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

Presentata da:
Mattia Ceccarelli

Correlatore:
Prof./Dott. Nico Curti

Anno Accademico 2017/2018

Indice

Capitolo 1

Introduzione

In questo capitolo si introdurranno i principali mezzi utilizzati nello svolgimento del progetto di tesi, ossia Algoritmi Genetici per la ricerca dei minimi di una funzione non parametrica ????? e Reti Neurali *fully connected*, che svolgono il ruolo di funzione a molti parametri da ottimizare in un problema di classificazione.

1.1 Algoritmi Genetici

Gli algoritmi genetici sono software di ricerca ispirati dalla selezione naturale applicata ad una popolazione di individui, chiamati soluzioni, caratterizzati da un *cromosoma*, spesso rappresentato da una lista di numeri binari o da una stringa. 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.

1.1.1 Operatori

Mitchell [1999] I principali operatori che compongo un semplice algoritmo genetico sono:

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 Mitchell [1999] 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*.

1.1.3 Applicazioni

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 ottimizzare.
Da migliorare

1.1.4 Drift genetico

posso scrivere qualcosa su questo visto che viene osservato.

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*. Gurney [1997]

1.2.1 Perceptron

Nielsen [2015] 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:

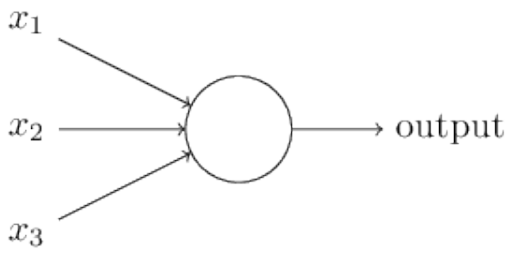

Figura 1.1: *Perceptron con 3 input ed un output*

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 [Nielsen [2015]], 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:

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

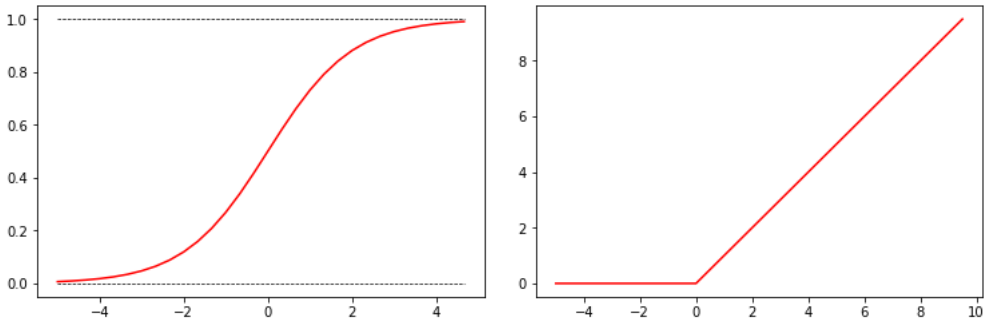

Figura 1.2: *confronto tra due funzioni di attivazione: a sinistra sigmoidale e a destra ReLU*

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:

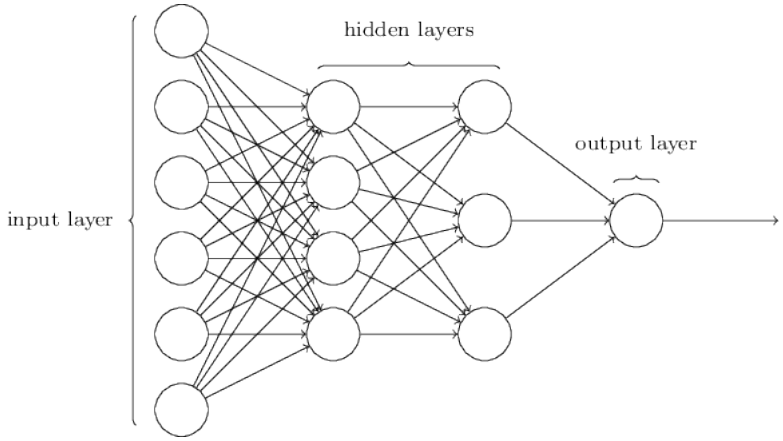

Figura 1.3: *Esempio di rete neurale fully connected in cui viene mostrata la distinzione tra input layer, hidden layer e output layer*

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.

Da completare con accenno a BackPropagation

1.2.3 Adam

1.2.4 Evoluzione di una Rete Neurale

1.3 Dirichlet Process o Chinese Restaurant

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.

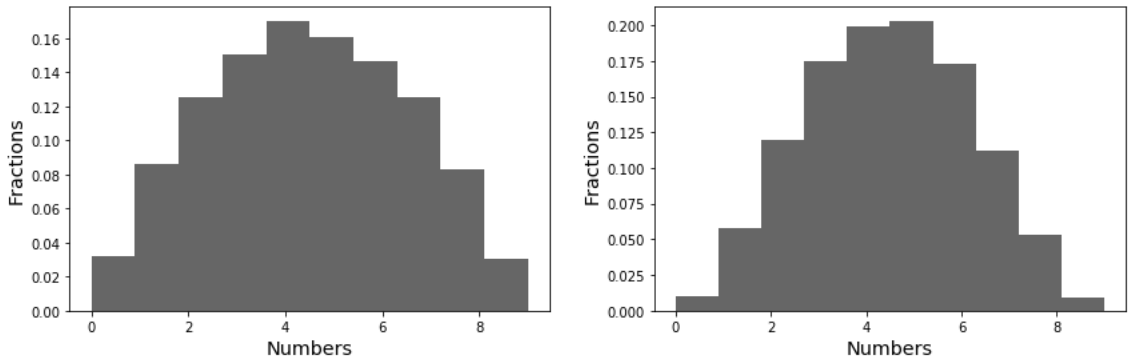

Figura 2.1: *In figura viene mostrato il risultato di 10000 estrazioni di 3 (a sinistra) e 5 (a destra) interi casuali dai quali viene estratta la mediana. Aumentando il numero di numeri casuali da cui estrarre la mediana il picco si stringe e si alza*

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* (Pedregosa et al. [2011]).

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:

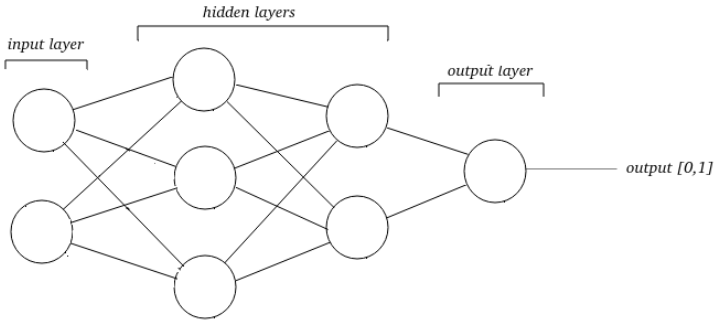

Figura 2.2: *Esempio di MLPClassier generato da un individuo con cromosoma [3,2]*

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 retio o ,più in generale, a quanti collegamenti 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).

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

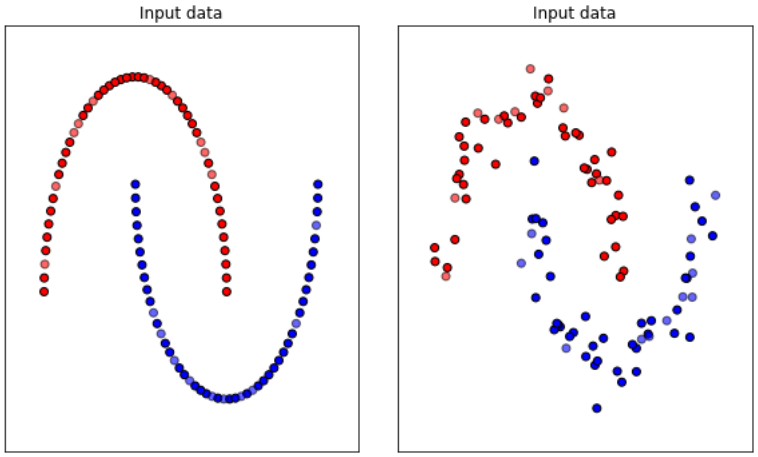

Figura 2.3: *Nell'immagine sono messe a confronto due versioni del dataset moons: a sinistra priva di noise e a destra con noise = 0.1*

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

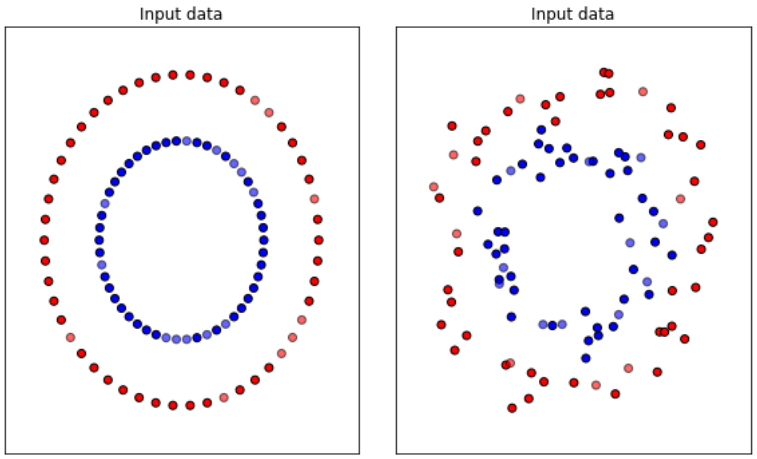

Figura 2.4: *Nell'immagine a confronto due versioni del dataset circles: a sinistra senza noise e a destra con noise = 0.1. Entrambe con lo stesso fattore tra i cerchi*

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

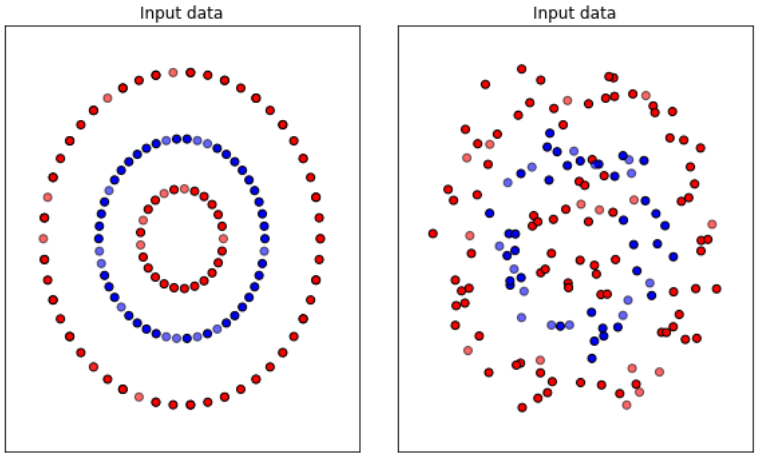

Figura 2.5: *Nell'immagine a confronto due versioni del dataset circles+: a sinistra con noise = 0 e a destra con noise = 0.1*

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 e la deviazione standard.

Capitolo 3

Risultati

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

3.1 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*.

3.1.1 *Accuracy score*

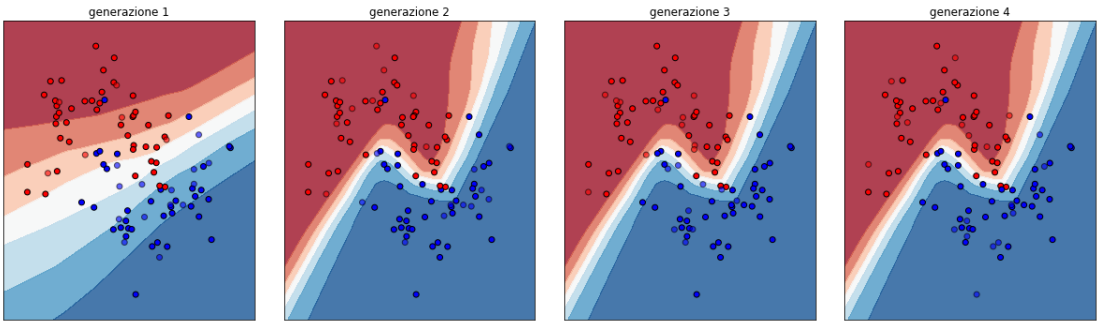

Figura 3.1: *L'immagine mostra una run tipica usando l'accuracy score e la random mate selection e noise = 0.25*

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 1.0 / 1.0) 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. 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

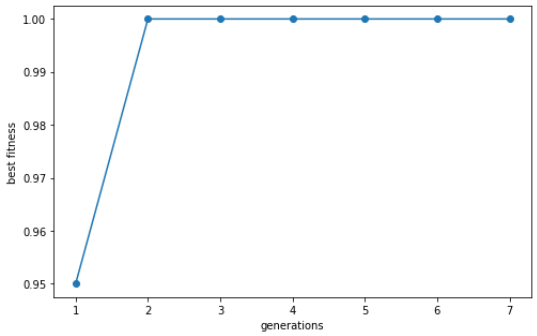

Figura 3.2: *Best fitness di ogni generazione della popolazione con accuracy score e random mate*

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:

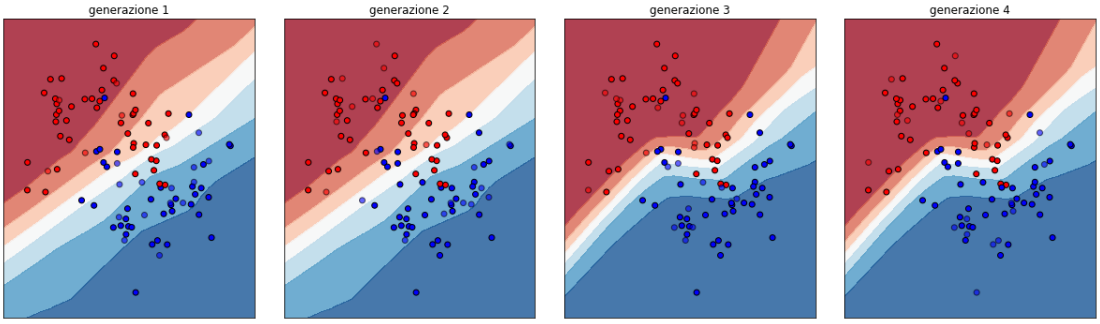

Figura 3.3: *In figura viene mostrata una run d'esempio utilizzando lo stesso noise e stessa funzione di fitness ma diverso algoritmo di selezione*

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):

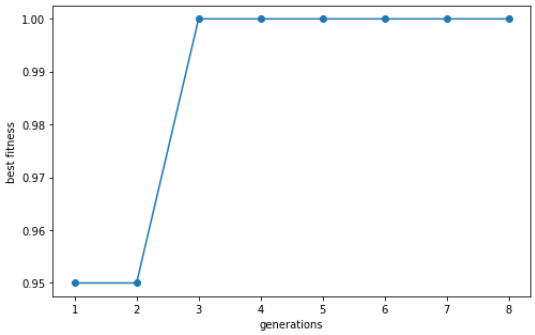

Figura 3.4: *In figura viene mostrato l'andamento del best fitness in funzione della generazione*

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

3.1.2 *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

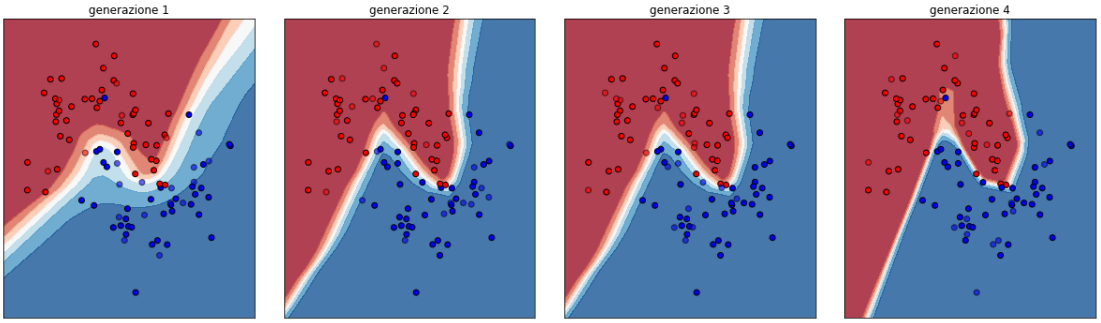

Figura 3.5: *Evoluzione di una rete neurale seguendo la logarithmic loss function sul dataset moons*

3.2 Circles Dataset

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

3.2.1 *Accuracy score*

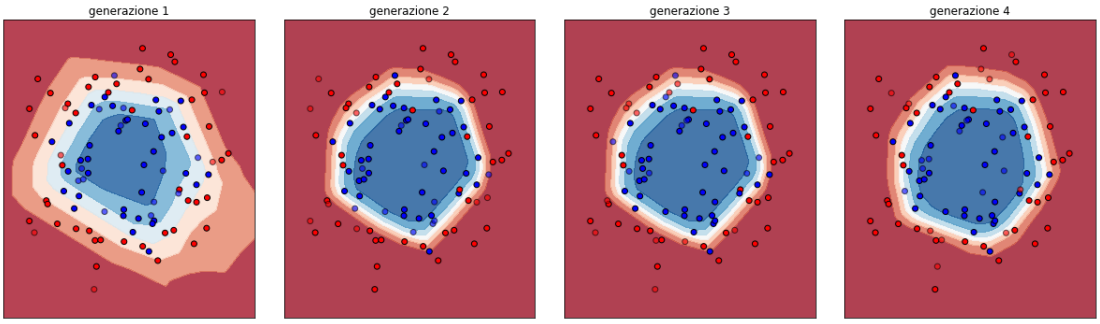

Figura 3.6: *L'immagine mostra una run d'esempio nel dataset circles usando l'accuracy score*

Anche in questo caso la convergenza dell'algoritmo è rapida.

3.2.2 *logarithmic loss*

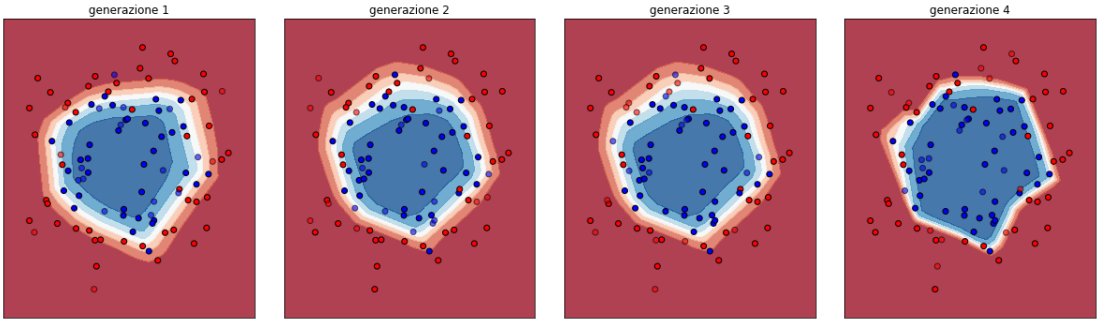

Figura 3.7: *L'immagine mostra una run d'esempio nel dataset circles usando la log loss*

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

3.3 Circles+ Dataset

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

3.3.1 *Accuracy score*

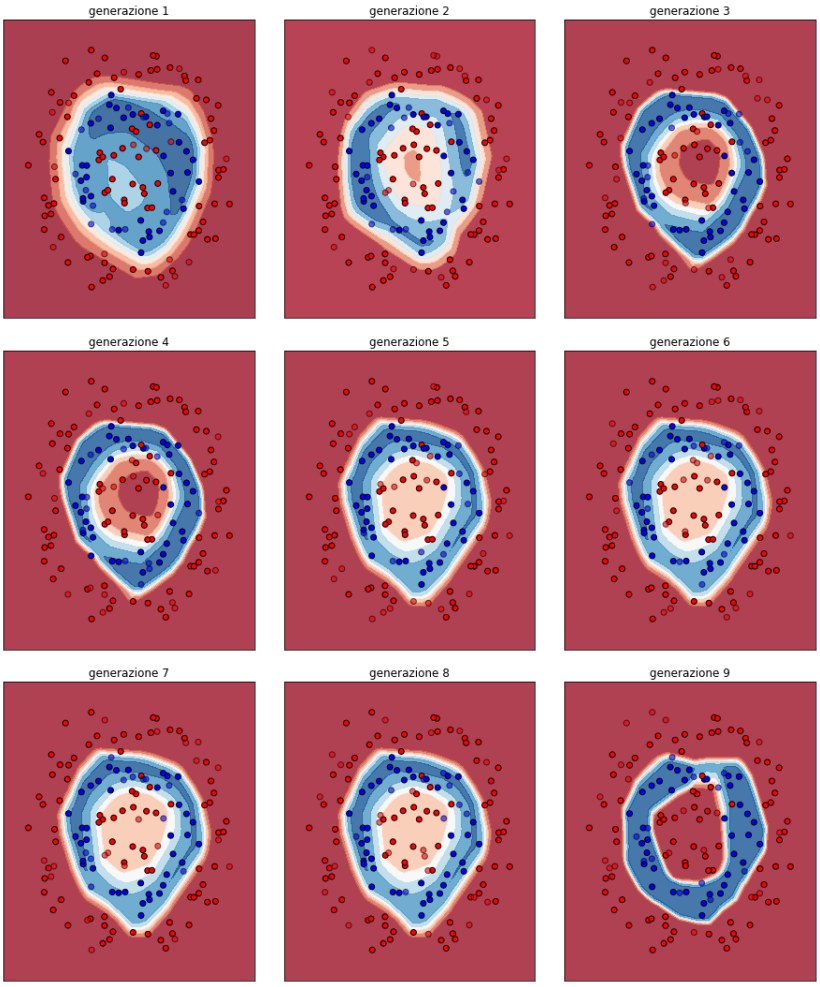

Figura 3.8: *L'immagine mostra una run d'esempio nel dataset circles+ usando l'accuracy score*

3.3.2 *Logarithmic loss*

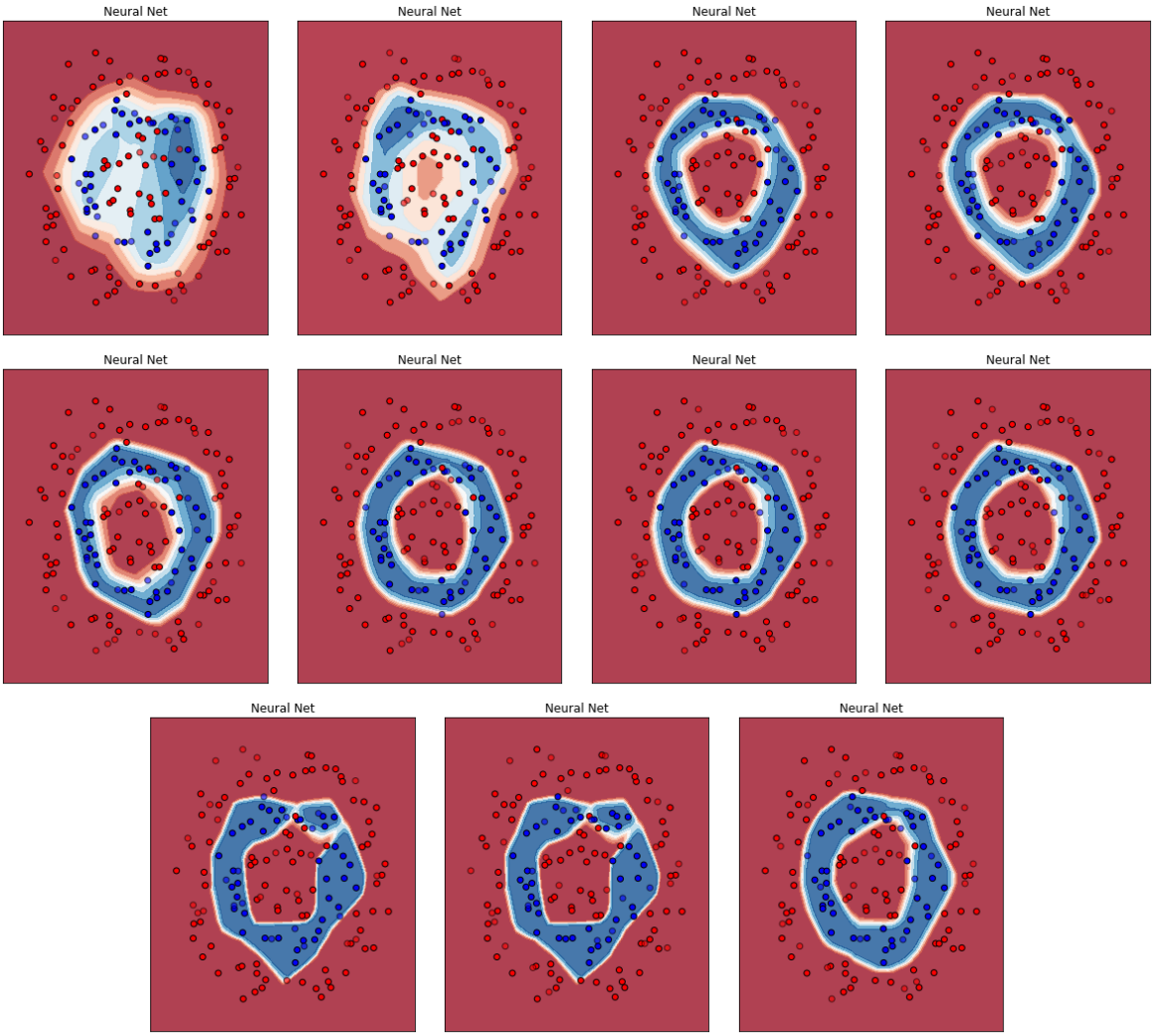

Figura 3.9: *L'immagine mostra una run d'esempio nel dataset circles+ usando la log loss*

In questo caso la convergenza è molto simile nelle due funzioni di fitness. In entrambi i casi è possibile osservare un'evoluzione più lenta rispetto ai primi due dataset.

3.4 Complessità della rete

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

- Collegamenti
- Numero di neuroni del layer con 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.

In figura 3.10 sono mostrati i dati relativi al dataset *moons*:

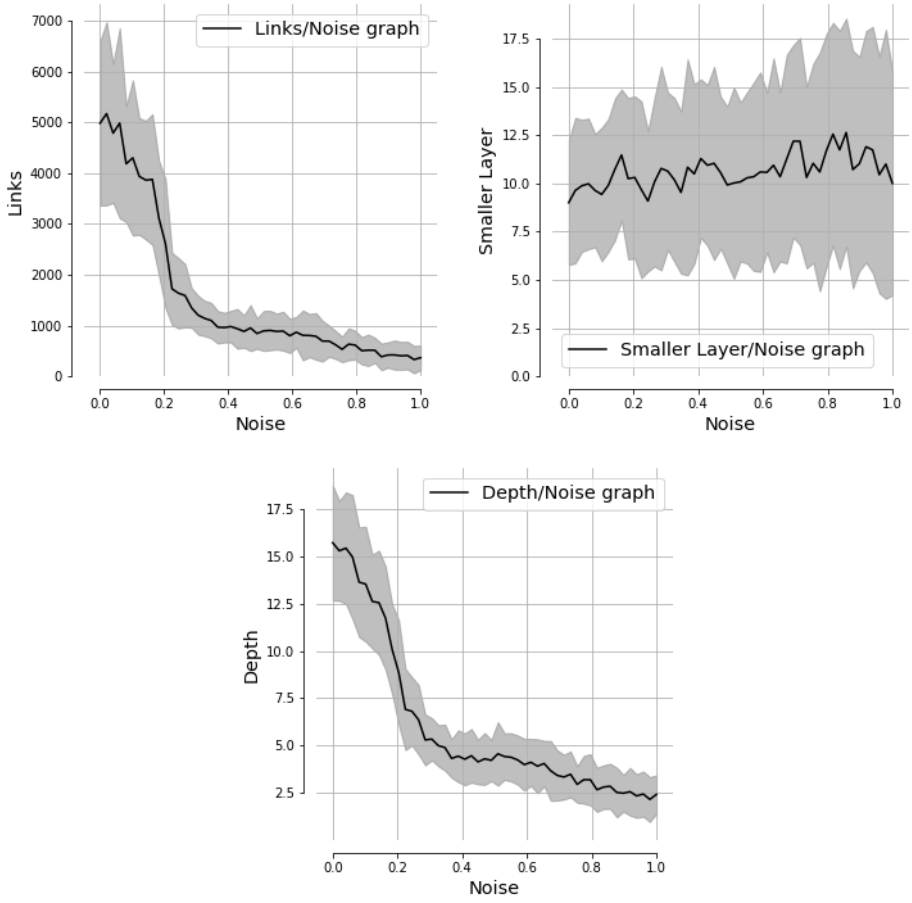

Figura 3.10: *Studio dell'andamento del numero di links(a sinistra), del layer più piccolo(a destra) e della profondità della rete (in basso) in funzione del noise per il dataset moons*

Links (in alto a sinistra) a differenza di quanto atteso, ad un dataset più difficile non corrisponde una rete più complessa. 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.

Smaller Layer : (in alto a destra) il numero di neuroni del layer più piccolo sembra rimanere costante.

Profondità (in basso) anche il numero di *hidden layers* della soluzione seguono un andamento simile ai collegamenti, con la differenza che sembra mostrare un plateu nel range di noise = [0.3;0.5].

Vengono riportate in figura 3.11 alcuni esempi di come si mostra un classificazione efettuta 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]

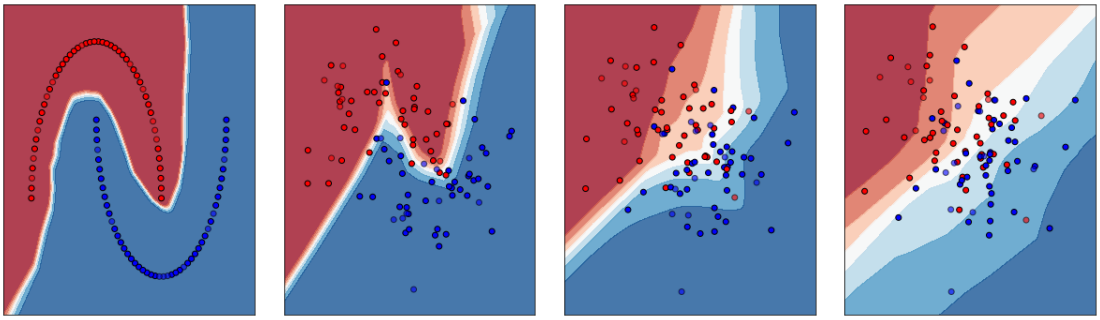

Figura 3.11: *in figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,noise = 0.2 , noise = 0.6 e infine noise = 1*

Il secondo dataset analizzato è *circles*, in figura 3.12 vengono mostrati i dati ottenuti:

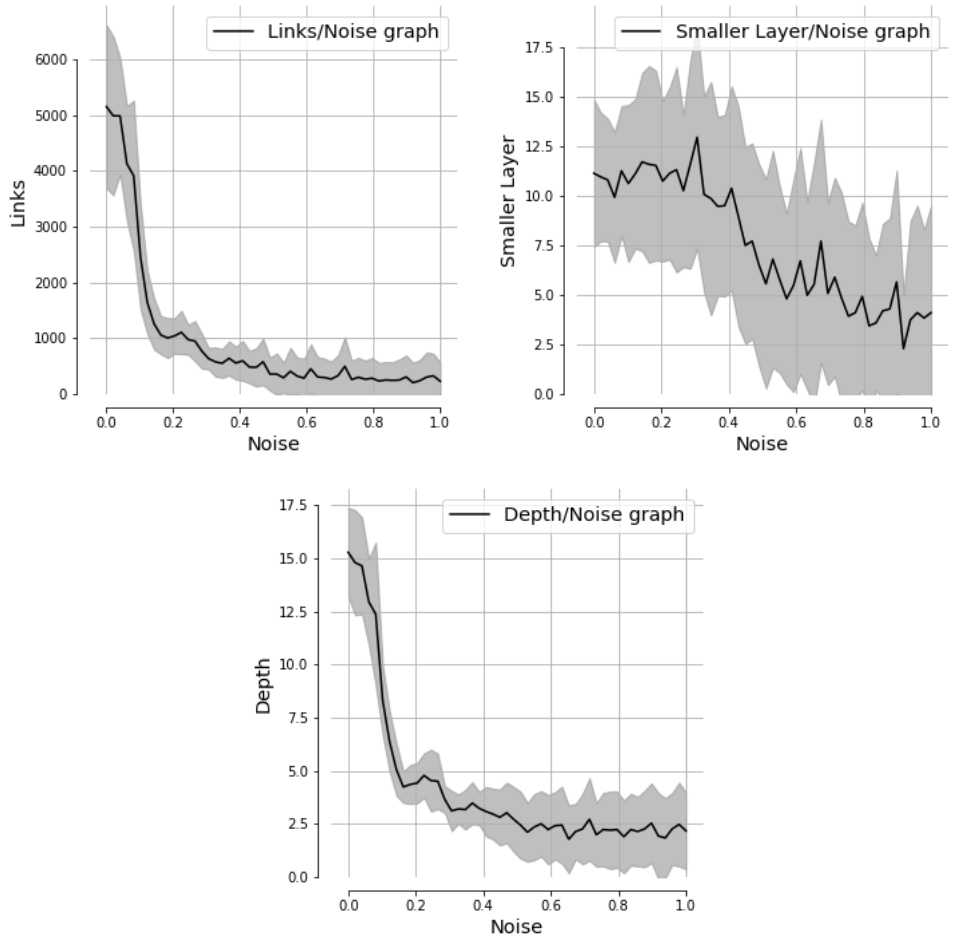

Figura 3.12: *Studio dell'andamento del numero di links(a sinistra), del layer più piccolo (a destra) e della profondità della rete (in basso) in funzione del noise per il dataset circles*

Links (in alto a sinistra) vediamo che il numero di link è in rapida discesa all'aumentare del noise fino a stabilizzarsi quando i due cerchi sono indistinguibili. Si nota un plateau in corrispondenza dei valori di noise = 0.2

Smaller Layer (in alto a destra) in questo caso il numero di neuroni nel layer più piccolo diminuisce all'aumentare del rumore.

Profondità (in basso) l'andamento è ancora una volta discedente e presenta un plateau attorno a valore di noise = 0.2.

In figura 3.13 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], [18, 24, 18, 10] e [2]:

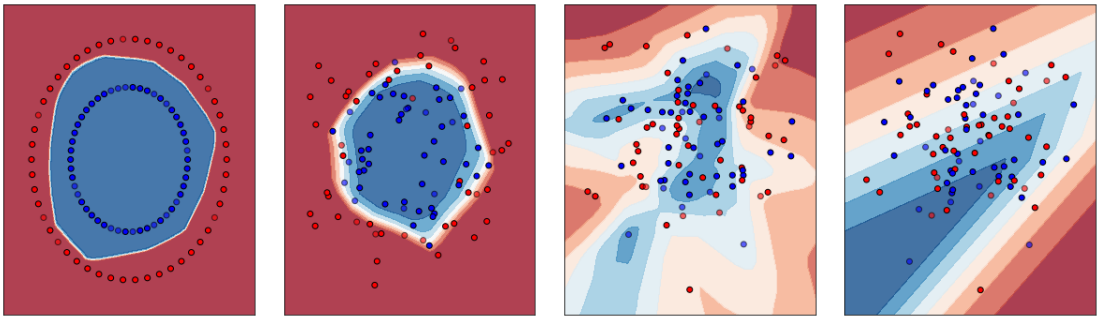

Figura 3.13: *in figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,noise = 0.2 , noise = 0.6 e infine noise = 1*

I quattro 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.

Il terzo ed ultimo dataset è *circles+*

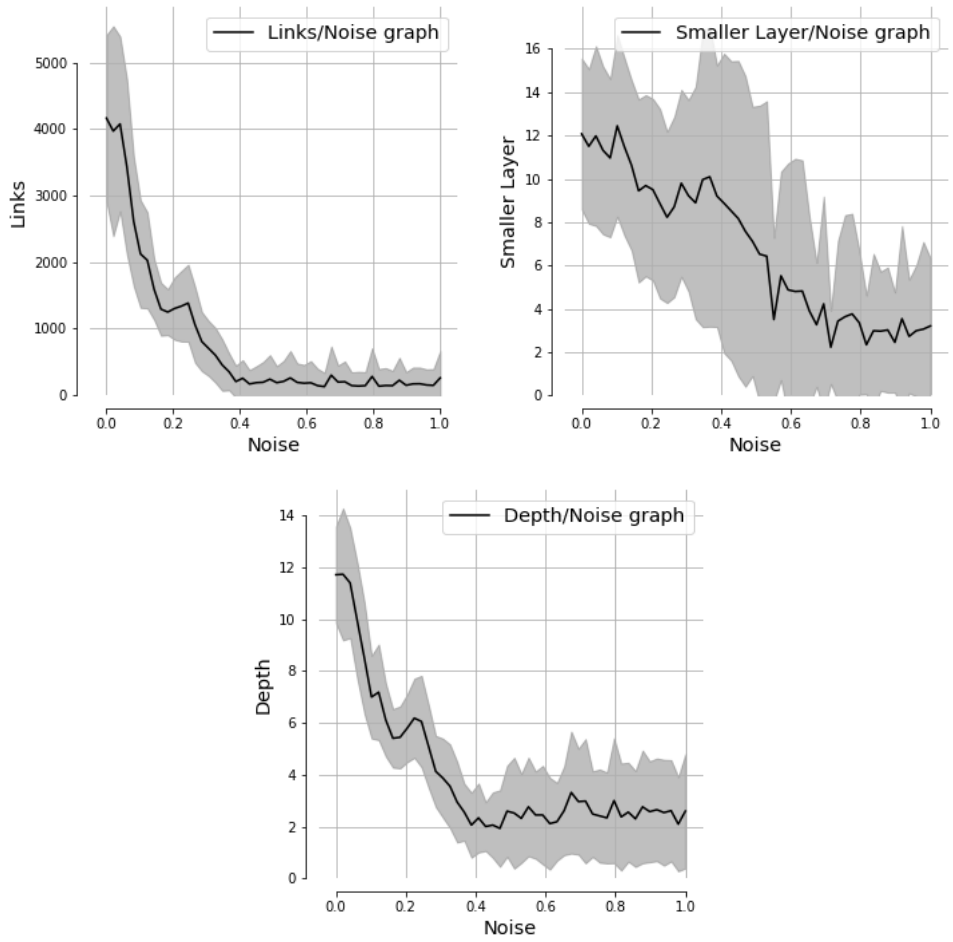

Figura 3.14: *Studio dell'andamento del numero di links(a sinistra), del layer più piccolo(a destra) e della profondità della rete (in basso) in funzione del noise per il dataset circles+*

Links (in alto a sinistra) il numero di link diminuisce rapidamente con l'aumentare del noise,sino a stabilizzarsi. Si nota una discontinuità in un range di noise = [0.1,0.3]

Smaller layer (in alto a destra) anche in questo dataset il numero di neuroni nel layer più piccolo diminuisce all'aumentare della complessità del dataset.

Profondità la lunghezza della rete diminuisce rapidamente fino a stabilizzarsi ad un valore di noise = 0.4. Anche qui si vede un andamento particolare in corrispondenza di valori di noise = [0.2,0.3].

In figura 3.15 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 []

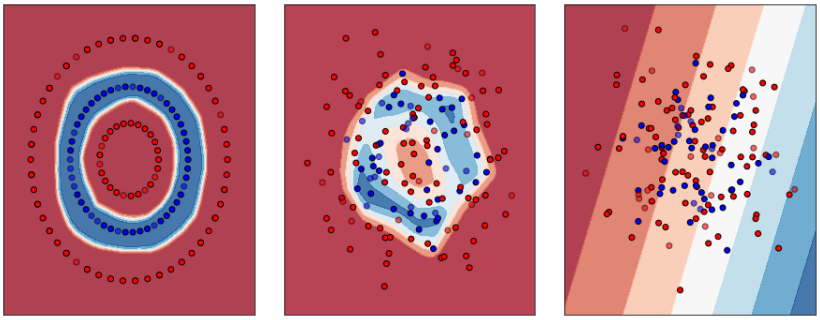

Figura 3.15: *In figura viene mostrata una classificazione d'esempio su diversi livelli di noise, da sinistra : noise = 0,noise = 0.2 e noise = 0.4*

Nel caso di *circles+* si raggiunge in fretta un livello di noise tale da rendere impossibile all'algoritmo una classificazione anche approssimativa del dataset. È facile immaginare che le classificazioni per noise $>$ 0,4 non siano molto diverse da quella mostrata.

Capitolo 4

Conclusioni

In questo capitolo verranno discussi i principali risultati presentati nel capitolo precedente

Elenco delle figure

Bibliografia

Kevin Gurney. *An Introduction to Neural Networks*. UCL Press, 1997. ISBN 0-203-45151-1.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Melanie Mitchell. *An Introduction to genetic algorithm*. The MIT Press, 1999. ISBN 0262133164.

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

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.