Playing *Connect-4* with Deep Neural Networks

Luca Arnaboldi

Corso "Computing Methods for Experimental Physics and Data Analysis"

14 ottobre 2021

Introduzione

- *Forza-4* è un gioco ad informazione completa, con 4 531 985 219 092 stati possibili [Tro12].
- Il numero di stati limitato permette di risolvere il gioco con algoritmi di ricerca completa.
- Non è un problema in cui le reti neurali sono utili ai fini pratici.

È una buona palestra per testare l'abilità delle DNN, potendo fare un confronto con gli algoritmi classici. Seguiamo due approcci:

- Supervised Learning: gli algoritmi classici generano il dataset per allenare la rete;
- Reinforcement learning: il giocatore basato sulla rete neurale gioca contro se stesso ed impara dalle proprie mosse.

Introduzione

- *Forza-4* è un gioco ad informazione completa, con 4 531 985 219 092 stati possibili [Tro12].
- Il numero di stati limitato permette di risolvere il gioco con algoritmi di ricerca completa.
- Non è un problema in cui le reti neurali sono utili ai fini pratici.

È una buona palestra per testare l'abilità delle DNN, potendo fare un confronto con gli algoritmi classici. Seguiamo due approcci:

- Supervised Learning: gli algoritmi classici generano il dataset per allenare la rete;
- Reinforcement learning: il giocatore basato sulla rete neurale gioca contro se stesso ed impara dalle proprie mosse.

Motore di gioco

Ho implementato in `Python3` il motore di gioco.
Paradigma ad oggetti, uno per ogni componente fondamentale del gioco:

- `Board`: tavola di gioco, contenente tutti i metodi necessari per aggiungere/togliere pedine, calcolare il vincitore e validare le configurazioni di gioco.
- `Game`: classe che gestisce una partita tra due giocatori. Gestisce l'alternanza dei turni e in generale regola il l'ordine con cui vengono chiamati i metodi sugli oggetti `Player`.
- `Player`: è la classe classe base per tutti gli altri giocatori; è puramente astratta.

Classi Extra: `Tournament`

Motore di gioco

Ho implementato in `Python3` il motore di gioco.
Paradigma ad oggetti, uno per ogni componente fondamentale del gioco:

- `Board`: tavola di gioco, contenente tutti i metodi necessari per aggiungere/togliere pedine, calcolare il vincitore e validare le configurazioni di gioco.
- `Game`: classe che gestisce una partita tra due giocatori. Gestisce l'alternanza dei turni e in generale regola il l'ordine con cui vengono chiamati i metodi sugli oggetti `Player`.
- `Player`: è la classe classe base per tutti gli altri giocatori; è puramente astratta.

Classi Extra: `Tournament`

Players

- Basic Players: giocatori senza una vera strategia:
  - `RandomPlayer`: gioca una mossa random tra quelle consentite.
  - `ConsolePlayer`: legge dallo `stdin` le mossa da fare.
  - `FixedMovesPlayer`: gioca una sequenza predeterminata di mosse.
- Search Players: giocatori basati sull'esplorazione dell'albero delle configurazioni. Algoritmo:

$$\text{minimax} + \text{alpha-beta pruning} + \text{euristiche}$$

`PerfectPlayer`

Implementazione in C++ di una ricerca completa[Pon17b; Pon17a]. Incluso nella repository come submodule.

- Combination Players: combinazione di 2 o più giocatori.
- TensorFlow Players: valutazione della tavola con un modello di *TensorFlow*.

Players

- Basic Players: giocatori senza una vera strategia:
  - `RandomPlayer`: gioca una mossa random tra quelle consentite.
  - `ConsolePlayer`: legge dallo `stdin` le mossa da fare.
  - `FixedMovesPlayer`: gioca una sequenza predeterminata di mosse.
- Search Players: giocatori basati sull'esplorazione dell'albero delle configurazioni. Algoritmo:

$$\text{minimax} + \text{alpha-beta pruning} + \text{euristiche}$$

`PerfectPlayer`

Implementazione in C++ di una ricerca completa[Pon17b; Pon17a]. Incluso nella repository come submodule.

- Combination Players: combinazione di 2 o più giocatori.
- TensorFlow Players: valutazione della tavola con un modello di *TensorFlow*.

Players

- Basic Players: giocatori senza una vera strategia:
  - `RandomPlayer`: gioca una mossa random tra quelle consentite.
  - `ConsolePlayer`: legge dallo `stdin` le mossa da fare.
  - `FixedMovesPlayer`: gioca una sequenza predeterminata di mosse.
- Search Players: giocatori basati sull'esplorazione dell'albero delle configurazioni. Algoritmo:

$$\text{minimax} + \text{alpha-beta pruning} + \text{euristiche}$$

`PerfectPlayer`

Implementazione in C++ di una ricerca completa[Pon17b; Pon17a]. Incluso nella repository come submodule.

- Combination Players: combinazione di 2 o più giocatori.
- TensorFlow Players: valutazione della tavola con un modello di *TensorFlow*.

Players

- Basic Players: giocatori senza una vera strategia:
  - `RandomPlayer`: gioca una mossa random tra quelle consentite.
  - `ConsolePlayer`: legge dallo `stdin` le mossa da fare.
  - `FixedMovesPlayer`: gioca una sequenza predeterminata di mosse.
- Search Players: giocatori basati sull'esplorazione dell'albero delle configurazioni. Algoritmo:

$$\text{minimax} + \text{alpha-beta pruning} + \text{euristiche}$$

`PerfectPlayer`

Implementazione in C++ di una ricerca completa[Pon17b; Pon17a]. Incluso nella repository come submodule.

- Combination Players: combinazione di 2 o più giocatori.
- TensorFlow Players: valutazione della tavola con un modello di *TensorFlow*.

Players

- Basic Players: giocatori senza una vera strategia:
  - `RandomPlayer`: gioca una mossa random tra quelle consentite.
  - `ConsolePlayer`: legge dallo `stdin` le mossa da fare.
  - `FixedMovesPlayer`: gioca una sequenza predeterminata di mosse.
- Search Players: giocatori basati sull'esplorazione dell'albero delle configurazioni. Algoritmo:

$$\text{minimax} + \text{alpha-beta pruning} + \text{euristiche}$$

`PerfectPlayer`

Implementazione in C++ di una ricerca completa[Pon17b; Pon17a]. Incluso nella repository come submodule.

- Combination Players: combinazione di 2 o più giocatori.
- TensorFlow Players: valutazione della tavola con un modello di *TensorFlow*.

Extras

- Testing: `PyTest`
- Linting: `Flake8`
- Documentazione: `Sphinx`
- Continuous Integration: `GitHub Workflows`
- Continuous Deployment: `GitHub Pages`

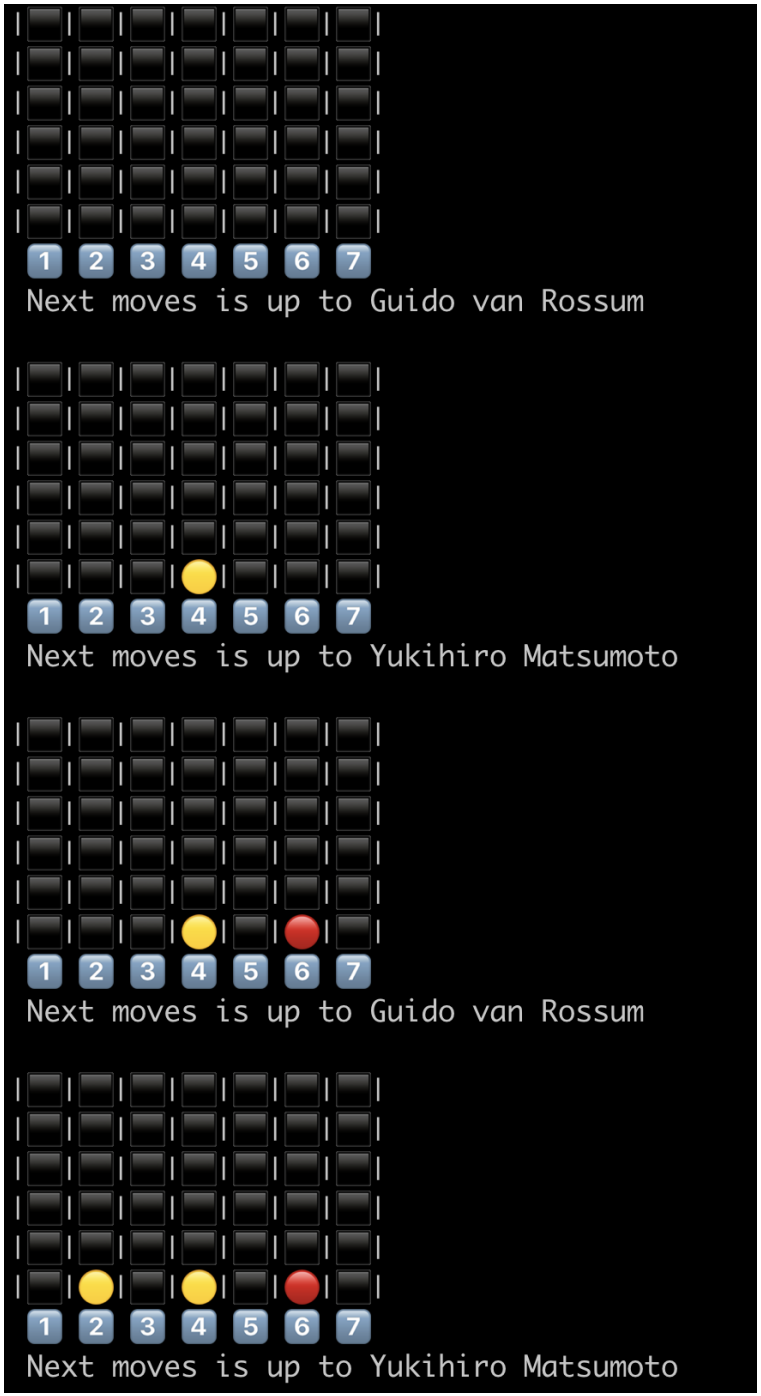
`https://arn4.github.io/connect4/`

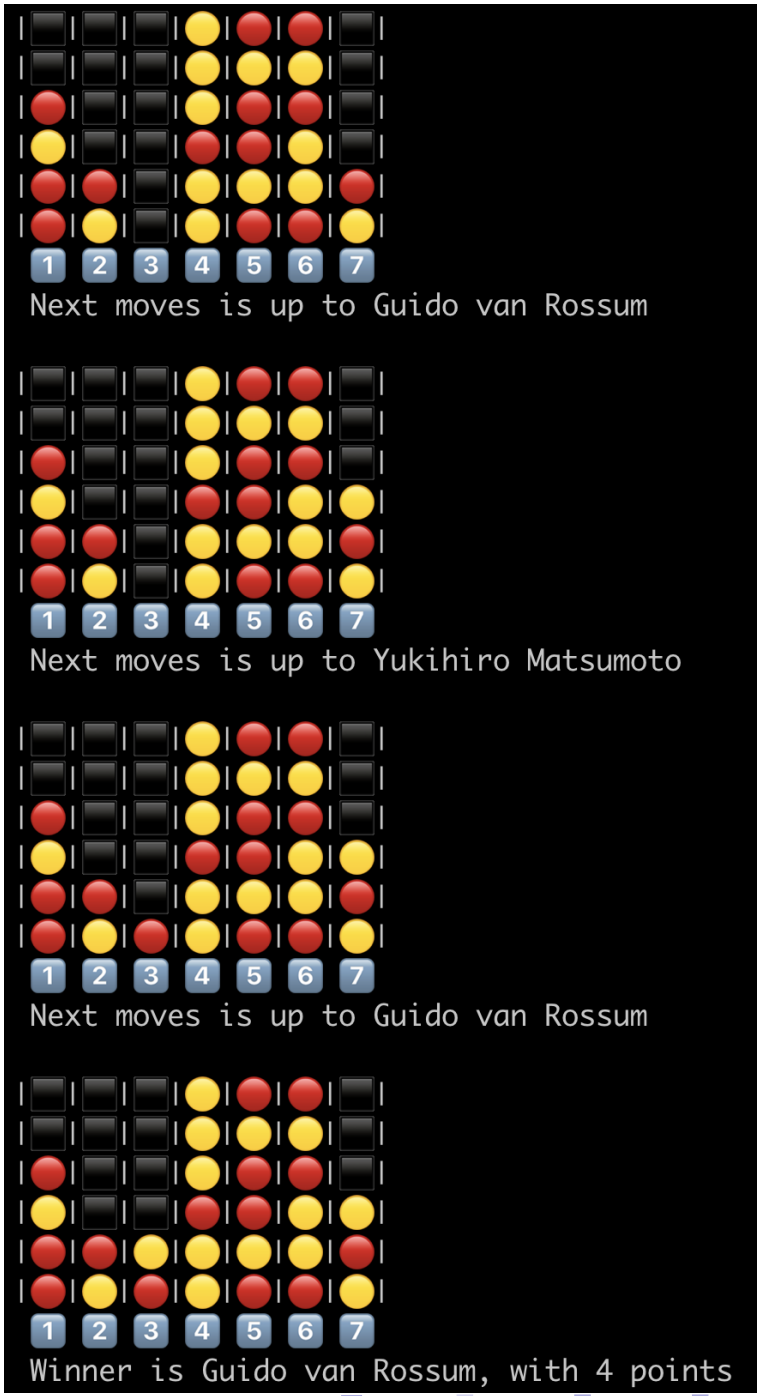
Codice d'esempio

```
from connect4.GameEngine import Game
from connect4.Players import NoisyAlphaBetaPlayer, PerfectPlayer
from connect4.costants import P2

g = Game(
  NoisyAlphaBetaPlayer(depth = 7, noise = 0.5, name = 'Yukihiro Matsumoto'),
  PerfectPlayer('./perfect-player', name = 'Guido van Rossum'),
  starter = P2
)

while not g.finish:
  print(g)
  g.next()
print(g)
```


1 2 3 4 5 6 7
Next moves is up to Guido van Rossum
1 2 3 4 5 6 7
Next moves is up to Yukihiro Matsumoto
1 2 3 4 5 6 7
Next moves is up to Guido van Rossum
1 2 3 4 5 6 7
Next moves is up to Yukihiro Matsumoto


1 2 3 4 5 6 7
Next moves is up to Guido van Rossum
1 2 3 4 5 6 7
Next moves is up to Yukihiro Matsumoto
1 2 3 4 5 6 7
Next moves is up to Guido van Rossum
1 2 3 4 5 6 7
Winner is Guido van Rossum, with 4 points

Bibliografia

[Pon17a] Pascal Pons. *Connect 4 Game Solver*. https://github.com/PascalPons/connect4. 2017.

[Pon17b] Pascal Pons. "Solving Connect 4: How to Build a Perfect AI". In: (2017). URL: http://blog.gamesolver.org/.

[Tro12] John Tromp. *A212693: Number of legal 7 X 6 Connect-Four positions after n plies*. https://oeis.org/A212693. 2012.