5000 collegamenti in media per noise = 0), questo crolla quando il noise raggiunge certi valori e continua a decrescere poi.

Nel secondo grafico si nota una rapida divergenza nel numero di links avvicinandosi a valori di score prossimi a 0: se l'algoritmo riesce a classificare molto bene il dataset allora tende a cercare reti con un numero di collegamenti più elevato.

Nella figura successiva, la 3.12, verrà presentato l'andamento del numero di neuroni nel layer più piccolo della rete in funzione del noise e dello score.

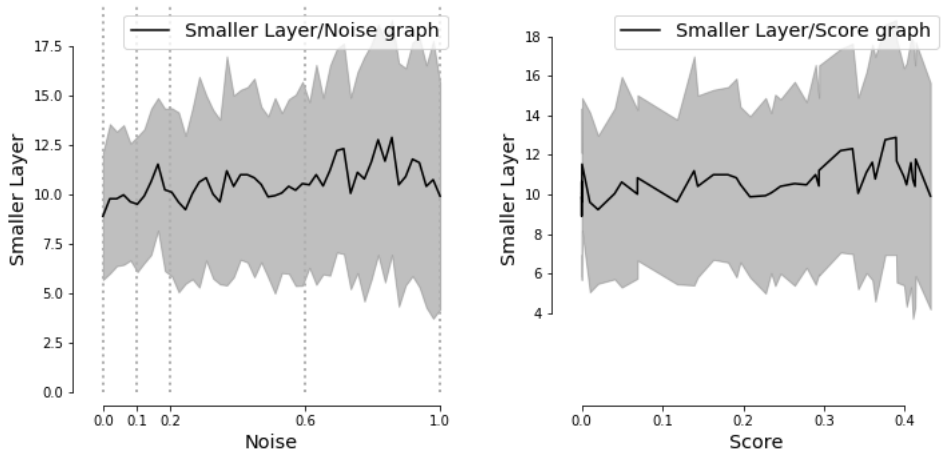

Figura 3.12: *Nei due grafici sono mostrati gli andamenti del numero di neuroni nel layer più piccolo della rete in funzione del noise di moons e dello score*

In questo dataset, il numero di neuroni del layer più piccolo della rete mostra un andamento costante sia in funzione del noise e sia in funzione dello score.

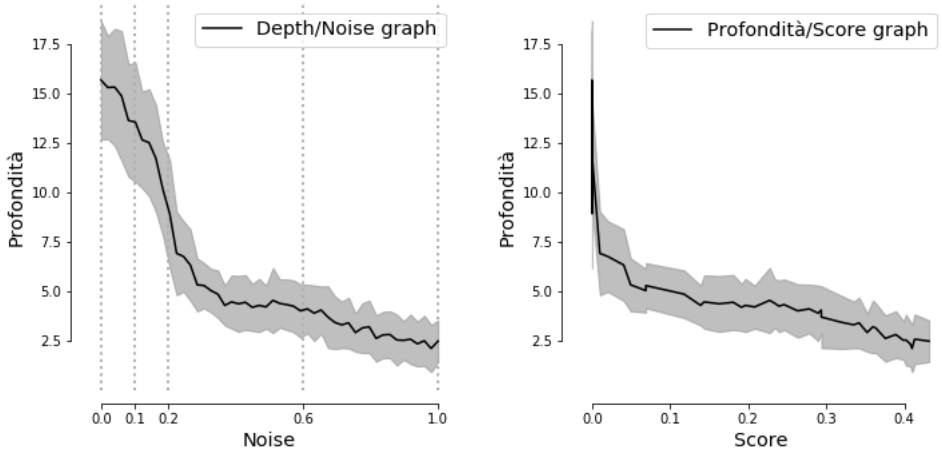

Figura 3.13: *Studio dell'andamento del numero della profondità della rete in funzione del noise e dello score per il dataset moons. In evidenza i punti di noise che mostrano andamenti interessanti(rispettivamente 0.1,0.2 e 0.6)*

Come per i link, anche in questo l'andamento della profondità in funzione del Noise è fortemente decrescente anche se sembra mostrare un accentuato plateau nel range di noise da 0.3 fino a noise = 0.6. La lunghezza delle reti avvicinandosi a score molto bassi tende a divergere, proprio come il numero di collegamenti.

Vengono riportate di seguito in figura 3.14 alcuni esempi di come si mostra un classificazione effettuta a diversi livelli di noise per *moons*. Le reti che hanno effetuato tali classificazione sono[17, 20, 24, 23, 24, 26, 20, 24, 23, 24, 26, 17, 9, 14, 6],[7, 12, 13, 24, 22], [24, 23, 13] e [6, 21]

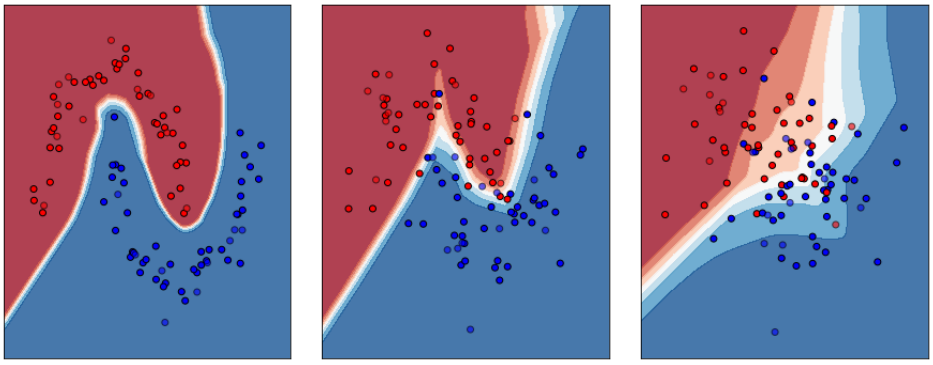

Figura 3.14: *in figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0.1,noise = 0.2 e noise = 0.6*

In figura 3.14 sono mostrati 3 momenti significativi del dataset: prima della presunta transizione, quando i punti sono ancora perfettamente separabili, durante la transizione a noise = 0.2 quando i punti iniziano a mischiarsi e a non essere più separabili dalla rete e infine a noise = 0,6, ben oltre la transizione, quando i punti sono definitavemente impossibili da separare se non approssimativamente.

3.2.2 Circles Dataset

Il secondo dataset analizzato è *circles*, di seguito vengono mostrati i dati ottenuti: In figura 3.15 viene mostrato l'andamento dello score in funzione del noise del dataset.

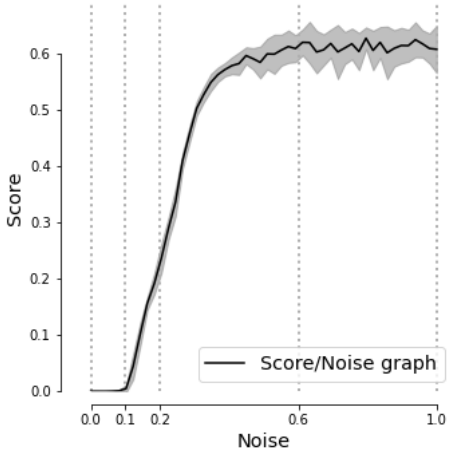

Figura 3.15: *Nel grafico viene mostrato l'andamento dello score in funzione del noise del dataset. In evidenza alcuni punti interessanti dell'andamento*

Notiamo che anche per *circles* esiste una zona di classificazione perfetta in cui lo score sembra indipendente rispetto al noise al dataset. Inoltre, a differenza di *moons*, qui lo score sembra raggiungere un livello di saturazione, simbolo del fatto che raggiunto un certo noise, i punti blu e rossi sono impossibili da separare per la rete.

Di seguito sono presentati i grafici delle tre grandezze che caratterizzano la complessità della rete per il dataset *circles*:

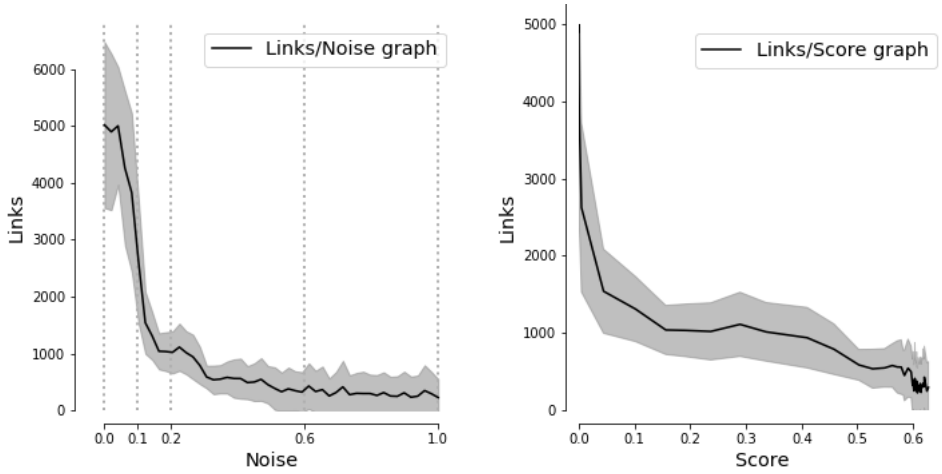

Figura 3.16: *Andamento dei collegamenti delle reti in funzione di noise e score*

In figura 3.16 vengono mostrati gli andamenti dei link in funzione del rumore del dataset ed in funzione dello score ottenuto dalla rete di convergenza. Vediamo che il numero di link tende a decrescere rapidamente nonostante nel range di noise [0;0.1] lo score della classificaziione sia costante. A differenza che nel caso di *moons* il numero di collegamenti sembra convergere. Si nota un plateau intorno a valori di noise = 0.2.

Nel secondo grafico invece osserviamo ancora una volta una divergenza per score bassi e quindi buone classificazioni, mentre un andamento molto rumoroso intorno ad un noise = 0.6

In figura 3.17 vengono mostrati gli andamenti relativi al numero di neuroni con layer più piccolo nella rete:

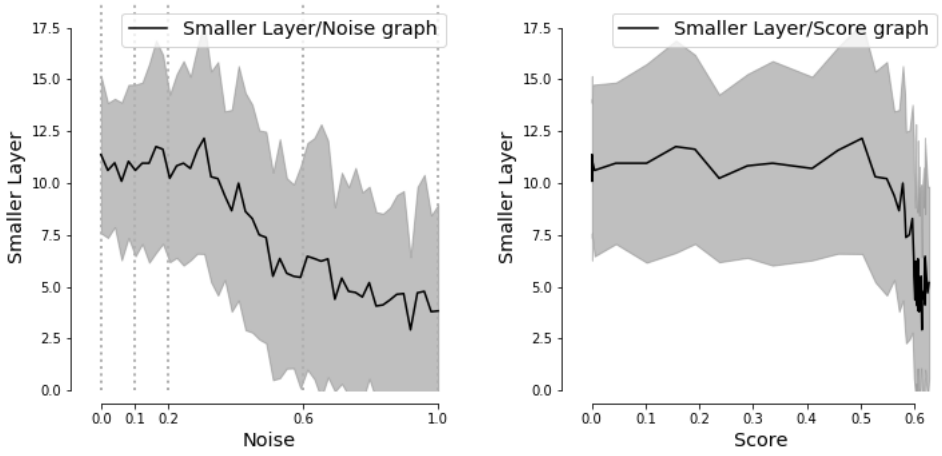

Figura 3.17: *Andamento dello smaller layer in funzione di noise e score.*

Notiamo un andamento decrescente nel numero di neuroni del layer più piccolo a differenza di quanto osservato per il precedente dataset. Ancora una volta la parte per score più alti e classificazioni peggiori risulta molto rumorosa per questo dataset.

In figura 3.18 vengono riportati i dati ottenuti sulla profondità della rete in funzione di noise e score.

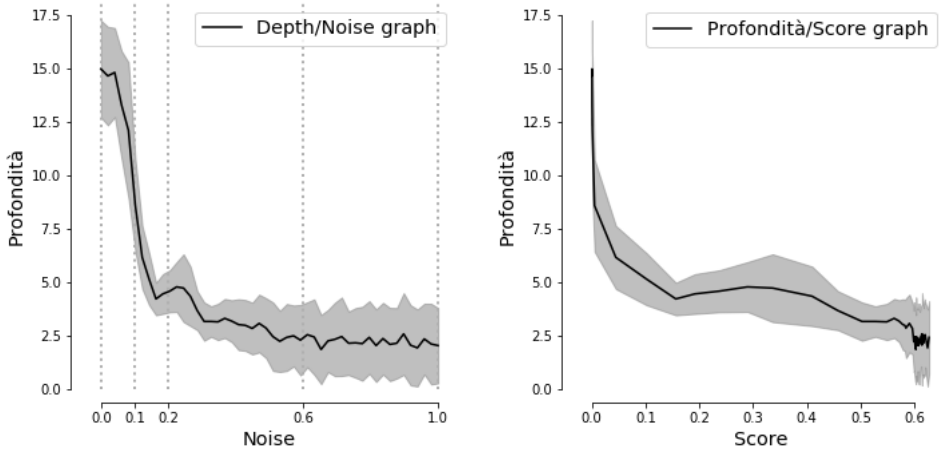

Figura 3.18: *In figura viene mostrato l'andamento della profondità delle reti in funzione del noise e dello score*

Gli andamenti appaiono simili a quelli osservati per i link nello stesso dataset.

In figura 3.19 sono raffigurati 4 esempi di classificazione di *circles* a livelli diversi di noise. Le reti che hanno compiuto queste classificazioni sono rispettivamente [16, 24, 24, 20, 24, 23, 24, 24, 23, 22, 24, 14, 24, 16, 14, 20], [19, 22, 19, 5, 14] e [18, 24, 18, 10]:

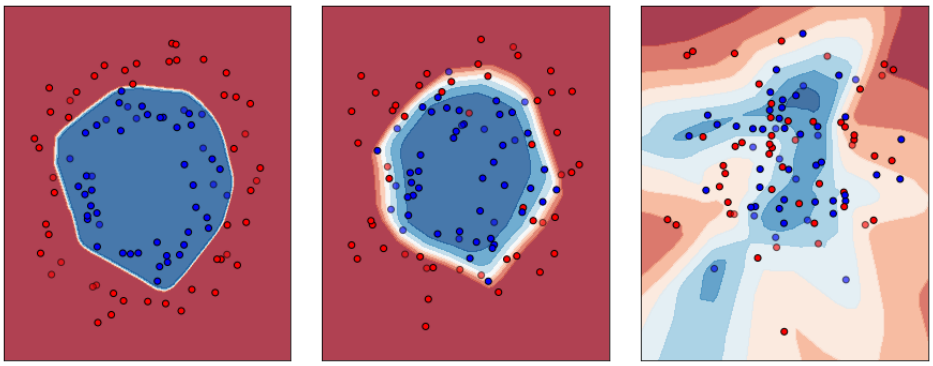

Figura 3.19: *in figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,1 quando ancora i punti sono separabili, noise = 0.2 e noise = 0.6 quando i punti sono pressochè indistinguibili dalla rete*

I tre esempi mostrano bene il cambiamento che compie il dataset durante la prova. Notiamo inoltre come l'algoritmo dopo un certo noise non riesca più a classificare efficacemente il dataset.

3.2.3 Circles+ Dataset

Terzo ed ultimo dataset presentato è *circles+* che combina due dataset *circles*. Nel grafico in figura 3.20 viene mostrato l'andamento del fitness della rete di convergenza in funzione del noise del dataset.

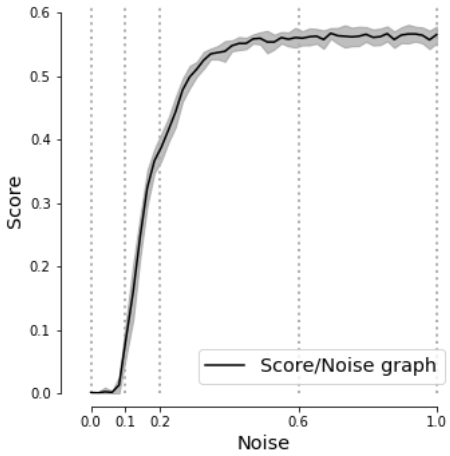

Figura 3.20: *Andamento dello score in funzione del noise per il dataset circles+*

Anche in questo caso si nota un range di noise per cui lo score della classificazione rimane costante a 0(più ristretto di quelli precedenti) e infine si nota un valore di saturazione per lo score.

In figura 3.21 sono comparati gli andamenti dei link in funzione di noise e score:

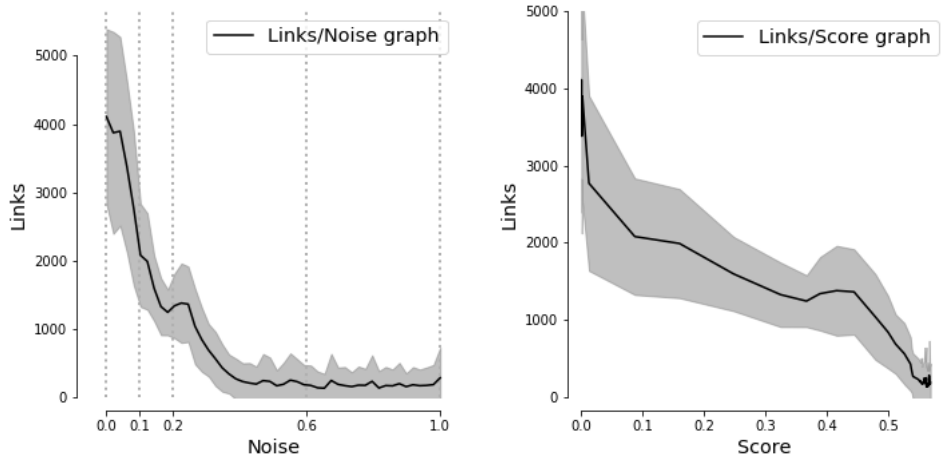

Figura 3.21: *Andamento dei links in funzione del noise del dataset e dello score ottenuto*

L'andamento del grafico dei links in funzione del noise è simile a quelli precedenti ma presenta uno *spike* intorno al valore di noise = 0.2. Come per circle, sembra convergere ad un valore stabile.

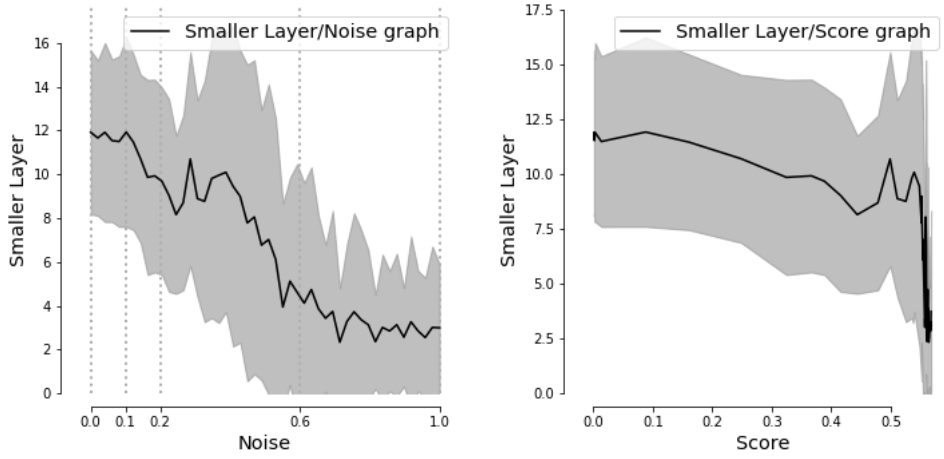

Figura 3.22: *Andamento del numero di neuroni nel layer più piccolo in funzione del noise e dello score ottenuto*

Come per il dataset *circle* anche qui, osserviamo che il numero di neuroni nel layer minore diminusce all'aumentare del noise.

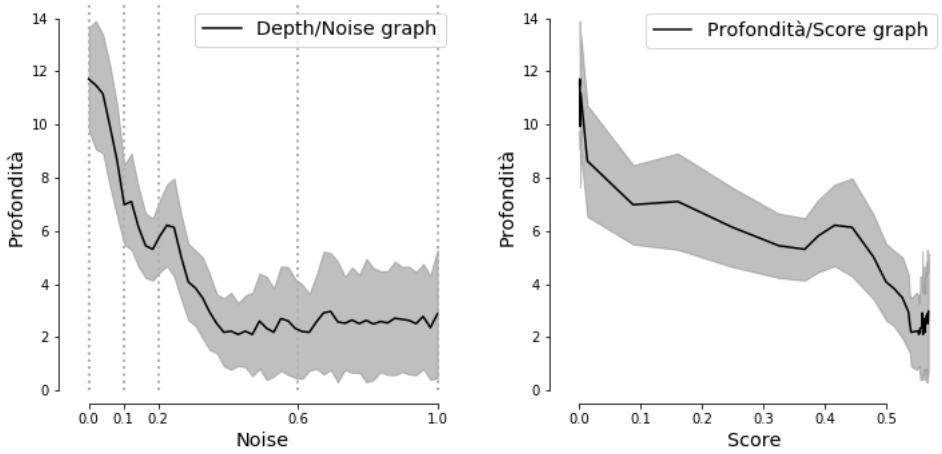

Figura 3.23: *Andamento della profondità in funzione del noise e dello score ottenuto*

Anche in questo caso, l'andamento della lunghezza in funzione del noise è molto simile all'andamento del numero di links.

In figura 3.24 vengono mostrati degli esempi di classificazione con diversi valori di noise. Le reti alle quali l'algoritmo converge in questi casi sono rispettivamente: [24, 21, 20, 21, 20, 21, 4, 14, 21, 24, 6, 21], [22, 24, 24, 12] e []

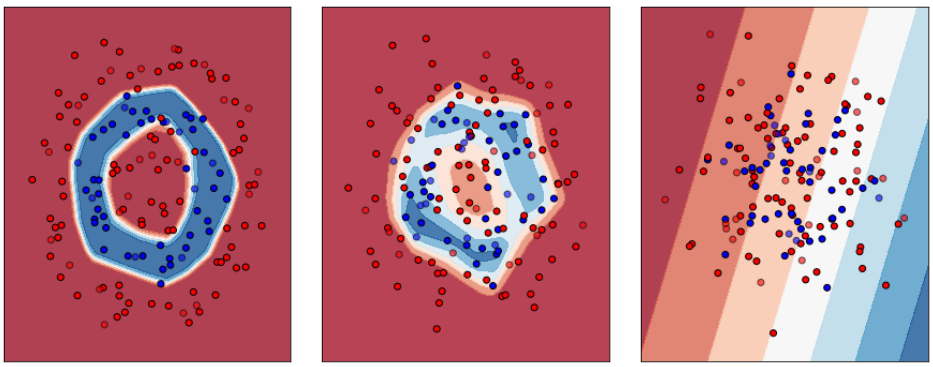

Figura 3.24: *In figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,noise = 0.2 e noise = 0.6*

Nel caso di *circles+* si raggiunge in fretta un livello di noise tale da rendere impossibile all'algoritmo una classificazione anche approssimativa del dataset, tuttavia è ancora possibile ottenre una classificazione accurata per noise = 0.1 È facile immaginare che le classificazioni per noise $>$ 0,6 non siano molto diverse da quella mostrata.

Capitolo 4

Conclusioni

In questo capitolo verranno discussi i principali risultati presentati nel capitolo precedente. La separazione compiuta dall'algoritmo sui dataset composti da due classi di punti sembra essere buona sotto una certa soglia di rumore e ciò vale sia per l'*accuracy score* sia per la *logarithmic loss*, nonostante esistano delle nette differenze tra le due in capacità di classificazione, almeno per quanto osservato nei dataset *circles* e *moons*, nei quali la seconda risulta più precisa e meno approssimativa.

Nel terzo dataset invece il comportamento sembra simile anche se, ancora una volta, la *log loss* raggiunge un risultato più preciso se si considerano le zone di bordo tra punti rossi e blu. In generale, possiamo dire che per quanto riguarda questi semplici casi di classificazione di punti l'algoritmo riesce a trovare un individuo in grado di compiere una separazione più o meno netta (ciò dipende dal noise) nonostante lo spazio di ricerca sia molto vasto, ottenendo risultati migliori se invece di considerare la semplice risposta binaria della rete (blu/rosso) si considera la sicurezza con la quale la rete attribuisce tale risposta e si cerca di massimizzare tale certezza.

Naturalmente ciò non significa che l'MLP restituito dalla convergenza dell'algoritmo sia il migliore in termini di classificazione o semplicemente in termini di costo computazionale, sia per il suo addestramento sia nel calcolare la sua risposta, che in caso di problemi complessi rappresenta sicuramente un problema.

Per questo si è voluto analizzare anche la complessità della rete, quantificandola in tre parametri: Link, Numero di hidden layer (profondità o lunghezza) e numero di neuroni nel layer più piccolo. Ricordando che non esiste nell'algoritmo genetico nessuna forma di penalità per reti troppo "grandi", sembra scontato affermare che in qualsiasi caso l'algoritmo tenderà a preferire un numero molto grande di hidden layer aumentandolo di generazione in generazione dato che in generale reti di questo tipo dovrebbe avere performance di classificazione migliore. In realtà ciò accade solo nel caso di rumore molto basso o nullo: diminuendolo infatti in tutti e tre i dataset la lunghezza e il numero di link della rete crescono molto rapidamente, il che sembra rappresentare un comportamento totalmente anti-intuitivo.

In altre parole la complessità della rete aumenta al diminuire della complessità del dataset: questa caratteristica potrebbe essere dovuta ad una sorta di penalizzazione interna all'addestramento degli MLP. Infatti, la separazione tra *train set* e *test set* fa in modo che le reti che ottengono alti score sul primo (*overfitting*) vengano in seguito penalizzate sul secondo.

Questo si osserva su noise elevati ed in tutti e tre i casi sembra esistere un valore di soglia oltre il quale

Elenco delle figure

1.1 *Perceptron con 3 input ed un output* 7
1.2 *confronto tra due funzioni di attivazione: a sinistra sigmoidale e a destra ReLU* 8
1.3 *Esempio di rete neurale fully connected in cui viene mostrata la distinzione tra input layer, hidden layer e output layer* 8

2.1 *In figura viene mostrato il risultato di 100000 estrazioni di serie da 3(in nero) e 5(in blu) interi casuali (da 1 a 100) dalle quali viene estratta la mediana. Aumentando il numero di numeri casuali da cui estrarre la mediana, il picco si stringe e si alza al centro. In viola un'estrazione uniforme.* 11
2.2 *Esempio di MLPClassier generato da un individuo con cromosoma [3,2]* . 13
2.3 *Nell'immagine sono messe a confronto due versioni del dataset moons: a sinistra priva di noise e a destra con noise = 0.1* 14
2.4 *Nell'immagine a confronto due versioni del dataset circles: a sinistra senza noise e a destra con noise = 0.1. Entrambe con lo stesso fattore tra i cerchi* 15
2.5 *Nell'immagine a confronto due versioni del dataset circles+: a sinistra con noise = 0 e a destra con noise = 0.1* 15

3.1 *L'immagine mostra una run tipica usando l'accuracy score e la random mate selection e noise = 0.25* 18
3.2 *Best fitness di ogni generazione della popolazione con accuracy score e random mate* 19
3.3 *In figura viene mostrata una run d'esempio utilizzando lo stesso noise e stessa funzione di fitness ma diverso algoritmo di selezione* 19
3.4 *In figura viene mostrato l'andamento del best fitness in funzione della generazione* 20
3.5 *Evoluzione di una rete neurale seguendo la logarithmic loss function sul dataset moons* 20
3.6 *L'immagine mostra una run d'esempio nel dataset circles usando l'accuracy score* 21
3.7 *L'immagine mostra una run d'esempio nel dataset circles usando la log loss* 22

3.8 *L'immagine mostra una run d'esempio nel dataset circles+ usando l'accuracy score* 23
3.9 *L'immagine mostra una run d'esempio nel dataset circles+ usando la log loss con noise = 0.1* 24
3.10 *Nel grafico viene mostrato l'andamento dello score in funzione del noise del dataset. In evidenza alcuni punti interessanti dell'andamento* 26
3.11 *A confronto gli andamenti del numero di collegamenti in funzione del noise ed in funzione dello score* 26
3.12 *Nei due grafici sono mostrati gli andamenti del numero di neuroni nel layer più piccolo della rete in funzione del noise di moons e dello score* . 27
3.13 *Studio dell'andamento del numero della profondità della rete in funzione del noise e dello score per il dataset moons. In evidenza i punti di noise che mostrano andamenti interessanti(rispettivamente 0.1,0.2 e 0.6)* . . . 28
3.14 *in figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0.1,noise = 0.2 e noise = 0.6* 28
3.15 *Nel grafico viene mostrato l'andamento dello score in funzione del noise del dataset. In evidenza alcuni punti interessanti dell'andamento* 29
3.16 *Andamento dei collegamenti delle reti in funzione di noise e score* 30
3.17 *Andamento dello smaller layer in funzione di noise e score.* 30
3.18 *In figura viene mostrato l'andamento della profondità delle reti in funzione del noise e dello score* 31
3.19 *in figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,1 quando ancora i punti sono separabili, noise = 0.2 e noise = 0.6 quando i punti sono pressochè indistinguibili dalla rete* 31
3.20 *Andamento dello score in funzione del noise per il dataset circles+* 32
3.21 *Andamento dei links in funzione del noise del dataset e dello score ottenuto* 33
3.22 *Andamento del numero di neuroni nel layer più piccolo in funzione del noise e dello score ottenuto* 33
3.23 *Andamento della profondità in funzione del noise e dello score ottenuto* . 34
3.24 *In figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,noise = 0.2 e noise = 0.6* 34

Bibliografia

[1] Melanie Mitchell. *An Introduction to genetic algorithm.* The MIT Press, 1999.

[2] Kevin Gurney. *An Introduction to Neural Networks.* UCL Press, 1997.

[3] Michael A. Nielsen. *Neural Network and Deep Learning.* Determination Press, 2015.

[4] Christopher M. Bishop. *Pattern Recognition and machine learning.* Springer, 2006.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[7] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* New edition edition, 2002.