Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Ottimizzazione Combinatorica mediante Deep Reinforcement Learning: Sperimentazione nella Logistica di Magazzino.

Relatore:
Prof.
MORO GIANLUCA

Co-relatore:
Prof.
MINGOZZI ARISTIDE

Co-relatore:
Prof.
PASOLINI ROBERTO

Presentata da:
MAGNINI MATTEO

Sessione di Dicembre
2017 - 2018

Il perder tempo a chi più sa più spiace
(Dante, Purgatorio III,78)

Introduzione

Il Deep Reinforcement Learning acquista sempre più importanza tra gli algoritmi di apprendimento dopo i brillanti risultati ottenuti da DeepMind con AlphaGo e AlphaZero, rispettivamente riuscendo a battere il campione mondiale di Go (2015) [3] e il chess engine Stockfish (2017) [4]. Negli ultimi anni sono stati sviluppati maggiormente programmi in grado di giocare a numerosi giochi Atari e, nell'ambito della robotica, di ottenere ottimi risultati nell'autoapprendimento di specifici comportamenti [1, 2]. Sebbene questa nuova tecnologia sia stata applicata con successo in questi ambiti, sono scarse le ricerche inerenti l'utilizzo del Deep Reinforcement Learning su problemi di ottimizzazione combinatorica. Questa tesi si pone l'obiettivo di esplorare una possibile soluzione al problema reale dell'allocazione di prodotti in un magazzino, confrontando i risultati ottenuti con la Ricerca Operativa e con l'allocazione dell'azienda presa in considerazione per comprenderne la bontà. Nel corso di questo lavoro verrà dapprima introdotto il Reinforcement Learning in generale e in particolare il Q-learning. Successivamente verrà mostrato il Deep Reinforcement Learning, con un esempio applicandolo al gioco Catch. Infine sarà presentato il progetto e i risultati ottenuti del Deep Reinforcement Learning applicato a varie istanze del problema di allocazione di prodotti in magazzino.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Reinforcement Learning

1.1 Caratteristiche generali

Il Reinforcement Learning (RE) è un algoritmo di apprendimento automatico che rientra nel grande dominio del Machine Learning (ML). Si differenzia dagli altri algoritmi di ML principalmente per i seguenti punti:

- non necessita di un training set di dati;
- l'iterazione non avviene quindi direttamente con i dati, bensì con l'**ambiente**;
- l'**agente** interagisce con l'ambiente al fine di raggiungere un obiettivo.

L'**ambiente** è tipicamente uno scenario del mondo reale, simulato in 2 o 3 dimensioni, oppure uno scenario game-based (scacchi, go, labirinti, ecc.). Esso fornisce una rappresentazione dello stato del mondo rappresentato per un determinato istante temporale. Ad esempio, per quel che riguarda il gioco degli scacchi, lo stato è derivato dalla posizione dei pezzi sulla scacchiera 8x8 e dall'insieme delle possibili mosse legali che si possono effettuare in quel determinato turno. L'ambiente è in grado di ricevere un input dall'esterno che viene interpretato di volta in volta come un'azione in grado di modificare lo stato interno.
L'**agente** è un'entità intelligente che interagisce con l'ambiente compiendo delle azioni su di esso. Ogni volta che viene effettuata un'azione, l'ambiente reagisce fornendo all'agente

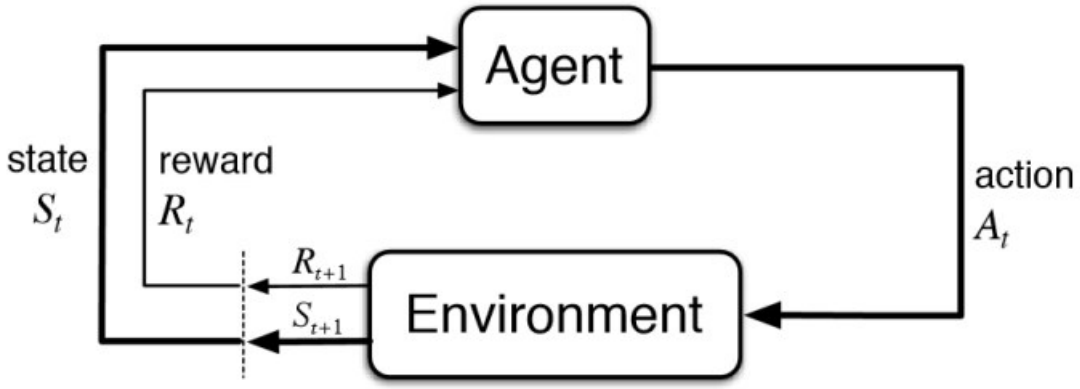

Figura 1.1: Interazione agente-ambiente

il nuovo stato che è stato raggiunto e un punteggio (oppure ricompensa o rinforzo). Il punteggio è un valore numerico che qualifica la bontà dell'azione appena compiuta nello stato passato.
Ogni qual volta l'ambiente riceve un'azione dall'agente, questa viene interpretata all'interno dell'ambiente stesso mediante delle politiche che non sono note al di fuori di esso. Compito dell'agente è quello di apprendere quali siano le azioni necessarie per giungere da uno stato iniziale ad uno finale massimizzando nel complesso tutte le ricompense ricevute.

1.2 Descrizione formale

Il RL è modellato tramite il **Processo Decisionale di Markov** (MDP). Un MDP è un modello matematico per la modellazione del processo decisionale in situazioni in cui i risultati sono in parte casuali e in parte sotto il controllo decisionale. In particolare un MDP è un processo di controllo stocastico a tempo discreto. Se gli spazi degli stati e delle azioni sono finiti, allora il problema è chiamato MDP finito. Un MDP è definito da una tupla (S,A,P_a,R_a,γ):

- S è l'insieme (finito) degli stati;
- A è l'insieme (finito) delle azioni;

- $P_a(\text{s},\text{s}^1) = P(s_{t+1} = s^1 | s_t = s, a_t = a)$ è la probabilità che un'azione a performata nello stato s conduca allo stato s^1 (probabilità di transizione);
- $R_a(\text{s},\text{s}^1)$ è una funzione che restituisce il valore atteso della ricompensa dovuta alla transizione dallo stato s ad s_1;
- γ è il fattore di sconto ($\gamma \in [0, 1]$) che indica quanto considerare le ricompense future rispetto a quelle immediate.

Come precedentemente annunciato, l'agente deve massimizzare la ricompensa totale, la quale può essere espressa come la somma di ogni ricompensa ottenuta al tempo t:

$$\sum_{i=0}^{N} r_i$$

Ciò si ottiene massimizzando la somma dei valori attesi delle ricompense per ogni singola azione al tempo t:

$$R_t = \sum_{i=t}^{N} r_i = r_t + r_{t+1} + \cdots + r_n$$

Predire la ricompensa attesa tuttavia non è un compito semplice, più si osserva negli stati futuri e più diventa difficile. Per questo motivo è più facile per l'agente considerare le ricompense future attese scontate di un fattore γ:

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \\ &= r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \dots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

Per $\gamma = 1$ l'agente è completamente deterministico, per $\gamma = 0$ invece non vengono considerate le ricompense future. Sperimentalmente un buon valore di γ si aggira intorno a 0.9 nella maggior parte dei problemi.
Il problema centrale di un MDP è trovare una politica per scegliere un'azione tra quelle possibili per il determinato stato in cui si trova al momento l'ambiente. Questa politica π è quindi una funzione da S ad A che restituisce l'azione migliore, ovvero quella a cui è associata la ricompensa attesa maggiore.

1.3 Q-learning

Il Q-learning è uno degli algoritmi di apprendimento per rinforzo di maggiore utilizzo, usato per imparare una politica per la scelta delle azioni. A livello astratto il Q-learning permette all'agente di adattarsi all'ambiente in cui opera, migliorando la scelta delle azioni da compiere per ogni epoca di addestramento. Il Q-learning massimizza il valore della Q-function che rappresenta la massima ricompensa futura scontata quando viene effettuata l'azione a nello stato s. La Q-function esprime la qualità della combinazione stato-azione:

$$Q : SxA \longrightarrow R$$
$$Q(s_t, a_t) = Max(R_{t+1})$$

Il Q-learning è model-free, ovvero l'agente prova a sviluppare una funzione di controllo che guardando lo stato decide qual è l'azione migliore da prendere. Ciò è il contrario del model-based nel quale l'agente prova ad imparare il modello probabilistico (MDP) e determinare in base a quello l'azione migliore. Detto Q-value il valore della funzione Q, l'azione ottima risulta quella con Q-value maggiore, quindi è implementata una politica per il RL:

$$\pi(s) = argmax_a Q(s, a)$$

Per un punto di transizione (s_t, a_t, r_t, s_{t+1}) si può ridefinire la Q-function tramite la Q-funcion al punto successivo $(s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2})$ similmente per quanto fatto con la ricompensa:

$$Q(s_t, a_t) = r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Nell'implementazione della funzione Q si fa tipicamente uso di una lookup table, denominata Q-table, nella quale le righe corrispondono agli stati S, le colonne alle azioni A e gli elementi le ricompense $Q(s, a)$. La migliore azione per ogni stato è quella col valore più alto lungo la riga. Inizialmente la Q-table è inizializzata con zero per ogni cella, infatti l'agente non ha alcuna conoscenza delle conseguenze che le azioni avrebbero sull'ambiente. Ad ogni azione presa viene aggiornata secondo quanto segue:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Dove γ è il fattore di sconto precedentemente introdotto e α rappresenta il tasso di apprendimento ($\alpha \in [0, 1]$). Con tasso 0 la tabella non viene aggiornata, con tasso 1 il vecchio valore non viene tenuto in considerazione per l'aggiornamento. Valori di α comuni sono 0,1 o 0,01. Tale funzione è nota come **equazione di Bellman**

	a_0	a_1	...	a_m
S_0	0	0	...	0
S_1	0	0	...	0
...	...	...	...	...
S_n	0	0	...	0

Tabella 1.1: Inizializzazione Q-table

	a_0	a_1	...	a_m
S_0	0	0.15	...	−0.27
S_1	0.84	0.05	...	−0.52
...	...	...	...	...
S_n	0	−0.13	...	0

Tabella 1.2: Q-table dopo t iterazioni

1.3.1 Esplorazione

All'inizio dell'addestramento l'agente deve esplorare le varie azioni per comprendere quali possono risultare positive e quali no, ma non ha idea di quali effettuare. Inoltre, affinché si giunga ad una convergenza, è necessario che tutti gli stati vengano continuamente aggiornati. Per questo motivo è necessario introdurre un metodo di esplorazione come ad esempio ϵ-greedy. ϵ-greedy permette di scegliere l'azione di uno stato secondo la seguente legge:

$$\epsilon - greedy = \begin{cases} random, & \text{if } p < \epsilon \\ argmax_a(s_t, a), & \text{otherwise} \end{cases} \tag{1.1}$$

Con probabilità ϵ viene effettuata un'azione casuale, altrimenti viene scelta quella indicata dalla Q-table. Poiché con il passare delle epoche di addestramento la Q-table risulterà più accurata, è comune usare un valore di ϵ che decresce nel tempo. Ad esempio si può scegliere come valore iniziale 0.1 e decrescere fino a 0.0001.

Algorithm 1 Q-learning

$\epsilon \leftarrow initialize$
$\gamma \leftarrow initialize$
$\alpha \leftarrow initialize$
$epoches \leftarrow initialize$
$Q \leftarrow initialize$ ▷ Q è la q-table
$environment \leftarrow initialize$ ▷ Un ambiente con cui interagire
$s_t \leftarrow initialize$ ▷ Stato iniziale

function $choose\_action(s_t)$
    if $uniform\_random(0, 1) < \epsilon$ **then return** $random(0, num\_action)$
    elsereturn $argmax_a(Q[s_t, :])$

while $epoches > 0$ **do**
    $game\_over \leftarrow True$
    while $game\_over$ **do**
        $a_t \leftarrow chooseAction(s_t)$
        $r_t, s_{t+1}, gameOver \leftarrow environment.Step(a_t)$
            ▷ Data un'azione torna ricompensa, stato e se si ha terminato
        $Q[s_t, a_t] \leftarrow Q[s_t, a_t] + \alpha(r_t + \gamma max_a Q[s_{t+1}, :] - Q[s_t, a_t])$
    $epoches \leftarrow epoches - 1$

Capitolo 2

Deep Reinforcement Learning

Il Q-learning ha dei limiti significativi. In primo luogo se ad un agente è chiesto di effettuare un'azione in uno stato di cui non ha avuto esperienza non sa quale scegliere, poiché i valori della riga associata a quello stato sono equivalenti. In secondo luogo, all'aumentare della dimensionalità dell'ambiente, la Q-table si ingrandisce, spesso fino ad un punto tale in cui richiederebbe troppo tempo l'utilizzo del Q-learning per l'apprendimento o troppo spazio in memoria. Ad esempio, nel capitolo precedente la Q-table aveva 5x5 righe e 4 colonne, per un totale di 100 celle. Tuttavia se lo spazio del labirinto aumentasse a 50x50 e ci si potesse muovere verso tutte le 8 celle adiacenti si avrebbero 20000 celle. Per questi motivi nel Deep Reinforcement Learning (DRL) si utilizza come Q-function una rete neurale (NN), che viene addestrata al fine di approssimare tale funzione. Grazie all'utilizzo combinato di apprendimento per rinforzo e rete neurale si riescono ad affrontare problemi molto complessi, quali ottenere giocatori di Go e Scacchi in grado di superare gli attuali campioni del mondo (umani e motori "classici"). IL DRL ha avuto molto successo anche nella risoluzione di giochi Atari a partire dalle immagini grezze di output del gioco stesso tramite l'utilizzo di reti profonde con strati convoluzionali.

2.1 Implementazione

La strategia del DRL è quella di addestrare una rete neurale al fine di predire l'azione che conduce al migliore stato futuro dato lo stato presente. Sperimentalmente, questo tende a condurre la NN ad un minimo locale. Per impedire che ciò accada, durante l'esecuzione dell'addestramento, vengono collezionate le transizioni effettuate, ovvero tuple del tipo (s_t, a, r, s_{t+1}) (stato attuale, azione effettuata, ricompensa, stato futuro). Questa esperienza che viene accumulata nel tempo è detta **replay memory**. Quando si addestra la rete vengono estratte casualmente delle tuple dalla replay memory al posto di selezionare solamente le tuple più recenti. Poiché le batches sono composte da tuple (s_t, a, r, s_{t+1}) casuali prive di ordine la NN si addestra meglio e non resta bloccata in minimi locali.
Durante le epoche di addestramento, ogni qual volta che l'agente sceglie un'azione viene aggiornata la replay memory e viene addestra la NN. L'addestramento procede estraendo dalla replay memory un batch casuale di input X e di di output Y, modificando quest'ultima secondo il seguente pseudocodice:

Il batch di input rimane quello salvato in memoria, mentre quello di output è generato dalla NN. Di quest'ultimo viene modificato l'elemento di indice corrispondente all'azione che era stata intrapresa precedentemente, ciò viene fatto per ogni tupla del batch Y. La modifica avviene secondo l'equazione di Bellman descritta nel capitolo precedente. Una volta estratti i batches la rete viene addestrata su di essi aggiornando i propri pesi interni tramite back propagation.
Inizialmente la replay memory è vuota, la NN non ha ancora nessuna esperienza riguardo le azioni. Prima di iniziare l'addestramento effettivo della NN si effettuano delle epoche di esplorazione nelle quali vengono accumulate le tuple rappresentanti le transizioni eseguendo delle azioni casuali. Solo dopo queste epoche iniziano quelle di addestramento. Inoltre, come nel caso del RL classico, si utilizza l'iperparametro ϵ per continuare durante l'addestramento l'esplorazione stocastica degli stati.

Poiché ad ogni epoca, escluse quelle di osservazione, avvengono alternativamente predizione e addestramento, il DRL è un esempio di apprendimento online. A differenza delle altre tecniche di apprendimento per batch dove il modello è addestrato sull'intero training set di dati, con l'apprendimento online il modello migliora continuamente mano

Algorithm 2 Replay Memory

1: **function** $get\_next\_batch(rm, model, num\_action, gamma, batch\_size)$
2: $batch\_indices \leftarrow random(0, len(rm), batch\_size)$
3: $batch \leftarrow rm[batch\_indices]$
4: $X \leftarrow initialize$ ▷ dimensioni dell'input della rete
5: $Y \leftarrow initialize$ ▷ dimensioni dell'output della rete
6: **for** i in range(len(batch)) **do**
7: ▷ si tiene anche traccia se il gioco è terminato
8: $s_t, a, r, s_{t+1}, game\_over \leftarrow batch[i]$
9: $X[i] \leftarrow s_t$
10: $Y[i] \leftarrow model.prdict(s_t)$
11: $Q\_sa \leftarrow max(model.predict(s_{t+1}))$ ▷ massima ricompensa attesa
12: **if** $game\_over$ **then**
13: $Y[i, a] \leftarrow r$
14: **else**
15: $Y[i, a] \leftarrow r + gamma * Q\_sa$
16: $return \quad X, Y$

Algorithm 3 Ciclo DRL

```
1: for e in range(num_epochs) do
2:     game.reset()
3:     a ← random(0, num_actions)
4:     s_t, r, game_over ← game.step(a)
5:     while not game_over do
6:         s_{t-1} ← s_t
7:         if epsilon ¡= num_epochs_observe then
8:             a ← random(0, num_actions)
9:         else
10:            if uniform_random(0, 1) ≤ ϵ then
11:                a ← random(0, num_actions)
12:            else
13:                a ← argmax(model.predict(s_t))
14:        s_t, r, game_over ← game.step(a)
15:        rm.append((s_{t-1}, a, r, s_t, game_over))
16:        if e > num_epochs_observe then
17:            X, Y ← get_next_batch(rm, model, num_action, γ, batch_size)
18:            model.train_on_batch(X, Y)
```

a mano che si addestra su nuovi dati.

2.2 Applicazioni

Di seguito sono mostrati i risultati ottenuti addestrando una rete neurale convoluzionale, tramite l'algoritmo descritto precedentemente, al gioco Catch. Il gioco consiste semplicemente nel catturare una palla che scende dall'alto dello schermo fino alla base tramite una barra che è possibile muovere a sinistra e a destra, oppure non muoverla. Il codice del programma, ripreso dagli autori di *Deep Learning with Keras* [5], è stato scritto in Python 3.6 con l'utilizzo di librerie scientifiche esterne (appendice A).

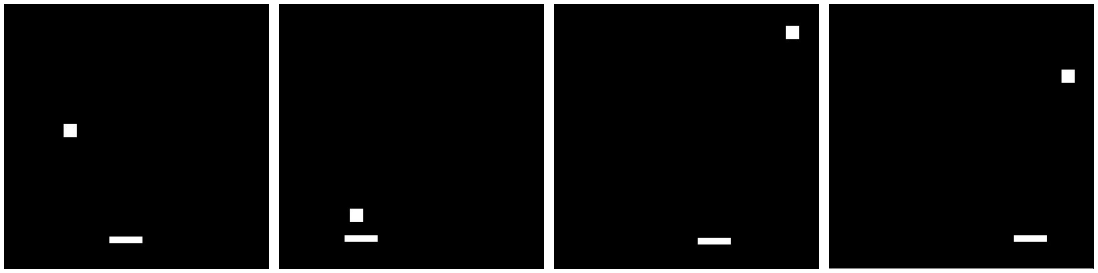

Figura 2.1: Frame del gioco Catch

La NN è stata addestrata su 5000 epoche (100 iniziali aggiuntive di esplorazione), ognuna equivalente ad una partita. Come iperparametri sono stati utilizzati: batch size 32, γ 0.99, ϵ iniziale 0.1, ϵ finale 0.0001, learning rate 10^{-6}. La NN ha come input quattro frame di dimensione 80 x 80 raffiguranti lo stato presente e i tre precedenti stati del gioco, in totale la rete ha 2560 valori di input. In output la NN ha tre neuroni, uno per possibile azione. La rete è composta da uno strato convoluzionale di input, da due strati convoluzionali e da uno strato denso come livelli nascosti, e da uno strato denso come output. Come funzioni di attivazione si è usata la funzione ReLu, ottimizzatore Adam, loss mse. L'ambiente, per ogni azione che riceve dalla rete, restituisce i quattro frame più recenti e la ricompensa. Se la palla non è arrivata alla base viene restituito 0, se la palla viene intercettata +1, se viene mancata -1.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 20, 20, 32)        8224
_________________________________________________________________
activation_1 (Activation)    (None, 20, 20, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 64)        32832
_________________________________________________________________
activation_2 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3277312
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1539
=================================================================
Total params: 3,356,835
Trainable params: 3,356,835
Non-trainable params: 0
_________________________________________________________________
```

Figura 2.2: Morfologia NN Catch

Al termine dell'addestramento la NN è stata fatta giocare in autonomia mantenendo i valori dei parametri invariati. Su 100 partite è stata in grado di catturare la palla 93 volte. Visto l'andamento crescente della curva delle vittorie, con un addestramento più lungo, ad esempio 8000 o 10000 epoche, è certamente possibile migliorare le prestazioni fino a risolvere il gioco.

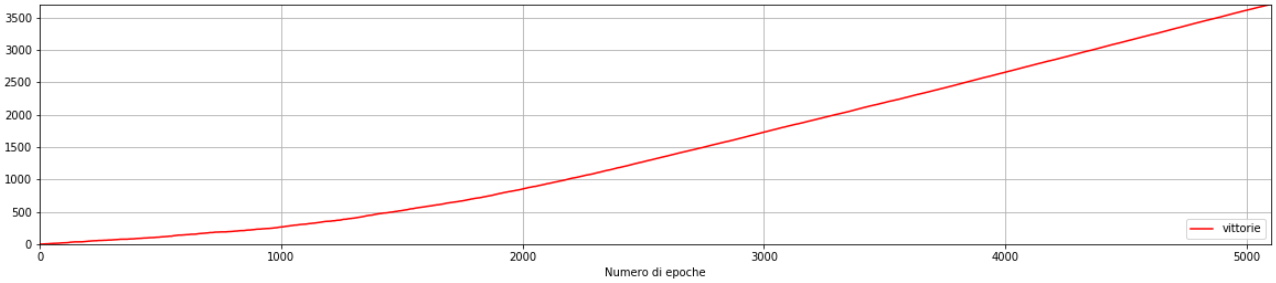

Figura 2.3: Vittorie accumulate Catch

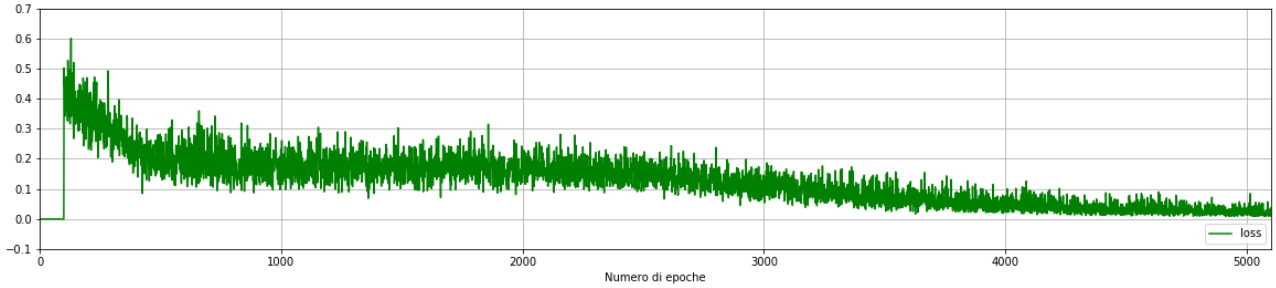

Figura 2.4: Andamento della loss Catch

Capitolo 3

Deep Reinforcement Learning applicato al PAP

Il problema dell'allocazione di prodotti (PAP) consiste nel trovare l'allocazione migliore dei prodotti nelle locazioni di un magazzino secondo una certa funzione di costo da minimizzare. Nel caso in esame si vuole minimizzare il tempo di prelievo, quindi la distanza, dei prodotti quando viene effettuato un ordine. Se il numero di prodotti è uguale al numero di locazioni i vincoli che sussistono sono i seguenti: un prodotto deve essere collocato in un'unica locazione, una locazione contiene esattamente un prodotto. Nella pratica possono esserci altri fattori da minimizzare, oppure il problema potrebbe essere soggetto a numerosi vincoli aggiuntivi [10, 11]. Nel corso di questo capitolo i dati utilizzati per l'applicazione del DRL sono stati forniti da un'azienda del campo medicale e farmaceutico, nel seguito ci si riferirà ad essa genericamente col termine di azienda.
PAP è un problema NP-hard, la cui complessità è estremamente elevata anche per un numero limitato di prodotti e locazioni. Sia PAP(i) un'istanza del problema con esattamente i prodotti ed i locazioni, il numero di possibili allocazioni differenti che si possono effettuare sono i fattoriale. Per questo motivo non è possibile applicare soluzioni che si basano sulla forza bruta o sulla ricerca operativa per un i grande per ottenere l'ottimo esatto. Nella pratica si utilizzano delle euristiche, come l'Analisi ABC basata sulla legge di Pareto [8, 9], utilizzata dall'azienda.

3.1 Analisi

3.1.1 PAP

L'obiettivo ultimo è quello di affrontare il problema del PAP modellato nel paragrafo precedente attraverso il DRL. Per fare ciò è necessario definire ogni componente del modello in modo corretto.
Il magazzino di un PAP contiene al suo interno un numero n di locazioni, tipicamente disposte lungo la sua lunghezza e separate da corridoi percorribili dagli addetti al prelievo dei prodotti. Per semplicità si suppone che ci sia un unico punto di ingresso e uscita dal magazzino. Ogni locazione ha una data distanza tra le altre e tra il punto di entrata. Tali distanze sono i valori delle celle della matrice delle distanze d usata nelle formule precedenti.
Il magazzino può essere rappresentato da un piano bidimensionale nel quale sono presenti zone di diverso tipo: le locazioni, l'entrata, i corridoi ed eventuali pareti. Sia $Z = \{z_{1,1}, z_{1,2}, \ldots, z_{1,y}, \ldots, z_{2,1}, z_{2,2}, \ldots, z_{x,y}\}$ l'insieme delle zone e x, y le dimensioni spaziali. Per poter utilizzare le distanze tra le locazioni occorre definire una metrica. Per semplicità si sceglie la metrica cityblock, nota anche come Manhattan. Usando questa regola tutte le zone considerate a distanza unitaria da una zona $z_{x,y}$ sono: $z_{x+1,y}$, $z_{x-1,y}$, $z_{x,y+1}$ e $z_{x,y-1}$.
Per ogni locazione è allocato un prodotto, identificato univocamente dal codice aziendale. La funzione di assegnamento f può essere interpretata come una matrice prodotti per locazioni le cui celle hanno valore 1 se il prodotto i-esimo è in locazione j-esima, 0 altrimenti. Ne risulta una matrice sparsa la cui somma degli elementi è uguale a n, inoltre, per ogni riga e colonna c'è un unico 1.
Dallo storico degli acquisti è possibile costruire la matrice dei pesi w prodotto per prodotto in modo che nella cella $w_{i,j}$ sia presente la frequenza delle transazioni di acquisto in cui il prodotto i e il prodotto j compaiono assieme. Sempre da questi dati è possibile calcolare il costo complessivo di prelievo per tutte le transazioni. Tale storico è un file in cui ogni riga contiene il codice della transazione e il codice del prodotto coinvolto oltre ad altre informazioni di supporto. Allora basta collassare le righe per codice di transazione, ottenere tutti i prodotti coinvolti nell'operazione e calcolarne la distanza

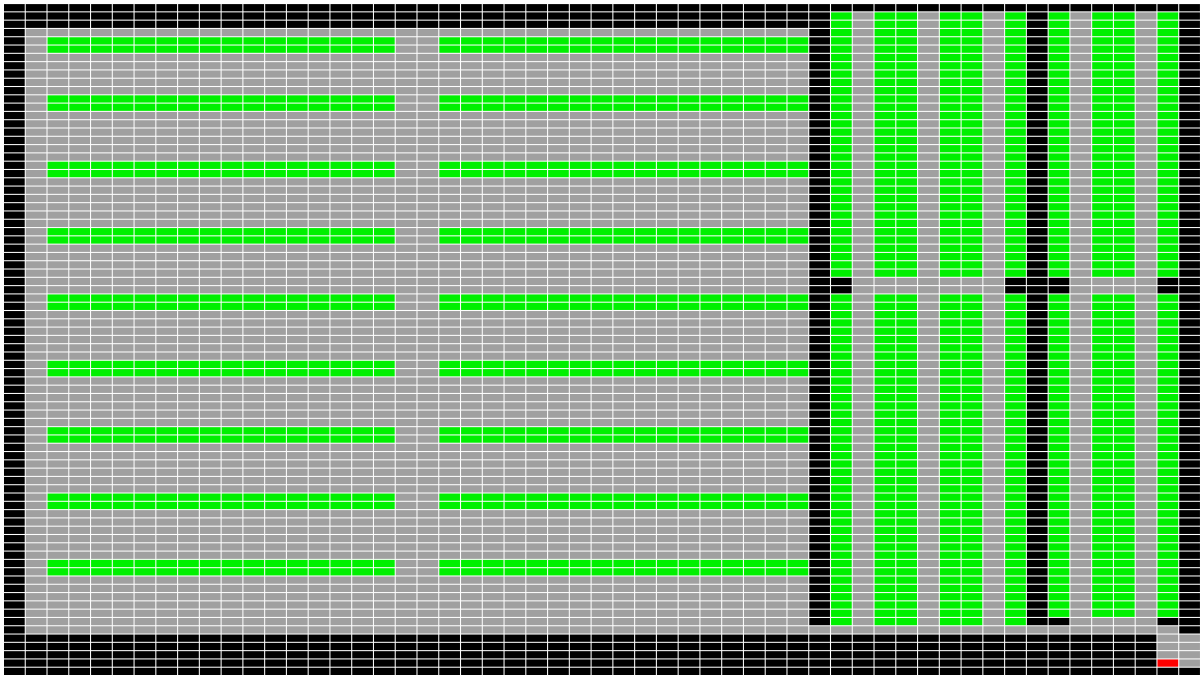

Figura 3.1: Planimetria magazzino aziendale

tramite l'algoritmo del commesso viaggiatore (TSP). La somma di tutti i valori ottenuti determina il costo complessivo di prelievo.
Per gli esperimenti che verranno mostrati nel seguito di questo capitolo, è stata utilizzata la pianta di un magazzino reale con un sottoinsieme di prodotti e storico acquisto dell'azienda.

3.1.2 DRL

Dal punto di vista del DRL il magazzino funge da ambiente. Lo stato del magazzino ad un determinato istante temporale è determinato univocamente dall'allocazione dei prodotti nelle locazioni. Pertanto per transitare da uno stato iniziale s_t ad s_{t+1} è necessario che avvenga uno scambio di posizione tra due prodotti. Sia $S = \{s_1, \ldots, s_k\}$ l'insieme degli stati dell'ambiente. Le azioni che l'ambiente riceve sono dunque una coppia di indici rappresentanti i prodotti da scambiare tra loro. $A = \{a_{1,1}, a_{1,2}, \ldots, a_{1,n}, a_{2,1}, \ldots, a_{n,n}\}$ è l'insieme delle azioni, da notare che le azioni $a_{i,j}$ e $a_{j,i}$ portano allo stesso stato per ogni i e j.

La ricompensa da restituire deve essere un valore che rappresenti la bontà di uno scambio, ovvero se, a scambio avvenuto, il costo di prelievo delle transazioni di acquisto è diminuito o meno. Un esempio potrebbe essere la percentuale di scostamento del costo complessivo di prelievo attuale rispetto a quello dello stato precedente: $r = (1 - C(s_{t+1})/C(s_t)) * 100$, oppure la differenza (pesata) tra il costo dello stato precedente con quello dello stato attuale. Dove s è la funzione $C : S \rightarrow R$ che dato uno stato restituisce un reale rappresentante il costo associato.
L'agente sceglie un'azione da compiere secondo l'algoritmo del DRL, ovvero passa all'ambiente una coppia di indici di prodotti da scambiare. Come risposta riceve la nuova allocazione e la ricompensa. La q-function nel DRL è una rete neurale (NN) profonda la quale ha per input una particolare struttura dati che rappresenta lo stato del magazzino e per output un vettore, o una matrice, usato per determinare i prodotti da scambiare.

3.2 Progettazione

Di seguito sono descritti lo schema, le principali classi, strutture dati, algoritmi e metodi utilizzati per affrontare il problema. L'implementazione è stata effettuata utilizzando come linguaggio di programmazione Python 3.6, con l'ausilio di librerie scientifiche esterne (appendice A).

3.2.1 Schema

Per una chiara implementazione del DRL è necessario scomporre le varie parti in classi in base alle funzionalità. Le principali sono la classe rappresentante il magazzino/ambiente e la classe dell'agente; sono presenti altre classi ausiliare minori.

- Warehouse: contiene la logica del magazzino, al suo interno sono presenti strutture dati per la memorizzazione della planimetria, dei prodotti e delle relazioni che sussistono tra prodotti e locazioni. Ha funzionalità di caricamento dei dati da file, salvataggio e interazione con l'agente;

- Agent: è composto da una NN che funge da Q-function e da una struttura dati per il salvataggio dell'esperienza accumulata durante il training. Ha una funzione per l'addestramento della rete che interagisce con la classe Warehouse;
- Zone: rappresenta il concetto di zona di un magazzino, contiene quindi i dati caratterizzanti dell'elemento quali coordinate, tipologia, eventuali locazioni presenti, ecc;
- Graph: classe di utilità per la creazione di grafi e per il calcolo delle distanze tramite l'algoritmo di Dijsktra;

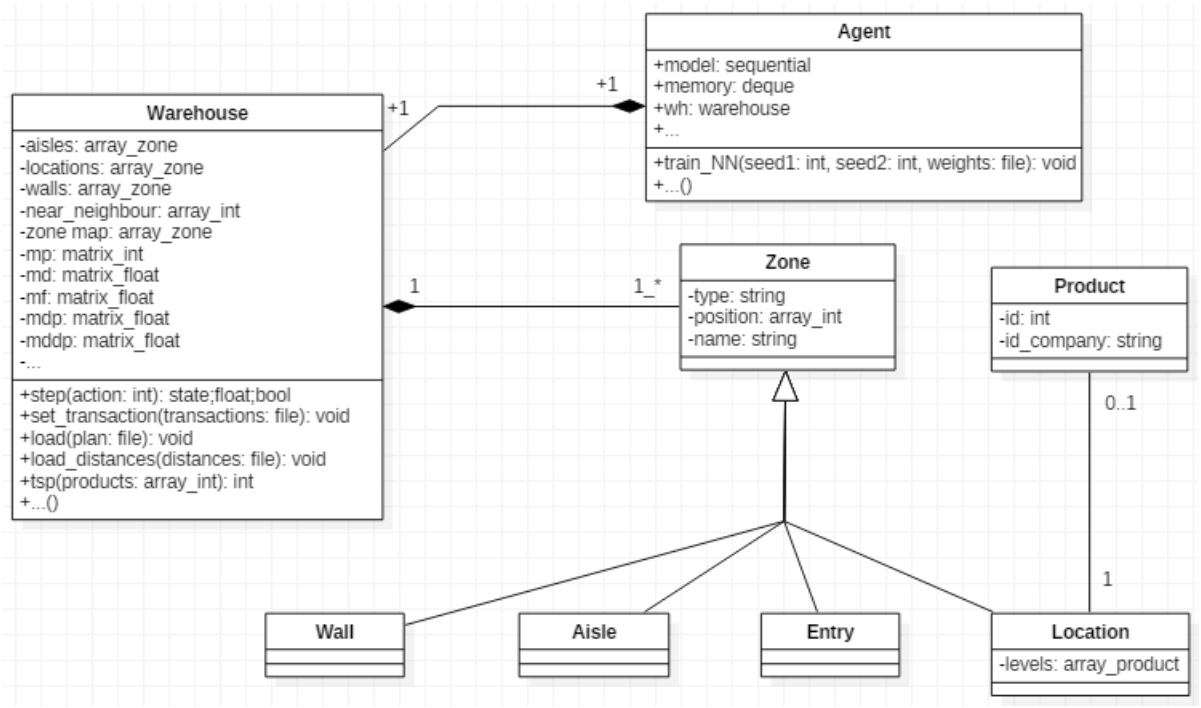

Figura 3.2: Schema UML: principali classi, attributi e metodi.

3.2.2 Frequent itemset

Per calcolare esattamente il costo di un'allocazione si deve leggere il file delle transazioni d'acquisto e per ognuna di esse calcolare tramite TSP la distanza corrispondente. Tali distanze sono poi moltiplicate per la frequenza con cui uno stesso insieme di prodotti

è acquistato e la somma dei valori ottenuti è il costo dell'allocazione. Per rendere più veloce il calcolo è opportuno tenere in memoria gli insiemi dei prodotti, il loro costo e la loro frequenza. Al momento di uno scambio è sufficiente ricalcolare le distanze per quegli insiemi che contengono almeno uno dei due prodotti scambiati e aggiornare il costo totale per differenza rispetto al precedente.
Tuttavia per istanze del problema complesse con un alto numero di prodotti, il numero di insiemi del file degli acquisti da considerare aumenta significativamente rendendo il calcolo del costo lento. Per questo motivo si possono approssimare con efficacia gli insiemi dei prodotti acquistati nella stessa transazione mediante i frequent itemset (FIS). Un FIS è un insieme di prodotti che occorre un numero di volte superiore ad una soglia prefissata, la soglia può essere assoluta o percentuale. Nel corso di questa tesi si è scelto di utilizzare dei FIS massimali, ovvero un insieme per essere un FIS non deve essere un sottoinsieme di un altro FIS. Per calcolare i FIS a partire dal file delle transazioni di acquisto si è utilizzata un'implementazione dell'algoritmo eclat. Tale algoritmo è descritto nella sezione 3.2.4.

3.2.3 Strutture dati principali

Al fine di memorizzare tutti i dati necessari per lo svolgimento dell'algoritmo e per ottimizzare i tempi di calcolo, la classe Warehouse contiene le seguenti strutture dati per il rapido accesso alle informazioni del magazzino:

- lista delle zone di corridoio (aisles): contiene tutte le zone di tipo corridoio presenti nel magazzino, gli indici della lista corrispondono agli identificativi stessi delle zone, in questo modo una zona è di immediato recapito sapendo il suo ID;
- lista delle zone di locazione (locations): contiene tutte le zone di tipo locazione presenti nel magazzino, gli indici della lista corrispondono agli identificativi stessi delle zone, in questo modo una zona è di immediato recapito sapendo il suo ID;
- lista delle zone non percorribili (walls): contiene tutte le zone di tipo muro presenti nel magazzino, gli indici della lista corrispondono agli identificativi stessi delle zone, in questo modo una zona è di immediato recapito sapendo il suo ID;

- lista delle vicinanze (near-neighbours): gli elementi sono un sottoinsieme degli identificativi delle zone di corridoio, gli indici corrispondono agli ID delle zone di locazione, in questo modo si rendono accessibili le locazioni da una zona di corridoio limitrofa attraversabile da un addetto al prelievo;
- dizionario del magazzino (map): per ogni stringa "(x,y)" corrispondente ad una posizione nel magazzino è associata la zona corrispondente;
- insieme di dizionari per la rapida conversione tra: identificativi dei prodotti dell'azienda con identificativi dei prodotti interni del magazzino, identificativi delle locazioni con il nome della locazione e della sua zona di locazione;

Durante la costruzione del magazzino sono note le distanze che intercorrono tra zone di corridoio vicine tra loro, ovvero a distanza unitaria. Una zona di locazione è accessibile da parte di un addetto al prelievo tramite una zona di corridoio ad essa adiacente. Per ottenere le distanze tra tutte le coppie è sufficiente costruire un grafo che abbia per nodi le zone e per archi le distanze unitarie note. Dal grafo, applicando Dijsktra, si ricavano le altre distanze. A questo punto si estraggono le distanze solo per le zone di corridoio usate per accedere a quelle di locazione e per il punto di entrata/uscita. Per sapere la distanza che deve essere percorsa per raggiungere un prodotto p (con p = 0,..., N-1) bisogna risalire alla zona di locazione (indice) tramite p usato come indice e leggere la distanza dalla matrice delle distanze.
Di seguito sono elencate le matrici utilizzate nel DRL nella classe Warehouse (tutte le matrici sono nella forma (N+1 x N+1) con N numero di prodotti/locaioni):

- matrice delle posizioni (MP) prodotti x locazioni: rappresenta l'allocazione dei prodotti nelle locazioni, gli elementi delle celle possono valere 1 se l'elemento i-esimo è allocato nella locazione j-esima, 0 altrimenti. La matrice ha una colonna e riga aggiuntiva di 0 e con $MP_{N,N}$ a 1. La somma degli elementi è costante nel tempo ed è N+1;
- matrice delle distanze tra locazioni (MD) locazioni x locazioni: in ogni cella della matrice è presente la distanza che intercorre tra le corrispettive due locazioni. La diagonale è nulla in quanto la distanza di una locazione e se stessa è 0. La matrice ha una colonna e una riga aggiuntiva contenente le distanze rispetto al punto di

ingresso/uscita del magazzino. Tale matrice è invariata durante l'esecuzione dell'algoritmo. Per facilitare l'apprendimento della rete la matrice è stata normalizzata in modo tale che i suoi valori siano compresi nell'insieme $[0, 1]$;

- matrice dei frequent itemset (MF) prodotti x prodotti: gli elementi sono il numero con cui i prodotti coinvolti compaiono assieme, ovvero negli stessi set, all'interno del file dei frequent itemset moltiplicato per la frequenza dei set stessi. La matrice ha una colonna e riga aggiuntiva, le celle aggiunte contengono il valore relativo alla frequenza del singolo prodotto nel file. La diagonale viene posta a 0. Tale matrice è invariata durante l'esecuzione dell'algoritmo. Per facilitare l'apprendimento della rete la matrice è stata normalizzata in modo tale che i suoi valori siano compresi nell'insieme $[0, 1]$;

- matrice delle distanze tra prodotti (MDP) prodotti x prodotti: è il risultato della seguente operazione $MP \cdot MD \cdot MP^T$. Moltiplicando la matrice delle posizioni per quella delle distanze tra locazioni e nuovamente per quella delle posizioni trasposta si ottiene che per ogni cella della nuova matrice vi è la distanza tra due prodotti, lungo l'ultima riga e colonna tra i prodotti e il punto di ingresso/uscita ((prodotti x locazioni) x (locazioni x locazioni) x (locazioni x prodotti) = (prodotti x prodotti)). I valori di questa matrice sono già compresi nell'intervallo $[0, 1]$;

- matrice delle distanze pesate tra prodotti (MDPP) prodotti x prodotti: ottenuta tramite la moltiplicazione elemento per elemento $MFxMDP$. Rappresenta le distanze tra prodotti pesate secondo la loro frequenza nei frequent itemset. I valori di questa matrice sono già compresi nell'intervallo $[0, 1]$.

Esempio delle matrici utilizzate per l'istanza del PAP con 10 prodotti.

	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	E
P0	0	0	0	0	0	0	0	0	1	0	0
P1	0	1	0	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	0	0	1	0	0	0
P3	1	0	0	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	1	0	0	0	0	0
P5	0	0	0	0	0	0	0	0	0	1	0
P6	0	0	1	0	0	0	0	0	0	0	0
P7	0	0	0	0	0	0	1	0	0	0	0
P8	0	0	0	0	1	0	0	0	0	0	0
P9	0	0	0	1	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	1

Tabella 3.1: Matrice MP

	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	E
L0	0	0.039	0.071	0.094	0.134	0.157	0.197	0.22	0.26	0.283	1
L1	0.039	0	0.031	0.071	0.11	0.134	0.173	0.197	0.236	0.26	0.976
L2	0.071	0.031	0	0.039	0.079	0.102	0.142	0.165	0.205	0.228	0.945
L3	0.094	0.071	0.039	0	0.039	0.079	0.118	0.142	0.181	0.205	0.921
L4	0.134	0.11	0.079	0.039	0	0.039	0.079	0.102	0.142	0.165	0.882
L5	0.157	0.134	0.102	0.079	0.039	0	0.039	0.079	0.118	0.142	0.858
L6	0.197	0.173	0.142	0.118	0.079	0.039	0	0.039	0.079	0.102	0.819
L7	0.22	0.197	0.165	0.142	0.102	0.079	0.039	0	0.039	0.079	0.795
L8	0.26	0.236	0.205	0.181	0.142	0.118	0.079	0.039	0	0.039	0.756
L9	0.283	0.26	0.228	0.205	0.165	0.142	0.102	0.079	0.039	0	0.732
E	1	0.976	0.945	0.921	0.882	0.858	0.819	0.795	0.756	0.732	0

Tabella 3.2: Matrice MD

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	E
P0	0.067	0	0	0.001	0	0	0	0	0.001	0	0.067
P1	0	0.055	0	0.003	0	0	0	0	0.001	0	0.055
P2	0	0	0.06	0	0	0.001	0	0.001	0.009	0.002	0.06
P3	0.001	0.003	0	0.062	0.001	0	0	0	0	0	0.062
P4	0	0	0	0.001	0.053	0.002	0.001	0.001	0.007	0.001	0.053
P5	0	0	0.001	0	0.002	0.053	0	0	0.012	0.001	0.053
P6	0	0	0	0	0.001	0	0.058	0.002	0.003	0	0.058
P7	0	0	0.001	0	0.001	0	0.002	0.055	0.004	0	0.055
P8	0.001	0.001	0.009	0	0.007	0.012	0.003	0.004	1	0.063	1
P9	0	0	0.002	0	0.001	0.001	0	0	0.063	0.176	0.176
E	0.067	0.055	0.06	0.062	0.053	0.053	0.058	0.055	1	0.176	0

Tabella 3.3: Matrice MF

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	E
P0	0	0.236	0.039	0.26	0.118	0.039	0.205	0.079	0.142	0.181	0.756
P1	0.236	0	0.197	0.039	0.134	0.26	0.031	0.173	0.11	0.071	0.976
P2	0.039	0.197	0	0.22	0.079	0.079	0.165	0.039	0.102	0.142	0.795
P3	0.26	0.039	0.22	0	0.157	0.283	0.071	0.197	0.134	0.094	1
P4	0.118	0.134	0.079	0.157	0	0.142	0.102	0.039	0.039	0.079	0.858
P5	0.039	0.26	0.079	0.283	0.142	0	0.228	0.102	0.165	0.205	0.732
P6	0.205	0.031	0.165	0.071	0.102	0.228	0	0.142	0.079	0.039	0.945
P7	0.079	0.173	0.039	0.197	0.039	0.102	0.142	0	0.079	0.118	0.819
P8	0.142	0.11	0.102	0.134	0.039	0.165	0.079	0.079	0	0.039	0.882
P9	0.181	0.071	0.142	0.094	0.079	0.205	0.039	0.118	0.039	0	0.921
E	0.756	0.976	0.795	1	0.858	0.732	0.945	0.819	0.882	0.921	0

Tabella 3.4: Matrice MDP

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	E
P0	0	0.007	0	0.037	0.002	0	0	0.002	0.013	0.006	5.028
P1	0.007	0	0	0.012	0	0.004	0	0	0.007	0.001	5.37
P2	0	0	0	0	0.001	0.009	0.005	0.005	0.095	0.029	4.743
P3	0.037	0.012	0	0	0.01	0.009	0	0	0	0.003	6.183
P4	0.002	0	0.001	0.01	0	0.022	0.011	0.003	0.028	0.005	4.554
P5	0	0.004	0.009	0.009	0.022	0	0.004	0.002	0.202	0.029	3.908
P6	0	0	0.005	0	0.011	0.004	0	0.031	0.025	0.001	5.457
P7	0.002	0	0.005	0	0.003	0.002	0.031	0	0.032	0.006	4.486
P8	0.013	0.007	0.095	0	0.028	0.202	0.025	0.032	0	0.248	88.189
P9	0.006	0.001	0.029	0.003	0.005	0.029	0.001	0.006	0.248	0	16.236
E	5.028	5.379	4.743	6.183	4.554	3.908	5.457	4.486	88.189	16.236	0

Tabella 3.5: Matrice MDDP *

*i valori delle celle sono rappresentati in formato 10^{-2}

3.2.4 Algoritmi

Di seguito sono presentati i principali algoritmi utilizzati nel corso del progetto.

- Dijsktra: utilzzato in fase di costruzione della classe magazzino per ottenere le distanze tra tutte le zone di corridoio. Le zone costituiscono i nodi del grafo e gli archi presenti sono quelli che intercorrono tra due zone limitrofe di distanza unitaria. L'arco ha peso 1 ed è bidirezionale;
- TSP: un'implementazione del problema del commesso viaggiatore è stata utilizzata per permettere di calcolare con esattezza il costo di prelievo di un generico insieme di prodotti. Dato il file degli acquisti, calcolati tutti i differenti insiemi di prodotti rappresentanti un ordine di acquisto e le loro frequenze, applicando il TSP ad ognuno di essi, moltiplicando per la rispettiva frequenza e sommando i risultati si ottiene il costo complessivo derivato dall'attuale allocazione in uso nel magaz-

zino. È stata utilizzata l'implementazione dell'algoritmo presente in *ORTOOLS* (appendice A);

- Eclat: il nome completo dell'algoritmo è Equivalence CLAss Transformation ed è ideato per migliorare le prestazioni di Apriori, un altro noto algoritmo per individuare i FIS. L'obiettivo di Eclat è quello di generare una lista di FIS con associata la frequenza con cui compaiono data una lista di transazioni e una soglia minima. Eclat opera creando un albero degli insiemi dei prodotti effettuando un'analisi depht first basata sulle intersezioni tra gli insiemi [7]. In questo lavoro è stata utilizzata l'implementazione *pyfim* (appendice A);
- DRL: descritto nel capitolo 2, con la differenza che al posto di avere come output della rete un vettore di lunghezza pari al numero di possibili azioni è stata utilizzata una matrice prodotto x prodotto. Il numero di celle totali è NxN con N uguale al numero di prodotti. La cella dal valore più alto indica quale coppia <prodotto,prodotto> debba essere scambiata;
- MFF: semplice euristica per individuare un'allocazione che in linea di principio è buona e in generale migliore di una soluzione casuale. Consiste nell'allocare i prodotti più frequenti nel file degli acquisti nelle locazioni più vicine al punto di entrata/uscita;
- BruteForce: per calcolare l'allocazione ottima è stata scritta una funzione che procedendo ricorsivamente esplora tutte le N! allocazioni possibili. Non è applicabile per istanze del problema superiori a PAP(10).

3.3 Risultati

3.3.1 Ricerca Operativa

Il problema dell'allocazione di prodotti (PAP) può essere espresso come un problema di assegnazione quadratica (QAP) semplificandone la funzione obiettivo per renderla al massimo di grado 2. Una descrizione informale del QAP problema è la seguente:

- c'è un insieme di n impianti e un insieme di n locazioni;

- è definita una matrice delle distanze per le locazioni;
- è definita una matrice di pesi corrispondente al flusso tra gli impianti.

L'obiettivo è trovare l'allocazione che minimizza la somma delle distanze moltiplicata per i pesi. Formalmente il problema è descritto come segue:

- P = $\{p_1, \ldots, p_n\}$, insieme degli impianti;
- L = $\{l_1, \ldots, l_n\}$, insieme delle locazioni;
- $w : PxP \to R$, matrice dei pesi;
- $d : LxL \to R$, matrice delle distanze;
- $f : P \to L$, funzione di assegnamento;

La funzione obiettivo da minimizzare è:

$$\sum_{p_1,p_2 \in P} w_{p_1,p_2} d_{f(p_1),f(p_2)}$$

Tale funzione si può espandere esplicitando le combinazioni $d_{f(a),f(b)}$, ovvero sia $x_{i,j}$ una variabile intera in $0, 1$ tale che abbia valore 1 se il prodotto p_i è allocato nella locazione p_j, 0 altrimenti. La funzione diventa:

$$z = \sum_{p_1,p_2 \in P} \sum_{l_1,l_2 \in L} w_{p_1,p_2} d_{l_1,l_2} x_{p_1,l_1} x_{p_2,l_2}$$

$$s.t. \quad \sum_{j=1}^{n} x_{i,j} = 1 \quad \forall i = 1, \ldots, n \tag{1}$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \quad \forall j = 1, \ldots, n \tag{2}$$

$$0 \leq x_{i,j} \leq 1 \quad \forall i, j = 1, \ldots, n \tag{3}$$

$$x_{i,j} \in N \quad \forall i, j = 1, \ldots, n \tag{4}$$

I vincoli (3) e (4) garantiscono che le variabili possono assumere o valore 0 oppure 1. Il vincolo (1) assicura che ogni impianto i deve essere allocato in un'unica locazione. Il

vincolo (2) impone che ogni locazione j contenga esattamente un impianto.
Usando la modellazione generica del QAP si ricava il PAP. Infatti il PAP consiste nel minimizzare i costi di prelievo di prodotti in magazzino in base allo storico delle transazioni d'acquisto. Alla tabella delle distanze viene aggiunta quella rispetto all'entrata del magazzino, mentre la tabella dei pesi corrisponde alla frequenza con cui una coppia di prodotti è acquistata nella stessa transazione. Alla funzione obiettivo vengono aggiunti i termini che rappresentano la distanza del prodotto i allocato nella posizione j moltiplicati per la loro frequenza assoluta, ovvero quante volte compare il prodotto i negli acquisti:

$$\sum_{p_1,p_2 \in P} \sum_{l_1,l_2 \in L} w_{p_1,p_2} d_{l_1,l_2} x_{p_1,l_1} x_{p_2,l_2} + w_{p1,p1} d_{l1,I} x_{p1,l1}$$

dove l'indice I è quello relativo all'entrata.
Questa modellazione non corrisponde esattamente alla descrizione del PAP, ma è una sua approssimazione. Poiché l'obiettivo è quello di minimizzare il costo complessivo di prelievo la corretta funzione obiettivo dovrebbe contenere termini di ordine superiore al 2°. Infatti sarebbe necessario esprimere tutte le combinazioni di prodotti acquistati in un'unica transazione presenti nel file degli acquisti. Tuttavia il problema diventerebbe tanto complesso da essere affrontabile unicamente tramite euristiche.
Per l'implementazione si è utilizzata la libreria *MIOSQP* [6], che si basa sul metodo branch-and-bound, e la si è applicata al PAP(10) e PAP(15). Per istanze del problema con un numero superiori di prodotti la matrice dei coefficienti dei termini quadrati risulta non avere tutti gli autovalori non nulli, per questo motivo non è semi definita positiva e il problema risulta essere non convesso.

3.3.2 Deep Reinforcement Learning

Di seguito verranno mostrati i risultati del DRL a istanze di complessità crescente del problema. Inizialmente si considera il problema del PAP(10), ovvero con 10 prodotti e 10 locazioni. Sebbene il numero di prodotti e locazioni sia a prima vista poco significativo, il numero totale di possibili allocazioni dei prodotti è 10!, ovvero 3.628.800 allocazioni differenti. Per la complessità del problema è possibile trovare l'allocazione ottima per forza bruta ed avere quindi un confronto diretto con l'ottimo. Per problemi di dimensioni anche poco più grandi non è in alcun modo possibile l'approccio a forza bruta poiché i

possibili stati del magazzino risulterebbero eccessivi per un calcolatore.
Per PAP(10) e PAP(15) si è utilizzata come funzione di costo il calcolo effettivo del costo sulle transazioni di acquisto in modo ottimizzato, ovvero ricalcolando la distanza degli insiemi che contengono almeno uno dei due prodotti scambiati. Per PAP superiori invece si è fatto uso dei frequent itemset al fine di rendere il processo di addestramento più rapido.
Come metro di paragone per confrontare i risultati del PAP verranno utilizzate le disposizioni dell'azienda e la disposizione che assegna le locazioni più vicine al punto di ingresso/uscita i prodotti più frequenti in assoluto (most frequent first, MMF). A fine capitolo sono riepilogati in forma tabellare tutti i risultati ottenuti.

- PAP(10): generazione di un'allocazione random per i 10 prodotti più frequentemente venduti dall'azienda. Le locazioni utilizzate sono quelle in cui l'azienda ha originariamente allocato tali prodotti. È stata utilizzata una rete neurale profonda di tipo denso utilizzando la libreria Keras. Lo strato di input contiene 512 neuroni, a seguire 4 strati da 256 neuroni e come strato di output 100 neuroni. Dopo ogni strato, ad eccezione dell'ultimo, è presente la funzione di attivazione ReLu. L'input della rete sono le matrici descritte precedentemente in 3.2.3 (11x11x5 = 605 valori): MP, MD, MF, MDP e MDDP. Ognuna di esse ha dimensione 11x11 per un totale di 605 valori. Prima di raggiungere i neuroni del primo livello l'input è modificato tramite Flatten. Anche la forma dell'output è modificata tramite Reshape in una matrice 10x10. Gli indici, x e y, della cella col valore più alto della matrice di output corrispondono ai prodotti da scambiare. La NN ha come ottimizzatore Adam, funzione di loss mse e learning rate = 10^{-6}. Sono state effettuate 10000 epoche nelle quali sono avvenuti per ciascuna 20 scambi di prodotti, per ogni scambio viene acquisita nuova esperienza e la rete viene addestrata. Alla fine di un'epoca viene ripristinata l'allocazione di partenza. Come ricompensa viene restituita la differenza della funzione costo dello stato precedente con quella dello stato attuale scontata di un fattore 10^{-5}. Il costo è calcolato interamente su tutte le transazioni che contengono unicamente i 10 prodotti considerati tra quelle del file delle transazioni (con tutti i prodotti aziendali). Il numero totale di transazioni considerate sono 9720. Le transazioni sono equivalenti a 54 insiemi di prodotti

con associata per ognuno di essi la frequenza con cui compaiono nel file. Come iperparametri del DRL sono stati scelti dimensione della batch = 32, $\gamma = 0.9$ e $\epsilon = 0.1$ come valore iniziale, 0.0001 come terminale scalando proporzionalmente ogni epoca. Come si può dedurre dall'andamento della curva nella figura 3.4 e 3.5, il DRL ha quasi raggiunto l'ottimo.

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 605)	0
dense_1 (Dense)	(None, 512)	310272
activation_1 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
activation_2 (Activation)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65792
activation_3 (Activation)	(None, 256)	0
dense_4 (Dense)	(None, 256)	65792
activation_4 (Activation)	(None, 256)	0
dense_5 (Dense)	(None, 256)	65792
activation_5 (Activation)	(None, 256)	0
dense_6 (Dense)	(None, 100)	25700
reshape_1 (Reshape)	(None, 10, 10)	0

Total params: 664,676
Trainable params: 664,676
Non-trainable params: 0

Figura 3.3: Riepilogo rete neurale PAP(10)

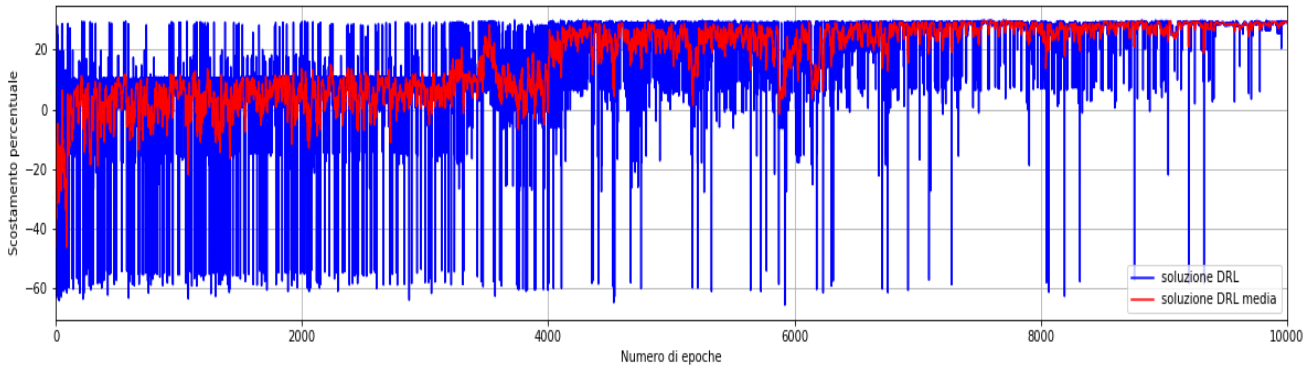

Figura 3.4: Scostamento percentuale per epoca di addestramento PAP(10)

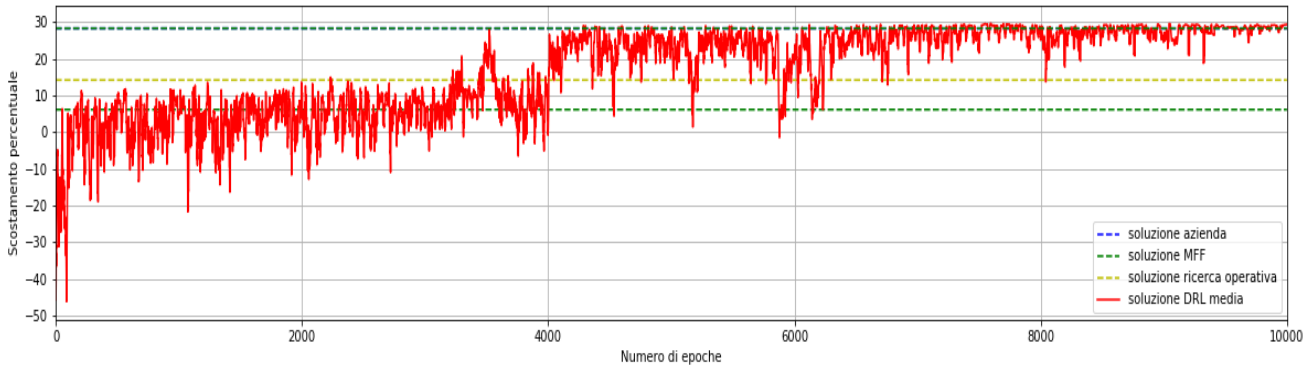

Figura 3.5: Confronto del DRL con soluzioni alternative PAP(10)

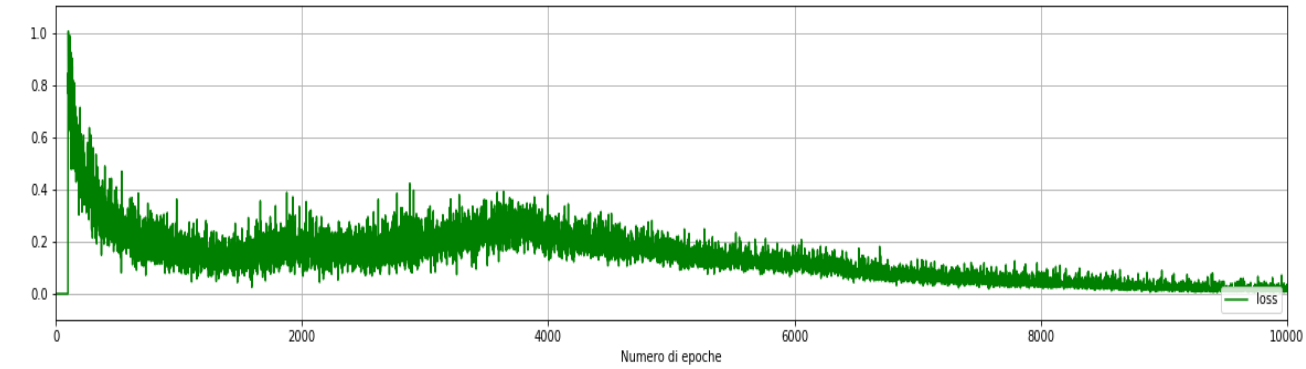

Figura 3.6: Andamento della loss (MSE) PAP(10)

- PAP(15): generazione di un'allocazione random per i 15 prodotti più venduti dall'azienda. Le locazioni utilizzate sono quelle in cui l'azienda ha originariamente allocato tali prodotti. È stata utilizzata la stessa rete del PAP(10), con input le 5 matrici in 3.2.3 (16x16x5 = 1280 valori) e output una matrice 15x15. Sono state effettuate 5000 epoche nelle quali sono avvenuti per ciascuna 25 scambi di prodotti, per ogni prodotto scambiato viene acquisita nuova esperienza e la rete viene addestrata. Alla fine di un'epoca viene ripristinata l'allocazione di partenza. Come ricompensa viene restituita la differenza della funzione costo dello stato precedente con quella dello stato attuale scontata di un fattore 10^{-5}. Il costo è calcolato interamente su tutte le transazioni che contengono unicamente i 15 prodotti considerati tra quelle del file delle transazioni (con tutti i prodotti aziendali). Il numero totale di transazioni considerate sono 10933. Come iperparametri del DRL sono stati scelti dimensione della batch = 32, $\gamma = 0.9$ e $\epsilon = 0.1$ come valore iniziale, 0.0001 come terminale scalando proporzionalmente ogni epoca. Come si può dedurre dall'andamento della curva nella figura 3.8 e 3.9, il DRL ha quasi raggiunto l'ottimo.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 1280)              0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               655872
_________________________________________________________________
activation_6 (Activation)    (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328
_________________________________________________________________
activation_7 (Activation)    (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_8 (Activation)    (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
activation_9 (Activation)    (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               65792
_________________________________________________________________
activation_10 (Activation)   (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 225)               57825
_________________________________________________________________
reshape_2 (Reshape)          (None, 15, 15)            0
=================================================================
Total params: 1,042,401
Trainable params: 1,042,401
Non-trainable params: 0
_________________________________________________________________
```

Figura 3.7: Riepilogo rete neurale PAP(15)

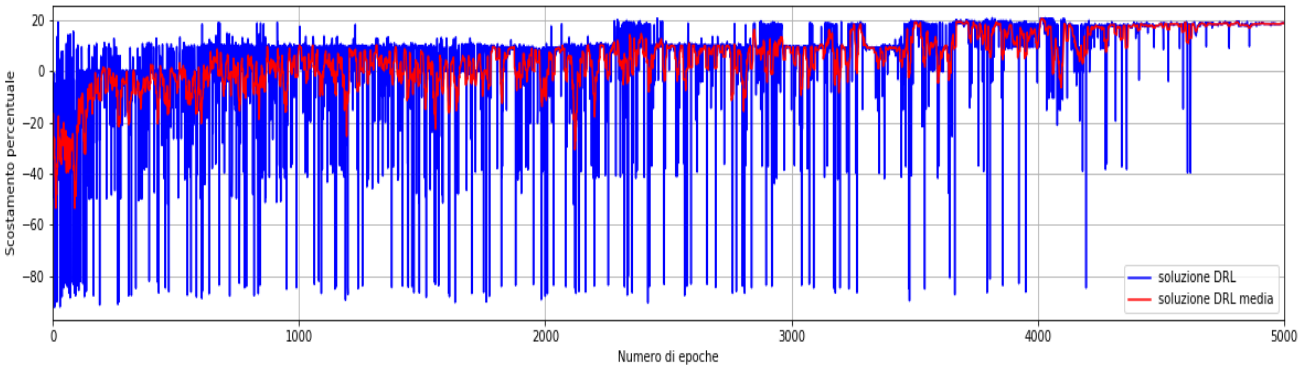

Figura 3.8: Scostamento percentuale per epoca di addestramento PAP(15)

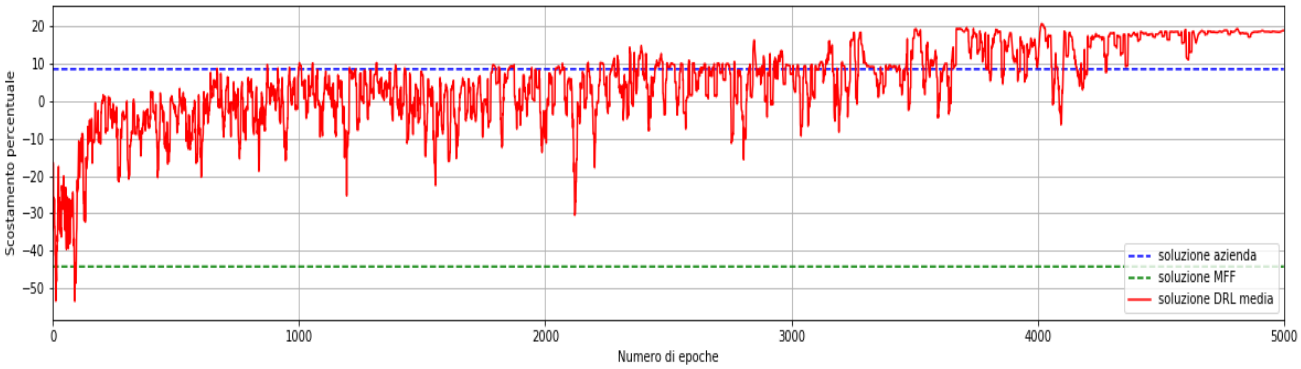

Figura 3.9: Confronto del DRL con soluzioni alternative PAP(15)

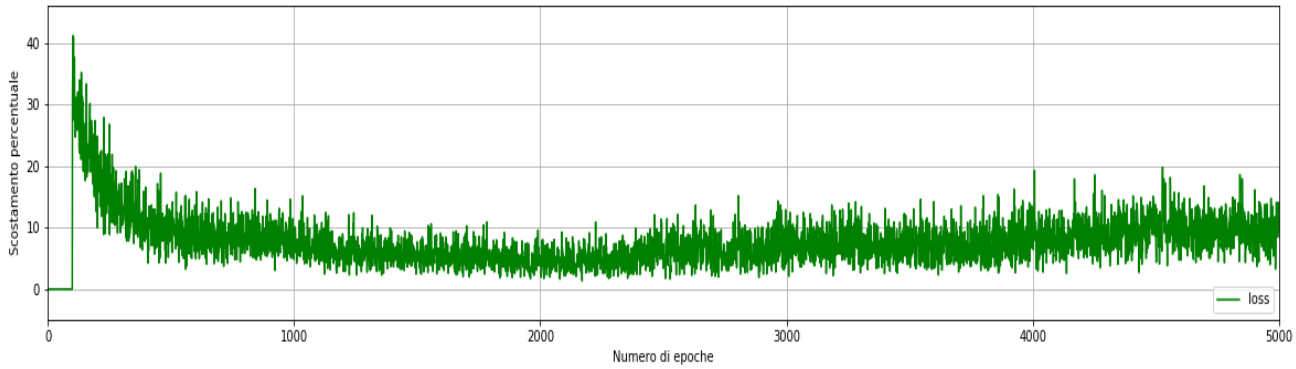

Figura 3.10: Andamento della loss (MSE) PAP(15)

- PAP(20): uso dell'allocazione scelta dall'azienda per i 20 prodotti più venduti. Le locazioni sono quelle in cui l'azienda ha originariamente allocato tali prodotti. È stata utilizzata la stessa rete del PAP(10) e PAP(15), con input le 5 matrici in 3.2.3 (21x21x5 = 2205 valori) e output una matrice 20x20. Sono state effettuate 5000 epoche nelle quali sono avvenuti per ciascuna 30 scambi di prodotti, per ogni prodotto scambiato viene acquisita nuova esperienza e la rete viene addestrata. Alla fine di un'epoca viene ripristinata l'allocazione di partenza. Come ricompensa viene restituita la differenza della funzione costo dello stato precedente con quella dello stato attuale scontata di un fattore 10^{-5}. Come funzione di costo si sono utilizzati i frequent itemset al posto di calcolarla sull'intero file degli acquisti. I frequent itemset sono stati calcolati a partire dal file degli acquisti inerenti soltanto alle transazioni che comprendevano unicamente i 20 prodotti considerati, 13304 transazioni. Come soglia per Eclat si è scelto di usare 0.01, eliminando quindi le transazioni di insiemi con frequenza minore a 2 ($13304 : 1.33 = 100 : 0.01$), ovvero transazioni di insiemi con frequenza unitaria. Così facendo si è utilizzato un file di frequent itemset di soli 104 insiemi a fronte di 270. In questo modo, a scapito dell'esattezza della funzione obiettivo, si velocizza il processo di apprendimento. Ogni scambio di prodotto impiega un tempo di calcolo dell'ordine della decina di millisecondi al posto del centinaio di millisecondi. Come iperparametri del DRL sono stati scelti dimensione della batch = 32, $\gamma = 0.9$ e $\epsilon = 0.1$ come valore iniziale, 0.0001 come terminale scalando proporzionalmente ogni epoca. Osservando le figure 3.12 e 3.13 si nota come la curva stia ancora crescendo e quindi, con un numero maggiore di epoche di addestramento, si potrebbero ottenere risultati migliori.

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 2205)	0
dense_1 (Dense)	(None, 512)	1129472
activation_1 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
activation_2 (Activation)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65792
activation_3 (Activation)	(None, 256)	0
dense_4 (Dense)	(None, 256)	65792
activation_4 (Activation)	(None, 256)	0
dense_5 (Dense)	(None, 256)	65792
activation_5 (Activation)	(None, 256)	0
dense_6 (Dense)	(None, 400)	102800
reshape_1 (Reshape)	(None, 20, 20)	0

Total params: 1,560,976
Trainable params: 1,560,976
Non-trainable params: 0

Figura 3.11: Riepilogo rete neurale PAP(20)

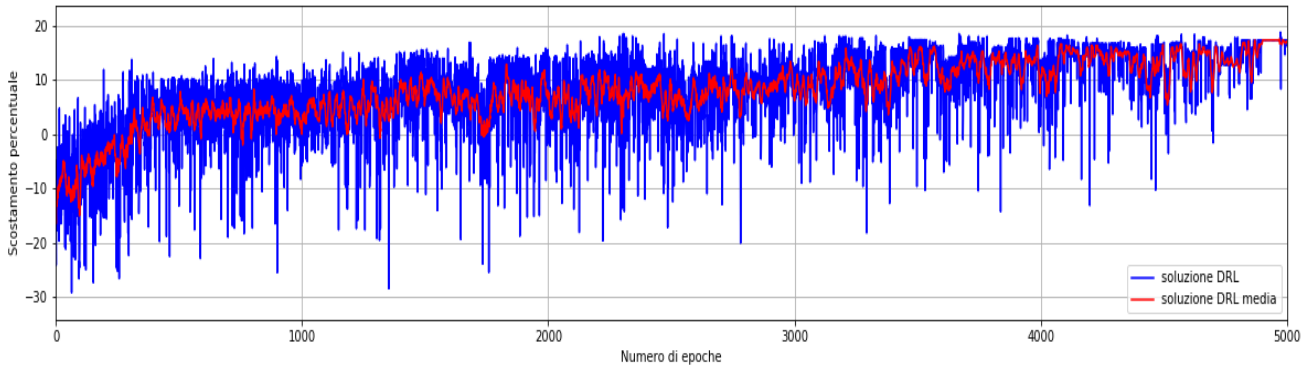

Figura 3.12: Scostamento percentuale per epoca di addestramento PAP(20)

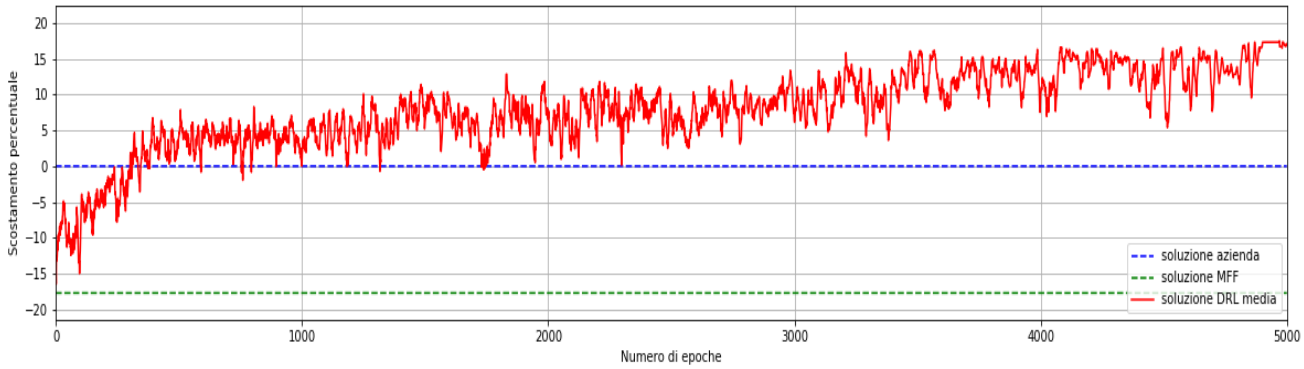

Figura 3.13: Confronto del DRL con soluzioni alternative PAP(20)

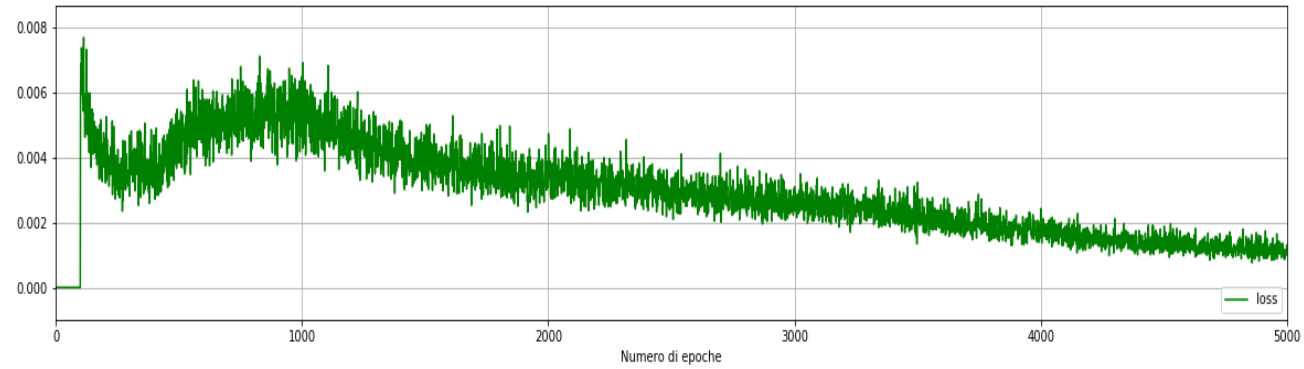

Figura 3.14: Andamento della loss (MSE) PAP(20)

- PAP(50): uso dell'allocazione scelta dall'azienda per i 50 prodotti più venduti. Le locazioni sono quelle in cui l'azienda ha originariamente allocato tali prodotti. È stata utilizzata la stessa rete usata precedentemente, con input le 5 matrici in 3.2.3 (51x51x5 = 13005 valori) e output una matrice 50x50. Sono state effettuate 3000 epoche nelle quali sono avvenuti per ciascuna 60 scambi di prodotti, per ogni prodotto scambiato viene acquisita nuova esperienza e la rete viene addestrata. Alla fine di un'epoca viene ripristinata l'allocazione di partenza. Come ricompensa viene restituita la differenza della funzione costo dello stato precedente con quella dello stato attuale scontata di un fattore 10^{-5}. Come funzione di costo si sono utilizzati i frequent itemset al posto di calcolarla sull'intero file degli acquisti. I frequent itemset sono stati calcolati a partire dal file degli acquisti inerenti soltanto alle transazioni che comprendevano unicamente i 50 prodotti considerati, 25477 transazioni. Come soglia per Eclat si è scelto di usare 0.5, eliminando quindi le transazioni di insiemi con frequenza minore a 128 (25477 : 127.385 = 100 : 0.5). Così facendo si è utilizzato un file di frequent itemset di soli 106 insiemi a fronte di 4444. In questo modo, a scapito dell'esattezza della funzione obiettivo, si velocizza il processo di apprendimento. Ogni scambio di prodotto impiega un tempo di calcolo dell'ordine della decina di millisecondi al posto di secondi. Come iperparametri del DRL sono stati scelti dimensione della batch = 32, $\gamma = 0.9$ e $\epsilon = 0.1$ come valore iniziale, 0.0001 come terminale scalando proporzionalmente ogni epoca. Osservando le figure 3.16 e 3.17 si nota come la curva stia ancora crescendo e quindi, con un numero maggiore di epoche di addestramento, si potrebbero ottenere risultati migliori.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 13005)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               6659072
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_4 (Activation)    (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_5 (Activation)    (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 2500)              642500
_________________________________________________________________
reshape_1 (Reshape)          (None, 50, 50)            0
=================================================================
Total params: 7,630,276
Trainable params: 7,630,276
Non-trainable params: 0
_________________________________________________________________
```

Figura 3.15: Riepilogo rete neurale PAP(50)

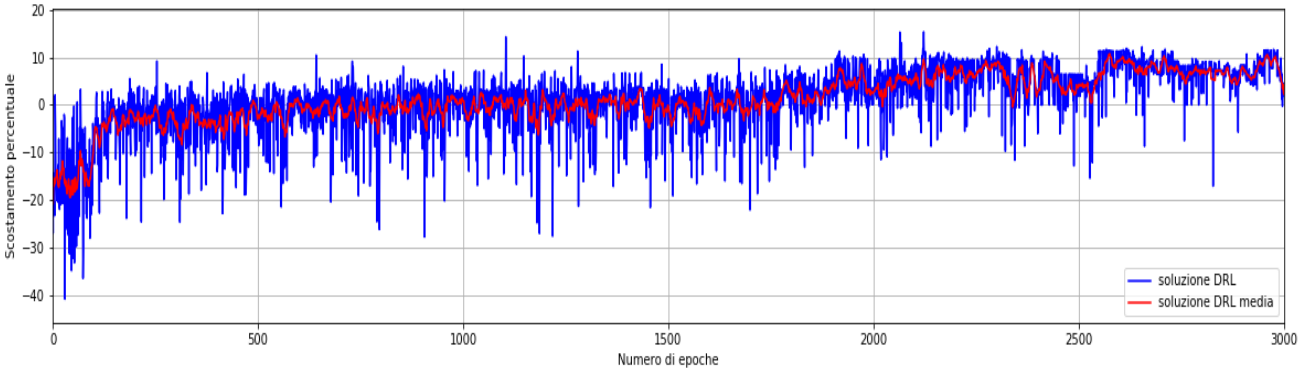

Figura 3.16: Scostamento percentuale per epoca di addestramento PAP(50)

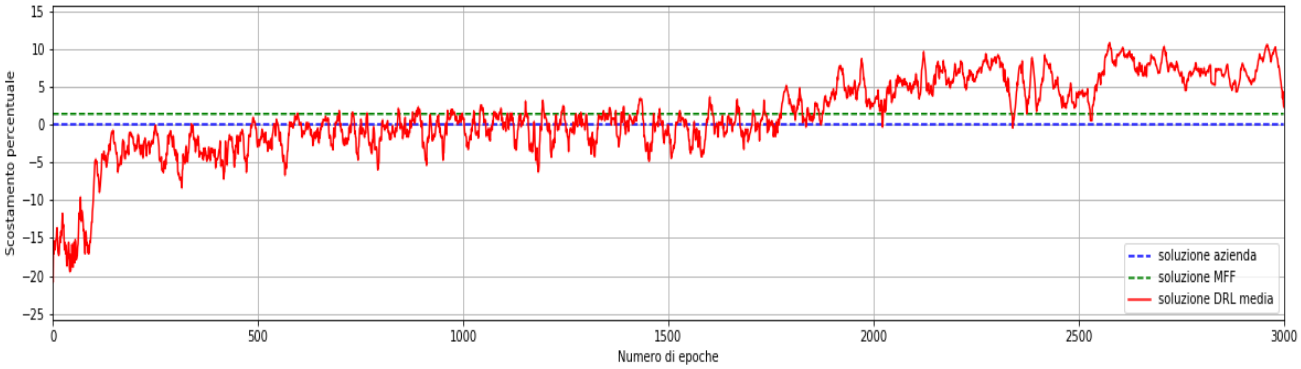

Figura 3.17: Confronto del DRL con soluzioni alternative PAP(50)

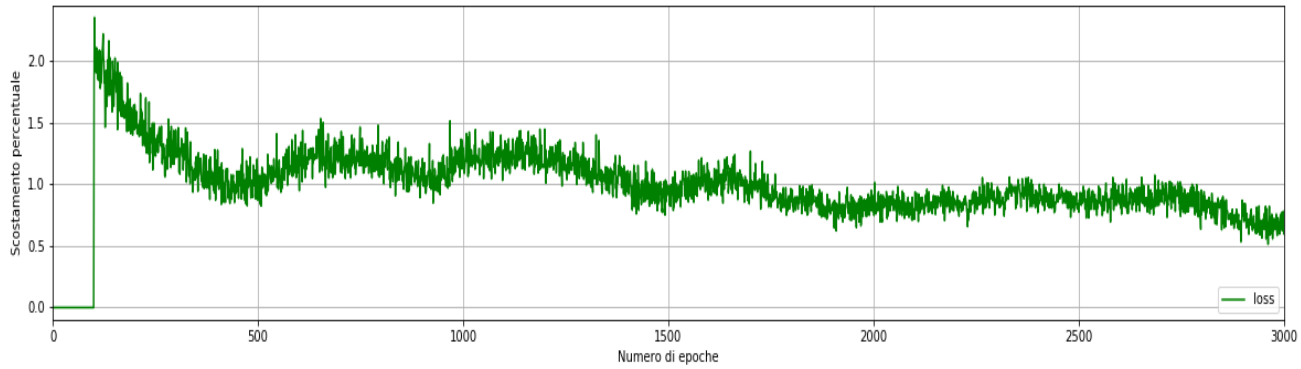

Figura 3.18: Andamento della loss (MSE) PAP(50)

- PAP(100): uso dell'allocazione scelta dall'azienda per i 100 prodotti più venduti. Le locazioni sono quelle in cui l'azienda ha originariamente allocato tali prodotti. È stata utilizzata la stessa rete usata precedentemente, con input le 5 matrici in 3.2.3 (101x101x5 = 51005 valori) e output una matrice 100x100. Sono state effettuate 3000 epoche nelle quali sono avvenuti per ciascuna 110 scambi di prodotti, per ogni prodotto scambiato viene acquisita nuova esperienza e la rete viene addestrata. Alla fine di un'epoca viene ripristinata l'allocazione di partenza. Come ricompensa viene restituita la differenza della funzione costo dello stato precedente con quella dello stato attuale scontata di un fattore 10^{-5}. Come funzione di costo si sono utilizzati i frequent itemset al posto di calcolarla sull'intero file degli acquisti. I frequent itemset sono stati calcolati a partire dal file degli acquisti inerenti soltanto alle transazioni che comprendevano unicamente i 100 prodotti considerati, 25477 transazioni. Come soglia per Eclat si è scelto di usare 2, eliminando quindi le transazioni di insiemi con frequenza minore a 694 (34677 : 693.54 = 100 : 2). Così facendo si è utilizzato un file di frequent itemset di soli 72 insiemi a fronte di 16971. In questo modo, a scapito dell'esattezza della funzione obiettivo, si velocizza il processo di apprendimento. Ogni scambio di prodotto impiega un tempo di calcolo dell'ordine della decina di millisecondi al posto di secondi. Come iperparametri del DRL sono stati scelti dimensione della batch = 8, $\gamma = 0.9$ e $\epsilon =$ 0.1 come valore iniziale, 0.0001 come terminale scalando proporzionalmente ogni epoca. Osservando le figure 3.20 e 3.21 si nota come la curva stia ancora crescendo e quindi, con un numero maggiore di epoche di addestramento, si potrebbero ottenere risultati migliori.

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 51005)	0
dense_1 (Dense)	(None, 512)	26115072
activation_1 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
activation_2 (Activation)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65792
activation_3 (Activation)	(None, 256)	0
dense_4 (Dense)	(None, 256)	65792
activation_4 (Activation)	(None, 256)	0
dense_5 (Dense)	(None, 256)	65792
activation_5 (Activation)	(None, 256)	0
dense_6 (Dense)	(None, 10000)	2570000
reshape_1 (Reshape)	(None, 100, 100)	0

Total params: 29,013,776
Trainable params: 29,013,776
Non-trainable params: 0

Figura 3.19: Riepilogo rete neurale PAP(100)

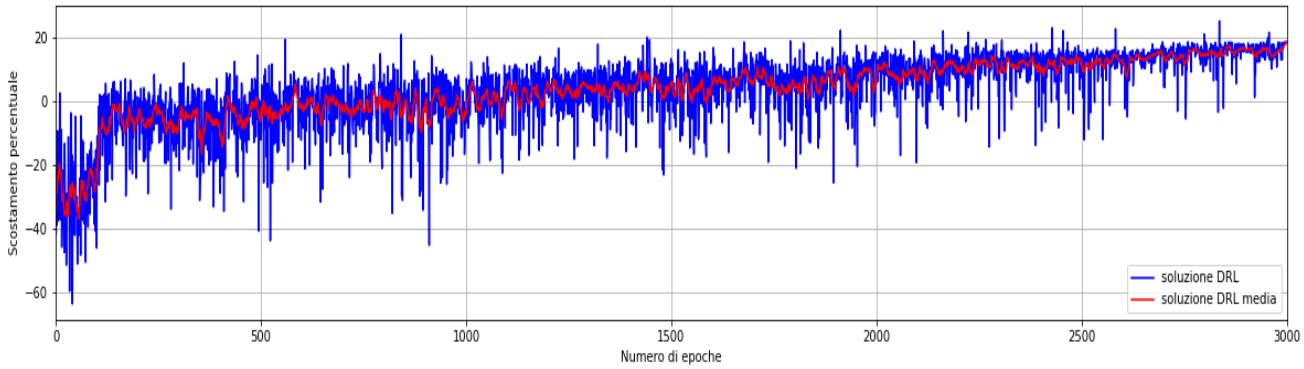

Figura 3.20: Scostamento percentuale per epoca di addestramento PAP(100)

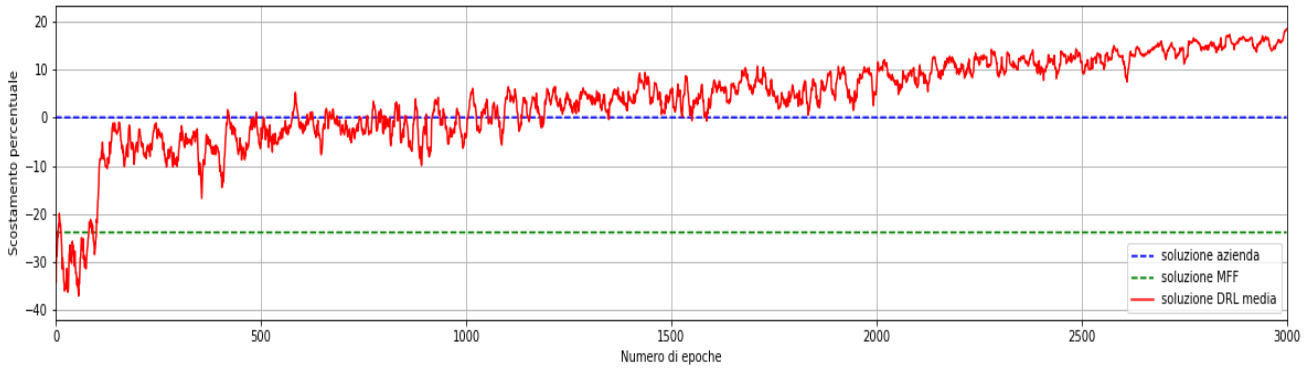

Figura 3.21: Confronto del DRL con soluzioni alternative PAP(100)

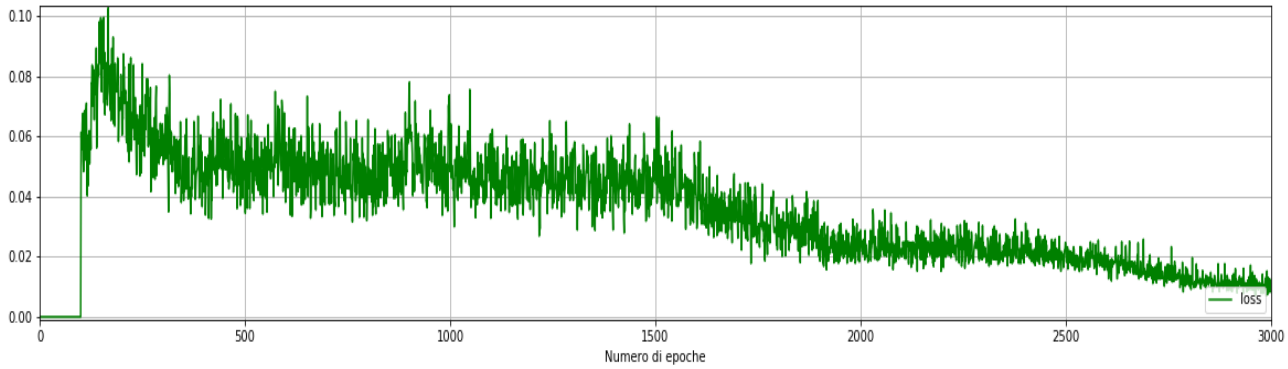

Figura 3.22: Andamento della loss (MSE) PAP(100)

Di seguito sono riportati in formato tabellare i risultati ottenuti sulle singole istanze del problema.

Istanza del problema	Allocazione	Frequent itemset	File acquisti	Diff. % *
PAP(10)	*Azienda*	–	827.320	0
	MFF	–	1.082.970	−30.9
	DRL	–	815.072	1.48
	Brute force	–	810.484	2.03
	MIQP	–	990.006	−19.66
PAP(15)	*Azienda*	–	958.118	0
	MFF	–	1.522.984	−58.95
	DRL	–	927.874	3.15
	MIQP	–	1.195.250	−24.75
PAP(20)	*Azienda*	72.322	1.421.076	0
	MFF	77.950	1.672.608	−17.7
	DRL	60.656	1.180.346	16.94
PAP(50)	*Azienda*	3.496.370	3.398.762	0
	MFF	3.447.940	3.470.542	−2.11
	DRL	3.083.206	2.959.704	12.92
PAP(100)	*Azienda*	8.160.848	5.725.976	0
	MFF	11.225.518	7.090.316	−23.83
	DRL	6.920.828	5.425.350	5.25

Tabella 3.6: Risultati dei costi

* le differenze percentuali si riferiscono al costo complessivo (File acquisti) rispetto al costo dell'allocazione utilizzata dall'azienda.
Si è sempre scelto di effettuare un numero di scambi per epoca pari al numero di prodotti del problema più 10. In linea teorica sono sufficienti un numero di scambi equivalente a quello dei prodotti per raggiungere qualsiasi allocazione, compresa quella (o quelle) ottima. Per dare maggior margine di manovra al DRL si sono aggiunti 10 scambi extra.

Conclusioni

All'aumentare del numero di prodotti la dimensionalità del problema aumenta in modo fattoriale, poiché il numero totale di possibili allocazioni di N prodotti in N locazioni è N! Questo significa che la PAP(100) è 10^{151} volte più complesso a livello spaziale di PAP(10). Tuttavia, il DRL si è dimostrato in grado di apprendere dai dati forniti in input, senza incorrere in punti di minimo locale, e ottenendo soluzioni migliori rispetto a quelle fornite dall'azienda e dalle altre euristiche. Si può osservare come da PAP(10) a PAP(100) il costo della soluzione del DRL in percentuale rispetto al costo di quella aziendale è sempre positivo. Inoltre, all'aumentare dello spazio delle possibili allocazioni anche la percentuale di guadagno tende ad essere più alta. Probabilmente ciò è dovuto al fatto che in spazi con dimensionalità maggiore ci sono più margini di manovra per il DRL e che la soluzione dell'azienda soffre l'aumento delle allocazioni. Questo fa supporre che l'aumento drastico della complessità del problema non è un limite per il DRL. Inoltre, dato l'andamento delle curve dei costi, è auspicabile che con un numero di epoche di addestramento superiore si ottengano risultati ancora migliori.

Tuttavia il DRL presenta anche delle limitazioni. In primis per PAP(i) con i maggiore di 15 è necessario operare con un'approssimazione della funzione di prelievo, ovvero con i frequent itemset. Scegliendo una soglia consona, si velocizza il calcolo del costo a scapito di precisione. Inoltre, mano a mano che i prodotti aumentano, anche la complessità della rete neurale si incrementa, in particolare aumenta il numero di parametri che la NN aggiorna durante il training. Semplificare troppo la rete per ridurre i parametri rischia di renderla inadeguata.
Questo limite lo si può notare in PAP(100) dove, per velocizzarne l'esecuzione, si è scelto

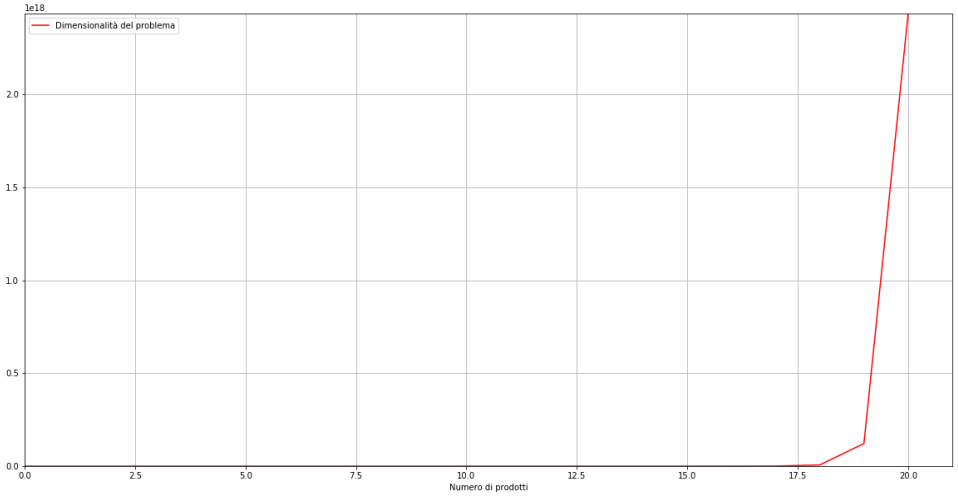

Figura 3.23: Crescita del numero di allocazioni per numero di prodotti

di ridurre la batch size da 32 a 8 e soprattutto di usare un numero di frequent itemset estremamente basso rispetto agli insiemi delle transazioni di acquisto, 72 contro 16971. Nonostante ciò il DRL riesce comunque ad ottenere un guadagno del 5.25% rispetto alla soluzione aziendale. Tuttavia se si analizza la differenza percentuale dei costi dei frequent itemset si ha un guadagno del (1-6.920.828/8.160.848)*100 = 15.19%. Ciò fa presupporre che non sono stati utilizzati abbastanza frequent itemset per approssimare correttamente il costo di prelievo, scegliendone in numero adeguato si otterranno risultati migliori a scapito del tempo di esecuzione.
Nelle istanze PAP(20) e PAP(50) invece il numero di frequent itemset risulta essere adeguato poiché le percentuali di miglioramento rispetto al costo di prelievo e al costo usando i frequent itemset sono molto simili tra loro: per il PAP(20) si ha un miglioramento effettivo del 16.94% e usando i frequent itemset del (1-60.656/72.322)*100 = 16.13%, mentre per il PAP(50) si ha un miglioramento effettivo del 12.92% e usando i frequent itemset del (1-3.083.206/3.496.370)*100 = 11.82%.

Appendice A

Librerie

Elenco delle principali librerie Python utilizzate durante lo sviluppo del progetto:

- numpy: libreria matematica per gestione di array e matrici;
- tensorflow: libreria per la creazione di reti neurali;
- keras: libreria di alto livello per la creazione di reti neurali, necessita di tensorflow
- collections: libreria per la gestione di strutture dati di grandi dimensioni, utilizzata per la realizzazione di memory replay;
- miosqp: libreria per la risoluzione di problemi di programmazione quadratica intera (link);
- matplotlib: libreria usata per plottare grafici;
- ortools: libreria di Google utilizzata per risolvere il TSP (link);
- pyfim: classe per calcolare i frequent itemset a partire da un file transazioni (link).

Bibliografia

[1] Mnih, V. et al. *Human-level control through deep reinforcement learning.* Nature 518, 529–533 (2015).

[2] Guo, X., Singh, S. P., Lee, H., Lewis, R. L. and Wang, X. *Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning* In Advances in Neural Information Processing Systems, 3338–3346 (2014).

[3] Silver, D., Schrittwieser, J. , Simonyan, K. *Mastering the Game of Go without Human Knowledge*

[4] Silver, D. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm* arXiv:1712.01815v1 [cs.AI] 5 (Dec 2017)

[5] Gulli, A., Pal, S. *Deep Learning with Keras* Packt Publishing Ltd (Apr 2017)

[6] Bartolomeo Stellato, Vihangkumar V. Naik, Alberto Bemporad, Paul Goulart and Stephen Boyd. *Embedded Mixed-Integer Quadratic Optimization Using the OSQP Solver* (2018)

[7] Mohammed J. Zaki. *Scalable Algorithms for Association Mining* IEEE transactions on knowledge and data engineering, VOL. 12, no. 3, May/Jun 2000

[8] Thomas E Vollmann, William Lee Berry, David Clay Whybark. *Manufacturing planning and control system for supply chain management* McGraw-Hill Education (2014)

[9] Tony Wild. *Best Practice in Inventory Management* Butterworth-Heinemann (2002)

[10] F. Guerriero, R. Musmanno, O. Pisacane, F. Rende. *A mathematical model for the Multi-Levels Product Allocation Problem in a warehouse with compatibility constraints* Elsevier (2012)

[11] Chiun-Ming Liu. *Optimal Storage Layout And Order Picking For Warehousing* International Journal of Operations Research Vol. 1, No. 1, 37-46 (2004)