
Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Scienze Informatiche

ELABORATO FINALE

# PROGETTAZIONE DI UN MODELLO DI DEEP LEARNING PER LA PREDIZIONE DELLE FUTURE PERFORMANCE DEI GIOCATORI DI CALCIO

Supervisore
*Mauro Dragoni*

Laureando
*Mattia Boller*

Anno accademico 2019/2020

# Ringraziamenti

Desidero ringraziare il professor Mauro Dragoni, supervisore di questa tesi, per tutto il fondamentale aiuto fornitomi durante il lavoro svolto e la stesura dell'elaborato.

Ringrazio poi i miei genitori, per il supporto continuo datomi durante tutto il percorso di studi, permettendomi di raggiungere questo traguardo.

# Indice

# Sommario

In questa tesi si è cercato di utilizzare tecniche di deep learning per problemi di predizione in campo sportivo. L'obiettivo è stato quello di prevedere come potessero evolvere le prestazioni di un calciatore durante l'ultima parte della partita a cui sta prendendo parte, basandosi sui dati riguardanti il giocatore raccolti durante il tempo già trascorso del match.

I dati utilizzati sono stati estratti attraverso tecniche di web scraping da un sito web specializzato nell'analisi approfondita dei dati calcistici, per poi essere inseriti in un database, struttura ideale per la raccolta e il successivo studio dei dati. Le informazioni sono state poi modellate in modo da essere utilizzate per risolvere il problema posto in principio, in un formato adatto ad essere sfruttate da tecniche di deep learning.

Come approccio al problema è stato deciso di utilizzare delle reti di tipo RNN (recurrent neural network), più precisamente delle reti LSTM (long short-term memory), data la capacità delle stesse di trattare sequenze di dati correlati tra loro. L'obiettivo principale iniziale è stato quello di riuscire a dividere le performance dei calciatori in casi di calo delle prestazioni e casi di miglioramento delle prestazioni, mentre successivamente si è mirato a cercare di riconoscere casi di calo, miglioramento e mantenimento costante delle prestazioni, con l'obiettivo di ridurre al minimo il confondere peggioramenti con miglioramenti e viceversa.

Sono stati eseguiti numerosi test, utilizzando diverse architetture della rete, diversi parametri e diverse strutture dei dati per cercare di ottenere le migliori performance dal modello. I risultati migliori sono stati generati dalla rete basata sulla classificazione in "peggioramento" e "miglioramento", con un'accuratezza di predizione sui dati del campionato Premier League del 63,35%, e dalla rete basata sulla classificazione in "peggioramento significativo", "miglioramento significativo" e "mantenimento costante delle performance", con una probabilità di confondere tra loro le classi "peggioramento significativo" e "miglioramento significativo" del circa 8% nel dataset del campionato Premier League.

I risultati sopra citati, dimostrano come il deep learning possa essere utilizzato efficacemente nel prevedere la prestanza fisica e l'evolversi della mentalità di un atleta durante competizione sportiva, basandosi su come si è comportato e su ciò che è successo in generale fino a quel momento. In futuro possono essere utilizzati dati più specifici riguardanti il singolo giocatore, in modo da entrare più a fondo nei particolari comportamenti e reazioni a diversi eventi da parte del calciatore, inoltre potrebbe essere esteso il modello a più sport, come il basket o la pallavolo, discipline in cui mantenere delle performance adeguate fino all'ultimo secondo è estremamente importante.

# Introduzione

Negli ultimi anni si sente parlare sempre di più di intelligenza artificiale, ma che di cosa si tratta? L'intelligenza artificiale è una branca dell'informatica interessata allo studio e alle tecniche di costruzione di macchine intelligenti, in grado di eseguire attività associate generalmente all'intelligenza umana, come per esempio imparare, osservare e prendere decisioni. Quindi non si parla solo di intelligenza vista come capacità di calcolo o di elaborazione dei dati, ma di tutti quei tipi di intelligenza descritti dalla teoria di Gardner, come quella spaziale, quella sociale o quella naturalistica. Ciò che un tempo sembrava fantascienza, oggi sta diventando realtà ed inizia a ricavarsi uno spazio importante in tantissimi settori diversi e uno di questi è lo sport. La tecnologia è già ampiamente sfruttata in ogni tipo di sport come aiuto ai giudici di gara e arbitri, basti pensare al sistema Hawk-Eye, utilizzato nel calcio e nel tennis per determinare la posizione della palla e quindi la sua validità in base alle regole, o al sistema VAR, e come aiuto nell'analisi delle prestazioni e dei risultati ottenuti nei giocatori. In una situazione del genere, l'introduzione dell'intelligenza artificiale è inevitabile, dato come rende possibile lavorare con enormi quantità di dati, intrecciandoli e mettendoli in relazione come mai è stato possibile prima. Un esempio del futuro verso il quale ci si sta dirigendo è la squadra italiana di calcio A.S. Roma che, come riporta il quotidiano La Stampa, ha stretto una partnership con Acronis (multinazionale attiva nel campo della cyber security) per la fornitura di soluzioni software basate sulla IA che serviranno ad ottimizzare le attività sportive e gestionali. [5]

Con il progetto trattato in questa tesi, si è cercato di sfruttare l'intelligenza artificiale nell'analisi e nella predizione delle performance dei giocatori di calcio. Più precisamente si è utilizzato il deep learning, campo di ricerca dell'IA e insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa. Gli obiettivi di questo progetto sono stati quelli di creare un modello di machine learning che fosse in grado, in seguito ad un opportuno allenamento, di predire, durante il corso di una partita, le performance di un giocatore di calcio nell'ultima parte dell'incontro a cui sta partecipando, utilizzando le informazioni ricavate dai minuti giocati fino a quel momento. Un sistema del genere può avere numerosi utilizzi, partendo dai più immediati come decidere quali giocatori sostituire durante un match e capire come la squadra potrebbe comportarsi nei minuti finali, ma anche aiutare a comprendere quanto possano influire gli eventi che accadono in campo sulla mentalità del calciatore e quindi sulla sua capacità di performare bene. Nei prossimi paragrafi è presente un'introduzione sulla teoria alla base del seguente progetto e nei capitoli successivi è possibile entrare più a fondo nelle specifiche di realizzazione del sistema, partendo dai dati utilizzati fino ad arrivare ai risultati ottenuti.

# 1 Teoria utilizzata

In questo capitolo viene fatta una breve introduzione ai concetti di teoria riguardanti le reti neurali sui quali si basa il lavoro fatto, dando più attenzione alle parti direttamente collegate con il progetto.

## 1.1 Reti neurali

Una rete neurale artificiale (in inglese ANN, artificial neural network, o più semplicemente NN, neural network) è un modello computazionale basato sul modo in cui una rete neurale biologica prende decisioni. Il cervello umano è composto da miliardi di neuroni, unità cellulari interconnesse fra loro che ricevono segnali da altri neuroni attraverso i dendriti, processano le informazioni nel nucleo e le trasmettono ad altre cellule dello stesso tipo attraverso un condotto chiamato assone. Analogamente, una rete neurale artificiale è composta da un certo numero di neuroni artificiali (chiamati anche percettroni) strutturati in modo simile ai neuroni biologici.

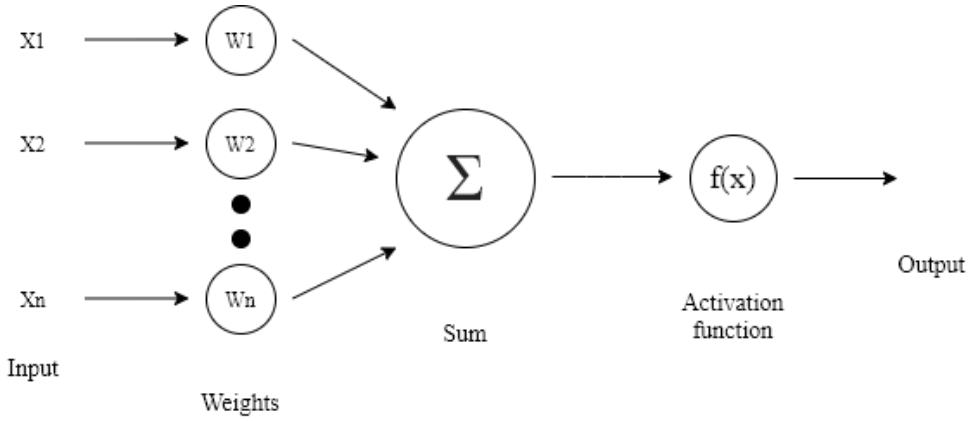


**Figura 1.1:** Schema della struttura di un percettrone

Come si può vedere dalla figura 1.1, un neurone artificiale riceve degli input che vengono moltiplicati per i rispettivi pesi (weights), sommati tra loro e il risultato passa da una funzione di attivazione, di cui si parlerà più in specifico nei successivi paragrafi.

Una rete neurale artificiale può essere vista come un insieme di neuroni divisi su più livelli (layer), suddivisi in:

- **Input layer:** insieme dei neuroni che ricevono i dati di input.
- **Output layer**: insieme dei neuroni che restituiscono il risultato finale computato dalla rete.
- **Hidden layer:** livelli posti tra l'input layer e l'output layer

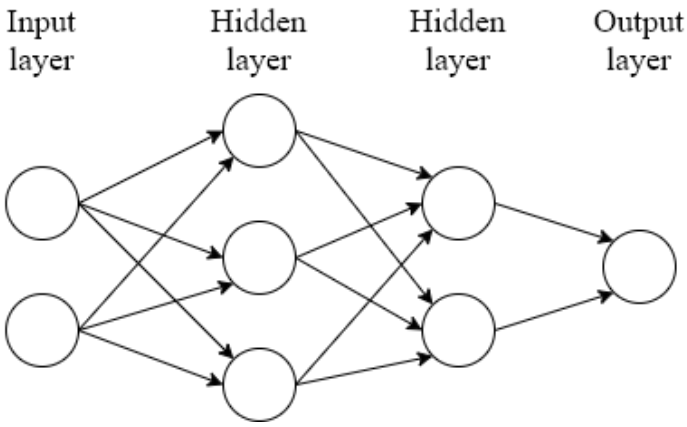


**Figura 1.2:** Esempio di rete neurale con 2 neuroni nel livello di input, 3 neuroni nel primo livello nascosto, 2 neuroni nel secondo livello nascosto e 1 neurone nel livello di output

Secondo il teorema di approssimazione universale, data una qualsiasi funzione *f*(x), esisterà sempre una rete neurale con un hidden layer che, per ogni input x, riesca a produrre come risultato *f*(x), che sia quindi in grado di riprodurre (o ad approssimare fedelmente) la funzione iniziale. Questa è una delle ragioni per la quale, questi modelli computazionali, vengono ampiamente sfruttati in problemi di intelligenza artificiale.

### 1.1.1 Reti neurali ricorrenti

Una rete neurale di base, come quella descritta dalla figura 1.2, viene detta feedforward neural network, proprio per il fatto che l'informazione viene inserita nella rete attraverso l'input layer e viene poi "portata in avanti" fino all'ultimo livello. La rete quindi non presenta cicli.

In una rete neurale ricorrente (in inglese RNN, recurrent neural network), a differenza delle feed-forward neural network, le connessioni tra i neuroni formano un digrafo, che ammette quindi dei cicli. Grazie a questa caratteristica, dati inseriti in precedenza possono influenzare il processo di dati inseriti successivamente, introducendo quindi una sorta di stato interno della rete e di memoria.

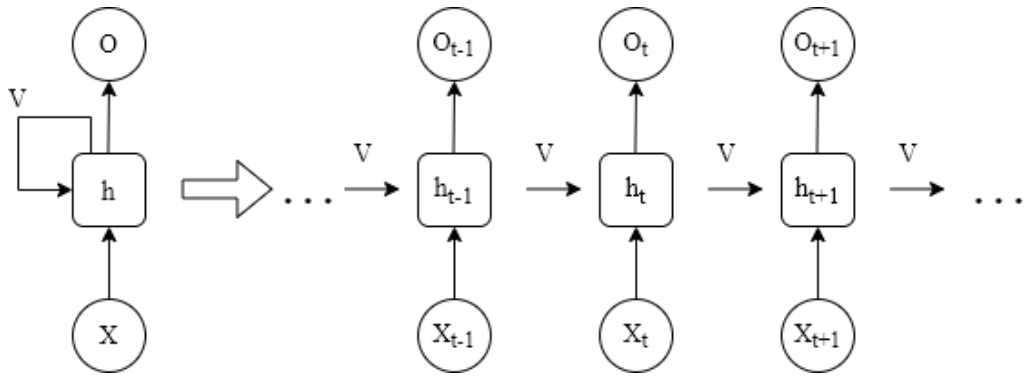


**Figura 1.3**: Schema della struttura di una RNN many-to-many

Questa classe di reti neurali, consente quindi di analizzare sequenze di dati, quindi successioni di esempi anziché esempi singoli. Questa peculiarità le rende adatte a compiti di analisi predittiva su serie consecutive di dati come possono essere video, registrazioni audio, documenti scritti. Dalla figura 1.3 è possibile osservare il comportamento di una RNN in versione many-to-many, nella quale ogni input

genera un output che viene reinserito nella rete per influenzare i dati inseriti successivamente. In versione many-to-one invece, l'output è unico e viene generato all'ultimo step della sequenza, variante utile per esempio nella classificazione di serie di dati, come potrebbe essere capire di che genere musicale faccia parte una certa canzone.

### 1.1.2 LSTM networks

Le reti neurali ricorrenti soffrono il problema della memoria a breve termine, quindi faticano a portare avanti informazioni utili se si ha a che fare con sequenze troppo estese. Le long short-term memory networks (LSTM) sono state create per risolvere questo problema sfruttando delle celle in grado di salvare informazioni utili all'esterno del flusso della rete. Queste celle sono formate da 3 componenti principali:

- **Input gate:** aggiunge informazioni alla cella.
- **Forget gate**: rimuove le informazioni non più necessarie.
- **Output gate**: seleziona e fornisce informazioni alla rete.

## 1.2 Funzioni di attivazione

La funzione di attivazione è una componente fondamentale dei neuroni artificiali. Essa determina la forma dell'output del neurone, quindi anche la forma dell'output dell'intera rete e stabilisce se il neurone deve essere attivo o spento (quindi se deve contribuire o meno al calcolo del risultato finale). Una funzione di attivazione è solitamente una funzione non lineare, dato che, con solo funzioni lineari, non importa il numero di layer nella rete, l'ultimo livello sarebbe semplicemente una funzione lineare del primo livello, dato che combinazioni lineari di funzioni lineari restituiscono sempre una funzione lineare.

Alcuni esempi di funzioni di attivazione non lineari sono:

- Sigmoide
- Tangente iperbolica
- ReLU (Rectified Linear Unit)
- Leaky ReLU

Una funzione di attivazione molto usata nel layer di output è la Softmax (1.4), funzione che trasforma un vettore di dimensione K contenente dei punteggi (logits), in un vettore di dimensione K contenente dei valori di probabilità, la quale somma è uguale ad 1. Si tratta di una funzione utilizzata in problemi di binary o multi class classification, dato che restituisce la probabilità calcolata dalla rete che un certo input corrisponda ad una certa classe.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \textbf{(1.4)}$$

## 1.3 Funzioni di perdita

La funzione di perdita (loss function) calcola essenzialmente il livello di prestazioni del modello confrontando ciò che il modello predice con il valore effettivo che dovrebbe produrre. La scelta della loss function è una parte cruciale in fase di

costruzione di un modello di deep learning ed è direttamente collegata alla forma dell'output che si decide di adottare. Durante il lavoro svolto in questo progetto, sono state usate principalmente 2 diverse funzioni di perdita:

- **Cross Entropy:** loss function usata in concomitanza alla softmax activation function in problemi di multiclass classification. La funzione (1.5) descrive l'entropia di una distribuzione di probabilità **y'** rispetto ad un'altra distribuzione **y**.

$$H(\mathbf{y}', \mathbf{y}) = -\sum_i y_i' \log y_i \quad \textbf{(1.5)}$$

- **Mean Absolute Error:** loss function utilizzata in problemi di regressione. La funzione (1.6) misura l'errore assoluto medio dei valori predetti dal modello di deep learning rispetto ai dati corretti.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n} \quad \textbf{(1.6)}$$

## 1.4 Algoritmi di ottimizzazione

Riprendendo i concetti di weight, funzione di attivazione e di funzione di perdita enunciati nei paragrafi 1.1, 1.2 e 1.3, si può riassumere che, l'obiettivo di una rete neurale, sia cercare di minimizzare il valore della loss function calcolata tra l'output della stessa (dipendente dagli input, dalle weight dei vari neuroni e dalle activation function) e l'output che avrebbe dovuto restituire. La minimizzazione della funzione di perdita avviene mediante certi algoritmi di ottimizzazione che permettono di trovare le weight che rendono il valore della loss function il più basso possibile.

Il Gradient Descent è l'algoritmo di ottimizzazione di base che minimizza una certa funzione muovendosi iterattivamente verso il minimo della stessa, sfruttando la derivata per individuare in quale direzione spostarsi.

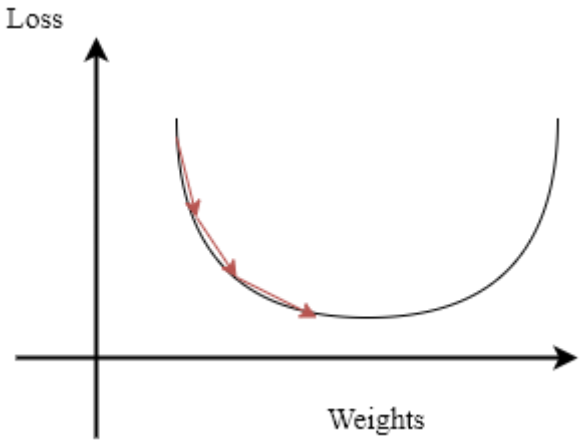


**Figura 1.7:** Rappresentazione grafica del funzionamento dell'algoritmo Gradient Descent

Dalla figura 1.7 è possibile intuire facilmente il funzionamento dell'algoritmo. Ogni freccia rappresenta uno step e la grandezza di ogni step è un valore fondamentale chiamato learning rate. La scelta di questo parametro determina il buon funzionamento dell'algoritmo. Con un learning rate troppo basso, il tempo di

convergenza rischia di essere troppo alto, con un valore troppo alto invece si rischia di non raggiungere la convergenza.

L'algoritmo utilizzato nel progetto di cui questo documento tratta, si tratta del Adam, metodo sviluppato da Diederik Kingma e Jimmy Ba, che, come gli stessi fanno notare, <<...è semplice da implementare, efficiente dal punto di vista computazionale, ha pochi requisiti di memoria, è invariante al riscalamento diagonale dei gradienti ed è adatto per problemi che sono grandi in termini di dati e/o parametri>>. [1]

# 2. Raccolta e trasformazione dei dati

In questo capitolo viene esposta tutta la fase di ricavo e di trasformazione dei dati, partendo dalla fonte dalla quale si sono estratti, mostrando poi la loro forma originale, i metodi per l'estrazione, l'inserimento dei dati in forma ordinata in un database e infine la loro modellazione in un formato adatto ad essere utilizzati nel modello di deep learning creato.

## 2.1 Fonte e forma originale dei dati

I dati utilizzati nel progetto sono stati ricavati dal sito it.whoscored.com, portale dedicato al gioco del calcio che fornisce ogni genere di informazione riguardante ogni partita di ogni campionato o coppa di una qualsiasi nazione. Dalla parte del sito dedicato ai dettagli di una singola partita, accedendo alla sezione "Match Centre" si possono consultare un grande numero di informazioni riguardanti ogni singolo giocatore presente nell'incontro in questione. Tutti questi dati sono contenuti in un file JSON, visualizzabile nel sorgente della pagina.

In dettaglio, nel file, sono presenti le seguenti informazioni:

- Partita:
    - Data e ora
    - Stadio
    - Informazioni sull'arbitro
    - Punteggio a metà partita e finale
    - Squadre partecipanti
- Squadre:
    - Nome del team
    - Nome dell'allenatore
    - Lista dei giocatori
    - Campo di gioco (casa o trasferta)
    - Formazioni utilizzate (in ordine di utilizzo durante la partita)
    - Statistiche generali del team (rating, tiri, passaggi...)
- Giocatori:
    - Nome del giocatore
    - Posizione di partenza
    - Età, altezza, peso
    - Uomo partita
    - Statistiche generali del giocatore (rating, tiri, passaggi...)
- Eventi:
    - Istante in cui è avvenuto l'evento
    - Punto del campo in cui è avvenuto l'evento
    - Squadra coinvolta nell'evento
    - Tipo di evento (passaggio corretto, tiro sbagliato, fallo...)


**Figura 2.1:** Esempio di alcune parti del file JSON di una partita di Premier League. In ordine, informazioni sulla squadra, informazioni su uno specifico giocatore, informazioni su uno specifico evento.

In particolare, in questo progetto, sono state sfruttate molto le informazioni riguardanti le variazioni dei rating dei giocatori lungo il corso della partita e gli eventi in cui sono coinvolti.

## 2.2 Web Scraping

Il web scraping, detto anche web harvesting o web data extraction, è una tecnica automatizzata di estrazione dei dati in formato leggibile da siti web. Il web scraping (in italiano letteralmente “raschiare la rete”) può essere fatto manualmente dall’utente, ma solitamente con il termine ci si riferisce a procedure automatizzate attraverso l’utilizzo di bot o web crawler. Si tratta di una tecnica di data mining che consiste nella visita di un certo sito da parte di un software, simulando il comportamento umano, e nell’estrazione dalla pagina web delle informazioni ricercate, analizzando direttamente l’interfaccia presentata all’utente o esaminando il sorgente della pagina. Solitamente, i dati raccolti vengono poi salvati in database per la successiva fase di studio.

Nel caso del progetto trattato in questo documento, per effettuare web scraping è stato usato Selenium, libreria utilizzata principalmente per l’automazione dei test delle pagine web, ma, date le funzioni che mette a disposizione, utilizzabile anche per effettuare del data mining. Selenium permette la navigazione automatica di un qualsiasi sito, simulando il comportamento umano interagendo con la pagina web, per esempio, premendo bottoni o inserendo informazioni nelle form. Attraverso lo sfruttamento di questa libreria in Java, sono state visitate tutte le pagine dedicate ad ogni partita dei 4 maggiori campionati europei (Serie A, Premier League, Bundesliga, LaLiga) nell’anno 2019/2020, ricavando da ognuna un file JSON contenente tutte le informazioni necessarie allo studio (vedi paragrafo 2.1 per i dettagli sul file JSON). Tutti i documenti sono stati salvati localmente per essere poi sottoposti ad una procedura di parsing dei dati e di inserimento in un database, di cui si possono conoscere le specifiche nel paragrafo 2.3 dedicato.

## 2.3 Database

Il database è stato realizzato in PostgreSQL, uno dei più diffusi database relazionali open source, preferito ad altri DBMS per performance, esperienze personali passate e per la presenza del tool pgAdmin, il quale rende le interazioni dirette con il database molto semplici, veloci e chiare, fondamentale in fase di modellazione dei dati. L'inserimento delle informazioni all'interno della base di dati è stato completamente automatizzato attraverso uno script Java, il quale, durante il parsing dei file JSON, inserisce tutti i nuovi dati riguardanti le squadre, i giocatori, le partite e gli eventi.

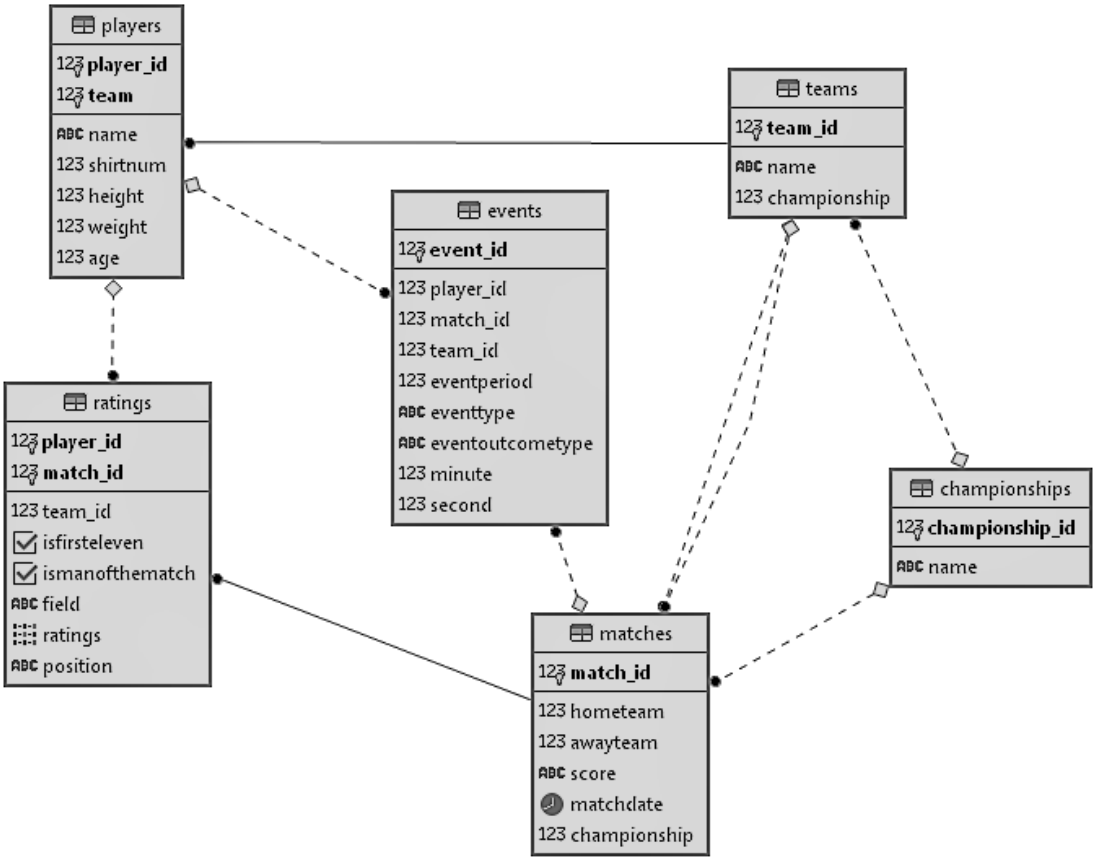


**Figura 2.2:** Struttura del database utilizzato.

In figura 2.2 è possibile analizzare l'organizzazione logica del database. La tabella ratings contiene ogni performance di ogni giocatore, individuata appunto dall'id del giocatore e dall'id della partita giocata. La tabella events contiene ogni singolo evento rilevante accaduto in ogni singola partita, individuato da un id univoco e contenente 3 chiavi esterne per indicare il giocatore coinvolto nell'evento (e per quale squadra giocava in quella partita) e la partita in cui l'evento è avvenuto. L'attributo eventtype indica il tipo di evento (passaggio, tiro, fallo...) mentre l'attributo eventoutcometype ne dà una breve descrizione (successo, insuccesso, subito...). Come si può notare, la tabella players presenta 2 chiavi, l'id del giocatore e l'id del team, questo per gestire il fatto che un giocatore può cambiare squadra nel corso del campionato.

## 2.4 Modellazione in formato CSV

Per ricavare informazione dai semplici dati inseriti nel database, è stato necessario un lavoro di modellazione degli stessi, raggruppando valori correlati fra loro e trascrivendoli in una forma adatta ad essere usata come input in un modello di deep learning. Il formato scelto è stato il CSV (comma separated value), facile da interpretare dal punto di vista umano e di facile utilizzo in ambito di machine learning, dato l'ampio supporto che le librerie di deep learning forniscono per lavorare con dati strutturati in questo modo. I dettagli sulla modellazione dei dati sono specificati nel prossimo capitolo dedicato alla progettazione della rete neurale, più precisamente nel paragrafo 3.1 dedicato ai dati di input.

# 3 Progettazione del modello

Questo capitolo è dedicato alla progettazione del modello di deep learning per risolvere il problema della predizione delle performance dei giocatori di calcio nel corso di una partita. Vengono mostrati i dettagli riguardanti il dataset di input, l'architettura della rete neurale e le tecniche di training utilizzate.

Come si leggerà nel capitolo, per affrontare il problema è stato scelto un approccio con una rete LSTM, data la natura dei dati da analizzare e prevedere (sequenze di dati in ordine temporale).

Il progetto è stato sviluppato completamente in Java, attraverso l'utilizzo della libreria open source DeepLearning4j.

## 3.1 Input della rete

Come anticipato nel capitolo 2, paragrafo 2.4, i dati di input utilizzati sono stati precedentemente modellati in formato CSV e di seguito ne vengono specificati i dettagli. Sono state prese in analisi ogni performance di ogni giocatore in ogni partita, con il vincolo che il calciatore abbia partecipato per più di 45 minuti in quel determinato incontro (nei casi in cui ciò non era rispettato, la performance è stata scartata), utilizzando i dati riguardanti il 75% del tempo giocato come features (input) della rete e il restante 25% come label (dati da prevedere e restituire in output). Per ogni performance diversa (quindi coppia giocatore/partita) sono stati creati 3 file CSV:

- Uno contenente le informazioni sul 75% del tempo giocato dal calciatore.
- Due contenenti le informazioni sul restante 25% del tempo giocato dal calciatore (codificate in 2 diversi modi)

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Rating | PlayerId | GoalScored | GoalTaken | Card | Foul | PassSuc | PassUnsuc | BallTouchSuc | BallTouchUnsuc |
| 2 | 6.01 | 362555 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 6.09 | 362555 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 4 | 6.17 | 362555 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 6.3 | 362555 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| 6 | 6.3 | 362555 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 7 | 6.3 | 362555 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

| K | L | M | N | O | P | Q | R | S | T | U |
|---|---|---|---|---|---|---|---|---|---|---|
| BallRecovery | TackleSuc | TackleUnsuc | TakeOnSuc | TakeOnUnsuc | AerialSuc | AerialUnsuc | Goal | MissedShot | SavedShot | Save |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figura 3.1:** Esempio delle prime 7 righe di un file CSV contenente i dati sulla performance di un giocatore in una partita

Come si può vedere dalla figura 3.1, la prima colonna contiene i rating del giocatore calcolati lungo il corso della partita, in ordine cronologico. Le altre colonne contengono il conteggio degli eventi successi dall'ultimo rating misurato fino al rating corrente, divise per tipo di evento. Per esempio, nel caso della figura 3.1, tra il primo rating e il secondo, il giocatore in questione è migliorato di 0.08 punti e ha eseguito 4 passaggi corretti. Gli altri due file contengono uno o 0 o 1, per indicare se il giocatore è peggiorato o migliorato nell'ultimo 25% del tempo giocato, l'altro contiene la media del rating sempre nell'ultimo 25% del tempo giocato. Per capire se il giocatore è migliorato, si confronta la media contenuta nel secondo file con l'ultimo rating del primo 75% del tempo giocato (ultimo rating presente nel file dei

dati di input). Si è deciso di salvare due label diverse (una binaria e una reale) per confrontare modelli basati sulla classificazione e modelli basati sulla regressione.

### 3.1.1 Bilanciamento del dataset

Durante le prime prove di utilizzo del dataset, si è notato un problema di sbilanciamento tra le classi 0 e 1. Per ogni campionato sono presenti circa un totale di 8000 performance, nel circa 66% di queste, il giocatore è migliorato nei minuti finali da lui giocati, implicando quindi un valore pari a 1 come classe. Ciò ha portato ad una classificazione errata nei primi modelli testati, i quali mostravano un'apparente buona accuratezza sulle predizioni pari a circa 0.75, ma causata solamente dal fatto che ogni singola sequenza veniva classificata come appartenente alla classe 1, la più numerosa. Per risolvere la seguente problematica si è fatto ricorso alla tecnica dell'oversampling, la quale consiste nell'aumentare il numero degli esempi appartenenti alla classe meno numerosa semplicemente creando delle copie degli esempi che già si posseggono. Questa tecnica è stata preferita al undersampling (diminuire il numero di esempi della classe più numerosa) per evitare la perdita di informazione.

### 3.1.2 Masking e padding

Come anticipato all'inizio del capitolo, come modello per risolvere il problema della predizione delle performance di un giocatore di calcio, è stata scelta un'architettura LSTM. Una rete neurale di questo tipo è in grado di ricevere in input e di restituire in output sequenze di dati. Di base, una LSTM riceve sequenze di lunghezza predefinita e restituisce sequenze di uguale dimensione (configurazione many-to-many), ma grazie alle tecniche di masking e padding è possibile controllare a proprio piacimento il comportamento della rete per quanto riguarda input e output.

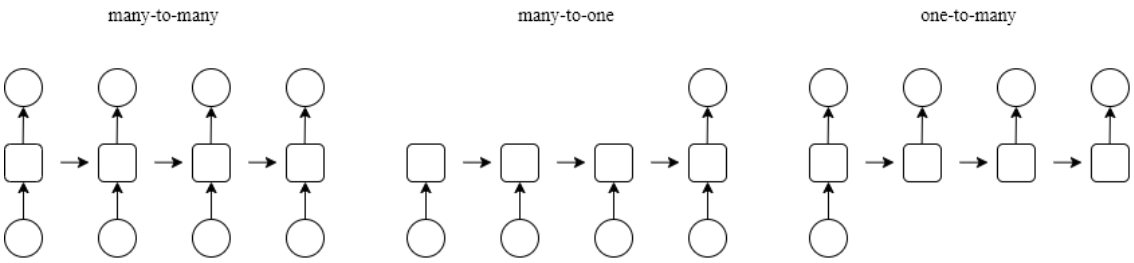


**Figura 3.2:** Rappresentazione grafica di alcune possibili configurazioni di una rete LSTM.

Nel caso del problema trattato in questa tesi, è necessaria una struttura many-to-one, dato che l'obiettivo è, data una sequenza di rating ed eventi in cui un giocatore è coinvolto, predire correttamente se il giocatore migliorerà o peggiorerà (0 o 1) in seguito a questa serie. Combinando masking e padding è facilmente ottenibile un'architettura come quella citata.

L'idea dietro al padding è semplice, aggiungere all'inizio o alla fine di una sequenza una serie di 0, in modo da rendere ogni successione (sia in input che in output) della stessa lunghezza. Limitarsi a questa tecnica creerebbe dei problemi durante il training. Attraverso il masking è possibile specificare se un certo dato di input o di output è presente o se si tratta di semplice padding. Ciò viene implementato con due array (uno per l'input e uno per l'output) contenenti 0 se il corrispondente timestep non è presente, 1 altrimenti.

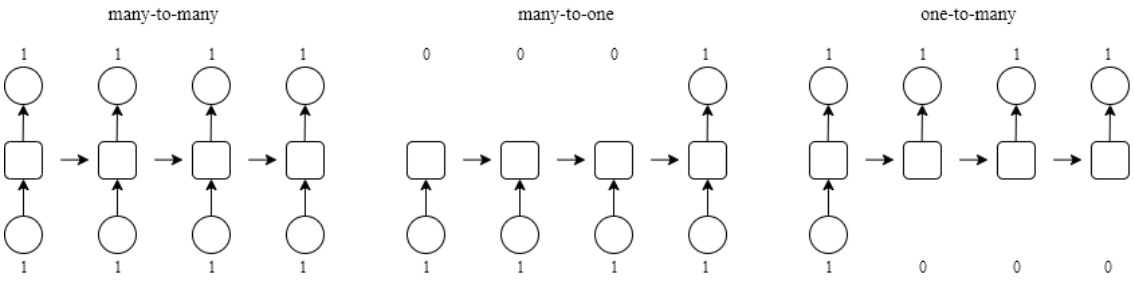


**Figura 3.3:** Esempio di array di masking per 3 diverse configurazioni una rete LSTM

## 3.2 Architettura della rete

Come già anticipato nei precedenti paragrafi, l'approccio scelto per cercare di risolvere il problema della predizione delle performance dei giocatori di calcio, è stato quello di affidarsi ad una rete la cui parte principale è l'iniziale layer LSTM. Sono stati costruiti 2 modelli diversi basati su questa architettura, uno incentrato sulla classificazione e uno sulla regressione, in modo da testare i diversi risultati dei due approcci. I dettagli sulle differenze tra le due soluzioni proposte sono definiti nei prossimi paragrafi.

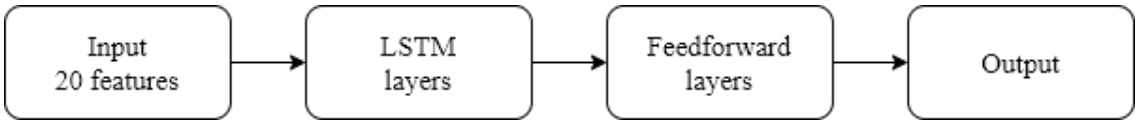


**Figura 3.4:** Rappresentazione grafica dell'architettura della rete

La figura 3.4 fornisce una rappresentazione riassuntiva dell'architettura adottata. Sono presenti un livello di input, che riceve sequenze di dati sulla performance di un giocatore, con ogni timestep contenente 20 informazioni diverse (features) e un livello di output, diverso a seconda se l'obiettivo sia la classificazione o la regressione. La parte centrale della rete è composta da N layer di tipo LSTM, con il compito di trovare pattern nascosti nelle sequenze e da M layer semplici di tipo feedforward, con l'obiettivo di migliorare la capacità di apprendimento della rete. I numeri di layer N e M sono stati variati durante il training come è stato variato anche il numero di neuroni per ogni singolo livello in modo da trovare la configurazione migliore. In fase iniziale è stato testato un modello composto da un unico layer LSTM ma già dai primi test è stato chiaro che un approccio simile non consentiva alla rete di apprendere a sufficienza, portando quindi alla decisione di aumentare non solo il numero di livelli LSTM, ma di aggiungere anche dei livelli di feedforward. Nei livelli LSTM, è stato impostato un valore di dropout come forma di prevenzione dell'overfitting. Si tratta di un metodo di regolazione che consiste nell'escludere dalla rete alcuni nodi in modo randomico (secondo un certo valore di probabilità), così da simulare un'architettura di rete leggermente diversa ad ogni iterazione.

Il valore iniziale dei pesi è stato assegnato attraverso la Xavier initialization, quindi in maniera casuale (non conoscendo i dati a priori) selezionandoli da una distribuzione Gaussiana.

L’algoritmo utilizzato per l’aggiornamento dei pesi è l’Adam, di cui si parla nel capitolo 1, paragrafo 1.4.

### 3.2.1 Classificazione

Per il modello basato sulla classificazione, nel layer di output è stata usata una funzione di attivazione Softmax in combinazione alla funzione di perdita Multiclass Cross Entropy. Il livello di output restituisce quindi 2 valori, rappresentanti la probabilità di una sequenza di appartenere alla classe 0 (il giocatore è peggiorato) o alla classe 1 (il giocatore è migliorato).

### 3.2.2 Regressione

Per il modello basato sulla regressione, nel layer di output è stata usata la funzione di attivazione tangente iperbolica in combinazione alla funzione di perdita Mean Absolute Error. Il livello di output restituisce quindi 1 valore, rappresentante la media del rating del giocatore nei minuti successivi a quelli inseriti come input.

### 3.2.3 Feedforward

È stato inoltre progettato un modello più semplice, composto di soli layer feedforward, per poter misurare le prestazioni di una rete LSTM in confronto ad una rete classica in casi in cui si trattano sequenze temporali di dati. Come input per questo modello sono stati utilizzati solo i valori di rating del calciatore, inserendoli contemporaneamente nella rete come features diverse di un unico esempio. Per rendere ogni input della stessa lunghezza, è stata scelta la grandezza della sequenza maggiore e a tutte le serie di dimensioni minori sono stati aggiunti in coda tanti 0 quanti necessari a raggiungere l’estensione voluta.

## 3.3 Training

La fase di training e tuning dei parametri si è rivelata essere la più dispendiosa in termini di tempo, data la grande quantità di test diversi necessari da eseguire e la bassa velocità di training effettivo del modello a causa della complessità dello stesso e dell’hardware utilizzato. Oltre alla modifica dei vari parametri delle reti, sono stati eseguiti test utilizzando diversi dataset, ognuno contenente performance di un determinato campionato. Si hanno avuto a disposizione 4 insiemi diversi di dati, provenienti da Premier League, Serie A, LaLiga e Bundesliga, ognuno contenente circa 11000 performance in seguito al bilanciamento, divise quindi a metà fra classe 0 e classe 1. La forma dei dataset utilizzati per il training è specificata nel paragrafo 3.1.

### 3.3.1 Ottimizzazione dei parametri

L’ottimizzazione della rete si è basata sulla ricerca delle migliori combinazioni tra numero di livelli e neuroni e valore di certi parametri fondamentali (hyperparameters) che controllano vari aspetti del training del modello. Di seguito si parla più approfonditamente dei parametri sopra citati:

- **Learning rate:** come è stato detto nel paragrafo 1.4, il learning rate determina il buono o cattivo funzionamento dell’algoritmo di minimizzazione di una funzione. Chiaramente, durante i test, si è visto che un valore di learning rate troppo alto portava il modello a non riuscire ad abbassare il valore della funzione di perdita, quindi di fatto a non imparare nulla dai dati

durante l'allenamento, mentre un valore troppo basso portava a tempi di convergenza troppo lunghi. Si è visto che, generalmente, nel caso dei modelli costruiti in questo progetto, i valori migliori stavano tra 0.001 e 0.0001, dando il giusto compromesso tra velocità di training e buone performance della rete.

- **Batch size:** la grandezza della batch indica con quanti esempi alla volta la rete viene allenata. Per esempio, se si hanno in totale 1000 esempi per la fase di training e si sceglie una batch size pari a 50, ogni 50 esempi inseriti nella rete i pesi della stessa verranno aggiornati e si passerà ai successivi 50 esempi. Saranno necessarie 20 iterazioni per allenare il modello su tutti i dati dedicati al training. Un ciclo completo di training su tutti i dati, viene detto epoca. I vantaggi di usare un valore di batch size minore del numero totale del dataset di training, sono che è necessaria meno memoria, dato che si usano mediamente un numero minore di dati, e che la rete impara più velocemente, dato che i pesi vengono aggiornati alla fine di ogni batch e non solo dopo la fine di un'intera epoca. Nel caso del progetto trattato in questo documento, i test sono stati effettuati con valori di batch size compresi tra 10 e 128 e sono stati ritenuti ottimali valori pari a 32/64.
- **Dropout:** come anticipato nel paragrafo 3.2, il dropout è un metodo di controllo dell'overfitting. Per overfitting si intende la tendenza del modello ad adattarsi perfettamente ai dati di training, apprendendone caratteristiche specifiche, rendendolo quindi meno generale e meno performante quando tratta dei dati sconosciuti. Attraverso il dropout, casualmente alcuni nodi non vengono utilizzati durante alcune fasi del training, evitando l'eccessivo apprendimento dal dataset di training. Durante i vari test, è risultato ottimo un valore di dropout pari a 0.8, il quale indica una probabilità dell'80% di mantenere attivo un certo neurone (da notare che la libreria DeepLearning4j utilizza una probabilità di dropout inversa rispetto a molte altre librerie, che spesso richiedono la probabilità che un nodo venga escluso, quindi in questo caso sarebbe stato usato un valore pari a 0.2).
- **Numero di epoche:** introdotte parlando delle batch, le epoche sono un altro elemento fondamentale nell'allenamento di una rete neurale. Nella maggior parte dei casi, una singola epoca non basta per allenare la rete neurale, è quindi necessario ripetere più volte la fase di training sugli stessi dati per permettere al modello di avere delle buone performance. Con un numero troppo basso di epoche si rischia l'underfitting, con un numero troppo alto l'overfitting. Un modo per scegliere questo parametro è l'utilizzo di una configurazione early stopping, come è stato scelto di fare in questo progetto. Una configurazione di questo tipo consiste nel continuare ad avanzare con le epoche finchè le performance continuano a migliorare e fermarsi quando peggiorano o quando i miglioramenti sono nulli o non rilevanti.

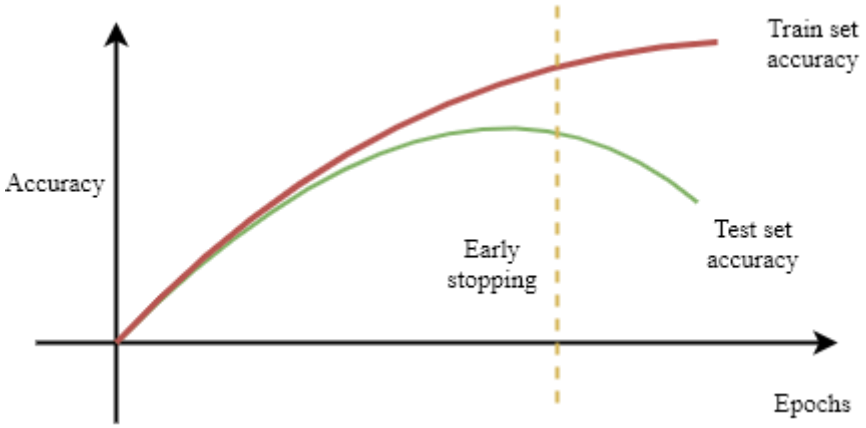


**Figura 3.4:** Rappresentazione grafica del funzionamento di una configurazione early stopping.

Per misurare le performance del modello si possono utilizzare varie tecniche e per il caso dei modelli trattati in questa tesi, è stata usata l'accuracy nel modello basato sulla classificazione e il Mean Absolute Error per il modello basato sulla regressione. Nel primo caso, il training veniva fermato dopo 5 epoche senza miglioramenti, nel secondo invece dopo 3 epoche senza un miglioramento minimo di 0.00001. La libreria consente inoltre di impostare un tempo massimo di training, di scegliere ogni quante epoche verificare le performance e di salvare il miglior modello ottenuto durante il training.

# 4 Valutazione e confronto

In questo capitolo vengono analizzate le valutazioni dei 3 modelli utilizzati su vari campionati diversi e vengono confrontati tra loro i 3 diversi approcci al problema. Sono stati selezionati i risultati più rilevanti tra i tanti ottenuti dalle varie configurazioni testate. Nella parte conclusiva viene mostrato un quarto modello, derivato dal primo, in cui cambia leggermente l'approccio al problema, nel tentativo di ottenere dei risultati migliori.

## 4.1 Classificazione

Per ogni modello, il training è iniziato testando la rete su uno stesso dataset (Premier League) per avere una prima idea di quale configurazione potesse funzionare al meglio con il problema che si vuole risolvere. Sono state testate varie configurazioni della rete, variando il numero dei layer e dei neuroni dei singoli layer.

| Test | LSTM layers | LSTM layer size | Dense layers | Dense layers size | Learning rate | Batch size | Training accuracy | Test accuracy |
|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 50 | 1 | 150 | 0.001 | 32 | 0.6194 | 0.6335 |
| **2** | 1 | 50 | 2 | 150 | 0.001 | 32 | 0.6210 | 0.6280 |
| **3** | 2 | 50 | 1 | 150 | 0.001 | 32 | 0.6254 | 0.6290 |
| **4** | 1 | 100 | 1 | 200 | 0.001 | 32 | 0.6206 | 0.6240 |
| **5** | 2 | 50 | 2 | 150 | 0.001 | 32 | 0.6279 | 0.6115 |
| **6** | 1 | 100 | 2 | 150 | 0.0005 | 32 | 0.6284 | 0.6300 |

**Tabella 4.1:** La tabella mostra i risultati ottenuti dal modello basato sulla classificazione, con diverse configurazioni, utilizzando il dataset dedicato alla Premier League. I test 1 e 6 hanno mostrato i risultati migliori.

In seguito ai test descritti dalla tabella 4.1, è emerso il fatto che le configurazioni meglio performanti fossero la numero 1 e la numero 6. Si è proceduto con l'approfondimento della configurazione 1 rispetto agli altri 3 dataset (Serie A, LaLiga, Bundesliga).

| Campionato | Train accuracy | Test accuracy |
|---|---|---|
| Premier League | 0.6194 | 0.6335 |
| LaLiga | 0.6043 | 0.5665 |
| Serie A | 0.6019 | 0.5980 |
| Bundesliga | 0.5910 | 0.5815 |

**Tabella 4.2:** La tabella mostra i risultati ottenuti dalla configurazione 1 della tabella 4.1 rispetto ad ogni campionato preso in considerazione.

Come si può vedere dalla tabella 4.2, la configurazione 1 della rete ha avuto dei pessimi risultati sugli altri campionati. Ciò potrebbe indicare che l'alta accuratezza del modello sui dati riguardanti la Premier League sia solo un caso. Si è deciso quindi di provare ad approfondire la configurazione 6.

| **Campionato** | **Train accuracy** | **Test accuracy** |
|---|---|---|
| Premier League | 0.6284 | 0.6300 |
| LaLiga | 0.5971 | 0.5795 |
| Serie A | 0.6019 | 0.6115 |
| Bundesliga | 0.6022 | 0.5950 |

**Tabella 4.3:** La tabella mostra i risultati ottenuti dalla configurazione 6 della tabella 4.1 rispetto ad ogni campionato preso in considerazione.

Analizzando le tabelle 4.2 e 4.3, sembra che per alcuni campionati la classificazione delle performance sia molto più complicata. Per Premier League e Serie A si riescono a raggiungere valori di accuracy sopra al 60% facilmente, con poche epoche di training, mentre per quanto riguarda LaLiga e Bundesliga, è difficile superare il 60% di accuracy anche dopo numerose epoche di training. La configurazione 6 è comunque quella che è dimostrato performance migliori mediamente su tutti i campionati.

## 4.2 Regressione

Il processo di training per il modello basato sulla regressione è stato analogo a quello per il modello basato sulla classificazione. Nelle tabelle viene riportata l'accuracy del modello, calcolata in questo modo: il modello restituisce in output una predizione del futuro valor medio di rating del giocatore, il quale viene messo a confronto con il valor medio di rating calcolato sui dati reali, se indicano entrambi un miglioramento o un peggioramento, la predizione viene valutata come corretta, viene valutata errata se indicano uno un miglioramento e l'altro un peggioramento. Non viene quindi presa direttamente in considerazione la distanza tra il valore predetto e il valore reale in fase di valutazione, ma viene però utilizzata in fase di training per migliorare il modello (Mean Absolute Error).

| Test | LSTM layers | LSTM layer size | Dense layers | Dense layers size | Learning rate | Batch size | Test MAE | Test accuracy |
|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 50 | 1 | 150 | 0.001 | 32 | 0.0536 | 0.5415 |
| **2** | 1 | 50 | 2 | 150 | 0.001 | 32 | 0.0522 | 0.5285 |
| **3** | 1 | 50 | 0 | - | 0.001 | 32 | 0.0513 | 0.5435 |
| **4** | 2 | 50 | 0 | - | 0.001 | 32 | 0.0518 | 0.5350 |
| **5** | 1 | 100 | 0 | - | 0.0005 | 32 | 0.0508 | 0.5400 |
| **6** | 1 | 50 | 0 | - | 0.001 | 64 | 0.0558 | 0.5540 |
| **7** | 1 | 50 | 0 | - | 0.0005 | 64 | 0.0553 | 0.5075 |

**Tabella 4.4:** La tabella mostra i risultati ottenuti dal modello basato sulla regressione, con diverse configurazioni, utilizzando il dataset dedicato alla Premier League. I test 3 e 6 hanno mostrato i risultati migliori.

In seguito ai risultati ottenuti dai test ed esposti nella tabella 4.4, è emerso che le configurazioni migliori fossero quelle prive di livelli di tipo feedforward. Si è quindi proceduto con l'approfondimento delle suddette architetture. In seguito vengono riportati i risultati ottenuti dalla configurazione numero 6.

| Campionato | Test accuracy |
|---|---|
| Premier League | 0.5540 |
| LaLiga | 0.5190 |
| Serie A | 0.5450 |
| Bundesliga | 0.4745 |

**Tabella 4.5:** La tabella mostra i risultati ottenuti dalla configurazione 6 della tabella 4.4 rispetto ad ogni campionato preso in considerazione.

## 4.3 Feedforward

L'ultimo modello testato, è stato quello basato solamente su livelli di tipo feedforward. I dettagli sugli input utilizzati in questo modello sono specificati nel paragrafo 3.2.3, diversi dai dati utilizzati negli altri 2 modelli a causa del fatto che un layer feedforward non può gestire di base delle sequenze. Lo scopo di questo modello è stato quello di, più che risolvere il problema posto all'inizio del progetto, verificare quanta differenza può fare sfruttare una rete di tipo LSTM (adatta a trattare serie di dati, grazie all'implementazione di una sorta di memoria) rispetto ad una rete classica quando si ha a che fare con successioni di dati correlati tra loro. Il seguente modello testato si basa sulla classificazione, analogamente al primo modello testato. Non è stata implementata una rete neurale feedforward basata sulla regressione.

| Test | Dense layers | Dense layers size | Learning rate | Batch size | Training accuracy | Test accuracy |
|---|---|---|---|---|---|---|
| **1** | 2 | 100 | 0.001 | 32 | 0.5795 | 0.5835 |
| **2** | 2 | 100 | 0.0001 | 32 | 0.5761 | 0.5845 |
| **3** | 2 | 200 | 0.0001 | 32 | 0.5724 | 0.5840 |
| **4** | 2 | 200 | 0.0001 | 64 | 0.5747 | 0.5835 |
| **5** | 2 | 400 | 0.0001 | 32 | 0.5779 | 0.5830 |
| **6** | 3 | 400 | 0.0001 | 32 | 0.5749 | 0.5865 |

**Tabella 4.6:** La tabella mostra i risultati ottenuti dal modello feedforward basato sulla classificazione, con diverse configurazioni, utilizzando il dataset dedicato alla Premier League. I test 3 e 6 hanno mostrato i risultati migliori.

Analizzando la tabella 4.6, si nota che nonostante le varie modifiche all'architettura, l'accuratezza del modello non cambia in modo significativo. La configurazione 6 è quella con le performance leggermente migliori, è quindi stata approfondita rispetto agli altri campionati.

| Campionato | Train accuracy | Test accuracy |
|---|---|---|
| Premier League | 0.5749 | 0.5865 |
| LaLiga | 0.5681 | 0.5615 |
| Serie A | 0.5711 | 0.5790 |
| Bundesliga | 0.5469 | 0.5600 |

**Tabella 4.7:** La tabella mostra i risultati ottenuti dalla configurazione 6 della tabella 4.6 rispetto ad ogni campionato preso in considerazione.

## 4.4 Confronto

In seguito ai numerosi test eseguiti, è possibile comparare tra loro i vari modelli adottati. Analizzando i risultati riportati nei precedenti paragrafi, è immediato notare che il modello LSTM basato sulla classificazione performi molto meglio del modello LSTM basato sulla regressione, con la prima architettura che mediamente raggiunge valori di accuratezza maggiori dell'8.25% rispetto alla seconda. L'operazione di predire lo specifico valore di rating futuro sembra quindi più complicato rispetto a capire semplicemente se un giocatore migliorerà o peggiorerà, potrebbe richiedere un numero di informazioni maggiori o un training specifico per giocatore, per apprendere meglio come uno specifico giocatore si comporta in differenti scenari. Per quanto riguarda il modello feedforward, presenta delle performance peggiori della sua controparte basata su layer LSTM in ogni campionato analizzato, dando così la conferma che le reti neurali di tipo RNN riescono a gestire meglio flussi di dati, sequenze temporali di informazioni, rispetto alle reti neurali di base.

## 4.5 Miglioramenti al modello

In un secondo momento, per cercare di apportare dei miglioramenti al modello, sono stati testati altri differenti approcci al problema. Nella prima versione del modello basato sulla classificazione, le cui valutazioni sono riportate nel paragrafo 4.1, l'obiettivo era prevedere un miglioramento o un peggioramento nelle performance di un giocatore durante l'ultimo quarto di partita, in confronto all'ultimo rating disponibile dei primi ¾ di partita. Si è deciso di testare una seconda versione della rete, il cui obiettivo è prevedere un miglioramento o un peggioramento significativo oppure un mantenimento costante delle performance nell'ultimo quarto di partita, in confronto al rating medio del terzo quarto di partita. Si è deciso di definire un miglioramento o peggioramento significativo come un rating medio dell'ultima parte di partita superiore o inferiore al rating medio del terzo quarto di partita di 0.1. Il dataset è passato quindi da avere sequenze appartenenti a 2 classi, ad avere sequenze appartenenti a 3 classi diverse. Questa modifica è stata decisa per gestire meglio situazioni in cui il rating del giocatore varia di poco, ma che sarebbero state comunque classificate come miglioramenti o peggioramenti dalla prima versione del modello, rendendo poco utili i risultati prodotti e più difficile l'apprendimento, dato che minime differenze e differenze significative vengono trattate allo stesso modo con un approccio di questo tipo.

In questa nuova versione del modello, si è deciso che la cosa più importante fosse evitare di classificare un peggioramento come un miglioramento e viceversa, ritenendo invece più accettabile confondere un miglioramento o un peggioramento con un mantenimento costante delle performance e viceversa. Questa scelta è stata fatta riflettendo sui possibili utilizzi della rete, per esempio come strumento di decisione delle sostituzioni dei giocatori durante una partita. Sostituire un giocatore che sarebbe migliorato o lasciare in campo un giocatore che peggiorerà ha un peso molto maggiore rispetto a sostituire o lasciare in campo un giocatore che avrebbe mantenuto o che manterrà costanti le proprie performance.

Come nel caso del dataset diviso in 2 classi, anche con la divisione in 3 classi è presente un forte sbilanciamento tra esse, con ogni campionato composto per la maggior parte da casi di miglioramento (circa 65% dei casi totali), poi da casi di mantenimento delle performance (circa il 25% dei casi totali) e infine da una piccola quantità di casi di peggioramento (mediamente meno del 10% dei casi totali). Per gestire questa situazione, in questa versione del modello si sono utilizzati dei pesi all'interno della loss function, in modo da rendere le varie classi di ugual importanza in proporzione alla loro dimensione nel calcolo del rendimento della rete. Il training è stato effettuato utilizzando una rete composta da 1 layer LSTM e 2 layer feedforward mentre per il numero di epoche si è utilizzata una configurazione early stopping.

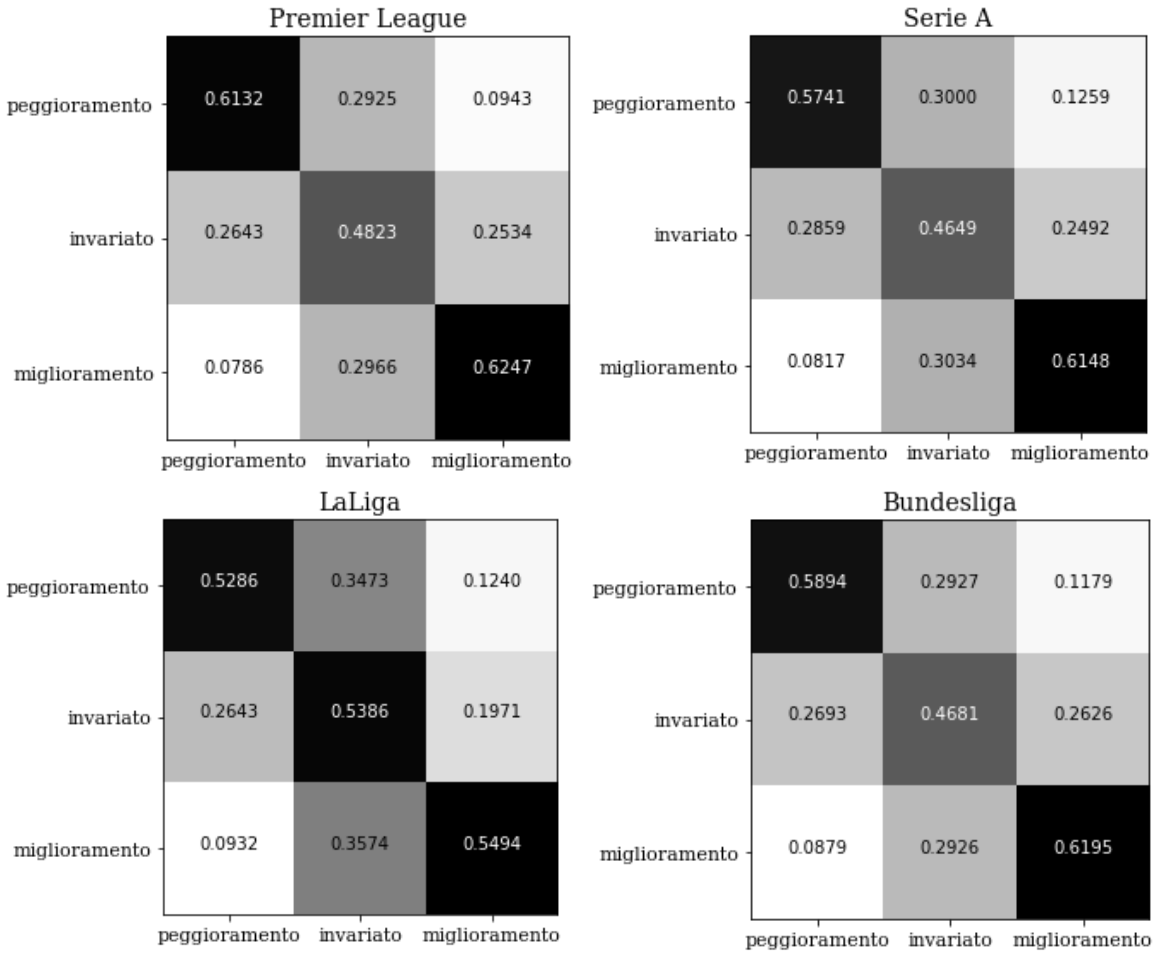


**Figura 4.1:** Matrici di confusione normalizzate rappresentati le performance del modello rispetto ai 4 principali campionati.

Analizzando le matrici di confusione a figura 4.1 prodotte dalla rete in seguito al training, possiamo notare come gli obiettivi posti prima della creazione del modello siano stati raggiunti. Difficilmente un peggioramento e un miglioramento possono essere confusi tra loro, mentre può capitare più di frequente di confonderli con un mantenimento costante delle performance, errore che, come è stato detto in precedenza, è accettabile purché non sia troppo frequente. Da notare che le matrici sono state riempite con valori normalizzati tra 0 e 1, per dare una visione migliore delle performance del modello rispetto ad ogni classe.

# 5 Conclusione

Attraverso i diversi test dei vari modelli proposti, è stato dimostrato come il deep learning possa essere utilizzato in campo sportivo, diventando un nuovo strumento di analisi utilizzabile dalle squadre. Le reti neurali LSTM si sono dimostrate ottime nell'analisi di sequenze di dati correlati tra loro. Utilizzando l'ultimo modello proposto, basato sulla classificazione delle performance dei giocatori in 3 classi, si è riusciti ad ottenere una probabilità molto bassa di confondere un caso di miglioramento con un caso di peggioramento e viceversa (mediamente una probabilità del 10%) nonostante l'estrema difficoltà nel prevedere come un giocatore potrebbe comportarsi in certe situazioni.

In futuro potrebbero essere aggiunti ulteriori dati per migliorare il modello, dati specifici sui vari giocatori, come la posizione in campo o lo sforzo fisico, oppure dati sull'andamento generale della partita. Nell'ottica di utilizzo di una tecnologia simile da parte delle squadre di calcio, potrebbero essere studiati modelli specifici per i singoli giocatori, avendo a disposizione una quantità molto maggiore di dati, provenienti anche da allenamenti e test, in modo da poterne ricavare un quadro dettagliato sulla mentalità e sulle capacità fisiche del calciatore. Ovviamente ciò potrebbe essere esteso a molti altri sport, come ad esempio il basket, sport in cui gli ultimi minuti sono ancora più fondamentali rispetto al calcio.

# Bibliografia


[1] https://arxiv.org/abs/1412.6980

[2] https://blog.osservatori.net/it_it/deep-learning-significato-esempi-applicazioni

[3] https://deeplearning4j.konduit.ai

[4] https://deeplearningdemystified.com/article/fdl-3

[5] https://www.lastampa.it/tecnologia/news/2020/05/28/news/l-a-s-roma-si-affida-all-intelligenza-artificiale-con-una-nuova-partnership-1.38892318