ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria Informatica

Attività progettuale di Intelligenza Artificiale M

Reti neurali applicate al gioco da tavolo Hanabi

Ilaria Crivellari
Riccardo Varotto

Anno Accademico 2021-2022

Indice

1 Introduzione

Hanabi è un gioco di carte cooperativo per 2-5 giocatori.

Questa attività si propone come obiettivo la progettazione e la realizzazione di una rete neurale in grado di giocare ad Hanabi nella versione con 2 giocatori.

Si è scelto di implementare la rete attraverso la libreria Tensorflow e il dataset è stato generato utilizzando un software di gioco già realizzato da altri studenti in una precedente attività progettuale.

L'attività è consistita quindi nelle seguenti fasi:

- Generazione del dataset con il software di gioco già realizzato in formato JSON
- Modifica del formato del dataset per avere una rappresentazione compressa e una gestione facilitata nell'addestramento della rete
- Progettazione, realizzazione e addestramento di varie reti neurali
- Creazione di un nuovo modulo nell'ambiente di gioco che calcola la nuova azione tramite la rete migliore

Regole del gioco I componenti del gioco consistono in:

- Un mazzo di 50 carte suddivise in cinque colori (rosso, bianco, blu, verde, giallo) e per ogni colore sono presenti tre carte con il numero 1, due carte con i numeri 2,3 e 4 e una carta con il numero 5.
- 8 gettoni indizio
- 3 gettoni errore

Ogni giocatore ha 5 carte in mano a lui coperte, mentre vede quelle degli altri giocatori. Lo scopo del gioco consiste nel giocare in ordine (da 1 a 5) le carte di ognuno dei 5 colori, a seguito di indizi ricevuti collaborando con gli altri giocatori. Le pile delle carte giocate di ogni colore, sono in comune tra tutti i giocatori.

Ogni giocatore al proprio turno può eseguire una delle azioni seguenti:

- Giocare una carta: la carta si posiziona sopra la pila di carte di quel colore se è giocabile (la carta deve seguire numericamente quella sul tavolo di quel colore) altrimenti viene scartata e si segna un errore perdendo un gettone errore. Non è necessario sapere colore e numero della carta che si sta giocando, basta che essa sia giocabile in una qualunque delle pile sul tavolo. Ogni carta giocata vele 1 punto.
- Scartare una carta: la carta si scarta, si pesca una nuova carta e si guadagna un nuovo gettone indizio. Le carte scartate sono sempre visibili dai giocatori durante tutta la durata della partita.
- Dare un indizio ad un altro giocatore: comporta l'utilizzo di un gettone indizio, si può suggerire o un colore o un numero indicando tutte le carte della mano di un giocatore di quel colore o numero.

.

La partita termina con punteggio 0 quando si sono commessi 3 errori, se ciò non accade, quando finiscono le carte del mazzo tutti i giocatori hanno a disposizione un ultimo turno dopo di che il gioco finisce con punteggio uguale al numero delle carte giocate.

2 Stato dell'arte

Negli ultimi anni c'è stato un crescente interesse della comunità informatica verso il gioco Hanabi con l'implementazione di vari giocatori con diverse tecniche. In particolare, la combinazione tra il fatto che sia un gioco in cui i giocatori hanno una conoscenza asimmetrica dello stato e la natura cooperativa lo rendono un gioco molto diverso dai classici problemi come Go e scacchi e quindi i tipici approcci di questi giochi non funzionano [1].

A seguito dell'interesse per questo gioco sono state create varie competizioni per la creazione di agenti in grado di giocare, ma nessuno è ancora riuscito a creare un bot in grado di giocare correttamente con giocatori esperti cogliendo tutti gli indizi e la conoscenza nascosta in ogni indizio [2]. Questo è dovuto al fatto che generalmente gli indizi contengono una conoscenza nascosta che è immediata da capire per un giocatore esperto ma difficile da codificare [2].

Software di gioco Per un'altra attività progettuale altri due studenti hanno creato un'implementazione del software di gioco e 4 bot, di cui solo 2 completi, che sono in grado di giocare ad Hanabi con diverse strategie. Il software permette di giocare da 2 a 5 giocatori scegliendo una qualsiasi combinazione tra i bot e il giocatore umano.

3 Generazione e modellazione del dataset

Per la generazione del dataset è stato utilizzato il software di gioco prodotto dalla precedente attività progettuale nella vesione a 2 giocatori e creato il nuovo modulo `hanabi-dataset`.

Il giocatore scelto per la generazione dei dati è il `Bot2` che risulta essere quello con migliori performance e ha una media di 14.37 punti su 100 partite.

Sono state generate 36.000 partite, ognuna delle quali consiste in media di 75-80 stati. Il numero totale degli stati generati è 2.737.394.

La generazione del dataset è consistita in una prima fase di generazione in formato JSON attraverso il software di gioco ed una seconda fase di formattazione per trasformare ogni stato in una stringa composta di soli double in modo da avere un input immediatamente pronto per la rete neurale ed eliminare i campi inutili.

3.1 Fase 1: Dataset in formato JSON

Il software di gioco consente di trasformare lo stato del gioco in formato JSON, che viene utilizzato per mandare lo stato ai giocatori ad ogni turno.

E' stato deciso di creare due file per ogni partita, il primo contenente tutti gli stati uno per ogni riga e il secondo contenente in ogni riga l'azione corrispondente allo stato della stessa riga nel file degli stati.

La rappresentazione dello stato in formato JSON contiene i seguenti campi:

- *hints*: numero di indizi rimasti.
- *finalround*: vale sempre -1 fino a che non si pesca l'ultima carta, in cui assume il numero del round corrente.
- 5 fireworks (*white, blue, red, yellow, green*): indicano quale è l'ultima carta giocata di ogni colore.
- *deck*: numero di carte rimaste nel mazzo.
- *handentropy_other*: valore dell'incertezza assegnato alla mano dell'altro giocatore.
- *handentropy_current*: valore dell'incertezza assegnato alla mano del giocatore corrente.

- *fuse*: numero di errori rimasti.
- *round*: numero del round corrente.
- *current*: nome del giocatore che deve effettuare la prossima mossa.
- Per ogni carta di entrambi i giocatori denominati *current* e *other*:
  - *playability*: punteggio di giocabilità assegnato a quella carta.
  - *color*: colore della carta.
  - *value*: valore della carta.
  - *cardentropy*: punteggio di incertezza assegnato alla carta.
  - *uselessness*: punteggio di utilità assegnato alla carta.
  - *poss_values*: array di 5 percentuali che rappresentano la probabilità di una carta di essere di ogni numero.
  - *poss_colors*: array di 5 percentuali che rappresentano la probabilità di una carta di essere di ogni colore.
- *discarded*: array di 5 numeri che rappresentano le carte scartate intesa come 5 interi a 6 cifre in cui la prima cifra rappresenta il colore; le altre 5 rappresentano il numero di carte scartate per ogni valore.

La rappresentazione delle azioni in formato JSON contiene i seguenti campi:

- *type*: contiene uno dei seguenti valori in base all'azione effettuata *hintvalue, hintcolor, discard, play.*
- *player*: giocatore che effettua l'azione.
- Nel caso di azione *hintvalue o hintcolor*:
  - *hinted*: giocatore che riceve l'indizio.
  - *value*: intero che rappresenta il valore o il colore dell'indizio.
- Nel caso di azione *discard o play*:
  - *card*: posizione della carta da scartare o giocare.

Conseguentemente ci sono 20 azioni possibili: 5 di scarto (*discard*) in base alla posizione della carta scartata, 5 di gioco (*play*) in base alla posizione della carta giocata, 5 di suggerimento di valore (*hint value*) e 5 di suggerimento di colore (*hint color*).

3.2 Fase 2: Formattazione del dataset

Il processo di formattazione consiste nel leggere ogni riga in formato JSON dai file (*RawSatate*) precedentemente prodotti e trasformarla in una lista di numeri (*FinalState*). Per quanto riguarda gli stati, si ottengono 195 valori per ogni riga. Sono stati generati due dataset per confrontare le prestazioni: uno tiene in considerazione le simmetrie dei colori mentre l'altro no. Per la generazione di entrambi i dataset è stato deciso di:

- Eliminare *finalround, round, current.*
- Codificare i colori con numeri binari.
- Codificare le carte scartate di ogni colore con 5 numeri rappresentanti il numero di carte scartate di ognuno dei cinque valori.
- Eliminare i dati riferiti a colore e numero delle proprie carte, in quanto il giocatore non ne ha conoscenza. Se gli è stato dato un indizio con questa informazione, essa sarà rappresentata dalla percentuale dei *poss_values* e *poss_colors.*

Per le azioni è stato deciso di assegnare un numero da 0 a 19 ad ogni azione possibile e utilizzare il One-Hot Encoding, ottenendo 20 valori per ogni riga.

Per generare il dataset che tiene in considerazione le simmetrie dei colori sono stati ordinati tutti i valori riferiti a colori (valore del firework, carte scartate, carte in mano di entrambi i giocatori) e di conseguenza codificate le azioni in base al nuovo ordinamento di carte e colori.

Gestione delle simmetrie Gli stati di Hanabi possono presentare delle simmetrie, questo succede poiché i colori sono intercambiabili tra di loro. Per gestire queste situazioni è stato deciso di dare un ordine ai colori in ogni stato della partita e ordinare i dati relativi ad ogni colore secondo questo ordine che varia ad ogni stato. Per definire l'ordine sono stati usati rispettivamente: valore del firework, carte scartate dal 5 all'1, carte in mano all'altro giocatore dal 5 all'1, *poss_color* di ogni carta dell'altro giocatore sia del colore ordinando le possibilità in ordine crescente, *poss_color* di ogni carta del giocatore corrente e ordinando le possibilità in ordine crescente.

Per quanto riguarda le azioni, esse risentono ugualmente di questo ordinamento: le azioni di gioco o scarto richiedono il calcolo della posizione della carta che si scarta o gioca dopo l'ordinamento, mentre per le azioni di indizio colore, i cinque bit che indicano i colori sono ordinati secondo lo stesso ordine dato ai colori.

Gestione degli stati ripetuti Inizialmente era stato deciso di eliminare gli stati ripetuti dal dataset, andando a rileggere tutto il file degli stati ogni volta per controllare la presenza di una riga uguale a quella attuale. Questo processo richiedeva molto tempo per rileggere ogni volta tutto il file degli stati già convertiti.

Dopo aver convertito 2.200 partite su 36.000, per un totale di circa 170.000 stati convertiti sui 2.7 milioni disponibili, gli stati ripetuti incontrati erano 2. Dato che per convertire 100 partite nel dataset finale in questo modo sono necessarie circa 15 ore e il tempo aumenta notevolmente all'aumentare delle dimensioni del dataset, si è deciso che non valeva la pena effettuare questo controllo e quindi di tenere anche stati ripetuti.

3.3 Analisi del dataset

La distribuzione delle classi nel dataset senza ordinamento dei colori è mostrata nella tabella 1 e nel grafico in figura 1.

Classe	**Percentuale**
PLAY_1st	3,57%
PLAY_2nd	3,50%
PLAY_3rd	3,68%
PLAY_4th	6,59%
PLAY_5th	1,81%
DISCARD_1st	6,82%
DISCARD_2nd	6,99%
DISCARD_3rd	7,05%
DISCARD_4th	6,82%
DISCARD_5th	7,21%
HINT_VALUE_1	7,41%
HINT_VALUE_2	5,98%
HINT_VALUE_3	4,90%
HINT_VALUE_4	3,98%
HINT_VALUE_5	1,64%
HINT_WHITE	4,51%
HINT_BLUE	4,41%
HINT_RED	4,37%
HINT_YELLOW	4,43%
HINT_GREEN	4,34%

Tabella 1: *Distribuzione classi nel dataset*

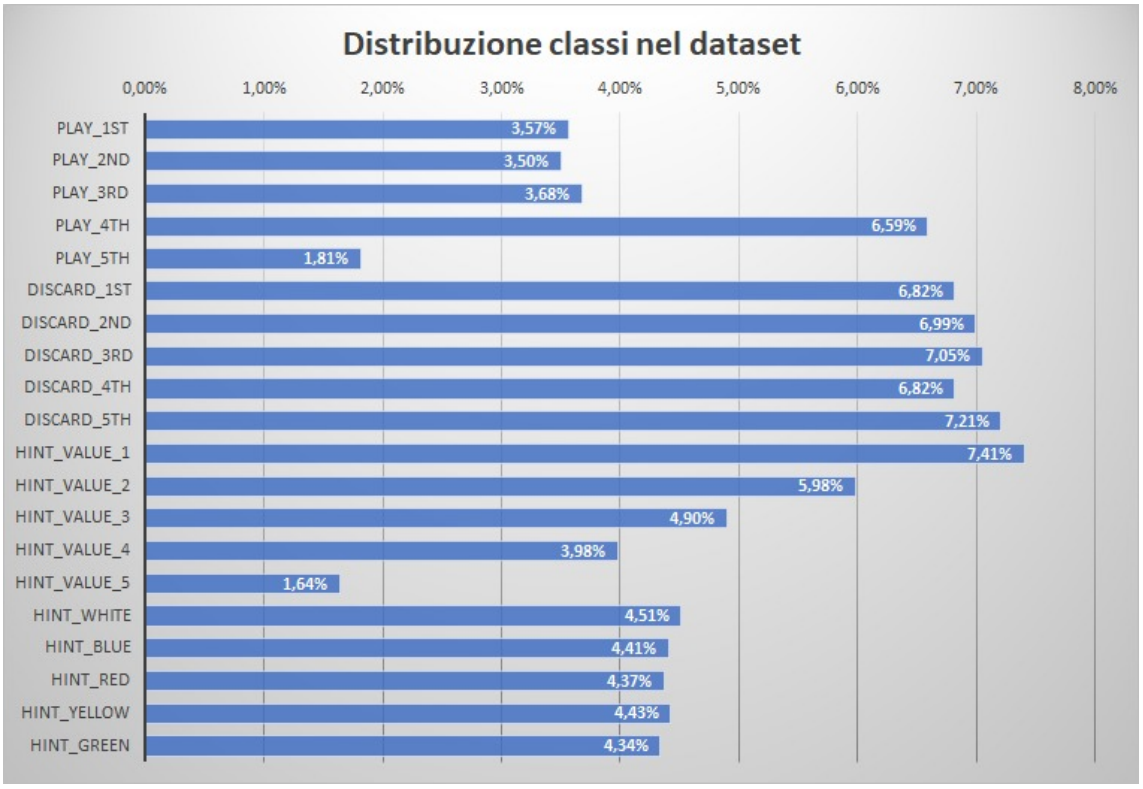

Figura 1: *Grafico della distribuzione delle classi nel dataset*

4 Rete neurale

4.1 Modello

Il modello della rete neurale si basa su una semplice architettura, che prevede il posizionamento in sequenza di diversi livelli `Dense` con ciascuno un numero diverso di neuroni. Il livello iniziale di input è costituito da 195 nodi, ovvero dallo stesso numero di features che compongono lo stato di una partita. Seguendo lo stesso ragionamento, il numero di nodi che costituiscono il livello finale di output è 20, uno per ogni possibile azione che può essere presa durante una partita. A seconda dello stato ricevuto, la rete deve infatti decidere quale delle 20 azioni compiere eseguendo una classificazione multi-classe. A questo scopo, il livello di output utilizza come funzione di attivazione la *softmax*, rendendo quindi possibile l'attivazione di uno solo tra i 20 neuroni di questo livello, ovvero quello corrispondente all'azione da effettuare. Per quanto riguarda invece i livelli intermedi (*hidden layers)*, il numero di neuroni in ciascun livello decresce progressivamente più si entra in profondità, partendo da 128 (a seguito della `Dense` di input), 64 e infine 32 (immediatamente prima della `Dense` di output).

Per evitare il problema di overfitting, sono state applicate tecniche di regolarizzazione introducendo dei livelli di `Dropout` dopo ogni `Dense`. Tramite questi livelli vengono "spenti" casualmente il 10% dei neuroni permettendo alla rete di migliorare in stabilità. Infine, si è notato sperimentalmente che l'introduzione di un livello di `BatchNormalization` prima della `Dense` di output aumenta notevolmente le prestazioni, la velocità e la stabilità della rete neurale. Per una visualizzazione grafica più chiara dell'architettura, si osservino le figure 2 e 3.

L'addestramento della rete viene effettuato suddividendo il dataset in 3 insiemi: *training* 70%, *validazione* 20%, *test* 10%. Si è deciso di effettuare questa partizione con lo scopo di conciliare da un lato la capacità della rete di imparare da un insieme elevato di stati, dall'altro la flessibilità della stessa in presenza di nuovi stati precedentemente non incontrati.

La procedura che ha portato alla scelta di questo modello si è basata sulla sperimentazione di diversi tipi di architetture, modificando di volta in volta diverse variabili come: numero di livelli

di `Dense`, numero di nodi in ciascun livello, percentuale di neuroni "spenti" nei livelli di `Dropout`. Il modello della rete scelta si è dimostrato il migliore tenendo in considerazione in primo luogo la *accuracy* e la *loss* durante l'allenamento sui set di validazione e test, ma anche la complessità generale della rete (a parità di performance, si sono preferite reti più semplici) e la velocità di addestramento. Si veda la sezione 4.3 per maggiori informazioni.

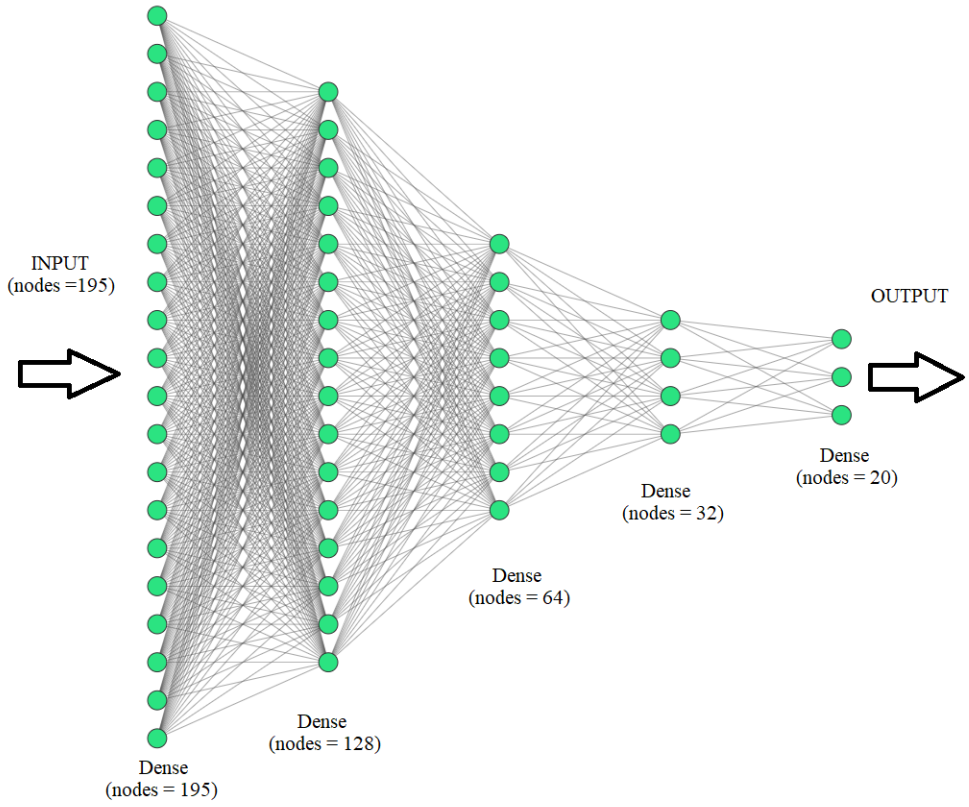

Figura 2: *Illustrazione dell'architettura della rete neurale, con le dimensioni dei livelli.*

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 195)]	0
x1 (Dense)	(None, 195)	38220
dropout (Dropout)	(None, 195)	0
x2 (Dense)	(None, 128)	25088
dropout_1 (Dropout)	(None, 128)	0
x3 (Dense)	(None, 64)	8256
dropout_2 (Dropout)	(None, 64)	0
x4 (Dense)	(None, 32)	2080
batch_normalization (BatchNormalization)	(None, 32)	128
output (Dense)	(None, 20)	660

Figura 3: *Sommario del modello della rete.*

4.2 Performance

Il modello così realizzato è stato addestrato per 100 epoche raggiungendo una *accuracy* del 79,4% con una *loss* di 0,55 nell'insieme di test, come mostrato in figura 4.

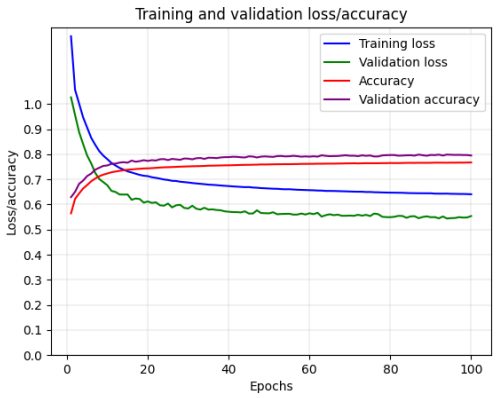

Figura 4: *Andamento di accuracy e loss durante l'addestramento.*

Entrando maggiormente nel dettaglio delle performance della rete neurale, in figura 5 è mostrato il *classification report*, in cui vengono analizzate, per ognuna delle 20 azioni possibili, una serie di caratteristiche che indicano la capacità della rete di selezionare l'azione giusta. In particolare:

- **Precision**: indica in numero di volte che la rete ha predetto l'azione x correttamente rapportato al numero di volte totale che ha predetto l'azione x.
- **Recall**: indica il numero di volte che la rete ha predetto l'azione x correttamente rapportato al numero totale di occorrenze dell'azione x.
- **F1-score**: misura che combina i due valori precedenti mediante media armonica.
- **Support**: numero di occorrenze dell'azione x.

	precision	recall	f1-score	support
play_1st	0.99	1.00	0.99	9760
play_2nd	1.00	1.00	1.00	9581
play_3rd	1.00	1.00	1.00	10063
play_4th	1.00	1.00	1.00	18038
play_5th	1.00	1.00	1.00	4968
discard_1st	0.84	0.97	0.90	18670
discard_2nd	0.88	0.96	0.92	19147
discard_3rd	0.91	0.96	0.93	19306
discard_4th	0.96	0.92	0.94	18664
discard_5th	0.96	0.97	0.96	19729
hint_value_1	0.81	0.68	0.74	20273
hint_value_2	0.63	0.66	0.64	16378
hint_value_3	0.55	0.55	0.55	13406
hint_value_4	0.53	0.50	0.52	10897
hint_value_5	0.57	0.23	0.33	4480
hint_white	0.57	0.65	0.61	12355
hint_blue	0.60	0.57	0.58	12068
hint_red	0.58	0.60	0.59	11963
hint_yellow	0.61	0.53	0.57	12115
hint_green	0.56	0.57	0.57	11879
accuracy			0.79	273740
macro avg	0.78	0.77	0.77	273740
weighted avg	0.79	0.79	0.79	273740

Figura 5: *Classification report.*

4.2.1 Performance di gioco

Una volta raggiunta, tramite l'addestramento, una accuracy considerata soddisfacente, l'agente è stato testato nel software di gioco in coppia rispettivamente con `Bot1`, `Bot2` (che è stato l'insegnante nella rete neurale) e infine anche con se stesso. Sono state effettuate 100 partite e raccolte alcune statistiche: media punti raggiunta, deviazione standard, punteggio massimo e minimo. Nella tabella 2 sono illustrati i risultati raggiunti per ogni casistica; a titolo di confronto, nelle ultime due righe vengono riportati anche i risultati ottenuti dalle partite giocate dall'insegnante `Bot2`.

Giocatori	Media	Deviazione standard	Massimo	Minimo
Rete Neurale - Bot2	13,32	2,88	21	6
Rete Neurale - Bot1	11,89	2,92	18	4
Rete Neurale - Rete Neurale	11,51	2,74	17	4
Bot2 - Bot2	14,37	3,07	20	8
Bot2 - Bot1	12,71	2,90	19	8

Tabella 2: Tabella che mostra punteggi medi, deviazione standard, massimo e minimo raggiunti su 100 partite

Si può quindi osservare come le performance con la rete neurale calino leggermente rispetto alle partite giocate esclusivamente dai Bot già presenti, essendo che le azioni sbagliate si accumulano rendendo la partita più complicata. Tuttavia, la discreta accuracy raggiunta nella selezione delle azioni permette alla rete neurale di effettuare spesso una scelta intelligente e di ottenere un punteggio finale accettabile. La media punti si abbassa infatti di circa 1 solo punto passando dalla coppia `Bot2` - `Bot2` (14,37) alla coppia `Rete Neurale` - `Bot2` (13,32) analogamente a quanto accade passando dalla coppia `Bot2` - `Bot1` (12,71) alla coppia `Rete Neurale` - `Bot1` (11,89).

Infine, la media punti su 100 partite giocate da due reti neurali in coppia è di 11,51 punti; questo punteggio risulta essere il più basso e questo potrebbe essere dovuto all'accumularsi in cascata degli errori da parte di ambo i giocatori, invece che da parte di un singolo giocatore.

4.3 Esperimenti

Si sono testate diverse varianti della rete realizzata, sia dal punto di vista dell'architettura che del dataset fornito alla rete durante l'addestramento.

4.3.1 Colori ordinati

Come già più dettagliatamente spiegato nella sezione 3.2, alcuni stati durante le partite possono presentare delle simmetrie in quanto i colori sono intercambiabili tra di loro. Si è quindi provveduto ad effettuare un ordinamento dei colori ad ogni stato della partita al fine di rendere l'apprendimento della rete neurale più semplice e quindi aumentare (teoricamente) le performance dell'agente. Tuttavia i risultati non sono stati quelli sperati, raggiungendo una *accuracy* complessiva solamente del 56%. Maggiori dettagli sono illustrati nel *classification report* mostrato in figura 6.

Si può notare come in realtà la capacità della rete di predirre azioni di tipo *hint-color* sia migliorata rispetto alla versione senza ordinamento dei colori, passando da circa il 60% al 70% di accuracy. La capacità di predirre azioni di tipo *hint-value* resta pressochè invariata, mentre ciò che cala drasticamente è l'abilità nel scegliere azioni di tipo *play* e *discard*: qui il valore *f1-score* passa da un range 90-100% a 26-70%.

Il notevole peggioramento suggerisce la possibilità di errori nell'algoritmo di ordinamento delle carte: la rete neurale "capisce" che deve giocare/scartare una carta ma seleziona molto spesso una carta sbagliata, causando di conseguenza un errore che può poi diventare irreversibile portando alla perdita della partita.

4.3.2 Diverse architetture della rete

No BatchNormalization Inizialmente l'architettura della rete non prevedeva un livello di `BatchNormalization` ma al suo posto un altro livello di `Dropout`. In questo modo l'addestramento raggiungeva un'*accuracy* del 65%.

	precision	recall	f1-score	support
play_1st	0.54	0.39	0.45	9760
play_2nd	0.31	0.37	0.34	9581
play_3rd	0.32	0.21	0.26	10063
play_4th	0.58	0.77	0.66	18038
play_5th	0.65	0.44	0.53	4968
discard_1st	0.54	0.60	0.57	18670
discard_2nd	0.37	0.35	0.36	19147
discard_3rd	0.39	0.28	0.32	19306
discard_4th	0.42	0.35	0.38	18664
discard_5th	0.61	0.82	0.70	19729
hint_value_1	0.78	0.77	0.77	20273
hint_value_2	0.56	0.60	0.58	16378
hint_value_3	0.52	0.41	0.45	13406
hint_value_4	0.47	0.50	0.48	10897
hint_value_5	0.52	0.32	0.39	4480
hint_white	0.68	0.80	0.74	13033
hint_blue	0.77	0.63	0.69	13295
hint_red	0.69	0.68	0.69	12955
hint_yellow	0.62	0.78	0.69	11883
hint_green	0.72	0.76	0.74	9214
accuracy			0.56	273740
macro avg	0.55	0.54	0.54	273740
weighted avg	0.55	0.56	0.55	273740

Figura 6: *Classification report con i colori ordinati.*

Variazioni Dropout Per ogni livello di `Dropout` sono state sperimentate diverse percentuali di "spegnimento" dei neuroni. Si è notato come salendo sopra al 10% l'*accuracy* raggiunta dal modello comincia a calare leggermente (circa 76%) mentre con percentuali comprese tra il 5 e il 10% l'*accuracy* rimane pressoché invariata (78-79%).

Add Un'altra variante riguarda l'utilizzo di un livello di `Add`, fornendo una scorciatoia e sommando direttamente dopo il livello di input. La struttura ha subito una sostanziale modifica, come si può notare in figura 7. La rete così addestrata ha raggiunto una *accuracy* del 78%, non portando quindi significativi miglioramenti; tuttavia il modello risulta più pesante e la fase di addestramento ha richiesto il doppio del tempo.

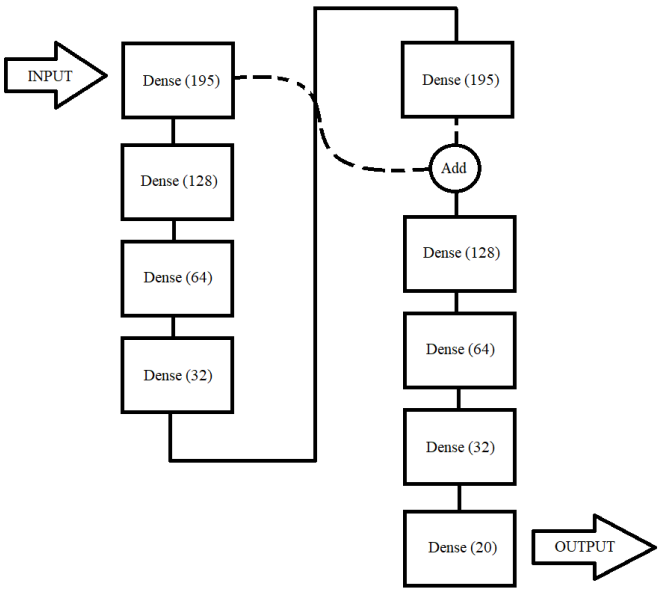

Figura 7: *Architettura con il livello di Add.*

Deep Dense In questa architettura, sono stati "sdoppiati" i livelli di `Dense`, portando ad avere in sequenza 2 livelli da 195 neuroni, poi 2 livelli da 128, 64 e infine 32. Questa soluzione con un modello più profondo non si è però rivelata efficace, andando ad ottenere un'*accuracy* del 70% aumentando notevolmente e il tempo di addestramento.

Variazioni Dense Infine, nel modello scelto sono state tentate diverse configurazioni del numero di neuroni in ciascun livello di `Dense`. Le variazioni dal punto di vista dell'*accuracy* sono state molto lievi, ciò che ha subito un notevole incremento è stato il tempo di addestramento e la pesantezza della rete.

Per esempio, in una rete dove i livelli di `Dense` erano formati rispettivamente da 512, 128 e 32 neuroni, i parametri allenabili erano più di 200'000 risultando molto maggiori rispetto ai circa 75'000 del modello finale. Le performance raggiunte da questo modello erano leggermente migliori a quelle della rete realizzata, arrivando ad ottenere un'*accuracy* del 79.8%; tuttavia, si è voluto mantenere un modello più semplice e leggero tenendo in considerazione che dal punto di vista delle performance di gioco, tale miglioramento era impercettibile.

5 Integrazione del nuovo modulo con il software di gioco

Per integrare la rete neurale con il software di gioco è stato creato un nuovo modulo `hanabi-neural-network` che contiene un nuovo bot capace di utilizzare la rete neurale per calcolare la prossima azione. In particolare il modulo contiene una nuova classe java `Bot` che è in grado di comunicare con la rete neurale e uno script python che contiene il calcolo dell'azione.

Quando si seleziona la rete neurale (NN) come giocatore, il server lancia il jar relativo che è in grado di comunicare con la rete neurale attraverso una socket.

Il funzionamento è il seguente:

- Si lancia lo script python che apre la socket e si mette in attesa.
- Si seleziona NeuralNetwork come giocatore nel server di gioco.
- Bot instaura la connessione con NN su una socket TCP.
- Per ogni turno di NN:
  - Bot riceve dal server di gioco lo stato in formato JSON
  - Bot costruisce la stringa di double che rappresenta lo stato (*FinalState*)
  - Bot invia la stringa a NN
  - NN calcola l'azione da effettuare e restituisce un intero da 0 a 19 che rappresenta il codice dell'azione
  - Bot esegue il parsing del risultato e restituisce al server l'azione richiesta
- Al termine della partita Bot chiude la connessione e termina. In caso di partite multiple il server Pyhton riapre una socket per la partita successiva.

Azioni proibite Ci sono alcune azioni che non possono essere effettuate in determinate situazioni e vengono rilevate dal server di gioco come proibite portando alla terminazione della partita. Poiché la rete non ha un'accuratezza del 100% essa potrebbe decidere di compiere una di queste azioni in situazioni vietate. Per evitare ciò, Bot controlla che l'azione richiesta da NN sia legale e se è illegale decide arbitrariamente di effettuare un'altra azione che è la meno dannosa possibile.

Le azioni proibite e le azioni cablate in caso di necessità sono le seguenti:

- E' vietato scartare una carta se ci sono a disposizione 8 gettoni indizio. Nel caso si verifichi questa situazione, è stato deciso di suggerire il valore della carta in posizione 1 dell'altro giocatore.
- E' vietato fornire un indizio se non ci sono gettoni indizio a disposizione. Nel caso si verifichi questa situazione, è stato deciso di scartare la carta in posizione 1.

E' bene specificare che nei test effettuati la rete neurale non ha mai predetto un'azione proibita, tuttavia per sicurezza sono state prese misure preventive al fine di evitare il crash dell'applicazione.

6 Conclusioni

In questa attività progettuale è stato creato un modulo Java-Python, integrato con il server di gioco esistente, per giocare ad Hanabi che calcola le mosse da compiere attraverso una rete neurale.

L'attività è stata caratterizzata da tre fasi: la generazione del dataset, la sua formattazione e costruzione e allenamento della rete neurale.

La rete che ha mostrato migliori performance ha raggiunto una *accuracy* del 79,4% nella selezione delle azioni, usando come insegnante l'agent già realizzato `Bot2`. Il punteggio medio ottenuto giocando 100 partite in coppia (`Rete Neurale - Bot2`) è di 13.32, risultando leggermente inferiore rispetto alla media raggiunta da due insegnanti (`Bot2 - Bot2`), 14.37.

Limitazioni e lavori futuri Il modulo creato funziona solamente nel caso di partite a due giocatori, si potrebbe pensare di estendere il modulo con il supporto a 3 e/o 4 giocatori.

Il dataset utilizzato per l'allenamento contiene i dati generati da 36.000 partite e probabilmente la rete funzionerebbe meglio se avesse a disposizione più stati per allenarsi. In fase di addestramento si è infatti notato come l'*accuracy* raggiunta con 1/4 del dataset fosse del 65%, mentre con il dataset completo è stato possibile raggiungere il 79.4% finale.

La versione con ordinamento dei colori ha performance migliori per le azioni di suggerimento, mentre gioca e scarta molto male. Si potrebbe approfondire questa formattazione provando a capire se ci sono eventuali errori nell'ordinamento delle carte ed in caso affermativo risolverli per verificare un effettivo miglioramento nelle performance rispetto all'utilizzo del dataset standard.

Per concludere, al fine di migliorare ulteriormente la stabilità e le performance della rete neurale, si potrebbe ricorrere all'utilizzo di tecniche come *cross-validation*.

Riferimenti bibliografici

[1] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

[2] Ho Chit Siu, Jaime Peña, Edenna Chen, Yutai Zhou, Victor Lopez, Kyle Palko, Kimberlee Chang, and Ross Allen. Evaluation of human-ai teams for learned and rule-based agents in hanabi. *Advances in Neural Information Processing Systems*, 34, 2021.