ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA IN INGEGNERIA ELETTRONICA,
INFORMATICA E TELECOMUNICAZIONI

APPRENDIMENTO AUTOMATICO NEI GIOCHI DI STRATEGIA

Elaborata nel corso di: Fondamenti di Informatica B

Tesi di Laurea di:
MARCO STEFENELLI

Relatore:
Prof. ANDREA ROLI

ANNO ACCADEMICO 2013–2014
SESSIONE II

PAROLE CHIAVE

Apprendimento Automatico
Annotazioni di Gioco
Giochi di Strategia
Scacchi
Tris

Alla mia famiglia ed ai miei amici che mi sono stati accanto in tutti questi anni.

Indice

Introduzione

L'Apprendimento Automatico è quella area dell'Intelligenza Artificiale che si occupa di realizzare algoritmi e sistemi capaci di sintetizzare nuove conoscenze a partire da dati forniti in ingresso.

Lo scopo di questo elaborato consiste principalmente in una presentazione generale e teorica dei fondamenti di questa materia, dei suoi paradigmi principali e dell'applicazione concreta di una di queste tecniche di apprendimento automatico nei giochi di strategia.

Infatti i giochi, in particolare quelli di strategia, offrono all'Apprendimento Automatico un ambiente ideale e privilegiato per effettuare test, in quanto possono fungere da modelli per problemi reali.

Nel trattare questi argomenti ho considerato due casi studio differenti. In primis ho descritto il processo che si cela dietro l'addestramento di una rete neurale per giocare a Tris seguendo il lavoro di di H. Mannen[9]. Ho deciso di cominciare proprio con il Tris, poiché, pur non essendo un gioco particolarmente complesso, presenta tutte le caratteristiche necessarie per produrre un buon caso di studio iniziale. Successivamente ho illustrato l'esperienza di Christian Wirth and Johannes Furnkranz, descritta nell'articolo "On Learning From Annotations" del 2014 [12], nel quale viene proposto di impiegare in maniera produttiva una grande risorsa finora poco utilizzata, le annotazioni di gioco degli scacchi, che in tutti questi anni sono state raccolte e catalogate in libri e database. Inoltre è da sottolineare il fatto che tale lavoro è stato realizzato in un passato assai recente, e per questo motivo, mi ha offerto un'immagine attendibile dello stato dell'arte dell'Apprendimento Automatico nei giochi.

La struttura della tesi è la seguente:

Nel primo capitolo viene introdotto il concetto di apprendimento e di Apprendimento Automatico. Successivamente si definiscono le linee guida necessarie per formalizzare un generico problema di apprendimento e infine vengono presentati i passi progettuali necessari per procedere allo sviluppo di un semplice sistema di Apprendimento Automatico.

Nel secondo capitolo vengono illustrati i tre principali metodi di Apprendimento Automatico (Apprendimento Supervisionato, Apprendimento Non Supervisionato e Apprendimento per Rinforzo) e per ciascuno di essi è riportato un paradigma che applica il relativo metodo di apprendimento.

Nel terzo capitolo, dopo una breve introduzione sul ruolo dei giochi nell'intelligenza artificiale, vengono illustrati i casi di studio analizzati cominciando con quello relativo all'addestramento di una rete neurale per il gioco del Tris e concludendo con l'esperienza che mostra come sia possibile addestrare un giocatore di scacchi attraverso le annotazioni di gioco.

L'ultimo capitolo contiene invece considerazioni e conclusioni in merito ai casi di studio esaminati ed alcune previsioni sui possibili sviluppi futuri dell'Apprendimento Automatico nei giochi.

Capitolo 1

Apprendimento Automatico

Per la realizzazione di questa parte teorica introduttiva ho preso ispirazione da alcuni libri di testo quali "Machine Learning" di Tom Mitchell [1] e "Introduction to Machine Learning" di Ethem Alpaydin [2].

1.1 Apprendimento

L'Apprendimento Automatico è la branca dell'Intelligenza Artificiale che studia metodi di apprendimento per la creazione di sistemi intelligenti. Questi sistemi sono addestrati attraverso l'utilizzo di algoritmi di apprendimento specifici per ogni determinato problema o compito.

Una prima definizione ci viene fornita nel 1984 da *H.Simon* che affermò:

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time".

"L'apprendimento denota cambiamenti adattivi nel sistema, nel senso che essi abilitano il sistema a compiere uno stesso compito per una stessa popolazione in una maniera più efficiente rispetto a come era accaduto precedentemente ."

R.Michalski invece, nel 1986, lo definisce più semplicemente come:

"Learning is constructing or modifying representations of what is being experienced".
"L'apprendimento è costruire o modificare le rappresentazioni di ciò di cui si ha avuto esperienza".

Da queste definizioni si deduce come l'apprendimento rappresenti una forma di adattamento del sistema all'ambiente attraverso l'esperienza, analogamente a quanto avviene a noi esseri umani e in generale ad ogni essere vivente. Questo adattamento del sistema all'ambiente attraverso l'esperienza deve essere tale da portare a un miglioramento senza però contare sul continuo intervento della mano dell'uomo.

Per raggiungere tale obiettivo il sistema deve essere in grado di apprendere, cioè deve essere capace di estrarre informazioni utili su un determinato problema esaminando una serie di esempi ad esso associati.

Infatti, l'apprendimento è un processo induttivo, quindi è capace di dedurre un inseme di regole a partire dall'analisi di un insieme di esempi. Regole che verranno poi utilizzate per lo svolgimento di determinate attività. Questo tipo di processo induttivo si contrappone all'utilizzo "standard" di un calcolatore il quale comporta tipicamente il processo inverso, deduttivo, che passa dalle regole (i programmi) ai risultati. Provenendo queste regole dall'esterno viene preclusa perciò ogni possibilità di apprendimento.

Proseguendo su questa strada possiamo notare come l'apprendimento spesso consista anche nell'effettuare una ricerca in uno spazio di ipotesi e selezionare fra queste quella maggiormente appropriata, rispetto all'insieme di addestramento e ad eventuali conoscenze note a priori o vincoli. Possiamo quindi dire che l'estrazione di informazione utile da un inseme di esempi/dati è il compito centrale dell'apprendimento, che viene definito automatico per distinguerlo da quello dell'uomo che invece è naturale.

1.2 Analisi di un generico problema di apprendimento

Diamo ora una definizione più precisa di come un programma, attraverso l'esperienza, migliori la sua prestazione nel svolgere uno specifico compito:

"si dice che un programma apprende dall'esperienza E rispetto ad una certa classe di compiti T e ad una misura della performance P, se la sua performance nell'eseguire i compiti in T migliora con esperienza E."

È quindi necessario, per poter definire correttamente un problema di apprendimento, individuare tre entità:

- T, classe dei compiti che il programma dovrà svolgere;
- P, misura della performance;
- E, esperienza di addestramento.

Nel corso degli anni sono stati sviluppati diversi algoritmi utili all'apprendimento e questi hanno mostrato la loro efficacia e utilità in un vasto insieme di applicazioni:

- Data mining, in cui vi sono vasti database contenenti informazioni che contengono delle regolarità implicite che possono venire scoperte in automatico. Ad esempio, è possibile estrarre conoscenza medica attraverso l'analisi di cartelle cliniche.
- Software "difficili" da programmare. Questo può succedere quando si ha a che fare con settori poco conosciuti in cui non si hanno le conoscenze necessarie per definire algoritmi efficaci, ovvero non vi sono esperti umani di questo settore specifico. Ad esempio lo studio del DNA. Un'altra possibile applicazione è data da tutta una classe di compiti che gli esseri umani svolgono facilmente ma dei quali non

sono in grado di esprimere la metodologia adottata,come ad esempio il riconoscimento dei volti o il riconoscimento del parlato.

- Domini dove i programmi devono dinamicamente e continuamente adattarsi ai cambiamenti delle condizioni in cui devono operare. In quest'ultimo caso rientrano tutte quelle applicazioni che devono adeguarsi alle preferenze dell'utente. Ne sono un esempio le applicazioni che suggeriscono gusti musicali, pagine internet o prodotti, in base al comportamento passato dell'utilizzatore di tali servizi.

1.3 Definizione di un sistema di apprendimento

Per procedere allo sviluppo di un sistema di apprendimento automatico occorre quindi effettuare precise scelte progettuali.

La prima scelta che bisogna affrontare è la modalità di addestramento con cui il nostro sistema apprenderà. Questa scelta può avere un impatto significativo sul successo o il fallimento dell'addestramento.

Il primo attributo da tenere in considerazione è la tipologia di feedback che l'esperienza di addestramento fornisce in risposta alle scelte effettuate dal sistema. Questo feedback può essere diretto, cioè il sistema è supervisionato in ogni momento da un "maestro" che lo corregge volta per volta, oppure indiretto, quando è guidato da obiettivi e solamente al termine dell'esperienza si potrà essere in grado di valutare l'esito dell'addestramento.

Un secondo attributo è il livello con il quale il sistema di apprendimento controlla la sequenza degli esempi di addestramento. Il sistema può affidarsi ciecamente al "maestro" nella selezione degli esempi oppure mantenere un comportamento pro-attivo e di collaborazione con questo ultimo.

Occorre poi decidere la rappresentazione degli oggetti del dominio sui cui deve essere misurata la performance finale del sistema. In generale un sistema di apprendimento deve eseguire dei compiti in un specifico ambiente

oppure classificare oggetti del mondo reale. In questo secondo caso si devono rappresentare gli oggetti da raggruppare e solitamente vengono utilizzati vettori o grafi. Invece, per rappresentare l'ambiente, si adoperano matrici o rappresentazioni grafiche più complesse quali textures o contorni.

Un passo importante è determinare con esattezza che tipo di conoscenza deve essere appresa e come essa poi dovrà essere usata dal programma. Stiamo parlando della funzione obiettivo. Essa è un'espressione formale della conoscenza appresa e viene adoperata per determinare le prestazioni del programma di apprendimento. Esempi di tale funzione sono polinomi, alberi di decisione, reti neurali e insiemi di regole. È da notare che in casi reali è difficile ottenere un sistema in grado di apprendere perfettamente la funzione obiettivo. In questi casi si sceglie un'approssimazione che abbia la stessa forma prescelta (polinomio, regole etc.) e che approssimi il meglio possibile la funzione reale.

L'ultimo passo, in conseguenza della funzione obiettivo, è la scelta dell'algoritmo. Se la funzione obiettivo selezionata e di natura probabilistica, allora sarà scelto un algoritmo a base statistica, come ad esempio quello utilizzato dall'apprendimento Bayesiano. Se invece la funzione è algebrica (un polinomio), allora si utilizzerà una metodologia algebrica come può essere l'algoritmo di discesa del gradiente. Altrimenti, se la funzione obiettivo è composta da espressioni logiche, verrà utilizzato un algoritmo basato sulla conoscenza, per esempio uno che sfrutti gli alberi di decisione.

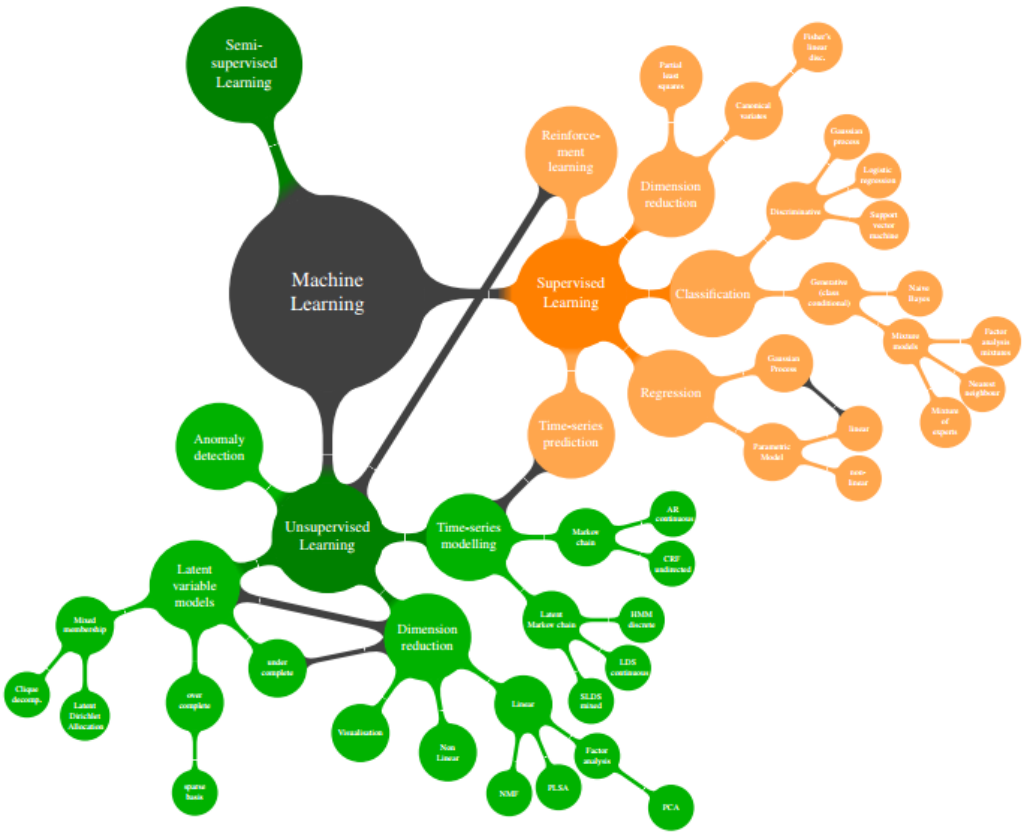


Figura 1.1: Overview su Apprendimento Automatico. Immagine tratta dal libro "Bayesian Reasoning and Machine Learning" [3].

Capitolo 2

Metodi di Apprendimento Automatico

Caratteristica comune ai vari algoritmi di apprendimento automatico è la presenza di dati in input, che possiamo chiamare *Training Pattern* che portano alla generazione di un output (*istanze dell'esperienza*).

Il nostro obiettivo e quindi fare in modo che dopo un certo numero di istanze il nostro sistema si comporti nel modo desiderato.

Il tipo di output emesso dall'algoritmo e la modalità con cui vengono presentati gli esempi di apprendimento invece caratterizzano i vari algoritmi.

Questi vari algoritmi possono quindi essere divisi in tre classi differenti:

- *Apprendimento Supervisionato*
- *Apprendimento Non Supervisionato*
- *Apprendimento per Rinforzo*

2.1 Apprendimento Supervisionato

L'apprendimento supervisionato è una tecnica di apprendimento automatico che presenta in ingresso un vettore di Input (il nostro Training Pattern), un vettore contente gli Output desiderati e una funzione f che ha il compito di associare ad ogni ingresso una determinata uscita. L'obiettivo finale di questa tipologia di algoritmi quindi è quella di estrapolare dagli esempi dati in ingresso, una funzione f' che approssimi il più fedelmente possibile la funzione di I/O f. Con un numero sufficiente di esempi, la funzione estrapolata offrirà una soluzione accettabile anche per nuovi Input che non facevano parte del Training Pattern.

Utilizzando un linguaggio più formale possiamo dire che:

vengono forniti alla macchina in ingresso un set di dati contenuti in un vettore, a ogni input I viene associato l'output O desiderato. Quindi l'obbiettivo dell'algoritmo consiste nell'estrapolare e migliorare, da ogni coppia (I, O), data come esempio, una funzione f' chiamata anche *Ipotesi Induttiva*, funzione che deve avvicinarsi il più possibile alla ipotetica funzione f chiamata *funzione obiettivo*. Indicheremo invece come S il parametro utilizzato come stima di efficienza dell'apprendimento, solitamente rappresentato come differenza tra gli output delle funzioni f e f'.

Se gli esempi dati in ingresso come Training Pattern sono sufficienti sia in qualità che in numero, la funzione estrapolata dall'algoritmo sarà sufficientemente simile alla funzione obbiettivo, permettendo così di approssimare in modo accettabile l'output fornito dall'algoritmo di fronte a un ingresso I non facente parte del Training Pattern iniziale. Quindi il buon funzionamento di tali algoritmi dipende in modo significativo dai dati in ingresso.

Fornendo pochi dati in ingresso, l'algoritmo potrebbe non aver abbastanza esperienza per dare un output corretto, viceversa avere molti dati in ingresso potrebbero renderlo eccessivamente lento, dato che la funzione f', generata dal gran numero di input, potrebbe essere molto complicata. Inoltre l'esperienza dimostra che questa tipologia di algoritmi è molto sensibile al rumore. Anche pochi dati errati potrebbero rendere l'intero sistema non affidabile e condurlo a decisioni errate.

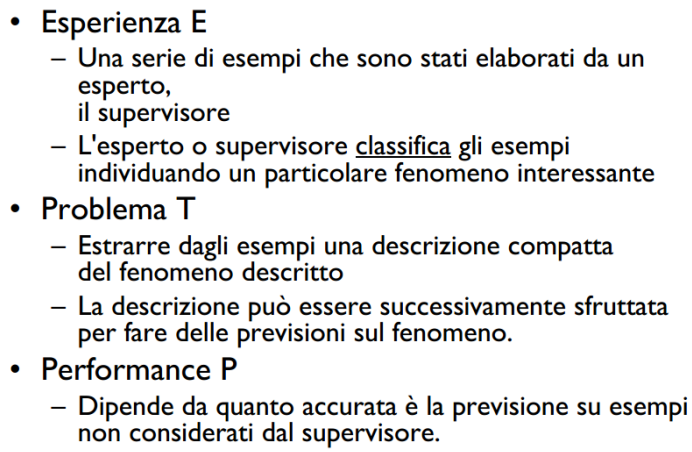


Figura 2.1: Struttura Apprendimento Supervisionato [1].

Questi algoritmi generalmente sfruttano i principi della distribuzione matematica e della funzione di verosimiglianza. Una volta individuata la funzione di distribuzione che lega l'input all'output, si generano dei parametri tali che massimizzino la probabilità di generare la funzione di verosimiglianza appropriata. Anche questi tipi di algoritmo sono utilizzati spesso in operazioni di Pattern Recognition, Speech Recognition e Handwriting Recognition, operazioni in cui, sopratutto negli ultimi due casi, l'algoritmo deve essere guidato a fornire un output accettabile, cosa che non è possibile per gli algoritmi non supervisionati.

2.1.1 Alberi di decisione

Un albero di decisione è un modello gerarchico per l'apprendimento supervisionato.

Tale apprendimento è un metodo per approssimare funzioni a valori discreti, attraverso cui la funzione appresa è rappresentata tramite un albero di decisioni. Gli alberi di decisione possono essere rappresentati anche come un insieme di regole if-then per migliorare la leggibilità all'uomo.

Questi metodi di apprendimento sono tra i più popolari algoritmi di inferenza induttiva e sono stati applicati con successo ad un'ampia gamma di funzioni nei più disparati ambiti (medicina, finanza). Gli alberi di decisione classificano le istanze disponendole lungo tutto l'albero, partendo dalla radice fino ad arrivare ai nodi foglia, i quali provvedono alla classificazione dell'istanza. Ogni nodo nell'albero specifica uno determinato attributo dell'istanza (funzione test), e ogni ramo discendente da quel corrispondente nodo rappresenta un possibile valore del attributo scelto (risultato della funzione test).

Una istanza è classificata a partire dal nodo-radice dell'albero. Dopo aver analizzato l'attributo specificato da questo nodo, ci si sposta giù per il ramo che corrisponde al valore dell'attributo selezionato. Tale processo è ripetuto su ogni sotto-albero la cui radice è il nuovo nodo, altrimenti se si è arrivati ad un foglia si restituisce l'etichetta ad essa associata.

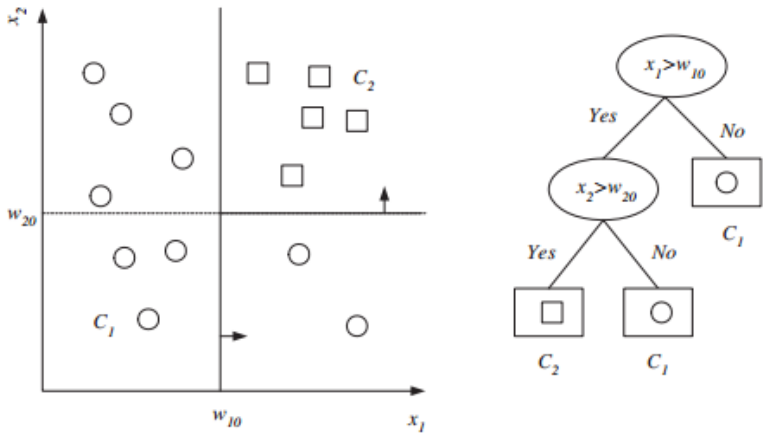


Figura 2.2: Esempio di un dataset e del corrispondente albero di decisione. I nodi ovali sono i nodi di decisione e quelli rettangolari sono i nodi foglia. Il nodo di decisione si divide prima lungo un asse, e le successive divisione sono l'un l'altra ortogonali [2].

2.2 Apprendimento Non Supervisionato

Questi algoritmi ricevono in input un vettore di dati o esempi, lo scopo dell'algoritmo e il relativo apprendimento consistono nell'individuare nel Training Pattern alcune caratteristiche distintive o comuni che permettono di eseguire un'operazione di classificazione sugli esempi di input.

Il processo di apprendimento però non riceve alcun feedback dall'esterno, quindi l'utente non ha modo di prevedere a priori l'output dell'operazione di classificazione. In pratica l'algoritmo realizza delle operazioni di tipo statistico per individuare una struttura ricorrente nei dati di ingresso che, successivamente, può essere riutilizzata per riconoscere e classificare altre istanze di input che non facevano parte del vettore di esempi.

La validità di questi algoritmi dipende dall'utilità delle informazioni che riescono ad estrarre dalle basi di dati. Si dimostrano molto efficienti con elementi di tipo numerico, ma molto meno prestanti con dati non numerici.

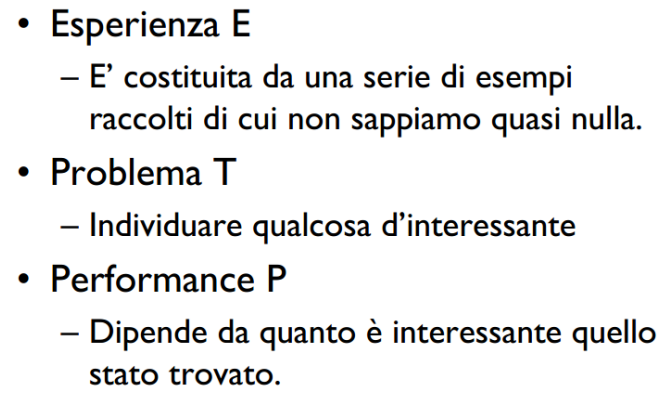


Figura 2.3: Struttura Apprendimento Non Supervisionato [1].

In generale essi lavorano correttamente in presenza di dati contenenti un ordinamento o un raggruppamento netto e chiaramente identificabile.

Questo genere di algoritmi trovano il loro massimo utilizzo nella risoluzione di problemi di Clustering o Data Mining.

$$P(h|D) = \frac{P(D|h)\,P(h)}{P(D)}$$

Figura 2.4: Formula del Teorema di Bayes

2.2.1 Apprendimento Bayesiano

Il ragionamento Bayesiano fornisce un approccio probabilistico alla soluzione del problema dell'apprendimento automatico. Esso è bastato sull'assunzione che tutte le grandezze di nostro interesse sono governate da distribuzioni di probabilità e che quindi una decisione ottimale può essere presa mediante l'analisi di queste probabilità e l'analisi dei dati osservati. Quindi partendo da una situazione in cui si hanno diverse ipotesi tra le quali scegliere, gli algoritmi Bayesiani calcolano le probabilità di ciascune di esse. Questo risultato viene poi utilizzato per selezionare quella più adatta.

La base teorica di tale approccio è il teorema di Bayes, che viene riportato qui di seguito.

Il Teorema di Bayes ci fornisce un metodo per il calcolo della probabilità di un'ipotesi conoscendo la sua probabilità a priori e la probabilità di osservare un certo insieme di dati, avendo a disposizione l'ipotesi ed i dati osservati.

Nella formula del Teorema di Bayes [Fig.2.4] possiamo identificare i seguenti parametri:

- P(h): rappresenta la probabilità iniziale dell'ipotesi h prima di aver esaminato l'insieme di allenamento. È anche detta probabilità a priori.
- P(D): è la probabilità che l'insieme D venga osservato.
- P(D—h): definisce la probabilità di osservare D in un mondo in cui si è verificata l'ipotesi h.

- P(h—D): esprime la probabilità che che si verifichi l'ipotesi h, avendo osservato D. E' chiamata anche probabilità a posteriori, e rappresenta la nostra confidenza che h si possa verificare in seguito all'osservazione di D.

Vi sono alcune caratteristiche comuni a tutti i metodi di apprendimento Bayesiano:

- Ciascun esempio contenuto all'interno del training set contribuisce ad incrementare o decrementare la probabilità che una certa ipotesi sia corretta. Ciò evidenzia quanto questo approccio all'apprendimento sia più flessibile rispetto ad altri algoritmi che invece eliminano completamente un'ipotesi se ritenuta inconsistente.
- La conoscenza a priori può essere combinata con l'analisi dei dati osservati per il calcolo della probabilità finale di una ipotesi.
- La conoscenza a priori viene ricavata fornendo, per ognuna delle ipotesi candidate, una probabilità a priori e una distribuzione di probabilità sui dati osservati.
- I metodi Bayesiani favoriscono ipotesi che effettuano predizioni probabilistiche.
- Si possono classificare nuove istanze combinando le predizioni di diverse ipotesi, pesate con le loro probabilità.

Una prima difficoltà pratica nell'applicazione dei metodi Bayesiani è il significativo costo computazionale che è richiesto per determinare l'ipotesi Bayesiana ottimale nella maggior parte dei casi.

Una seconda difficoltà pratica nell'applicazione dei metodi Bayesiani è che tipicamente richiedono una conoscenza a priori di molte probabilità. Quando tale conoscenza però non è nota, è comunque possibile fare delle stime probabilistiche su di essa basandosi sull'esperienza o su precise assunzioni riguardo la distribuzione delle probabilità.

2.3 Apprendimento per Rinforzo

Inizialmente l'Apprendimento per Rinforzo era considerato come caso particolare dell'apprendimento supervisionato. In realtà esso trova applicazioni in vari contesti nei quali l'apprendimento supervisionato risulta inefficiente ed inefficace. I problemi di interazione con l'ambiente ne sono un buono esempio.

Infatti, questa tecnica di apprendimento, punta a realizzare algoritmi di apprendimento capaci di adattarsi ai mutamenti dell'ambiente.

Tutto ciò è possibile poiché è prevista la ricezione di feedback (*reward signal*) esterni generati dall'ambiente a seconda delle scelte intraprese dall'algoritmo. Una scelta corretta comporterà un premio mentre una scelta scorretta porterà ad una penalizzazione del sistema. Tutto questo per poter raggiungere il miglior risultato ottenibile.

L'Apprendimento per Rinforzo differisce dall'Apprendimento Supervisionato nel modo in cui viene gestito un errore nell'output. Con l'Apprendimento Supervisionato l'informazione di feedback è esattamente ciò che serviva come output, invece il feedback usato con l'Apprendimento con Rinforzo ci dice solamente se l'output corrente è valido o meno.

E' possibile individuare due diverse tipologie di algoritmi ad apprendimento per rinforzo:

- *Algoritmi ad apprendimento continuo*: queste tecniche dispongono di un meccanismo semplice in grado di valutare le scelte dell'algoritmo e quindi "premiare" o "punire" l'algoritmo attraverso un segnale numerico di rinforzo a seconda del comportamento istantaneo del sistema.

  Un esempio di algoritmi ad apprendimento continuo sono i programmi di riconoscimento vocale o i programmi di riconoscimento di caratteri.

- *Algoritmi ad addestramento preventivo*: queste tecniche non dispongono della possibilità di valutare continuamente le azioni dell'algoritmo. In questo caso si applica una prima fase in cui si istruisce l'algo-

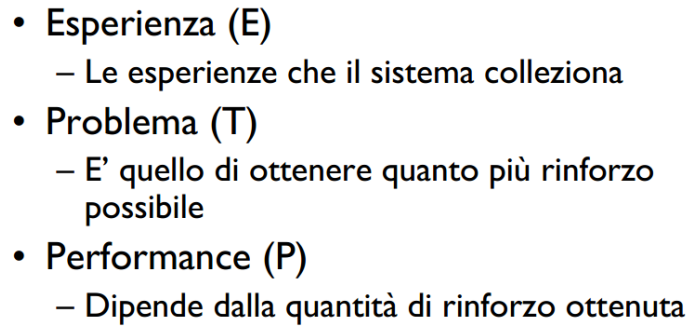


Figura 2.5: Struttura Apprendimento per Rinforzo [1].

ritmo e quando il sistema viene ritenuto affidabile viene cristallizzato e reso quindi immodificabile.

Si noti che le tipologie sopra elencate sono delle scelte implementative piuttosto che delle distinzioni concettuali dell'algoritmo. La scelta di una tipologia rispetto all'altra è a discapito del progettista che fa la sua scelta a seconda dei requisiti che richiede il problema da risolvere.

I principali algoritmi sono le Reti Neurali Artificiali, Algoritmi genetici e Classificatori.

2.3.1 Reti Neurali Artificiali

I metodi di apprendimento basati sulle reti neurali forniscono un solido approccio per l'approssimazione di funzioni a valori reali, discreti e vettoriali.

Per alcune categorie di problemi come ad esempio l'elaborazione dell'informazione proveniente da sensori ambientali, quindi dati complessi e spesso affetti da rumore e disturbi (microfono o telecamera), le reti neurali artificiali sono tra i metodi di apprendimento più efficienti finora conosciuti.

Le reti neurali artificiali (ANN) sono nate per riprodurre attività tipiche del cervello umano come il riconoscimento di forme, il riconoscimento di immagini, la comprensione del linguaggio.

Lo studio delle reti neurali artificiali è stato in parte ispirato dalla osservazione che ha evidenziato come i sistemi di apprendimento biologici siano costituiti da una fitta e complessa rete di neuroni interconnessi tra loro. Allo stesso modo le, reti neurali artificiali, sono realizzate edificando una fitta rete di semplici unità. I neuroni artificiali.

Tipicamente, il neurone artificiale ha molti ingressi ed una sola uscita. Ogni ingresso ha associato un peso, che determina la conducibilità del canale di ingresso. L'attivazione del neurone è una funzione della somma pesata degli ingressi.

Il metodo più usato per addestrare una rete neurale consiste nel presentare in ingresso alla rete un insieme di esempi (training set). La risposta fornita dalla rete per ogni esempio viene confrontata con la risposta desiderata, si valuta la differenza (errore) fra le due e, in base a tale differenza, si aggiustano i pesi Questo processo viene ripetuto sull'intero training set fino a quando le uscite della rete producono un errore al di sotto di una soglia prestabilita.

Riassumendo, le tecniche di apprendimento con ANN sono appropriate per problemi con le seguenti caratteristiche:

- Le istanze sono composte da più coppie attributo-valore.
- L'output della funzione obiettivo è composta da valori reali, discreti o vettoriali.
- Gli esempi di addestramento possono contenere errori. Infatti, come affermato in precedenza, i metodi di apprendimento ANN sono molto resistenti ai rumori.
- Sono accettati lunghi tempi per l'addestramento. A differenza degli alberi di decisione, i tempi di addestramento sono molto lunghi, possono

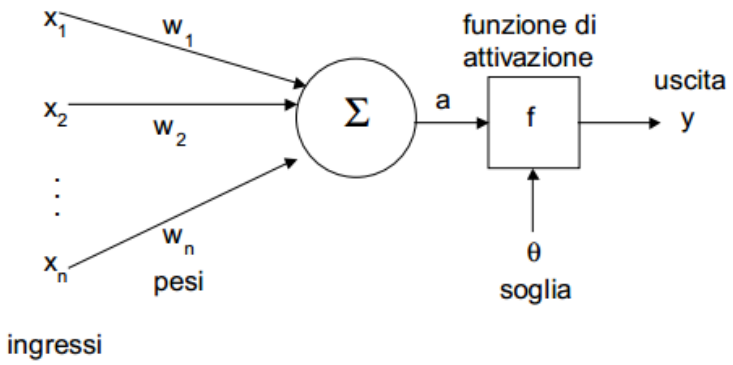


Figura 2.6: Modello di un neurone artificiale. Immagine tratta da "Introduzione alle Reti Neurali" di B. Lazzerini.

durare pochi secondi o anche diverse ore. Questo può dipendere da diversi fattori come il numero di esempi trattati nell'addestramento.

- E' richiesta una valutazione veloce della funzione obiettivo. Pur avendo dei lunghi tempi di addestramento, i tempi di valutazione sono molto veloci.
- Non è importante per gli esseri umani riuscire a comprendere la funzione obiettivo.

Le reti neurali trovano valida applicazione in settori quali predizione, classificazione, riconoscimento e controllo, portando spesso contributi significativi alla soluzione di problemi difficilmente trattabili con metodologie classiche.

Capitolo 3

Apprendimento Automatico nei Giochi

3.1 Giochi

Da centinaia di anni, i giochi sono un fenomeno della cultura umana dove le persone manifestano intelligenza, interazione e competizione (Huizinga, 1955; Bell, 1980). Ma i giochi sono anche un importante paradigma teorico nella logica, Intelligenza Artificiale, informatica, linguistica, biologia e ultimamente sempre di più nelle scienze sociali e in psicologia.

I giochi possono essere classificati secondo queste cinque caratteristiche:

- Il numero di giocatori (uno, due, multi-player).
- Il coinvolgimento o meno della componente casuale (deterministici o non-deterministici).
- La quantità di informazione a disposizione del giocatore (perfetta, se in ogni momento ogni giocatore ha a disposizione tutta l'informazione, imperfetta in caso contrario).
- Il tempo del sistema utilizzato (turn-based o real-time).
- Dominio delle variabili decisionali (discreto o continuo).

Ciò che rende i giochi di particolare interesse è la loro combinazione di semplicità e complessità. Da un lato, hanno regole bene definite, i vari stati sono facili da rappresentare e tutte le possibili azioni sono note, ma allo stesso tempo, permettono di sviluppare ragionamenti complessi ed articolati.

Lo studio di questi argomenti ha una ricca tradizione nella logica e nell'informatica. Qui la cognizione incontra la computazione, i giochi sfidano gli esseri umani con le loro complessità e lo studio delle macchine da parte dell'uomo evidenzia i problemi di fondo di tale complessità.

3.1.1 Giochi e Intelligenza Artificiale

I giochi costituiscono per l'Intelligenza Artificiale un campo di ricerca privilegiato, in quanto forniscono comodi modelli di problemi reali. I giochi, infatti, pur presentando problemi di complessità paragonabile a quelli reali, hanno regole ben definite e formalizzabili. Inoltre per ogni gioco esistono esperti in grado di giudicare la bontà dei risultati elaborati da una macchina. Risulta quindi comodo lavorare e sperimentare prima in un ambiente ben definito,come può essere il dominio dei giochi, per poi generalizzare e adattare i risultati ottenuti ad un ambiente più variabile come è quello dei problemi reali.

La ricerca dell'Intelligenza Artificiale nel dominio dei giochi ottenne un notevole sviluppo nel 1944, quando Von Neumann ripubblicò il suo articolo riguardante l'algoritmo Minimax (1928) insieme con Morgenstern nel libro "Teoria dei giochi e comportamento economico". Queste idee sono state portate avanti successivamente da Shannon (1950), Turing (1953) e Simon (1958), che scrisse i primi articoli su come i computer potessero giocare a scacchi in maniera intelligente.

Tra le varie tipologie di giochi esistenti quelli che hanno avuto maggiore successo, a livello di ricerca scientifica, sono i giochi ad informazione perfetta, deterministici, a due giocatori. Di questa categoria fanno parte gli scacchi, la dama e il Go. L'attenzione dei ricercatori si è però prevalentemente

soffermata sul gioco degli scacchi. Negli anni passati, grazie all'impegno e alla dedizione di diversi scienziati, è stato possibile realizzare un giocatore artificiale capace di giocare ad un livello da campione del mondo.

Il successo più importante ottenuto in questo gioco si ottenne nel 1997 quando Deep Blue sconfisse il campione mondiale di scacchi Garry Kasparov. Oggi uomini e macchine competono agli stessi livelli, ma il gioco è considerato ancora troppo complesso per essere compreso interamente dall'essere umano e troppo vasto per essere computato dal computer più potente in circolazione. Tuttavia, i ricercatori nel campo dell'intelligenza artificiale continuano a cercare di inventare nuovi modi per affrontare tale problema al fine di testare i loro algoritmi intelligenti.

Per il gioco del Go, antico gioco orientale, i programmi non sono ancora ad un livello da "maestro". Nonostante ciò, nel 2009, è stato raggiunto un importante risultato. Il programma MoGo è riuscito a sconfiggere un giocatore di livello 9 (livello massimo) con solo sette mosse di vantaggio.

Per quanto riguarda invece il gioco della dama, che ha un grado di complessità decisamente inferiore rispetto ai precedenti, nel 2007 Schaffer è riuscito nell'impresa di risolverlo. Cioè è riuscito a dimostrare che, se entrambi i giocatori giocano in maniera ottimale, la partita finisce in un pareggio.

Il Backgammon è anche esso un gioco a due giocatori ad informazione perfetta, ma a differenza dei precedenti è non deterministico, infatti contiene una componente casuale data dal lancio di due dadi per determinare lo spostamento delle pedine. Anche esso è stato oggetto di studi e di ricerca e, dagli anni Novanta, i computer sono più forti degli esseri umani (Tesauro, 1994). Uno dei programmi più famosi è TD-Gammon, realizzato da Gerald Tesauro, il quale impiega reti neurali e temporal-difference learning per raggiungere un alto livello di gioco [17].

3.2 Tris

Come primo caso di studio di apprendimento automatico nei giochi ho seguito il lavoro svolto da Henk Mannen [9] nel quale viene utilizzata una rete neurale per addestrare una funzione di valutazione per un giocatore di Tris.

Come caso di studio iniziale il gioco del Tris è l'ideale, infatti, rispetto gli altri giochi di strategia, presenta poche semplici regole. Inoltre è relativamente facile da programmare e, potendo una partita durare al massimo il tempo di nove mosse, l'addestramento di una funzione di valutazione risulta estremamente rapido.

Il Tris è un gioco a informazione perfetta a due giocatori, dove ad ognuno di essi viene assegnato un simbolo con cui giocare. I simboli solitamente utilizzati sono la *croce* ed il *cerchio*. La partita viene iniziata dal giocatore che adopera la *croce*.

La griglia di gioco ha una struttura 3x3 e presenta 9 celle inizialmente vuote. A turno, i giocatori scelgono una cella vuota e vi disegnano il proprio simbolo.

Vince il giocatore che riesce a disporre tre dei propri simboli in linea retta orizzontale, verticale o diagonale. Se la griglia di gioco viene riempita senza che nessuno dei giocatori sia riuscito a completare una linea retta di tre simboli, il gioco finisce in parità.

3.2.1 Architettura

L'obbiettivo di questo caso di studio è quello di allenare una rete neurale in grado poi di svolgere il compito di funzione di valutazione per le posizioni di gioco.

Scelta una possibile mossa, la posizione ad essa associata viene posta in ingresso alla rete da allenare. Il risultato prodotto in uscita fornisce un valore della posizione valutata. Più questo valore è alto, più la posizione è da considerarsi soddisfacente.

La struttura della rete adoperata nell'esperienza presenta tre livelli differenti: un livello di input, uno di output e uno nascosto.

Il livello di input è costituito a sua volta da 10 nodi di ingresso, dei quali uno identifica il giocatore che compie la mossa, mentre i restanti 9 rappresentano le nove celle della griglia di gioco. Queste 9 celle possono assumere 3 valori: 0 se la cella è vuota, 1 se è presente una *croce* , -1 se invece la cella è occupata da un *cerchio*.

Il livello di output è rappresentato da un unico nodo. Il livello nascosto, invece, presenta tre configurazioni diverse: a 40, a 60 e 80 nodi.

Per quanto riguarda le interconnessioni tra i vari livelli ogni nodo nel livello di input ha una connessione unica con ogni nodo presente nel livello nascosto e ciascuno di questi presenta una connessione con il singolo nodo di output.

3.2.2 Esperimento

Durante l'esperimento è stato modellato un giocatore *win-block-random* che si comporta come descritto in seguito:

Prima di tutto, controlla se nella griglia è presente una cella vuota che potrebbe portarlo subito alla vittoria. Se essa è disponibile, la occupa.

Se questa prima situazione non è riscontrata, il giocatore a questo punto controlla se è presenta una cella vuota che potrebbe far vincere la partita all'avversario, in caso affermativo la occupa, bloccando quindi le aspirazioni di vittoria del suo rivale (Boyan, 1992 Wiering 1995).

Altrimenti, se nessuna delle precedenti circostanze si è verificata, il giocatore procede scegliendo una cella in maniera casuale.

Il giocatore, che utilizza la rete neurale come sua funzione di valutazione, sceglie la sua prossima mossa in base ai risultati che otterrà in uscita dalla rete , che come abbiamo detto prima, ha il compito di valutare le mosse possibili. La mossa da lui selezionata sarà quella che ha ottenuto dalla rete il valore più alto.

learning rate	0.01
λ	$0.9 \rightarrow 0.0^2$
input nodes	10
hidden nodes	40/60/80
output nodes	1
bias hidden nodes	1
bias output node	0.25

Figura 3.1: Parametri delle differenti reti.

L'allenamento inizia una volta che la prima partita è terminata.

Si inizia con il valutare l'ultima posizione raggiunta nella griglia, che può avere valore 1, se a vincere è stato "croce", -1 se a vincere è stato cerchio, 0 se la partita è terminata in parità.

Questo input pattern è fornito alla rete assieme al valore di output desiderato (Il valore di output delle altre posizioni è invece calcolato con il metodo offline TD-learning).

La rete durante la prima partita, per la funzione di valutazione, utilizza pesi dal valore casuale, per tale motivo i risultati così ottenuti sono molto imprecisi. Dopo però un certo numero di partite i pesi iniziano a regolarsi e i risultati cominciano a essere precisi. Questa precisione continua a migliorare aumentando il numero di partite giocate.

3.2.3 Testing

Nella figura 3.1 sono raffigurati i parametri utilizzati nei vari esperimenti.

Nel primo esperimento è stata adoperata una rete con un livello nascosto di *40 nodi*. Dopo 16.000 partite la rete ha iniziato ad ottenere più vittorie che pareggi. Dopo 40.000 partite la percentuale di vittoria si aggirava al 53% e quella di sconfitta al 4%.

Nel secondo esperimento il livello nascosto presentava *60 nodi*, e le vittorie hanno superato i pareggi dopo 6.000 partite giocate. A 40.000 partite,

la rete aveva una percentuale di vittoria pari al 58% e di sconfitta del 3.8%.

Nell'ultimo esperimento è utilizzata una rete con *80 nodi* nel livello nascosto. Come nell'esperimento precedente il numero di vittorie ha superato il numero di sconfitte dopo le 6.000 partite giocate. Alla soglia delle 40.000 partite la percentuale di vittorie era pari a 62% mentre la percentuale di sconfitte era del 3.5%.

I vari risultati evidenziano come tutte e tre le configurazioni siano valide. Si può notare che dopo 40.000 partite la percentuale di sconfitte è pressoché identica per tutte e tre le configurazioni, mentre la percentuale di vittoria aumenta con il numero dei nodi utilizzati a discapito del numero di pareggi.

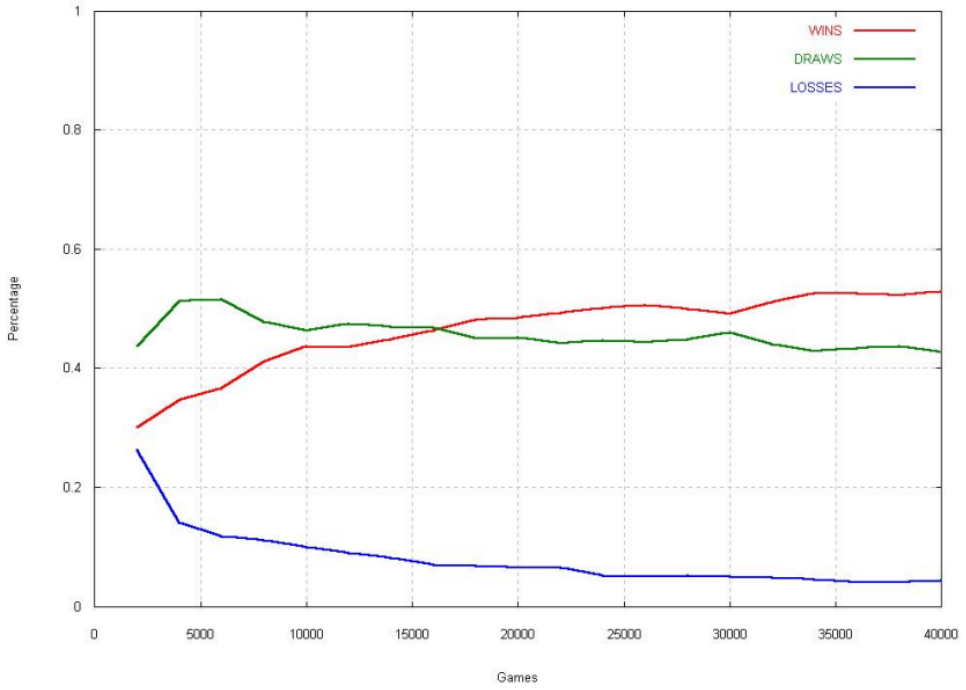


Figura 3.2: Livello nascosto che utilizza 40 nodi.

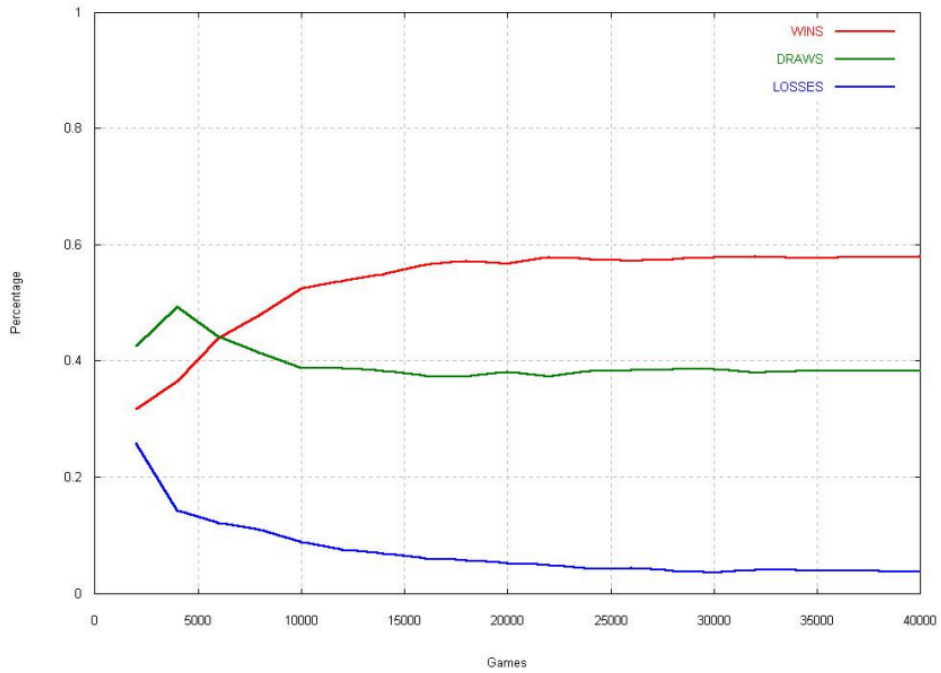


Figura 3.3: Livello nascosto che utilizza 60 nodi.

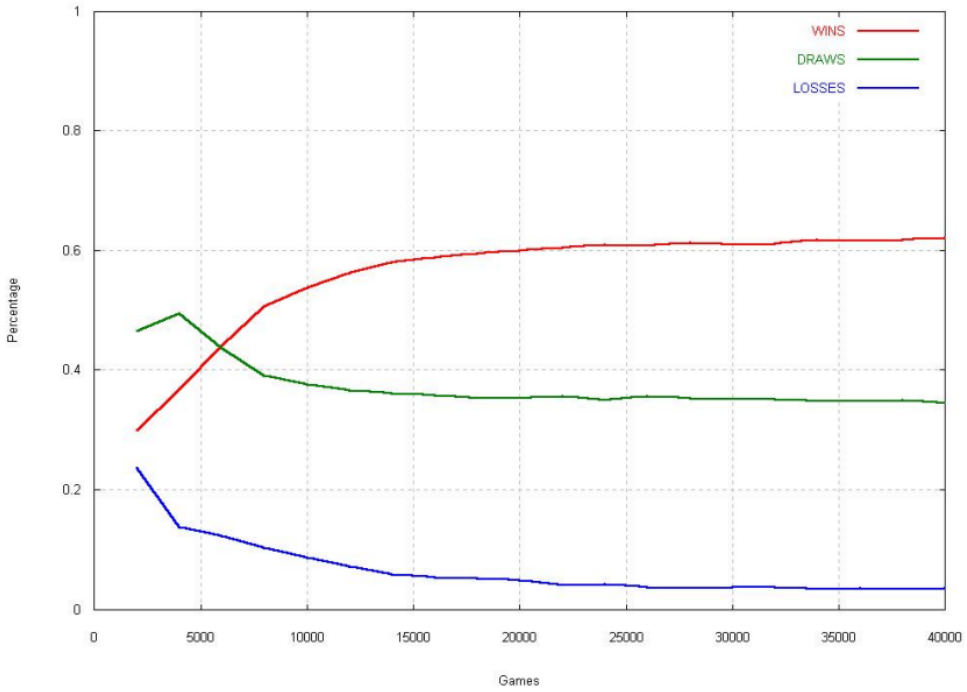


Figura 3.4: Livello nascosto che utilizza 80 nodi.

3.3 Apprendimento attraverso le annotazioni di gioco

Nel dominio dei giochi, la maggior parte della ricerca nell'area dell'addestramento della funzione di valutazione è concentrata sull'utilizzo di algoritmi di apprendimento per rinforzo. Nonostante ciò, in svariati giochi, come gli scacchi, sono numerosi i documenti che contengono feedback di giocatori esperti raccolti sotto forma di annotazioni di gioco.

Questi feedback, solitamente, vengono rappresentati mediante informazioni qualitative poiché gli annotatori umani trovano difficile stabilire precisi valori di utilità per gli stati di gioco. Una forma di conoscenza qualitativa di particolare interesse sono le cosiddette coppie di comparazione o di preferenza. Gli esseri umani, infatti, spesso non sono in grado di determinare un valore preciso di un'opzione ma, tipicamente, sono capaci di confrontare la qualità di due opzioni e quindi poi di selezionarne una rispetto l'altra. L'apprendimento preferenziale (Preference Learning) studia come questo tipo di informazioni qualitative può essere utilizzato per risolvere problemi di apprendimento automatico.

Per affrontare questo argomento ho seguito il lavoro svolto da Christian Wirth and Johannes Furnkranz e raccolto nell'articolo "On Learning from Game Annotations" [12].

L'obiettivo del loro studio è di cercare un modo per poter utilizzare quella ricca fonte di informazioni, ancora poco utilizzata nell'ambito dell'apprendimento automatico, costituita dalle annotazioni di gioco per gli scacchi. A tal fine, si mostra, nello specifico, come queste annotazioni di gioco possono essere tradotte in istruzioni preferenziali rispetto le mosse e le posizioni di gioco. Queste che, a loro volta, possono essere adoperate per l'apprendimento di una funzione di utilità che rispetti i suddetti vincoli preferenziali. Verrà poi valutata la funzione risultante creando più euristiche basate su sottoinsiemi diversi di dati d'allenamento (training data) e confrontando tra loro le differenti configurazioni nello scenario di un torneo.

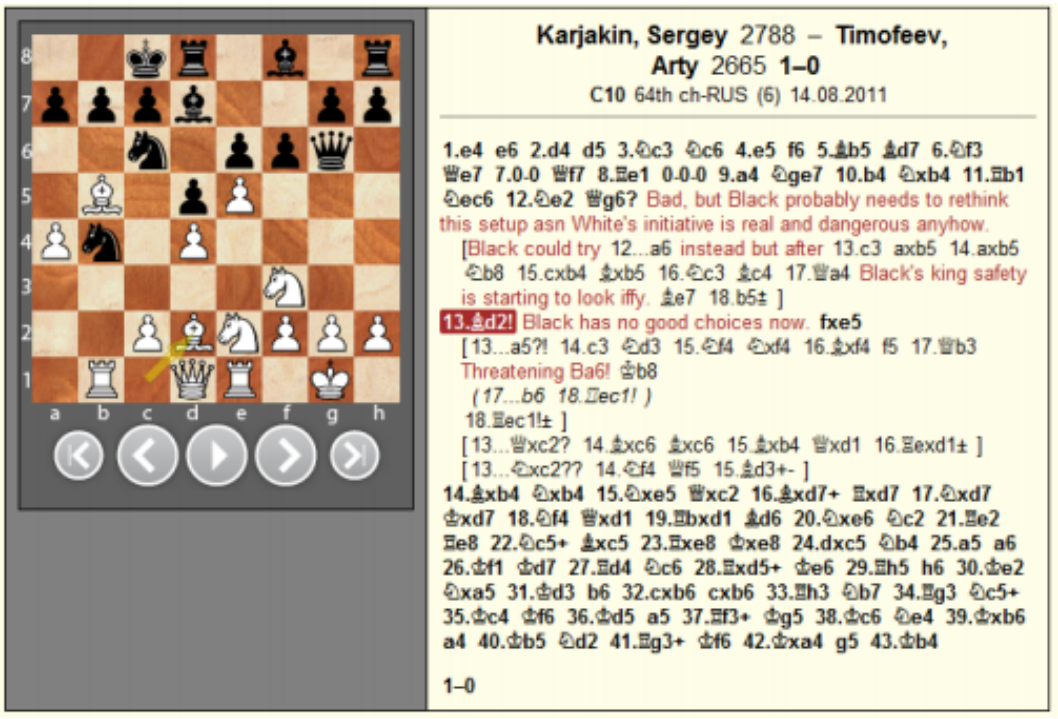
3.3.1 Annotazioni di gioco negli scacchi

Gli scacchi sono un gioco di grande interesse che ha generato una grande quantità di letteratura che analizza le sue dinamiche di gioco.

Nel gioco degli scacchi, i feedback qualitativi prodotti dall'uomo sono ampiamente disponibili sotto forma di annotazioni di gioco. Queste annotazioni sono state prodotte nei secoli da alcuni dei più forti giocatori, e sono ora ampiamente disponibili nei libri e nei database on-line specifici di tale gioco.

Le annotazioni riflettono l'analisi di una particolare partita giocata da un (tipicamente) forte giocatore di scacchi, sono prodotte senza vincoli di tempo e con la possibilità di ricorrere a tutti i mezzi ritenuti necessari per migliorare la valutazione, compresa la consultazione di colleghi, libri o computer. Per questi motivi queste annotazioni sono considerate di grande qualità e valore, e sono utilizzate regolarmente da ogni tipo di giocatore per fini di studio.

I giocatori di scacchi annotano le partite con una combinazione di sequenze di gioco e/o commenti testuali ad una certa posizione o mossa. Il Portable Game Notation (PGN) è uno dei formati utilizzati per salvare le partite e le annotazioni. Le partite in PGN sono rappresentate come una sequenza di mosse. Le mosse stesse sono salvate in una notazione algebrica standard in cui viene specificato il numero della mossa, il pezzo mosso e le caselle di partenza e di destinazione. Ad esempio, nella figura 3.5 la posizione della re nero è identificato da Kg6. Il NAG (Numerica Annotation Glyps) invece è uno standard utilizzato per codificare, con una set di simboli, i tipici eventi che possono accadere in una partita di scacchi. Questi possono far riferimento a proprietà delle posizioni (struttura dei pedoni, possibilità di attacco etc.) o a proprietà esterne al gioco (es. vincoli di tempo).



Karjakin, Sergey 2788 – **Timofeev, Arty** 2665 **1–0**
C10 64th ch-RUS (6) 14.08.2011

1.e4 e6 2.d4 d5 3.♘c3 ♘c6 4.e5 f6 5.♗b5 ♗d7 6.♘f3 ♕e7 7.0-0 ♕f7 8.♖e1 0-0-0 9.a4 ♘ge7 10.b4 ♘xb4 11.♖b1 ♘ec6 12.♘e2 ♕g6? Bad, but Black probably needs to rethink this setup asn White's initiative is real and dangerous anyhow.
[Black could try 12...a6 instead but after 13.c3 axb5 14.axb5 ♘b8 15.cxb4 ♗xb5 16.♘c3 ♗c4 17.♕a4 Black's king safety is starting to look iffy. ♗e7 18.b5±]
13.♗d2! Black has no good choices now. **fxe5**
[13...a5?! 14.c3 ♘d3 15.♘f4 ♘xf4 16.♗xf4 f5 17.♕b3 Threatening Ba6! ♔b8
(17...b6 18.♖ec1!)
18.♖ec1!±]
[13...♕xc2? 14.♗xc6 ♗xc6 15.♗xb4 ♕xd1 16.♖exd1±]
[13...♘xc2?? 14.♘f4 ♕f5 15.♗d3+-]
14.♗xb4 ♘xb4 15.♘xe5 ♕xc2 16.♗xd7+ ♖xd7 17.♘xd7 ♔xd7 18.♘f4 ♕xd1 19.♖bxd1 ♗d6 20.♘xe6 ♘c2 21.♖e2 ♖e8 22.♘c5+ ♗xc5 23.♖xe8 ♔xe8 24.dxc5 ♘b4 25.a5 a6 26.♔f1 ♔d7 27.♖d4 ♘c6 28.♖xd5+ ♔e6 29.♖h5 h6 30.♔e2 ♘xa5 31.♔d3 b6 32.cxb6 cxb6 33.♖h3 ♘b7 34.♖g3 ♘c5+ 35.♔c4 ♔f6 36.♔d5 a5 37.♖f3+ ♔g5 38.♔c6 ♘e4 39.♔xb6 a4 40.♔b5 ♘d2 41.♖g3+ ♔f6 42.♔xa4 g5 43.♔b4

1–0

Figura 3.5: Una partita di scacchi annotata tratta da http://chessbase.com/

Elencherò ora i simboli funzionali per la valutazione della qualità della mossa e della posizione:

Valutazione della mossa:

- ??, errore grave.
- ?, errore.
- !?, mossa interessante, forse non la migliore.
- ?!, mossa dubbia, ma sicuramente non errata.
- !, buona mossa.

- !!, ottima mossa.

Un possibile ordinamento parziale in base ai valori dei sopra citati simboli può essere : !! ; ! ; !? ; ?! ; ? ; ??

Valutazione della posizione:

- +-, il bianco ha un vantaggio decisivo, è prossimo alla vittoria.
- +/-, il bianco ha un vantaggio moderato.
- +/=, il bianco ha un lieve vantaggio.
- =, stesse possibilità per ambo le parti.
- =/+, il nero ha un lieve vantaggio.
- -/+, il nero ha un vantaggio moderato.
- -+, il nero ha un vantaggio decisivo,è prossimo alla vittoria.
- ???, la posizione non è chiara.

Un possibile ordinamento parziale, considerato dal punto di vista di un giocatore bianco, può essere il seguente +- ; +/- ; +/= ; =/+ ; -/+ ; -+

Si noti che, anche se c'è una certa correlazione tra le annotazioni della posizione e della mossa (buone mosse tendono a portare a posizioni migliori e cattivi mosse tendono a condurre a posizioni peggiori), queste non sono intercambiabili. Un'ottima mossa può essere l'unica mossa che salva il giocatore dalla rovina imminente, ma non deve necessariamente portarlo in una buona posizione. Al contrario, una mossa sbagliata può essere una mossa che perde occasione di sopraffare subito l'avversario, ma può portare ad una posizione comunque buona per il giocatore.

In aggiunta all'annotazione dei giochi con i simboli NAG, gli annotatori possono aggiungere commenti testuali, che possono essere di complemento alle mosse effettuate, o proporre variazioni (come ad esempio sequenze di mosse alternative che avrebbero potuto portare il giocatore ad una posizione migliore).

La figura 3.5 mostra un esempio di partita di scacchi annotata. La parte sinistra dell'immagine mostra la scacchiera dopo la 13esima mossa del bianco. Qui il nero è in difficoltà poiché la sua 12esima mossa è stata un errore (12.Kg6?). Dopo la buona risposta del bianco (12.Bd2!) il nero ha diverse possibili mosse a sua disposizione come e possibile vedere nell'immagine bisogna in qualche modo decidere quale mossa effettuare.

E' importante notare che i feedback sono tutti di natura qualitativa.Non è infatti possibile determinare la percentuale di vittoria data una posizione con valutazione +/- ma, grazie all'ordinamento parziale considerato precedentemente, possiamo affermare che le posizioni con valutazione +/- sono preferibili a quelle con valutazione +/=. Alla stesso modo possiamo, quindi, definire un ordine di preferenza tra tutte le mosse possibili che il nero può effettuare e scegliere quella a lui più profittevole.

Per identificare le posizioni, invece, utilizziamo lo standard FEN (Forsyth-Edwards Notation), che rappresenta la scacchiera in una maniera testuale e serializzata catturando tutti i dati e le informazioni utili per identificare univocamente uno stato di gioco.

3.3.2 Addestramento di una funzione di valutazione attraverso partite annotate

L'approccio utilizzato per apprendere da annotazioni qualitative dei giochi è basato sull'idea di trasformare i simboli di notazione in istruzioni preferenziali tra le coppie di posizioni. Ciascuna preferenza sarà poi vista come vincolo in una funzione di utilità per le posizioni degli scacchi, che può essere appresa mediante qualunque algoritmo di apprendimento automatico a noi noto, come ad esempio il support-vector machine.

Per andare avanti con il lavoro è necessario introdurre il concetto di Preference Learning.

Il Preference learning o Apprendimento preferenziale è una nuova area che si occupa di sviluppare modelli preferenziali a partire da dati empirici. Stabilisce un legame tra l'Apprendimento Automatico e campi di ricerca come " preference modeling" e il "decision making".

Si possono identificare due tipi di problemi affrontabili con il Preference learning: Label ranking, in cui bisogna imparare quali delle opzioni presenti nel set, identificate da etichette, sono da preferire in un determinato contesto. Object ranking, in cui le preferenze sono definite su un set di oggetti caratterizzati da un insieme di funzionalità. Qui il compito è quello di addestrare una funzione di utilità f(.) che sarà poi capace di ordinare gli oggetti in sottoinsiemi secondo un set di preferenze.

L'obiettivo intermedio da raggiungere, come detto in precedenza, è quello di generare preferenze a partire dai dati in nostro possesso (annotazioni di gioco), per questo motivo si definiscono, in primis, alcune preferenze a partire dalle annotazioni posizionali e successivamente altre inerenti alle preferenze di mossa:

- Generazione di preferenze a partire da Annotazioni posizionali.

  I dati di addestramento che sono necessari per allenare un algoritmo di object ranking possono essere generati dal tipo di annotazioni discusse in precedenza. In primo luogo, viene analizzata ogni partita contenuta nel database G, rivedendo tutte le mosse in modo tale di essere poi in grado di generare tutte le posizioni che si verificano. Per ogni mossa m a cui è associato un insieme di simboli di annotazione posizionale Ap(m) associamo i simboli di annotazione con la posizione p risultante dopo aver giocato ogni mossa.

  La figura 3.6 mostra l'algoritmo per generare preferenze di stato (Algoritmo 1). Il primo ciclo raccoglie per ogni partita una lista di tutte le posizioni che hanno annotazioni posizionali. Nel secondo ciclo, invece, tutte le posizioni incontrate sono confrontate.

Require: database of games $\mathcal{G}$, initial position $\boldsymbol{p}_0$

$prefs \leftarrow \emptyset$,
for all $\mathbf{g} \in \mathcal{G}$ **do**
  $pairs \leftarrow \emptyset$,
  $seen \leftarrow \emptyset$
  $\boldsymbol{p} \leftarrow \boldsymbol{p}_0$
  for all $\mathbf{m} \in \mathbf{g}$ **do**
    $\boldsymbol{p} \leftarrow \textsc{Move}(\boldsymbol{p}, \mathbf{m})$
    if $\exists \mathbf{a} \in \mathcal{A}_P(\mathbf{m})$ **then**
      $pairs \leftarrow pairs \cup \{(\boldsymbol{p}, \mathbf{a})\}$
      $seen \leftarrow seen \cup \{\boldsymbol{p}\}$
    end if
  end for
  for all $\boldsymbol{p}' \in seen$ **do**
    for all $\boldsymbol{p}'' \in seen$ **do**
      for all $(\boldsymbol{p}', \mathbf{a}'), (\boldsymbol{p}'', \mathbf{a}'') \in pairs$ **do**
        if $\mathbf{a}' \sqsupset \mathbf{a}''$ **then**
          $prefs \leftarrow prefs \cup \{\boldsymbol{p}' \succ \boldsymbol{p}''\})\}$
        end if
      end for
    end for

Figura 3.6: Algoritmo 1, algoritmo per generare preferenze di posizione

Per ogni coppia di posizioni (p',p"), dove p' ha ricevuto un valore di valutazione maggiore a p", viene generata una corrispondente preferenza posizionale. Questo significa che a', l'annotazione associata a p', porta il "bianco" a una posizione più vantaggiosa rispetto a quella fornita da a", annotazione di p" (ogni valutazione posizionale è valutata secondo il punto di vista del giocatore "bianco").

Come si può vedere, confrontiamo solo preferenze di posizione all'interno di una stessa partita e non su più partite. Uno dei motivi per cui è stato scelto di agire in questo modo è che un confronto di tutte le coppie di posizione annotate in tutto il database sarebbe computazionalmente impossibile.

- Generazione di preferenze da annotazioni delle mosse.

  Il problema della scelta della prossima mossa da effettuare in una partita può essere visto come un problema di label ranking, dove le posizioni sono il contesto e le mosse sono le etichette che devono essere classificate. Tuttavia, poiché ogni mossa corrisponde ad un'unica posizione successiva, le preferenze di mossa possono anche essere interpretate come preferenze di posizione tra le corrispondenti posizioni successive. E' stato scelto questo approccio perché permette di trattare le preferenze di mossa e preferenze di posizione in maniera uniforme.

  La Figura 3.7 mostra l'algoritmo che genera preferenze posizionali da annotazioni delle mosse (Algoritmo 2). Prima, ogni partita in formato PGN g, contenuta in G, è analizzata e tutte le posizioni sono esplorate. Dopo, poiché è possibile confrontare solo posizioni successive di una stessa posizione originale p, vengono salvate le triplette posizione/mossa/annotazione(p,m,a). Dopo aver raccolto questi dati per ogni partita, tutte le triplette contenenti la stessa posizione FEN vengono confrontate. I simboli NAG per tutte le mosse associate a questa posizione vengono confrontati a loro volta, e le corrispondenti preferenze tra le posizioni derivanti da queste mosse vengono aggiunte al set di preferenze. Anche qui c'è da tenere in considerazione che tutte le valutazioni sono fatte dal punto di vista del giocatore "bianco".

In molti casi, oltre ad un'annotazione NAG, è disponibile anche una sequenza di mosse suggerite che dovrebbero seguire successivamente. Questo significa, solitamente, che l'annotazione non punta a suggerire la posizione raggiunta dopo la prima mossa, ma quella che si raggiunge alla fine della sequenza. Per tale motivo, in queste circostanze, è stata presa in considerazione la posizione alla fine della sequenza. Negli Algoritmi 1 e 2 ciò è implementato nella routine MOVE.

Require: list of games $\mathcal{G}$, initial position $\boldsymbol{p}_0$

$prefs \leftarrow \emptyset$,
for all $\mathbf{g} \in \mathcal{G}$ **do**
  $triples \leftarrow \emptyset$,
  $seen \leftarrow \emptyset$
  $\boldsymbol{p} \leftarrow \boldsymbol{p}_0$
  for all $\mathbf{m} \in \mathbf{g}$ **do**
    if $\exists \mathbf{a} \in \mathcal{A}_M(\mathbf{m})$ **then**
      $triples \leftarrow triples \cup \{(\boldsymbol{p}, \mathbf{m}, \mathbf{a})\}$
      $seen \leftarrow seen \cup \{\boldsymbol{p}\}$
    end if
    $\boldsymbol{p} \leftarrow \textsc{Move}(\boldsymbol{p}, \mathbf{m})$
  end for
  for all $\boldsymbol{p} \in seen$ **do**
    for all $(\mathbf{m}_1, \mathbf{m}_2)$ s.t. $(\boldsymbol{p}, \mathbf{m}_1, \mathbf{a}_1), (\boldsymbol{p}, \mathbf{m}_2, \mathbf{a}_2) \in triples$ **do**
      if $\mathbf{a}_1 \sqsupset \mathbf{a}_2$ **then**
        $\boldsymbol{p}_1 \leftarrow \textsc{Move}(\boldsymbol{p}, \mathbf{m}_1)$,
        $\boldsymbol{p}_2 \leftarrow \textsc{Move}(\boldsymbol{p}, \mathbf{m}_2)$,
        if $\textsc{WhiteMove}(\mathbf{m})$ **then**
          $prefs \leftarrow prefs \cup \{(\boldsymbol{p}_1 \succ \boldsymbol{p}_2)\}$
        else if $\textsc{BlackMove}(\mathbf{m})$ **then**
          $prefs \leftarrow prefs \cup \{\boldsymbol{p}_2 \succ \boldsymbol{p}_1)\}$
        end if
      end if
    end for
  end for
end for
return *prefs*

Figura 3.7: Algoritmo 2, algoritmo per generare preferenze di mossa

3.3.3 Setup degli esperimenti

Per verificare l'utilità dei dati preferenziali raccolti, sono state addestrate alcune funzioni di valutazioni per gli scacchi basate sulle preferenze generate dalle annotazioni delle partite, e sono state impiegate nel performante motore scacchistico, open source, CUCKOO. La qualità delle varie preferenze è stata poi analizzata comparando la forza di gioco del nostro motore scacchistico modificato (ri-ponderato).

Mega Database 2012 è il database di scacchi utilizzato per l'esperimento ed è messo a disposizione da ChessBase. Questo è il più grande database in circolazione che contiene partite di scacchi annottate in maniera professionale. Queste annotazioni sono per lo più, ma non esclusivamente, realizzate da Gran Maestri di scacchi. Negli oltre 5 milioni di partite contenute nel database, sono state riscontrate 86.767 partite annotate, da cui si è calcolato, utilizzando gli algoritmo citati prima, 5.050.000 preferenze di posizione e 190.000 preferenze di mossa. A causa del grande numero di preferenze di posizione, sono state inizialmente considerate solo le preferenze di mossa, e successivamente si è indagato sull'utilità di aggiungere preferenze di posizione.

CUCKOO è il motore scacchistico utilizzato per effettuare i vari esperimenti. E' stato scelto questo per via della sua combinazione di elevata forza di gioco e buona modificabilità. La maggior parte dei motori scacchistici utilizza una funzione di valutazione della posizione euristica mentre per la ricerca della migliore posizione raggiungibile vengono utilizzati per lo più algoritmi di ricerca Alpha-Beta come NegaScout. La funzione di valutazione di CUCKOO può essere inquadrata su diversi livelli di astrazione.

Nella figura 3.8 è raffigurata la Tabella 1 che mostra le sue 16 caratteristiche di primo livello. Le caratteristiche non sono altro che proprietà elementari della posizione raggiunta, quali l'utilità dei vari pezzi nella loro corrente casella. Queste caratteristiche sono versioni già aggregate di funzioni di valutazione più complesse, che non sono direttamente disponibili, ma nascoste all'interno del codice del valutatore. E' stata anche creata una funzione di valutazione complessa sulla base di questo codice nascosto che utilizza le 650 caratteristiche mostrate nella Tabella 2 di della Figura 3.9.

Feature Type	# Features	Description
material difference	1	Difference in the sum of all piece values per player.
piece square	6	Position dependent piece values by static piece/square tables. A single value for every piece type.
pawn bonus	1	Bonus for pawns that have passed the enemy pawn line , while also considering its distance to the enemy king.
trade bonus	2	Bonus for following the "when ahead trade pieces, when behind trade pawns" rules.
castle bonus	1	Evaluates the castling possibility.
rook bonus	1	Bonus for rooks on (half-) open files.
bishops scores	2	Evaluating the bishops position by attack possibilities, if trapped and relative positioning.
threat bonus	1	Difference in the sum of all piece values under attack.
king safety	1	Evaluates the kings position relative to the rooks.

Figura 3.8: Tabella 1. Caratteristiche aggregate usate nella funzione di valutazione lineare del motore scacchistico CUCKOO.

Per concludere il discorso sul motore scacchistico possiamo aggiungere che è stata utilizzata una configurazione single-thread e che tutti gli esperimenti sono stati eseguiti in sistemi con due core o più, in modo tale da garantire l'indipendenza della potenza di calcolo a disposizione per ogni giocatore.

Per implementare il support-vector machine (SVM) è stato inizialmente utilizzato *SVMRank* data la sua implementazione particolarmente veloce ed adatta a problemi di classificazione. Successivamente, a causa della sua scarsa stabilità negli esperimenti preliminari, si è passati a *LIBLINEAR*.

Le*funzioni di valutazione* apprese sono state valutate in diversi tornei round-robin, dove ogni funzione addestrata gioca più partite contro le altre funzioni. Salvo indicazione contraria, ogni torneo è stato giocato utilizzando un controllo di tempo di due secondi per mossa.

Tutti i risultati sono riportati in termini di classificazione Elo [19], che è il sistema di valutazione comunemente usato per valutare i giocatori di scacchi. Esso considera non solo la percentuale assoluta di partite vinte, ma prende anche in considerazione la forza dell'avversario. Per avere un'idea della susa scala di valori una differenza di valutazione di 100 punti significa approssimativamente che il giocatore più forte ha un tasso di successo atteso di 5/8.

Feature Type	# Features	Description
material difference	1	Multiplier for the material difference.
kings square table midgame	64	Position of the king in the midgame.
kings square table endgame	64	Position of the king in the endgame.
pawns square table midgame	64	Position of the pawns in the midgame.
pawns square table endgame	64	Position of the pawns in the endgame.
knights square table midgame	64	Position of the knights in the midgame.
knights square table endgame	64	Position of the knights in the endgame.
bishops square table	64	Position of the bishop.
queens square table	64	Position of the queen.
rooks square table	64	Position of the rook, multiplied with the pawn count.
queens mobility	28	Positions the queen can reach.
rooks mobility	15	Positions the rooks can reach.
bishops mobility	14	Positions the bishops can reach.
pawn bonuses	5	Values for double, island, isolated, backward and passed pawns.
trade bonuses	2	Bonus for following the "if ahead, trade pieces, if behind, trade pawns rule". Multiplied by pawn count and material value.
rook bonuses	3	Bonus for rooks on open or half-open files and the 2th/7th row.
bishop bonuses	2	Bonus for a pair of bishops as fixed value and pawn based multiplier.
trapped bishop penalties	2	Penalty for bishops blocked by enemy pawns.
threat bonus	1	Multiplier for the sum of pieces under attack.
king safety	1	Multiplier for the king safety value.

Figura 3.9: Tabella 2. Insieme completo delle caratterisitche utilizzate nella funzione di valutazione lineare del motore scacchistico CUCKOO.

3.3.4 Risultati degli esperimenti

Preferenze di mossa

Nel primo studio sono state utilizzate solo le preferenze di mossa e le 16 caratteristiche aggregate per la valutazione (Tabella 1). Inoltre sono state allenate sei diverse funzioni di valutazione utilizzando rispettivamente il 5%, 10%, 25%, 50%, 75% e il 100% di partite disponibili. Sono state provati poi tre diversi set per ogni giocatore (ad eccezione del 100% motore), e si è addestrato una funzione di valutazione per ogni training set.

Le funzioni apprese sono state valutate in tre tornei round-robin, ciascuno comprendente il giocatore con la funzione di ponderazione originale e un altro scelto random. Il motore casuale preleva una nuova serie di pesi casuali per ogni valutazione di posizione. I pesi sono campionati uniformemente alla scala min / max dei valori osservati all'interno di tutti i modelli SVMRank appresi. Ogni coppia ha giocato 100 partite con un lasso di tempo 5 minuti e senza incrementi. La Figura 3.10 mostra lo sviluppo del rating in base alla percentuale di preferenze di mossa usate nei dati di addestramento.

E' riconoscibile che un aumento della quantità di dati di preferenza utilizzati conduce ad un motore scacchistico migliore. Da ciò è possibile affermare che le annotazioni di gioco forniscono informazioni utili per l'apprendimento di una funzione di valutazione. Il giocatore addestrato con il 5% dei dati rappresenta un'anomalia. Questa è evidentemente risultante dalla elevata varianza nei dati di addestramento in questo punto. Tutto ciò è illustrato nella figura 3.11, nella quale è mostrata la media e la varianza per le cinque caratteristiche tra i valori appresi con la più alta variazione.

Tuttavia, la maggior parte delle funzioni convergono verso un valore medio stabile. La Figura 3.12 mostra l'andamento della media, deviazione standard e min / max per le caratteristiche selezionate. Le 5 caratteristiche in figura 3.12(a), sono abbastanza stabili, mostrando valori medi per lo più stabili, mentre le funzionalità di 3.12(b) sono piuttosto instabili. Questo potrebbe essere un motivo per il loro sviluppo instabile. Invece le cinque caratteristiche non riportate erano stabili.

Preferenze di posizione

Come è stato osservato in precedenza, il database contiene circa un ordine di grandezza in meno di preferenze di mossa rispetto preferenze di posizione. Quindi, è ragionevole aspettarsi che se ci spostiamo dalle preferenze di mossa al molto più grande insieme di preferenze di posizione, i nuovi risultati saranno decisamente migliori rispetto ai precedenti.

Per verificare ciò, abbiamo generato sia preferenze di mossa che di posizione utilizzando rispettivamente il 5%, 10%, 25%, 50%, 75% e il 100% delle partite disponibili.

Abbiamo diviso i dati in k cartelle, ognuna delle quali contenente 1/k dei dati (es. 4 cartelle con il 25% dei dati ciascuna) e addestrato una funzione di valutazione per ogni cartella. I giocatori, che rappresentano diversi valori di k, prima di ogni partita, in maniera casuale, selezionano delle funzioni di valutazione in base all a loro dimensione. Successivamente, a differenza della configurazione precedente, dove i tornei sono stati giocati separati e poi mediati, qui è stato giocato un torneo in cui i risultati di diverse funzioni di valutazione sono stati implicitamente mediati attraverso la selezione casuale in ogni partita. La seconda e la terza colonna della tabella presente in figura 3.13 mostra il numero di preferenze nei dati di addestramento per ogni configurazione, come media di tutte le cartelle della stessa dimensione.

Il secondo obiettivo è stato quello di confrontare i diversi modi per integrare le valutazioni di posizione. Per fare ciò si può: utilizzare direttamente la posizione a cui è collegata l'annotazione, attaccando l'annotazione alla fine della sequenza fornita dal annotatore, o eseguire una ricerca quiescente come viene comunemente fatto nelle funzioni di valutazione dell'apprendimento per rinforzo. Abbiamo confrontato tutte le quattro possibili combinazioni di queste opzioni effettuando tornei tra di loro in ciascuna delle 12 impostazioni sopra descritte. Ognuna delle sei coppie in ognuno dei 12 tornei è stata selezionata giocando 75 partite. Questo è stato ripetuto sia per l'apprendimento di una funzione di valutazione semplice (Tabella 1, figura 3.8) sia per una funzione di valutazione complessa (Tabella 2, figura 3.9).

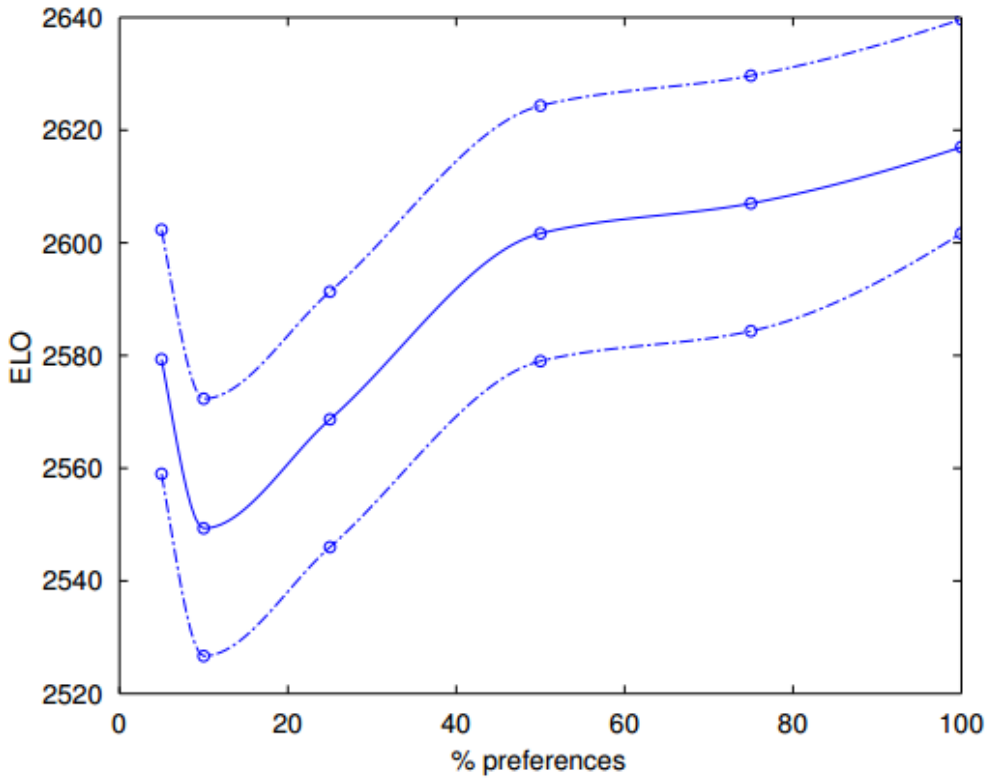


Figura 3.10: Forza di gioco misurata secondo la classificazione Elo, con limiti massimi e minimi sulle percentuali di preferenze di mossa utilizzare per l'allenamento.

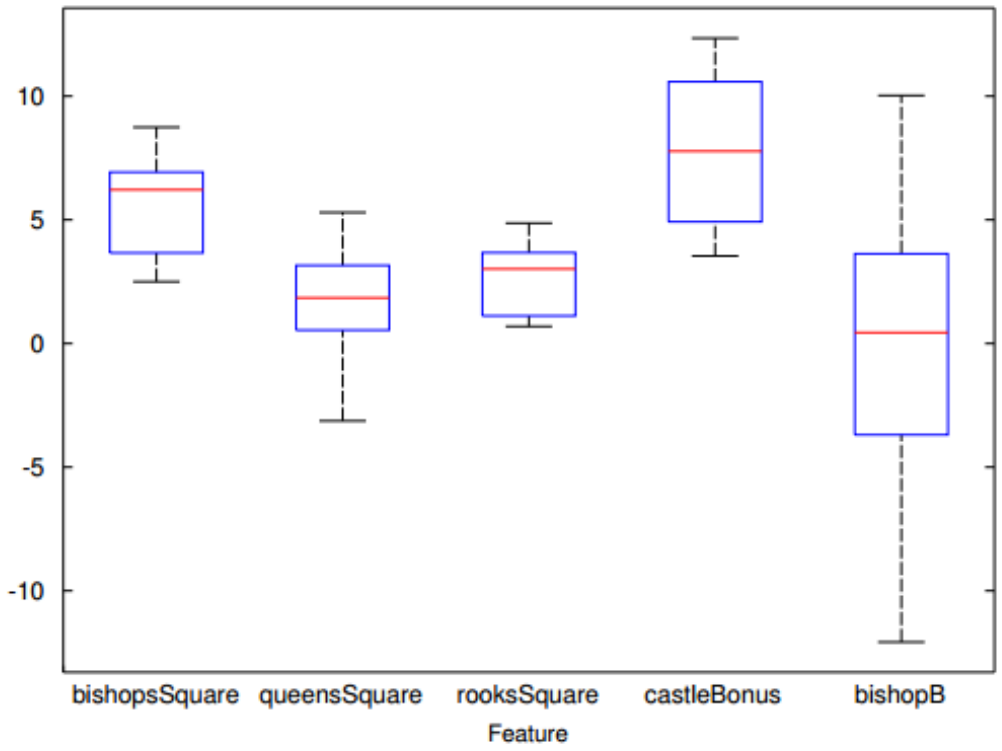
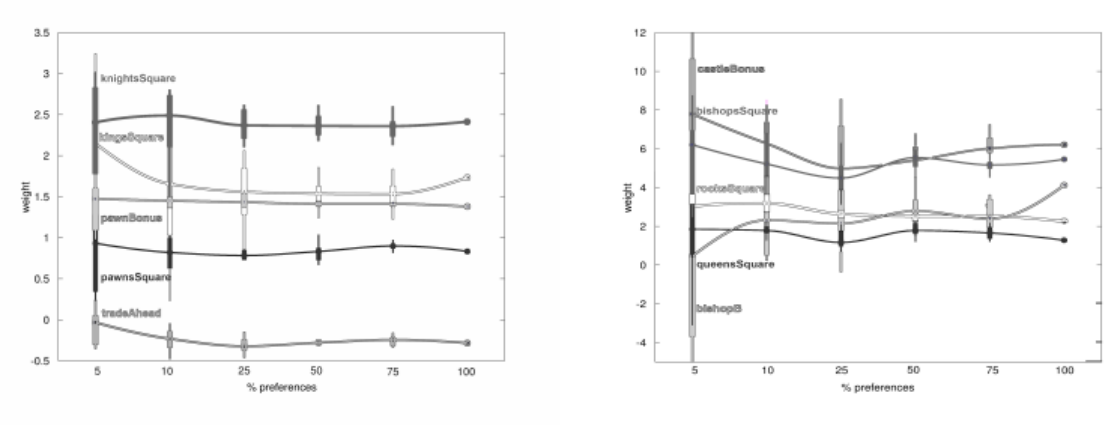


Figura 3.11: Varianza dei pesi appresi per le 5 caratteristiche più importanti, basate su 10 campioni casuali, ciascuno ottenuto adoperando il 5% dei dati disponibili.



(a) Stable (b) Unstable

Figura 3.12: Sviluppo della media e della varianza per caratteristiche di pesi selezionate calcolate su 10 campioni.

	Preferences			
#games	Move	Position	Augmented	Surrender
10%	0.02M	0.50M	0.34M	0.05M
25%	0.04M	1.26M	0.85M	0.12M
50%	0.09M	2.53M	1.71M	0.24M
100%	0.19M	5.05M	3.41M	0.48M

Figura 3.13: Numero medio di preferenze prodotte (in milioni).

Chiaramente, apprendendo da solo preferenze di mossa, vengono prodotti risultati molto peggiori che apprendendo anche da preferenze di posizione. Inoltre, è stato possibile dimostrare che l'aggiunta di preferenze di mossa alla preferenze di posizione non aumenta ulteriormente le prestazioni. Si deve tenere presente che il numero di preferenze di mossa è notevolmente inferiore al numero di preferenze di posizione. Tuttavia, vediamo anche dai risultati che l'apprendimento da preferenze di posizione è già saturo, nel senso che ci sono solo piccole variazioni di prestazioni aumentando la dimensione dei set di addestramento, vale a dire che un numero maggiore di preferenze di posizione non migliora ulteriormente le prestazioni. Lo stesso caso di riscontra quando si aggiungono alla preferenze di posizione le preferenze di mossa. Detto ciò è possibile quindi concludere affermando che le preferenze di mossa non aggiungono informazioni qualitativamente diverse da quelle di posizione e quindi il loro utilizzo non porta ad un miglioramento sostanziale dei risultati.

Aumentando le preferenze di posizione

Mentre si è potuto notare come l'apprendimento dall'assai più grande insieme di preferenze di posizione produce risultati notevolmente migliori rispetto ai primi esperimenti che utilizzavano esclusivamente le preferenze di mossa, è possibile però costatare ancora un notevole divario di prestazioni tra la performance del programma CUCKOO originale e quello che utilizza la funzione di valutazione modificata. In particolare, è stato rilevato che alcuni concetti fondamentali, come il valore materiale dei diversi pezzi, tende ad essere sottovalutato dalle euristiche da noi addestrate.

Una possibile ragione di ciò potrebbe risiedere nel fatto che i giocatori umani tendono ad annotare solo posizioni "interessanti", cioè quelle posizioni in cui la valutazione non è totalmente chiara o scontata. Perciò, situazioni di evidente svantaggio, quindi di non particolare interesse, come avere la regina a terra, non compaiono nelle annotazioni di gioco. Per testare questa ipotesi, si è cercato di aumentare i dati preferenziali con altre preferenze rappresentati situazioni di evidente vantaggio per uno dei

due giocatori. Sono state selezionate quelle posizioni annotate precedentemente come "di vantaggio decisivo" per uno dei due giocatori. Per ciascuna posizione,è stata generata una nuova posizione rimuovendo, al giocatore in svantaggio, un pezzo dalla scacchiera in maniera casuale (escludendo però re e pedoni). L'idea è che se si toglie un pezzo da una posizione già perdente, la posizione risultante dovrebbe essere ancora peggiore. Pertanto, tali posizioni sono auspicabilmente più estreme di quelle che di solito vengono annotata nei database. Applicando questa tecnica la quantità di preferenze di posizione è aumentata del 60% -70%, come si può vedere dalla quarta colonna della tabella di figura 3.13.

E' stato quindi confrontato il set di preferenza di posizione aumentate con il set di preferenze non-aumentate in un torneo con 100 partite per coppia. I risultati evidenziano una netta supremazia dei set di preferenza aumentato rispetto a quello non aumentato.

Infine vengono confrontate, per il giocatore originale, la migliori configurazioni di apprendimento generate dalle funzioni di valutazione semplice e complessa. Sono state giocate perciò 100 partite per coppia includendo le preferenze di mossa, posizione e quelle aumentate. Essenzialmente è constatabile che la funzione semplice surclassa quella complessa. Infatti, sembra che le informazioni disponibili di addestramento sono sufficienti per fare il tuning di un piccolo insieme di parametri, ma non sono abbastanza per fare il tuning del grande insieme di parametri necessitato dalla funzione complessa.

Concludendo, notiamo che persino la configurazione più performante non è riuscita a raggiungere le prestazioni della versione messa a punto manualmente, pur constatando che il divario tra le due configurazioni è diminuito rispetto allo studio preliminare che utilizzava solo le preferenze di mossa. Sembra sempre più evidente che tale fallimento derivi dal fatto che non tutte le informazioni deducibili da una partita di scacchi vengano annotate, ma solamente quelle considerate interessanti. Per questo prevalentemente motivo non possiamo ottenere dalle annotazioni di gioco la stessa qualità di risultati che sono stati raggiunti adoperando metodi più popolari come l'apprendimento per rinforzo con il self-play.

3.3.5 Conclusioni

I risultati evidenziati dagli esperimenti possono essere considerati contrastanti. Da una parte, si può dimostrare che è possibile apprendere informazioni utili a partire dalle annotazioni di gioco. Dall'altra, la prestazione del motore scacchistico non è direttamente proporzionale alla quantità crescente di informazioni fornita.

Un problema che è stato identificato è che gli annotatori umani tendono solo a concentrarsi su posizioni vicine e interessanti. Gli autori dell'articolo hanno potuto dimostrare che un aumento delle informazioni per l'allenamento con preferenze artificiali derivanti dalla produzione di cattive posizioni togliendo pezzi da scenari già perdenti, porta ad un significativo aumento della qualità della funzione appresa.

Quindi, possiamo concludere che importanti informazioni d'addestramento sono mancanti nelle annotazioni prodotte dall'uomo. Si potrebbe anche ipotizzare che le differenti performance osservate potrebbero essere dovute a diverse scale di giudizio utilizzate da diversi annotatori. Tuttavia, non sono state prodotte preferenze provenienti dall'incrocio di più partite, ma solo all'interno di una singola partita. Quindi, non possiamo soffrire di questo problema, purché ciascun esperto sia coerente con se stesso.

Si può concludere che affinché le annotazioni diano un concreto impulso al miglioramento delle prestazioni, esse dovrebbero essere integrate con altri metodi di apprendimento (ad esempio self-play con apprendimento per rinforzo). Ciò è in sintonia coi risultati ottenuti con i protocolli di addestramento, che sembrano anche essi indicare che questi approcci ibridi possono portare ad una migliore performance.

I possibili sviluppi futuri sono indirizzati proprio in questa direzione, cioè nel cercare di sviluppare approcci all'apprendimento ibridi, capaci di integrare metodi di apprendimento ormai consolidati, come l'apprendimento per rinforzo, con altri invece ancora poco conosciuti ma ricchi di potenzialità. Questo è il caso dell'apprendimento con annotazioni di gioco.

Dovrebbe anche essere poi possibile migliorare la scarsa performance dell'apprendimento con annotazioni dovuto al disinteresse dei giocatori es-

perti nell'annotare posizioni "non interessanti". Per esempio si potrebbero utilizzare le annotazioni di giocatori inesperti come preferenze aggiuntive. Queste annotazioni infatti conterrebbero anche quelle posizioni e mosse che per i grandi giocatori sono banali e poco interessanti.

Capitolo 4

Conclusioni e possibili sviluppi futuri

Questa tesi ha come oggetto l'Apprendimento Automatico nei giochi di strategia.

Nella prima parte è stata sviluppata un'introduzione teorica in cui sono stati illustrati i concetti base di questa disciplina, l'analisi di un problema di apprendimento e la definizione ed i metodi di un sistema di Machine Learning. Nella seconda parte sono stati sviluppati due differenti casi di studio. Le conclusioni che si possono trarre da questi casi sono le seguenti.

Nel primo caso è stato valutato, attraverso lo sviluppo di tre diverse reti neurali, un sistema di addestramento che massimizzi la percentuale di vittorie nel gioco del Tris. Il Tris è un ottimo caso di studio poiché combinando la semplicità delle sue regole ad una grande velocità di calcolo, che permette di effettuare un gran numero di partite in pochissimo tempo, fornisce un ambiente ideale per un primo apprendimento delle tecniche di Machine Learning. Nella fattispecie, si è affrontato lo sviluppo di tre diverse reti neurali, di 40, 60 e 80 nodi, effettuando fino 40.000 partite di addestramento. La rete di 40 nodi è stata la meno performante nel numero di vittorie; è stato inoltre constatato che, al raggiungimento delle 40.000 partite, il numero di vittorie, a discapito dei pareggi, aumenta al crescere del numero dei nodi di una rete, restando il numero di sconfitte pressoché uguale in tutte e tre le configurazioni.

Nel secondo caso di studio è stata riportata l'esperienza di C. Wirth e J. Furnkranz che mostra come sia possibile allenare un giocatore di scacchi attraverso le annotazioni di gioco.

Le annotazioni di gioco rappresentano per gli scacchi una grande fonte di informazioni che, se correttamente utilizzate, possono essere di grande aiuto per l'addestramento di una funzione di valutazione scacchistica.

I risultati di questo lavoro hanno mostrato che l'apprendimento da annotazioni di gioco è sì possibile ma le funzioni di valutazione così ottenute non sono state capaci di eguagliare le prestazioni della funzione originale del programma di scacchi la quale era stata regolata manualmente. La ragione di questo fallimento sembra risiedere nel fatto che gli annotatori umani si limitano ad annotare solo le posizioni "interessanti" di una partita, così che risulterà poi difficile imparare, adoperando esclusivamente le annotazioni, alcune situazioni di base, come ad esempio posizioni di palese vantaggio per uno dei due giocatori. Per questo motivo, ad oggi, non è possibile pensare di abbandonare i metodi d'apprendimento comunemente utilizzati finora in favore di un apprendimento che utilizzi esclusivamente le annotazioni di gioco. I propositi per il prossimo futuro prevedono lo sviluppo di metodi di apprendimento ibridi, dove l'apprendimento attraverso le annotazioni di gioco potrà essere di complemento a metodi di apprendimento ormai affermati nel dominio degli scacchi, come l'apprendimento per rinforzo.

Per quanto riguarda le prospettive future per l'Apprendimento Automatico nei giochi in generale si può dire che, negli ultimi anni, la ricerca ha trovato un nuovo ambiente di applicazione e test: I giochi commerciali. In confronto ai tradizionali giochi di strategia (come scacchi, backgammon o dama), questi giochi sono delle applicazioni più complesse e quindi più impegnative da realizzare. Ad esempio, in questo tipo di giochi, gli agenti, in genere, hanno necessità di interagire con un gran numero di altri agenti, partner o nemici in un ambiente real-time altamente dinamico, con una conoscenza incompleta riguardo i loro stati.

Ringraziamenti

Voglio ringraziare il mio relatore, il Professore Andrea Roli, per la sua disponibilitá e per l'aiuto che mi ha offerto nella elaborazione di questa tesi.

Ringrazio poi la mia famiglia per il supporto e la comprensione che mi ha dato in questi anni di Universitá ma sopratutto in questo ultimo periodo ricco di impegni e scadenze.

Infine, voglio ringraziare tutti i miei amici. In particolare quelli "nuovi" che ho avuto modo di conoscere all'Universitá, con i quali ho vissuto alcuni dei momenti più importanti, felici ma sopratutto divertenti di questi ultimi anni.

Bibliografia

[1] T. M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[2] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[4] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning (Vol. 2, No. 1)*. New York: Springer, 2009.

[5] S. J. Russell, and P. Norvig. *Intelligenza artificiale. Un approccio moderno. Vol. 1*. Pearson Italia Spa, 2005.

[6] S. J. Russell, and P. Norvig. *Intelligenza artificiale. Un approccio moderno. Vol. 2*. Pearson Italia Spa, 2005.

[7] M. P. D. Schadd. *Selective Search in Games of Different Complexity*. Ph.D. thesis, University of Maastricht, Maastricht, 2011.

[8] J. T. Saito. *Solving Difficult Game Positions*. Ph.D. thesis, University of Maastricht, Maastricht, 2010.

[9] Henk Mannen. *Learning to Play Chess Using Reinforcement Learning*. Master thesis, Cognitive Artificial Intelligence, University of Utrecht, Utrecht, 2003.

[10] T. M. Mitchell. *The Discipline of Machine Learning*. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[11] P. Domingos. *A Few Useful Things to Know about Machine Learning*. Communications of the ACM, 55(10), 78-87, 2012.

[12] C. Wirth, and J. Furnkranz. *On Learning From Game Annotations*. IEEE, 2014.

[13] H. Mannen, and M. Wiering. *Learning to Play Chess Using TD-learning with Database Games*. Cognitive Artificial Intelligence, University of Utrecht, 2004.

[14] C. Wirth, and J. Furnkranz. *First Steps Towards Learning from Game Annotations*. Proceedings of the ECAI-12 Workshop on Preference Learning: Problems and Applications in AI, 53-58, 2012.

[15] J. Furnkranz. *Recent Advances in Machine Learning and Game Playing*. OGAI Journal, 26(2), 19-28, 2007.

[16] A. L. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of Research and Development, vol. 3, no. 3, 211?229, 1959.

[17] G. Tesauro. *TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play*. Neural computation, 6.2, 215-219, 1994.

[18] J. Furnkranz, and E. Hullermeier. *Preference Learning*. Springer-Verlag, 2010.

[19] A. E. Elo. *The Rating of Chessplayers, Past and Present*, 2nd ed., New York: Arco, 1978.