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.

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.

Extra: classe `Tournament`, `costants`

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`

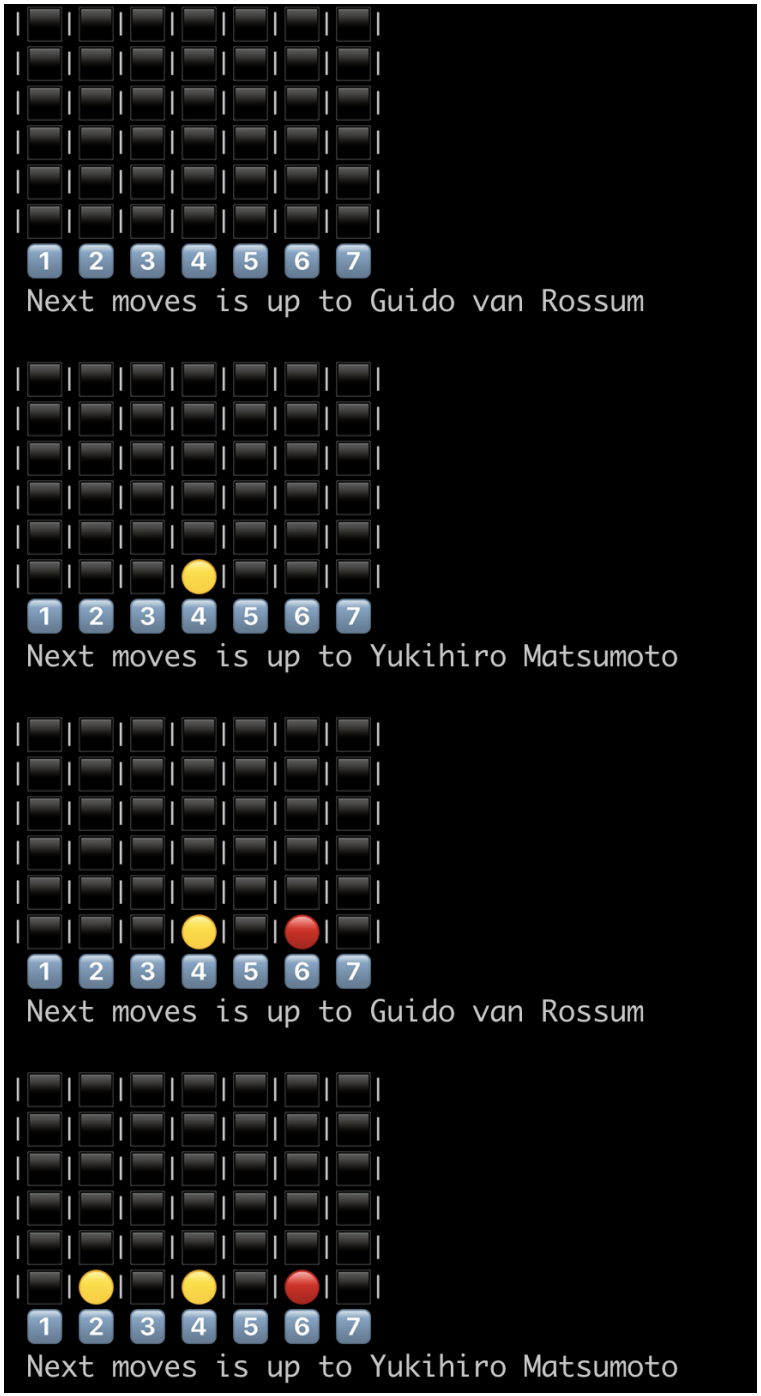
`https://arn4.github.io/connect4/`

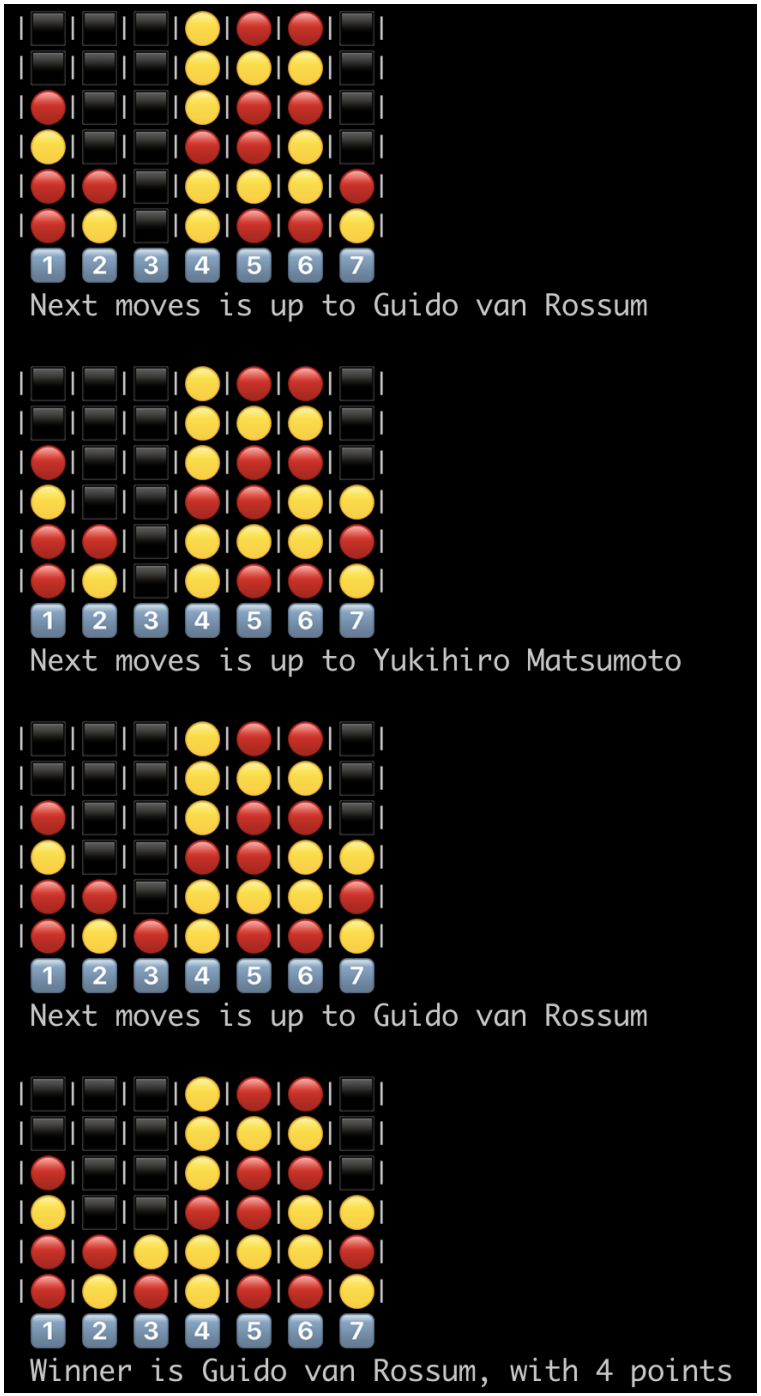
Codice d'esempio

```
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

Supervised Learning: creazione del training set

$$\texttt{GameEngine.Game} \xrightarrow[\text{derivata}]{\text{classe}} \texttt{SupervisedGame}$$

Oltre ad i due giocatori c'è un *supervisor*, che in ogni situazione di gioco dice la mossa che avrebbe fatto.

`DatasetGenerator` gioca in *parallelo* tante partite supervised e ritorna un dataset in formato `numpy.array`

Il dataset che ho generato

Giocatori:

- `RandomPlayer`
- `NoisyAlphaBetaPlayer` con diversi rumori e profondità.

Supervisor:
`CenteredAlphaBeta` con `depth = 6`

42 000 partite

Supervised Learning: topologia della rete

- Il problema ha già una struttura 2D: *convolutional neural network*.
- Kernel quadrati di dimensione 4. Uso *padding valid*.
- I layer che dunque ho usato sono:
  - Convolutional Layer con 100 feature
  - Flatten Layer
  - Dense Layer con 50 neuroni, `ReLU`
  - Dense Layer con 50 neuroni, `ReLU`
  - Dense Layer con 7 neuroni come output. Sono i *logits* delle probabilità che quella sia la miglior mossa.
- Loss function: *Binary Crossentropy*
- Adam Optimizer

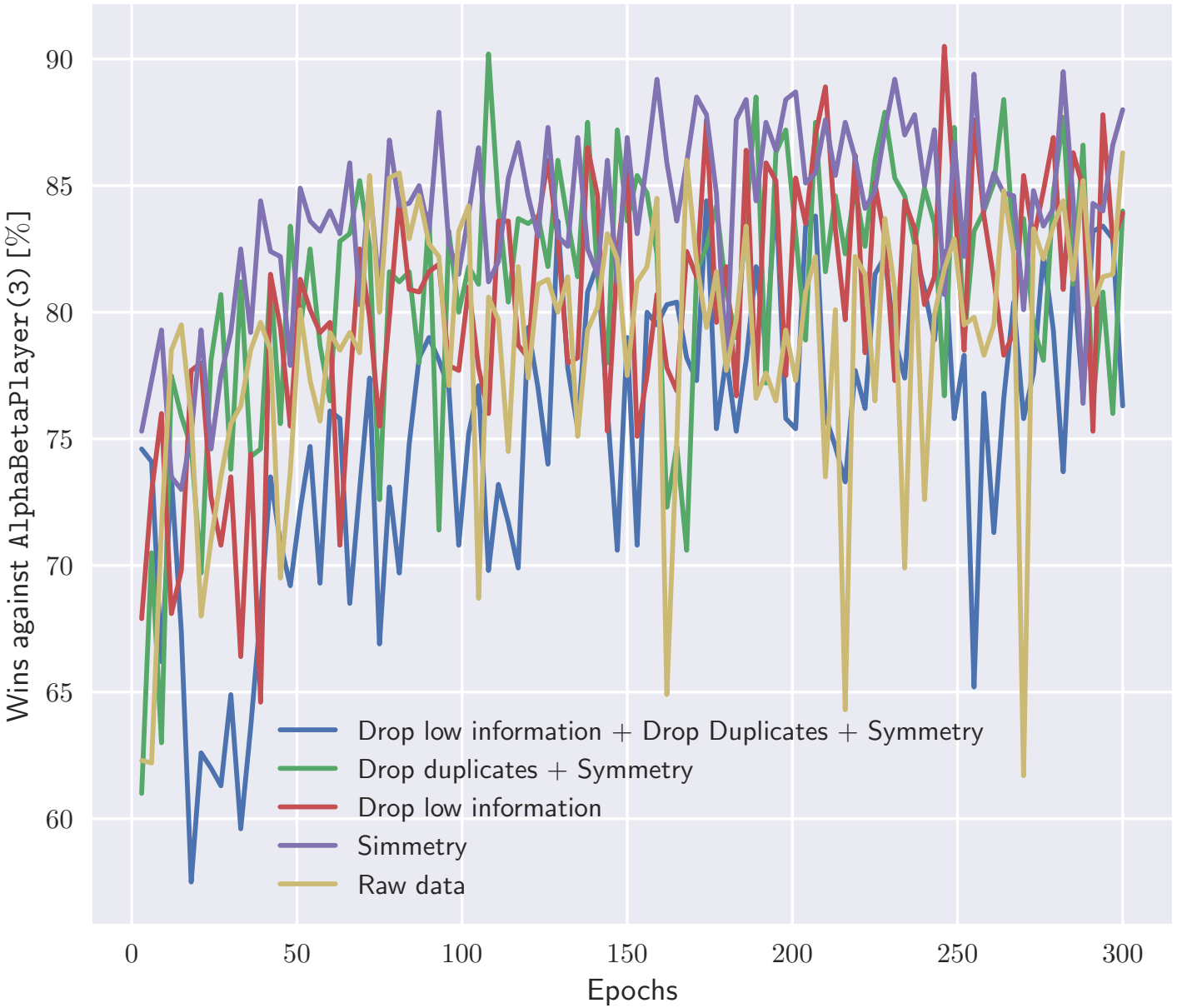
Supervised Learning: pulizia del dataset

Possibili correzioni che si possono applicare al dataset:

- Simmetria longitudinale: la tavola è simmetrica per riflessione rispetto alla colonna centrale.
- Duplicati: alcune configurazioni possono essere duplicate nel dataset, si possono rimuovere.
- Informazione minima: il supervisore potrebbe non essere in grado di fornire informazione sufficiente su alcune configurazioni, che dunque si possono scartare.

Alleno la rete per 300 epoche, con diverse combinazioni delle correzioni.

Supervised Learning: risultati



Supervised Learning: conclusioni

- Usare la simmetria è una buona idea, le altre correzioni invece non lo sono.
- Confronto con il supervisore: 15.4% di vittorie (su 1000 giocate).
  Il supervisore è al livello di un umano, ma la rete è lontana da poter competere con lui.
- Usare un supervisore e dei giocatori più forti sarebbe buona cosa, ma aumenterebbe esponenzialmente il tempo necessario per la generazione del dataset.

Nelle situazione reali non abbiamo possibilità di fare supervised learning.
La rete deve imparare da sola: *reinforcement learning*!

Reinforcement learning: setup

Topologia:

- 1 convolutional layer,
  2 hidden dense;
- Output layer ha un solo neurone:
  lo score della configurazione
- Loss Function: MSE

Punteggi:

Semplificazione del *deep-Q-learning*.
I punteggi vengono dati a posteriori:

- *vittoria*: $+1$
- *sconfitta*: -1
- *pareggio*: 0
- *configurazione intermedie*:
  $r_i = \gamma \cdot r_{i+1}$.

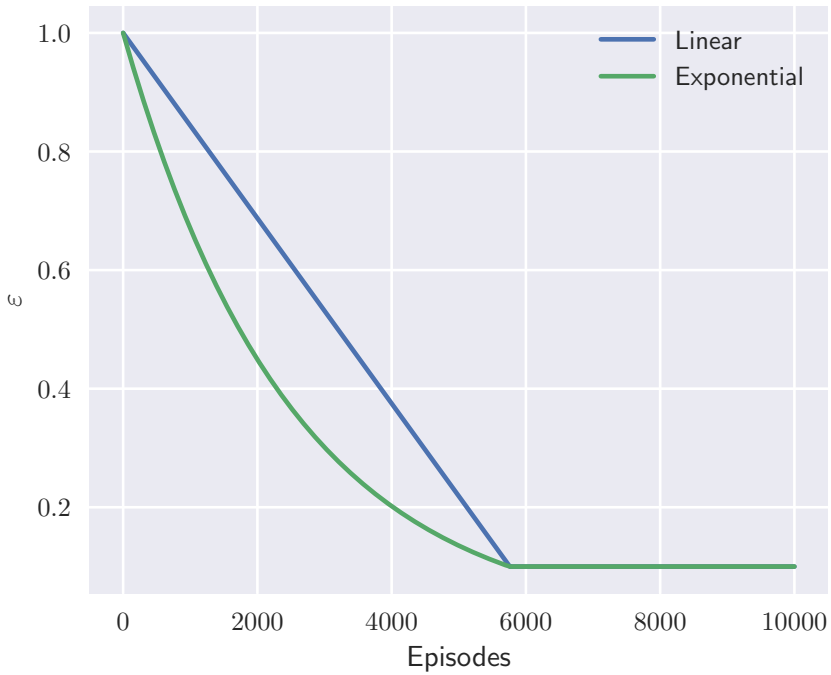
Trade-off Esplorazione-Apprendimento

Uso l'*epsilon-greedy strategy*:

- probabilità ε di fare una mossa random invece che quella scelta dalla rete.
- ε decresce allo scorrere degli episodi.

Reinforcement learning: altre idee

- Decadimento lineare: Il decadimento di ε è solitamente scelto esponenziale. Trasformarlo in lineare potrebbe beneficiare.
- Flattened NN: potrebbe essere che non usare il convolutional layer funzioni meglio.
- Multi-Traning: allenando un agent singolo potrebbe succedere che si adatti a se stesso per avere buoni punteggi. Se alleno un gruppo invece questo non dovrebbe succedere.



Reinforcement learning: risultati preliminari

Alleno per 10 000 episodi, per testare le idee

Reinforcement learning: risultati

Alleno per 3 agent in simultanea, per 50 000 episodi

Reinforcement learning: conclusioni

- Prendendo la miglior performance di ognuno dei 3 agent, e combinandola in un `PoolPlayer` si ottiene 57.0 % contro `AlphaBetaPlayer(3)`.
- La performance decresce dopo circa 10 000 episodi: è una manifestazione del *catastrophic forgetting*[Kir+17]

Idea per il futuro!

Usare la rete neurale combinata con il minimax per avere sia la capacità di vedere il futuro che di valutare una configurazione della tavola da gioco.

Bibliografia

[Kir+17] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[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.