Tecniche di Machine Learning per la Progettazione di Circuiti Integrati

Fusato Michele

14 novembre 2022

Indice

Capitolo 1

INTRODUZIONE

In questa tesi verrà trattato l'impiego del machine learning nel processo di progettazione di un circuito integrato (Integrated Circuit, IC). Perché si sta investendo in questo settore dell'informatica? Attraverso l'analisi dei costi nel capitolo 3 si è individuato nel costo del personale ingegneri la risposta alla domanda appena posta. Tale costo è destinato ad aumentare con l'aumento della complessità dei circuiti. Da qui il sempre più crescente interesse nella ricerca e sviluppo di metodi machine learning che hanno come scopo, non la semplice riduzione del personale specializzato, ma addirittura la totale eliminazione del "l'uomo". Definito questo, come si potrà mai arrivare ad un traguardo del genere? Sostituendo il personale specializzato con delle macchine (machine) che apprendono (learning) attraverso "l'esperienza" di algoritmi di apprendimento (solitamente questa "esperienza" si crea dando in pasto a tali algoritmi un set di dati, costruito dall'uomo, composto da esempi dove l'algoritmo viene allenato) costruiti per risolvere un problema a loro assegnatoli. Il machine learning è un argomento vasto ed in rapida evoluzione, sviluppato attraverso molteplici metodologie. Nello specifico, in questa tesi verranno trattati i seguenti argomenti:

- Nel capitolo 4 l'apprendimento per rinforzo dove si definisce un problema decisionale su cui si farà lavorare un'agente che eseguirà delle azioni.
- Nel capitolo 5 reti neurali, reti neurali convoluzionali, reti neurali a grafo. Se queste reti neurali saranno addestrate attraverso un set di dati composto da esempi si parlerà di apprendimento supervisionato, non supervisionato o semi-supervisionato.

Tali metodi verranno scelti in base al problema che si dovrà affrontare. Si è detto che l'obiettivo è la sostituzione dell'umano con la macchina. Le macchine utilizzano dei processori che hanno una velocità di calcolo decisamente

superiore al cervello umano, perciò si avrà anche una riduzione importante nei tempi di progettazione. Nel capitolo 2 dunque si farà una panoramica generale sulle fasi di progettazione di un circuito integrato andando a capire nel capitolo 3, in quali di queste fasi la tecnologia attuale consente l'impiego pratico del machine learning. Nel capitolo 6 infine si andrà a vedere l'applicazione del machine learning nell'ambito della progettazione di due tipologie di chip e i risultati ottenuti.

Capitolo 2

PROGETTAZIONE DI CIRCUITI INTEGRATI

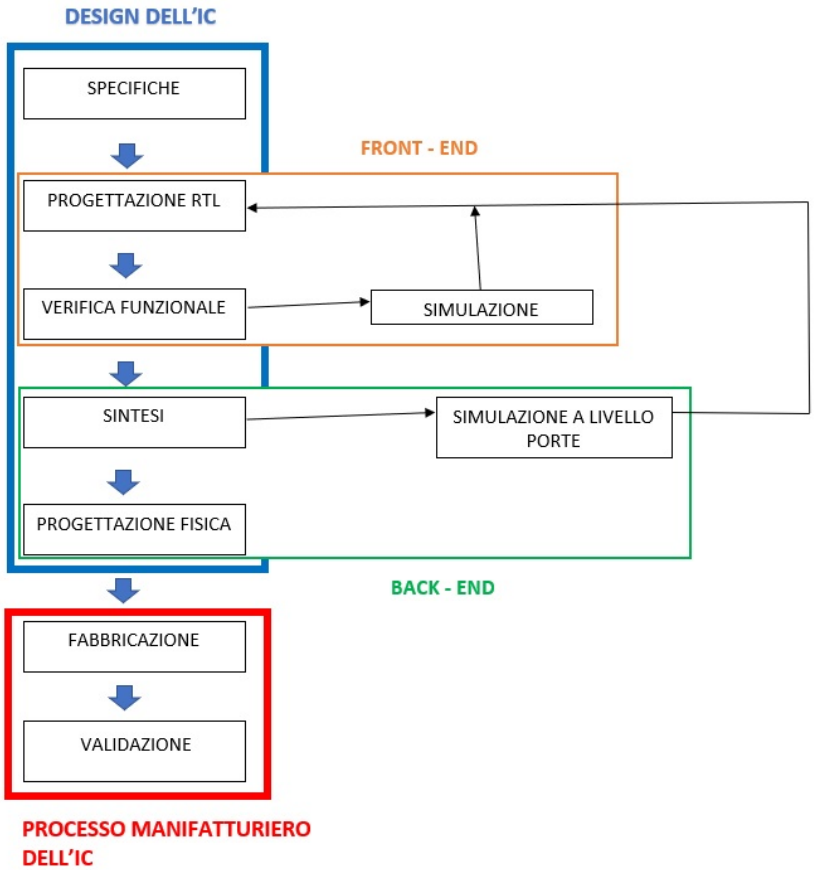

Figura 2.1: Fasi di progettazione di un IC[4]

Nella progettazione di un IC (figura 2.1), prima di arrivare al prodotto finale, ci sono varie fasi di sviluppo. Si parte dalle specifiche, cioè si andranno a definire funzioni, architettura e parametri che dovrà rispettare il circuito integrato per soddisfare la domanda che presenterà il mercato (oppure la richiesta di sviluppare un prodotto specifico con determinate caratteristiche da parte di un cliente) e sopratutto competere con la concorrenza. Fatto questo si passerà alla fase di Front-end dove si descriverà l'IC attraverso un linguaggio che permette la descrizione dell'hardware attraverso un codice andando poi a verificare attraverso delle simulazioni quello che si è scritto. La fase successiva viene chiamata Back-end. Consiste nel trasformare il codice in parti "fisiche" che verranno utilizzate per produrre un disegno (di come sarà fisicamente strutturato l'IC) del circuito su cui si utilizzerà un software, al cui interno sono presenti degli strumenti specifici (strumenti di automazione della progettazione elettronica, EDA tools), per verificare la presenza o meno di incongruenze e che vengano rispettate le specifiche imposte inizialmente. Infine si arriverà al processo manifatturiero per la produzione dell'IC.

2.1 FRONT-END

Con la definizione di funzioni, architettura e parametri dell'IC che si intende realizzare, si continua la progettazione di esso attraverso la scrittura di funzioni logiche (descrivono le funzioni del'IC) del tipo $f(a, b) = a + b$ grazie a linguaggi di descrizione dell'hardware come Verilog, SystemVerilog, VHDL (acronimo di VHSIC Hardware Description Language, dove "VHSIC" è la sigla di Very High Speed Integrated Circuits). Dopo di che si eseguirà una "prima conversione" del codice prodotto in cui si otterrà il Register Transfer Level (RTL), ovvero il circuito dell'IC è definito in termini di segnali, operazioni logiche tra i segnali e elementi di memoria dei segnali (registri generici). Allo stesso tempo attraverso le medesima procedura si svilupperà un programma di simulazione chiamato behavioral simulation (simulazione comportamentale). Verrà verificato se l'RTL ottenuto non abbia bug e che l'IC svolga le funzioni che dovrà possedere. Se non sarà cosi, si ritornerà al codice per modificarlo o correggerlo. I due processi descritti in figura 2.1 vengono chiamati progettazione RTL e verifica funzionale.

2.2 BACK-END

Verificata la progettazione RTL, avviene una "seconda conversione" del codice. I segnali, le operazioni logiche tra i segnali e gli elementi di memoria

dei segnali (registri generici) vengono convertiti, attraverso l'operazione di sintesi, in porte (gate): porte logiche e flip-flops (FF). La lista delle porte, le interconnessioni tra le porte, i flip-flops e i loro pin costituiscono quella che viene definita Netlist.

Figura 2.2: Fase di Back-End[4]

2.2.1 PROCESSO DI PROGETTAZIONE DEL DESIGN FISICO

È il processo che trasforma il codice scritto in elementi fisici che verranno utilizzati poi per realizzare il circuito dell'IC. Tale progetto verrà verificato attraverso gli strumenti EDA al fine di verificare se siano rispettate le

specifiche (ad esempio la potenza consumata) iniziali del progetto. Se tali strumenti daranno risposte negative, si andrà a modificare il progetto (tali modifiche o eventuali correzioni possono portare addirittura a riprendere in mano il codice scritto all'inizio del processo). Effettuata la verifica, il progetto verrà poi convertito in una maschera foto sensibile (photo-mask), che sarà impressa tramite raggi UV nel silicio per la creazione dell'IC. Il processo per implementare il design fisico del circuito si compone di vari step descritta nella figura 2.2.

Partizione: il circuito viene partizionato in vari blocchi, che saranno assegnati al personale specializzato per la realizzazione. La partizione viene eseguita secondo un certo criterio per semplificare i processi di piazzamento, instradamento del segnale (routing) e per ridurre la lunghezza totale dei fili in metallo che connettono i blocchi e le porte che ci sono al loro interno. Nei blocchi creati troviamo le definizioni di: Macro celle, ovvero celle senza dimensioni predefinite, rappresentate come "scatole nere (black-box)" e celle standard, molto più piccole rispetto alle macro celle con dimensioni predefinite, che rappresentano porte logiche o Flip-Flops.

Floor planing: si decide quali blocchi possono essere messi vicino tra di loro. Tutto questo cercando un compromesso tra l'ottimizzazione della richiesta di risorse, area che si andrà ad occupare e velocità.

Piazzamento: si decide in quali regioni del silicio dell'IC vengono collocati i blocchi trovati tramite partizione evitandone la sovrapposizione tra di essi. Processo di elevata importanza e complessità perché da esso dipenderanno le prestazioni del chip e degli errori in questa fase potrebbero addirittura rendere irrealizzabile l'IC nel silicio.

Sintesi del clock: il segnale del clock viene introdotto nel design del circuito cercando di minimizzarne il ritardo.

Instradamento del segnale (routing): Si connette ogni pin di ogni blocco attraverso dei conduttori metallici (metal wires), che seguiranno un percorso preciso. Tuttavia ci saranno dei vincoli da rispettare imposti dal design (design rules). Questo processo viene eseguito dal software attraverso li strumenti EDA.

Timing closure: Attraverso li strumenti EDA del software si andranno a eliminare le violazioni dei vincoli imposti che riguardano la temporizzazione e verrà eliminato anche il rumore che va a danneggiare l'integrità del segnale.

Capitolo 3

COSTI

La legge di Moore afferma che: "la complessità di un microcircuito, misurata ad esempio tramite il numero di transistor per chip, raddoppia ogni 18 mesi e quadruplica quindi ogni 3 anni". Possiamo associare questa crescita del numero di transistor ad un aumento di difficoltà nella progettazione e implementazione di un chip. Ovviamente questo corrisponderà ad una maggiorazione nell'esborso economico da parte delle aziende per tenere il passo nel loro settore di mercato. Secondo l'indagine dell'International Business Strategy Corporation[1] (IBS), l'aumento dei costi di progettazione per ogni generazione di tecnologia ha superato il 50% dopo il processo a 22 nm. Per esempio, il costo totale di progettazione del processo a 7 nm è di circa 300 milioni di dollari, mentre quello del processo a 3 nm è destinato ad aumentare di 5 volte fino a 1,5 miliardi di dollari. Questo trend viene riportato nella figura[3.1]. Come risposta all'aumento dei costi, l' agenzia statunitense per i progetti di ricerca avanzata di difesa (DARPA)[2] nel 2017 annunciò il progetto Electronics Resurgence Initiative (ERI). Lo scopo della DARPA, attraverso l'investimento di centinaia di milioni di dollari nel progetto ERI, sarà quello di fare ricerca e sviluppo in 3 settori che a parere loro sono le tre colonne fondamentali di un chip: materiali per la realizzazione, architetture, progettazione. Per la progettazione viene posta questa domanda: possiamo ridurre drasticamente il tempo e la complessità necessari per progettare i moderni sistemi su circuito integrato (SoC)? Il progetto ERI per dare risposta a questo quesito ha dato vita a due programmi (facenti sempre parte di ERI). Il primo denominato IDEA (Intelligent Design of Electronic Assets) si pone l'obbiettivo di arrivare addirittura a una progettazione 24 ore su 24 senza l'intervento di un umano **"no human in the loop"**, consentendo anche ai "non esperti" di progettare tecnologie di circuiti elettronici complessi. Mentre il secondo programma chiamato POSH (Posh Open Source Hardware), fornirà una piattaforma di progettazione e verifica open-source,

comprendente tecnologie, metodi e standard, che consentirà di progettare in modo economicamente vantaggioso SoC ultracomplessi.

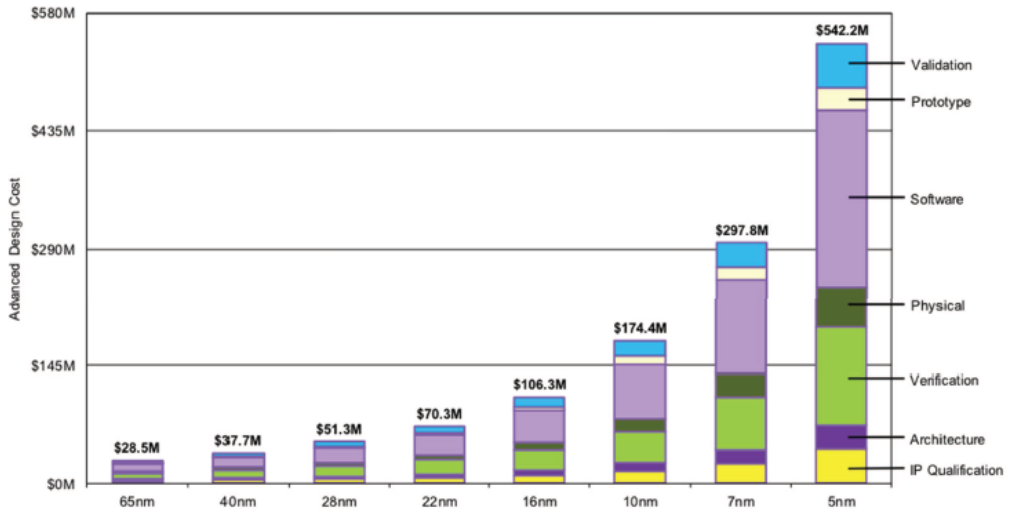

Figura 3.1: Diagramma tecnologia-costi[1]

3.1 PERCHÉ "NO HUMAN IN THE LOOP"?

L'investimento principale nel design dei circuiti integrati è il personale di ingegneria[12]. Come descritto prima, una delle sfide più importanti di oggi è appunto il controllo dei costi e di conseguenza quindi il controllo dei costi del personale di ingegneria. In media, la domanda di ingegneri progettisti IC/ASIC (figura 3.2)[1] è cresciuta a un CAGR (coumpounded annual growth rate) del 3% circa tra il 2007 e il 2020, mentre la domanda di ingegneri per la verifica funzionale IC/ASIC è cresciuta a un CAGR del $6,8\%$[3].

[1]Un circuito integrato per applicazione specifica (ASIC) viene creato con lo scopo di risolvere un unico problema, raggiungendo così velocità di processo ed efficienza energetica impossibili da ottenere con soluzioni generiche

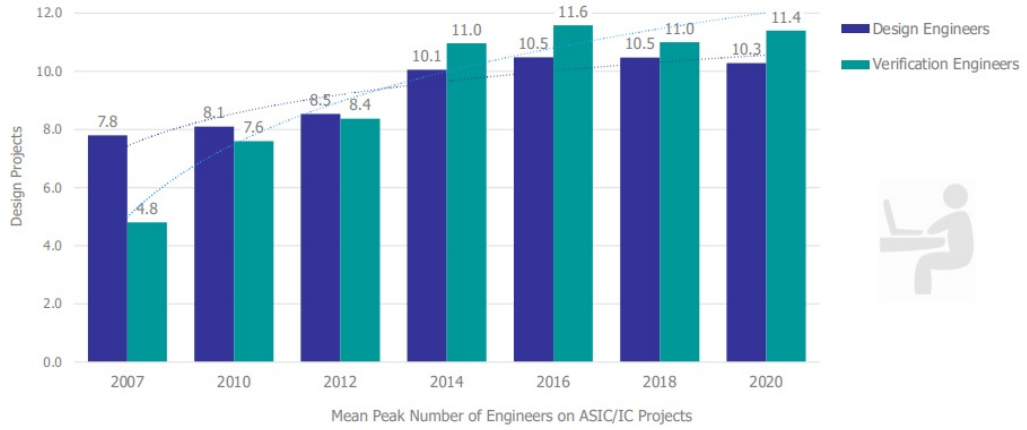

Figura 3.2: Crescita del numero di ingegneri impiegati nella progettazione e nella verifica funzionale[3]

Cosa molto importante è che gli ingegneri per la verifica funzionale non sono le uniche parti interessate al progetto che spendono il loro tempo nel processo di verifica. Anche gli ingegneri del design spendono una parte significativa del loro tempo nella verifica del disegno fisico dell'IC ottenuto (nella fase di piazzamento) durante la progettazione. Nel 2020 (figura 3.3), questi ingegneri hanno dedicato in media il 53% del loro tempo alle attività di progettazione e il 47% alla verifica[3]. Ovviamente questo porta ad un aumento consistente dei tempi di progettazione.

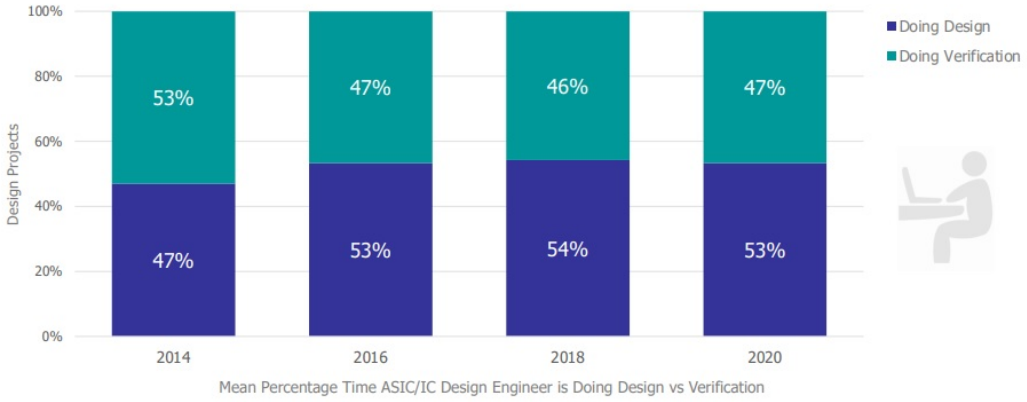

Figura 3.3: Percentuale di tempo speso dagli ingegneri di progettazione[3]

3.2 DOVE È POSSIBILE ATTUARE IL "NO HUMAN IN THE LOOP"?

Prendendo come riferimento la figura 3.1 e associandola a quanto visto nel capitolo 2 (figura 2.1) si cercherà di rispondere alla domanda posta nel titolo di questa sezione. Per le parti di specifica e sviluppo software è impensabile togliere la parte umana. Sarà più ragionevole affiancare agli ingegneri una intelligenza artificiale (AI), con cui interagire, che li faccia da supporto per semplificare i processi di sviluppo specifici e quindi ne riduca anche il tempo speso. Ma questo, ad oggi, fa parte di un futuro lontano. Invece per quanto riguarda la fase di validazione ci sono già metodi "automatici" utilizzati dagli ingegneri descritti in letteratura. In merito alla progettazione del design fisico di un chip, si sta facendo ricerca e sviluppo nel settore del machine learning (ML,che negli ultimi anni e tornato a far parlare di se in maniera prepotente), non solo per aiutare, ma per andare a sostituire la parte umana coinvolta in un processi lungo e complesso come il processo di piazzamento.

Capitolo 4

APPRENDIMENTO PER RINFORZO

La definizione dell'apprendimento per rinforzo arriva dalla letteratura: "L'apprendimento per rinforzo (RL) è una tecnica di machine learning in cui un computer (agente) impara a svolgere un'attività tramite ripetute interazioni di tipo "trial-and-error" (eseguite per tentativi ed errori) con un ambiente dinamico. Questo approccio all'apprendimento consente all'agente di adottare una serie di decisioni in grado di massimizzare una metrica di ricompensa per l'attività, senza essere esplicitamente programmato per tale operazione e senza l'intervento dell'uomo"[5]. Quindi il RL mira a sostituire la figura umana e la sua capacità di acquisire esperienza per la risoluzione di un qualche problema che si mira a risolvere, con l'evidente scopo di sfruttare il vantaggio in termini di potenza di calcolo che ha a disposizione un processore.

Da questa definizione possiamo estrapolare le prime informazioni base:

- **Agente**: il soggetto che impara e che prende le decisioni.
- Ripetute iterazioni tramite la scelta di **azioni**: portano dallo stato attuale a quello successivo.
- **Ambiente**: dove lavora l'agente.
- **Ricompensa**: valore di ritorno dopo aver scelto un'azione. Definisce la qualità dell'azione.

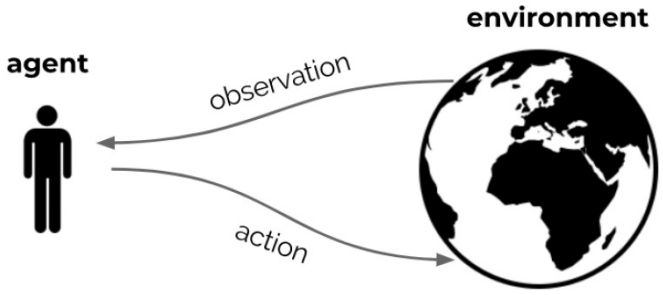

Figura 4.1: Interazione tra Ambiente e Agente[19]

Con l'utilizzo della figura 4.1 si andrà ad utilizzare una analogia fisica per spiegare meglio la terminologia appena estrapolata dalla definizione di RL. Si supponga che una persona decida di affrontare un giro del mondo in un tempo finito (ad esempio 6 mesi). Si supponga che questa persona di conseguenza formuli il seguente problema:"quali posti mi converrà visitare per **massimizzare** il piacere del viaggio?". **L'obiettivo** (task) viene definito nella massimizzazione del piacere. Quindi: la persona sarà l'agente, il pianeta terra sarà l'ambiente, le azioni saranno le decisioni sui luoghi da visitare da parte della persona e il luogo che la persona starà visitando sarà lo stato attuale mentre il luogo che dovrà visitare subito dopo sarà lo stato successivo. Ogni luogo visitato dovrà dare una ricompensa in termini di piacere. Il nuovo termine che introduce la figura è l'osservazione, che nell'esempio esposto rappresenterà tutte le sensazioni fornite dal luogo visitato e raccolte dalla persona attraverso i suoi "sensori". Perciò si può affermare che l'osservazione dipende sia dall'ambiente che dall'agente. Molto importante è definire che **l'ambiente non lo si conosce e non lo si controlla**.

Da questo esempio si può dire che nel RL:

- Non basta la formulazione del problema ma si deve definire anche l'obiettivo da raggiungere.
- Si ha un problema decisionale. Quali azioni eseguire in quell'ambiente? Azioni che hanno come fine il raggiungimento dell'obiettivo che si è posto.
- Con molta probabilità ci si ritroverà in un ambiente sconosciuto che azione dopo azione l'agente cercherà di conoscere.

- Si cerca di predire (il termine predizione verrà specificato più avanti). Siccome lo scopo è la risoluzione del problema in cui le scelte dell'agente dovranno portare a massimizzare l'obiettivo, si dovrà capire dove ci porteranno le azioni che si potranno scegliere. Questo processo viene definito apprendimento (learning).

- La somma di tutte queste ricompense ci da un ritorno totale. Perciò si vuole scegliere le azioni che massimizzano questo ritorno totale.

- Siccome c'è incertezza, non si può predire con sicurezza che cosa accadrà nel futuro. Tuttavia visto che lo scopo è la massimizzazione del ritorno, quello che si potrà fare sarà massimizzare il **valore atteso** di questo ritorno, cioè il suo **valore medio** sulla predizione del futuro. Quando si parla di valore nel RL si intende un valore atteso di una qualche variabile aleatoria che in genere rappresenta una somma di ricompense che si ricevono per certe azioni.

- Si discute parlando in termini di predizione e incertezza, quindi ci si ritroverà in un ambito stocastico.

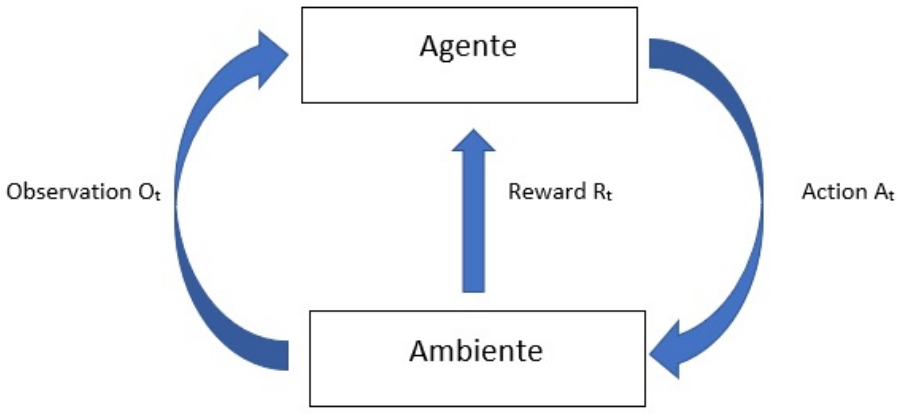

Figura 4.2: Interazione Ambiente-Agente[9]

Utilizzando la figura 4.2 si andrà a descrivere l'interazione che c'è tra ambiente e agente. L'agente riceve da parte dell'ambiente una osservazione e una ricompensa. Avvenuto questo, come conseguenza, si può eseguire un'azione. Eseguita questa azione A_t[1], l'ambiente a sua volta fornirà: un nuovo

[1]Tutte gli argomenti trattati in questo capitolo sono stazionarie, indipendenti dal tempo, che questo t sia 3-9-10 è invariante.

stato che darà una nuova ricompensa R_{t+1} e una nuova osservazione O_{t+1}. Tutto questo avviene ciclicamente. Quando l'agente riceve una osservazione e una ricompensa, con esse "costruisce" il uso stato S_t^{azione}, stato da cui verrà scelta l'azione che porterà a quello successivo, dove si avrà nuova ricompensa e nuova osservazione. Inoltre, l'ambiente ha un suo stato interno $S^{ambiente}$ che non si conosce. Si conosce soltanto quello che si riceve dall'ambiente. L'ambiente, del suo stato, fa soltanto vedere quello che l'agente è "pronto" a vedere che si chiama osservazione. Quindi possiamo definire la seguente tabella:

AGENT,step t	ENVIROMENT,step t
-Riceve l'osservazione O_t	-Riceve l'azione A_t
-Riceve reward R_t	-Computa il suo stato S^e
-Computa il suo stato S_t^a	-Emette l'osservazione O_{t+1}
-Esegue l'azione A_t	Emette reward R_{t+1}

Lo stato dell'agente nella sua definizione più generale possibile, dipende da tutta la storia. Si definisce storia H tutto quello che è successo fino a quel momento, la sequenza di osservazioni , azioni e ricompense fino al time step t:

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t \tag{4.1}$$

Quindi possiamo scrivere $S_t^a = f(H_t)$. Ovviamente si dovranno "riassumere" tutte le informazioni con una funzione f (decisa dall'agente), altrimenti computazionalmente non sarebbe possibile sostenere una tale mole di dati.
Lo stato dell'ambiente è diverso da quello dell'agente e il nostro ambiente soddisfa quella che si chiama proprietà di Markovianità, perciò lo stato del nostro ambiente è Markoviano. Dalla letteratura, si definisce processo stocastico Markoviano (o di Markov), un processo aleatorio in cui la probabilità di transizione che determina il passaggio a uno stato di sistema dipende solo dallo stato del sistema immediatamente precedente (proprietà di Markov) e non da come si è giunti a questo stato[1]. Quindi, l'informazione necessaria per avere il futuro è contenuta tutta nel presente (le cose che accadranno nel futuro dipendono dal passato, soltanto tramite il presente). **Per semplificazione, si assume di studiare ambienti completamente osservabili** (caso più semplice). Perciò, tutte le distinzioni e le particolarità dette prima si faranno coincidere. Nello specifico: $O_t = S_t^e = S_t^a$, cioè l'osservazione che

[1]Nella teoria della probabilità, la proprietà di Markov per un processo stocastico consiste nella dipendenza esclusiva dallo stato presente della variabile casuale dei futuri stati e, per esempio, non dagli stati passati ma soltanto dall'ultima osservazione. Un processo con la proprietà di Markov è chiamato processo Markoviano

viene restituita dall'ambiente è esattamente tutto il suo stato. Quindi l'agente che prenderà O_t, userà S_t^e come suo stato. Definendo che S_t^e è Markoviano non c'è bisogno di mantenere l'informazione precedente, è già contenuta nello stato.

Quindi si avrà:

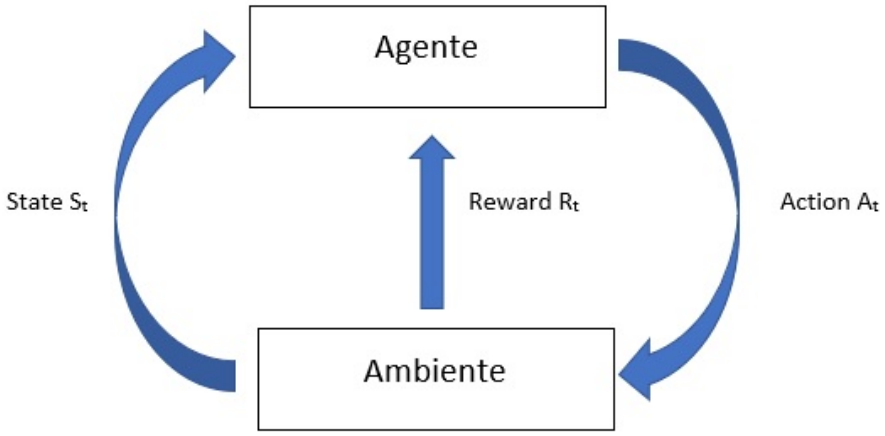

Figura 4.3: Interazione Agente-Ambiente dopo semplificazione

Si otterrà un'iterazione del tipo:

- Si estrae, da una distribuzione di probabilità di azioni (.) condizionata allo stato S_t nel quale ci si trova la decisione sull'azione da eseguire $A_t \sim \pi(., S_t)$ per andare nello stato successivo. Con π si indica il termine **policy**. La policy "dirà" sulla base della distribuzione di probabilità di azioni (.), che azione dovrà essere scelta in un certo stato.
- Da questa azione A_t si riceve un nuovo stato S_{t+1} e una nuova ricompensa R_{t+1}. Stato e ricompensa sono due cose diverse tra di loro, che però accadono congiuntamente, governate da una stessa distribuzione di probabilità.

4.1 PREDIZIONE, MIGLIORAMENTO, CICLO DI CONTROLLO

In generale gli algoritmi di RL vengono fatti lavorare per eseguire i seguenti compiti:

1. Predizione: si avrà una certa policy e si vorrà capire quanto questa policy valga. Da ciascuno stato, seguendo una certa policy, quanto ritorno si otterrà? Predire vuol dire calcolare il valore di una certa policy fissata.

2. Miglioramento: si avrà una certa policy, è possibile trovarne una migliore? Questa ricerca viene eseguita con un determinato **passo di miglioramento**.

3. Ciclo di controllo: si potrà mai trovare la policy migliore di tutte?

Con V_π si definisce il valore di quello stato relativamente ad una certa policy. Il valore di quello stato dipende dalle azioni che si sceglieranno da esso. V_π è la funzione valore della policy π. Di conseguenza questi tre problemi interagiranno così: partendo da una policy π, con la predizione si calcolerà il suo valore V_π, da V_π con il miglioramento si troverà una policy migliore/migliorata π', infine con il ciclo di controllo si eseguirà una iterazione partendo da π' interrompendola, quando si arriverà a convergenza[2] (si trova la policy migliore).

4.2 PROCESSO DECISIONALE DI MARKOV (MDP)

I processi decisionali di Markov (MDP), dal nome del matematico Andrej Andreevič Markov (1856-1922), forniscono un quadro matematico per affrontare un problema di tipo decisionale. Si consideri un problema la cui soluzione passerà dalle decisioni sulle azioni prese da un agente. Per risolverlo si dovrà trovare la distribuzione di probabilità sulla scelta delle azioni (policy) ottimali. Quindi per costruire il problema si utilizzerà un blocco base contenete: stato, azione, modello, ricompensa e che può essere unito ad altri blocchi analoghi. Come si uniscono questi blocchi? Utilizzando dei nodi (figura 4.4) dalle quali possono partire una o più azioni. Si partirà dal nodo u e con un arco (edge) lo si collegherà ad un altro nodo identificato da una dimensione più piccola (pallino pieno nero), chiamato nodo azione. Rappresenterà la possibilità che da quello stato si potrà eseguire quella azione. Scelta l'azione a, l'ambiente fornirà lo stato v. Tuttavia Il vero nodo del grafo non è lo stato oppure l'azione, ma è la coppia intera (figura 4.5). Essendo l'ambiente aleatorio e non deterministico, ho uno stato w alternativo. v e w possono comparire con due probabilità diverse, v con il 70% e w con il 30%. Se avessi

[2]Teorema del punto fisso di banach caccioppoli

w al 20% potrei avere un terzo stato con probabilità al 10%. La probabilità di scelta di v e la ricompensa che si otterrà vengono rappresentati nell'eq. 4.2:

$$p(v, +3|u, a) = Pr(S_{t+1} = v, R_{t+1} = 3|S_t = u, A_t = a) \tag{4.2}$$

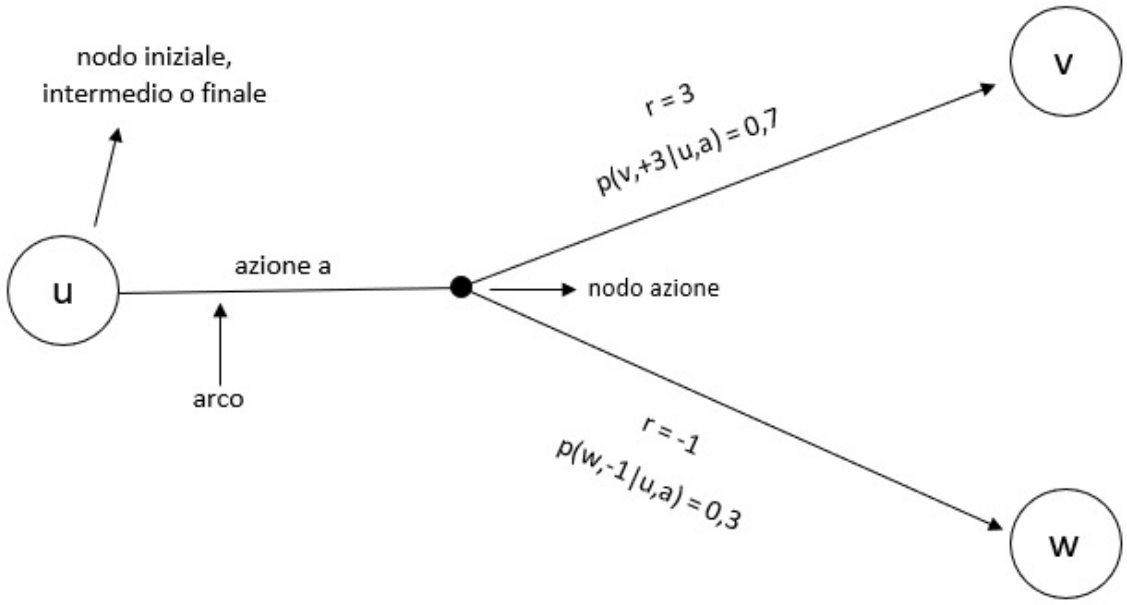

Figura 4.4: Blocchi base messi assieme[9]

Dopo varie azioni e vari stati, si potrebbe tornare allo stato iniziale u, quindi il grafo verrà definito ciclico. Può anche essere che ci sia un'altra azione (ovviamente con una sua probabilità) per arrivare a v con una ricompensa differente.

Figura 4.5: Nodo del grafo composto dallo stato u e dall'azione a[20]

Il processo decisionale di Markov è una tupla che consiste di:

- Uno spazio degli stati S.

- Uno spazio delle azioni A.
- Una funzione ricompensa $r : SXA \to \mathbb{R}$, cioè la variabile aleatoria ricompensa r avrà valori in questo insieme qui.
- Per ogni stato $s \in S$ e azione a $\in A$, si dovrà avere una probabilità $p(.,.|s,a)$ di SXR. Cioè se si è nello stato s e si esegue l'azione a, con quale probabilità otterrò uno stato s' ed una certa ricompensa r? p è quello che prima si è chiamato modello, è la probabilità che l'ambiente usa per restituire un nuovo stato e una nuova ricompensa. Quindi il blocco che prima si chiamava ambiente ora viene definito come la funzione di quattro variabili $p(.,.|s,a)$.
- Inoltre si avrà un fattore di sconto $\gamma \in [0,1]$. Rappresenta se le ricompense tra 1, 2, 3, 4 istanti temporali sono tutte uguali, oppure se quelle che si prenderanno tra 4 istanti temporali dovranno valere di meno di quelle che si prenderanno tra 4, 3, 2 o 1 istanti.

Per semplificazione, si assume che lo spazio degli stati, lo spazio delle azioni e l'insieme delle ricompense siano tutti **insiemi finiti**.

Dal modello di distribuzione p alle variabili randomiche S_t e R_t si avrà la dinamica dell'MDP:

$$Pr(S_t = s', R_t = r|S_{t-1}, A_{t-1} = a) := p(s'|s,a) \tag{4.3}$$

Attraverso la seguente equazione si esprime la ricompensa R_t come valore medio:

$$\mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] \tag{4.4}$$

Per ottenere il valore medio della ricompensa si prende la determinazione possibile della ricompensa r e la si moltiplica per la probabilità che accada questa ricompensa.

$$r * p(s', r|s, a) \tag{4.5}$$

Ma questa è la probabilità di ottenere tale ricompensa andando proprio in s'. Per avere la probabilità di ottenere solo questa ricompensa indipendentemente da s', basta sommare su tutti gli s' possibili.

$$r * \sum_{s' \in S} p(s', r|s, a) \tag{4.6}$$

Si ottiene che r viene pesata per la sua probabilità. Per avere la ricompensa media $r(s,a)$ per la coppia stato azione (s,a), si dovrà sommare su tutte le ricompense possibili.

$$\sum_{r\in R} r * \sum_{s'\in S} p(s', r|s, a) \tag{4.7}$$

Un'azione a $\in$ A darà una ricompensa media per ogni stato s:

$$R_s^a = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = r(s, a) \tag{4.8}$$

Dove R^a è il vettore di ricompensa media per ogni azione a.

Si definisce ora la policy. Una policy π è una distribuzione di probabilità dati gli stati. Se non c'è dipendenza dalla storia H_t la policy π si dice Markoviana, quindi dipenderà dallo stato.

$$\pi(a|s) := Pr(A_t = a|S_t = s) \tag{4.9}$$

Le decisioni da parte dell'agente sono prese in base alle distribuzioni di probabilità $\pi(a|s)$, cioè qual è la probabilità che dallo stato s l'agente scelga a? Questo processo da parte dell'agente viene chiamato agent decision.
Distinguiamo due casi del MDP. Primo caso: se c'è uno stato terminale chiamato per ipotesi C, l'MDP (il task) è episodico, perché si spezza in episodi, cioè un episodio è una scelta di percorsi (non tutti quanti) S_0, A_0, R_1, S_1 che ci porta in C. A quel punto faremo la conta delle ricompense. Attraverso questa somma si andrà a trovare il ritorno totale. Il ritorno totale di un episodio che termina al tempo T sarà la variabile randomica:

$$G_t = R_{t+1} + R_{t+2} + \ldots\ldots + R_T \tag{4.10}$$

Nell'eq. (4.10) le variabili R sono aleatorie.
Inoltre, si può trasformare lo stato terminale C nello stato assorbente, cioè si decide che da C si potranno continuare ad eseguire tutte le azioni che si vorrà e ciascuna di queste riporti con probabilità uno nello stesso Stato C e la ricompensa che si otterrà da questi passaggi dovrà essere zero. Facendo questo si potrà utilizzare la formula del ritorno totale per l'MDP continuo.
Secondo caso: se non ci fosse lo stato terminale, si direbbe che l'MDP è continuo. In questo caso per la variabile randomica G_t si renderà necessario l'utilizzo di un fattore di sconto (discount factor) γ:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots\ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+\gamma+1} \tag{4.11}$$

Con il ritorno totale si possono definire ora le funzioni valore per stato e azione. La funzione stato-valore (state-value function) $V_\pi(S)$ per un MDP è il ritorno che si suppone di accumulare partendo dallo stato s seguendo la policy π:

$$V_\pi(s) := \mathbb{E}_\pi(G_t|S_t = s) \tag{4.12}$$

La funzione azione-valore (action-value function) $q_\pi(s, a)$ per un MDP è il ritorno che si suppone di accumulare partendo dallo stato s, scegliendo l'azione a e poi seguendo la policy π:

$$q_\pi(s, a) := \mathbb{E}_\pi(G_t|S_t = s, A_t = a) \tag{4.13}$$

Per definire tali funzioni valore si dovrà ricorrere a delle equazioni ricorsive. Come si potrà trovare una formula per la funzione valore $V_\pi(s)$ in un certo stato s, come funzione di quello stesso valore negli stati successivi $V_\pi(s')$? Seguendo una certa policy, non si saprà mai se la policy scelta sia la migliore o la peggiore però si vorrà sempre quantificare quanto valga. Questa equazione rappresenterà il valore di questo stato per la policy scelta, quindi si può affermare che si stia valutando il valore di tale policy. Con $V_\pi(s)$ scritto come la (4.12) ci si ritroverebbe ad avere G_t infinita, ma utilizzando $G_t = R_{t+1} + \gamma G_{t+1}$ si otterrà una eq. ricorsiva:

$$V_\pi(s) := \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \tag{4.14}$$

La somma totale di ricompensa consisterà in una prima ricompensa iniziale e poi in tutte le ricompense successive dove si raccoglierà γ. Il valore trovato sarà il ritorno, però partendo dal tempo t+1. Si può riscrivere la (4.14) in tale modo:

$$\mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma\mathbb{E}[G_{t+1}|S_t = s] \tag{4.15}$$

Però si è fermi allo stato s, quindi la policy non è presente. Per inserirla si dovrà eseguire l'azione a, pesandola con la sua probabilità.

$$\pi(s, a)(\mathbb{E}_\pi[R_{t+1}|S_t = s, A_t = a] + \gamma\mathbb{E}[G_{t+1}|S_t = s, A_t = a] \tag{4.16}$$

Se si vorrà la probabilità di partire da s senza dipendenza dall'azione a, si dovrà sommare su tutte le azioni.

$$\sum_a \pi(s, a)(\mathbb{E}_\pi[R_{t+1}|S_t = s, A_t = a] + \gamma\mathbb{E}[G_{t+1}|S_t = s, A_t = a] \tag{4.17}$$

$$\sum_a \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V_\pi(s')) \tag{4.18}$$

Dove $\mathcal{P}_{ss'}^a$ è la matrice[1] che rappresenta la transizione tra gli stati, se si fissa un'azione a si ha una certa probabilità di passare da s a s'.

$$\mathcal{P}_{ss'}^a = p(s'|s,a) = Pr(S_t = s'|S_{t-1} = s, At - 1 = a) \tag{4.19}$$

Si è appena definita l'equazione di Bellman stato-valore. L'equazione di Bellman azione-valore invece sarà:

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s', a') \tag{4.20}$$

L'equazione di bellman è quella che permette all'agente di apprendere (learning). Come avviene l'apprendimento? Se si conosce il valore di s', attraverso delle formule ricorsive di questo tipo (ricorsive perché usano il valore dello stato successivo), e una ricompensa immediata, che si può verificare sul campo facendo una singola azione, sommando queste quantità (utilizzando i valori medi perché ci cono in gioco le probabilità) si potrà ottenere il valore dello stato in cui ci si troverà. In questo modo, partendo dalla fine, si riuscirà a ricostruire il valore della policy scelta in tutti gli stati. La funzione che permette di trovare i valori migliori per gli stati e per le azioni, si definisce funzione-valore ottimale. La funzione-valore ottimale stato-valore (optimal state-value function) $V_*(s)$ è il massimo stato-valore su tutte le policy:

$$V_*(s) := max_\pi V_\pi(s) \tag{4.21}$$

Non potrebbe esistere un massimo in quanto le policy, essendo stocastiche, tendono a essere infinite. Però fissato uno stato s e una policy stocastica π, esisterà sicuramente una policy deterministica π' che su quello stato s valga di più. Dati s e π $\exists$ π' det. t.c. $V_{\pi'}(s) \geq V_\pi(s)$. A questo punto il massimo lo si potrà determinare sulle policy deterministiche, perché tanto in quelle stocastiche per ciascuno stato s ce n'è una deterministica che migliora quello stato. E se le ipotesi sono deterministiche, ecco che si verifica l'ipotesi di MDP finito. Le policy deterministiche su stati e azioni finiti sono finite, quindi il massimo su un insieme finito esiste. La policy π, come detto fino a ora, sceglie le azioni con una certa probabilità. L'azione a ha un suo valore $q_\pi(s,a)$ che assumerà la definizione di q-value. Siccome ogni azione possiede un suo valore, allora sarà necessario calcolare $q_\pi(s,a)$ $\forall a$. Si assuma di essere

[1] quando si ha una matrice che rappresenta una transizione tra gli stati, c'è sottostante un processo decisionale di Markov. Ovvero si ha un task decisionale di Markov.

in uno stato s e di avere una policy π' det. che valga più della policy π. In s, π' sceglierà l'azione che renderà massimo il q-value. Fatto questo, cosa accadrà alle altre policy? Verranno scartate e π' quindi diventerà deterministica. Allo stesso modo, esisterà il valore massimo di $q_\pi(s, a)$. La funzione-valore ottimale azione-valore (optimal action-value function) $q_*(s, a)$ è la massima azione-valore su tutte le policy:

$$q_*(s, a) := max_\pi q_\pi(s, a) \tag{4.22}$$

Ogni policy in grado di ottenere la funzione-valore ottimale stato-valore o la funzione-valore ottimale azione-valore è chiamata policy ottimale. π^* è una policy ottimale se:

$$q_{\pi^*} = q_* \quad \text{oppure} \quad V_{\pi^*} = V_* \tag{4.23}$$

La policy ottimale non è unica[1] però ce n'è una deterministica. Come si può trovare questa policy? In uno stato s si sceglierà una azione a che massimizza q_*:

$$a = argmax_{a \in A} q_*(s, a) \tag{4.24}$$

La policy quindi è deterministica, perciò se si conoscerà q_* si otterrà la policy ottimale. Quindi si avranno le equazioni di ottimizzazione di Bellman:

$$V_*(s) = max_a(R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V_*(s')) \tag{4.25}$$

$$q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a max_{a' \in A}(q_*(s', a')) \tag{4.26}$$

I metodi che utilizzano le equazioni di Bellman vengono sono metodi che si basano sul valore (value-based).

4.3 APPROSSIMAZIONE

Si debba risolvere un problema utilizzando dei metodi di apprendimento per rinforzo e nel formalizzarlo risulti uno spazio di stati S avente dimensioni 10^{30} (o superiore). Computazionalmente sarebbe impossibile sostenerlo o comunque la funzione per descriverlo valore per valore sarebbe troppo complessa, quindi come ridurre la dimensionalità/complessità del problema? Si

[1] Ma loro funzione valore sia di stato che di azione, è univocamente determinata.

assumerà una forma funzionale nota a priori e la si userà per definire la funzione necessaria alla risoluzione del problema. Quindi d'ora in poi si userà la seguente notazione di stima:

$$V_\pi(s) \sim \hat{V}(s,w) \quad q_\pi(s,a) \sim \hat{q}(s,a,w) \tag{4.27}$$

Dove $w \in \mathbb{R}$ è un peso che si aggiorna.
Si andrà ad approssimare la funzione valore esatta attraverso delle supposizioni, ovvero si assocerà un certo valore a ogni stato dato, da uno stimatore, la cui forma funzionale si supponga nota ma parametrica (la forma funzionale è nota almeno nei parametri). Fatto ciò il problema diventerà la ricerca del parametro, quindi il parametro sarà l'incognita.
Come esempio si assuma di dover fare una predizione su una funzione y=f(x) che descrive la relazione tra x e y, avendo a disposizione il grafico di tale funzione dove in precedenza siano stati ricavati sperimentalmente delle relazioni tra numeri reali (si sta parlando di apprendimento supervisionato). Ovviamente, ci saranno svariati modi per descrivere tali relazioni. Tuttavia per qualche motivo si riesce a definire a priori che la f(x) è un polinomio di secondo grado, del tipo $ax^2 + bx + c$. Dalle svariate scelte per descrivere la relazione tra x e y, si riesce ad **approssimare** ad un polinomio di secondo grado dove si dovranno definire a,b,c, riducendo così le dimensioni del problema iniziale. Si avrà $f(x) = ax^2 + bx + c = y$ dove y nell'apprendimento per rinforzo è l'obiettivo (target) che si è scelto di raggiungere (nell'apprendimento supervisionato viene definito etichetta (label)), mentre x viene dato dalle relazioni trovate sperimentalmente. Perciò si può dire che i parametri a,b,c dovranno "andare" verso l'obiettivo y. Si potrà quindi scrivere $\hat{f}(x) \simeq$ obiettivo.

Ci sono varie famiglie di approssimatori:

- Approssimatori classici in ML: alberi di decisione (algoritmo di tipo nearest neighbor), approssimazione lineare. In questo caso si avranno le caratteristiche (features) anzichè le etichette.

- Reti neurali: utilizzando delle reti neurali come approssimatori, si rende l'apprendimento per rinforzo molto efficiente. Il modo di procedere con le reti neurali è sviluppare l' approssimatore (l'architettura, la rete neurale) in modo che lo stimatore alla fine dell'apprendimento su x sia circa il target. Spesso l'approssimatore, prima di utilizzarlo su un problema, viene addestrato tramite allenamento su un set di dati costruito su problemi simili.

Nella funzione valore si avrà x=stato S_t su cui si vorrà calcolare il valore, mentre y=$V_\pi(s)$ sarà l'obiettivo (o etichetta nel caso dell'apprendimento supervisionato). Perciò y sarà l'incognita da trovare. Quindi possiamo definire *state* $\rightarrow$ *update* dove lo stato s sarà lo stato di cui vogliamo calcolarne il valore, mentre l'obiettivo y è V_π. Però il valore esatto dell'obiettivo non lo si conosce, perciò lo si sostituirà con l'aggiornamento (update) u e si cercherà di "muovere" il valore di s verso il valore u. Nell'apprendimento supervisionato si andrà a creare un bacino di dati (esempi) che avranno il loro valore vero già definito e si andrà perciò ad allenare l'agente sulle y di tali esempi. Come? Si avranno gli esempi x associati alla loro corrispondente etichetta y. Quindi si avrà $x_1, y_1, x_2, y_2...$. Per trovare i parametri migliori viene utilizzato spesso lo scarto quadratico medio, perciò si andrà a minimizzare l'errore tra l'approssimazione $\hat{y}(x_i)$ e l'etichetta dell'esempio "preso in esame":

$$\frac{1}{N}\sum_{i=1}^{N}(\hat{y}(x_i) - y_i) \tag{4.28}$$

La somma viene fatta su tutti gli N esempi contenuti nel bacino di dati.
Questo procedimento lo si spiegherà meglio nel capitolo riguardante le reti neurali, però lo si può utilizzare per spiegare l'errore della funzione valore $\bar{VE}$:

$$\bar{VE} := \sum_{s\in S}\mu(s)(V_\pi(s) - \hat{V}(s,w))^2 = \mathbb{E}_\mu[(V_\pi(s) - \hat{V}(s,w))^2] \tag{4.29}$$

Dove $\mu(s)$ è la probabilità di distribuzione sugli stati, cioè ci dice l'importanza di ogni stato, mentre w è il parametro peso. Perciò abbiamo un problema di ottimizzazione dove si dovrà trovare w che minimizza $\bar{VE}$:

$$\bar{VE}(w) = arg_w MIN \tag{4.30}$$

Come si può trovare il parametro w che minimizza $\bar{VE}$? Definendo una funzione di costo (funzione di loss) parametrizzata per w e facendo il gradiente $\nabla J(w)$. J(w) totale è una somma, quindi ha senso partire dal singolo addendo:

$$J(w) = V_\pi(s) - \hat{V}(s,w) \tag{4.31}$$

Lo stato vien scelto dall'agente, quindi si ha dipendenza solo da w.

$$\Delta w = -\frac{1}{2}\alpha\nabla J(w) \tag{4.32}$$

Dove α è il passo di apprendimento (learning rate) e l'errore w è il gradiente della funzione di costo. Quindi utilizzando l'eq. (4.31) nella eq. (4.32) si otterrà:

$$\Delta w = \alpha(V_\pi(s) - \hat{V}(s,w))\nabla(\hat{V}(s,w)) \tag{4.33}$$

Δw è il valore di quanto si modificherà il valore iniziale del parametro w, cioè si quantifica l'errore fatto su w. L'errore quindi è una percentuale del gradiente, dipendente dal passo di apprendimento. Infine la regola di aggiornamento di w sarà $w + \Delta w$. Per minimizzare bisognerà far arrivare questo vettore w a un valore tale che l'approssimatore utilizzato sull'insieme di allenamento, abbia una funzione di costo il più bassa possibile. Questo metodo si definisce discesa del gradiente. Come detto non si è in possesso, del valore $v_\pi(s)$, quindi nella funzione di costo si sostituirà appunto $V_\pi(s)$ con l'obiettivo u. Visto che dall'insieme di dati che si costruisce lo stato S_t e il target u_t sono presi casualmente allora si parlerà di discesa del gradiente stocastica. Con $\bar{VE} = \sum_{s\in S} \dots$. la somma su tutti gli stati è ipotetica, infatti si è detto che se gli stati sono troppi allora si ricorre alla approssimazione. Nello specifico si andrà a campionare lo spazio degli stati (questo metodo si chiama mini-batch). Riassumendo: per minimizzare devo fare il gradiente totale di $\bar{VE}$, però non ci si riesce perchè S_t è troppo grande, quindi si stima. Come si farà a stimare tutta questa grande somma? Semplicemente la si stima con un singolo pezzo, con un singolo addendo $\nabla(V_\pi(s) - \hat{V}(s,w))^2$[1]. Come fa questa stima, man mano che la si fa più volte in maniera iterativa, a darci il gradiente totale? Per stimare $\nabla\bar{VE}$ si prendono gli stati uno ad uno stocasticamente di conseguenza il metodo di discesa del gradiente che stiamo utilizzando si chiamerà discesa del gradiente stocastico. Quando si trova il minimo sui campioni si dice che si è arrivati a convergenza. Utilizzare degli approssimatori su metodi basati sul valore (value-based) comporta ad avere problemi di convergenza.

4.4 METODI POLICY GRADIENT

Per quanto visto fino ad ora per trovare la policy migliore, si è costretti a passare prima da una funzione valore che deve essere trovata. Da qui sorge spontanea la domanda: non ci sono metodi che imparano direttamente la policy, senza quindi dover prima passare per un'altra cosa? La risposta a questo quesito la danno i metodi di apprendimento per policy (policy lear-

[1] $\mu(s)$ si calcola da sola, dal fatto che si campionano gli stati seguendo proprio questa distribuzione

ning). Si rimane comunque nell'ambito della approssimazione. La policy è una distribuzione di probabilità data sugli stati quindi sarà necessario un metodo e dei parametri per poter parametrizzare queste distribuzioni. Si utilizzerà un vettore di parametri $\theta \in \mathbb{R}^d$, perciò si avrà:

$$\pi_\theta(a|s) = \pi(a|s,\theta) = Pr(A_t = a|S_t = s, \theta_t = \theta) \tag{4.34}$$

Quindi π diventa dipendente da θ. Dato un θ, l'approssimatore sarà la forma funzionale scelta nella quale ci saranno questi parametri θ, interpretati come dei pesi (formalmente come w). In che modo si può costruire la forma funzionale approssimata vista nella sezione precedente? Per prima cosa si andranno a formare quelle che vengono definite preferenze per la coppia stato azione pesate h(s,a,θ) per θ. Si otterrà una successione di numerica dove si avranno dei numeri che tanto più sono alti, tanto più quell'azione verrà preferita alle altre (probabilità non normalizzate). Poi servirà un qualcosa che possa trasformare questa successione in una distribuzione di probabilità. Questo qualcosa ha il nome di soft-max (eq. 4.35). Alle preferenze h(s,a,θ) non si vogliono imporre vincoli di positività, eccetera... quindi le si trasforma tramite la funzione esponenziale e poi si normalizza con la somma su tutte le azioni per renderle una distribuzione di probabilità:

$$\pi_\theta(a|s) = \frac{e^{h(s,a,\theta)}}{\sum_{b \in A} e^{h(s,b,\theta)}} \tag{4.35}$$

Quindi attraverso il soft-max[1] si otterrà: preferenze più alte = probabilità più alte. I vantaggi del soft-max[2] sono:

- Si trasferisce l'apprendimento su una funzione di preferenze che non ha vincoli particolari.
- Questo tipo di scelta può imparare delle policy stocastiche. Cosa non possibile con un con metodi basati sul valore (value-based).
- Il soft-max è flessibile, quindi se la policy ottimale è deterministica anziché stocastica semplicemente la preferenza di quell'azione ottimale in quel determinato stato verrà mandata con valori sempre più alti, ovvero darà una probabilità $\pi_\theta(a|s)$ molto alta per l'azione deterministica rendendo tutte altre trascurabili, qundi la policy sarà essenzialmente deterministica.

[1]Distribuzione di Boltzmann, modo standard di trasformare dei numeri reali in distribuzione di probabilità.

[2]Nelle reti neurali, questo sarà implementato aggiungendo alla fine della nostra rete neurale uno strato soft-max.

Quanto la policy la si vuole più o meno deterministica (ad esempio certi θ anche se bassi si vuole usarli)?. Si inserisce il parametro temperatura T[1] nella soft-max:

$$\pi_\theta(a|s) = \frac{e^{\frac{h(s,a,\theta)}{T}}}{\sum_{b\in A} e^{\frac{h(s,b,\theta)}{T}}} \tag{4.36}$$

Se T tenderà a zero si avrà una distribuzione deterministica invece se T tenderà ad infinito si avrà una distribuzione molto lontana dall'essere deterministica.

4.4.1 FUNZIONE OBIETTIVO

Affinché l'agente possa imparare, si deve avere una soluzione che possa dire quando una policy è migliore dell'altra, quindi come si può misurare la qualità di una policy? Si utilizza una funzione obiettivo parametrizzata per θ (eq. 4.37) andando poi ad affrontare un problema di ottimizzazione:

$$J(\theta) = \sum_{s\in S} \mu_\theta(s) V_{\pi_\theta}(s) \tag{4.37}$$

Dove $\mu_{\theta,p}(s)$ è la distribuzione di probabilità su s che pesa l'importanza dello stato (quante volte lo si visita). $J(\theta)$ la si andrà a massimizzare o minimizzare utilizzando l'ascesa o la discesa del gradiente.

4.4.2 COME SI CALCOLA IL GRADIENTE

Consideriamo un MDP episodico avente un solo episodio. L'MDP episodico comincia da uno stato s_0 e termina dopo un certo time step T. Per l'MDP considerato si calcola il gradiente $\nabla J(\theta)$, utilizzando $q_{\pi_\theta}(s_0, a)$ come valore di ritorno, della funzione $J(\theta)$ del valore iniziale $J(\theta) = V_{\pi_\theta}(s_0)$. Si ricorda che la volontà è quella di andare a trovare direttamente la policy migliore senza dover passare dalla funzione valore, quindi il gradiente verrà calcolato sulla policy. Infine si sceglie di andare a massimizzare la funzione obiettivo:

$$\begin{aligned} \nabla J(\theta) = \nabla(V_{\pi_\theta}(s_0)) = \nabla(\sum_a \pi_\theta(a|s_0) q_{\pi_\theta}(s_0, a)) = \\ = \sum_a \nabla\pi_\theta(a|s_0) q_{\pi_\theta}(s_0, a) + \pi_\theta(a|s_0) \nabla q_{\pi_\theta}(s_0, a) \end{aligned} \tag{4.38}$$

[1]Nelle reti neurali T è un iperparametro.

Però si ha un solo episodio, quindi si avrà:

$$\sum_{a} \nabla \pi_\theta(a|s_0) q_{\pi_\theta}(s_0, a) \tag{4.39}$$

L'equazione (4.39), per gli MDP con più di un episodio, verrà utilizzata in maniera ricorsiva. Quindi il gradiente $\nabla J(\theta)$ della funzione del valore iniziale $J(\theta)V_{\pi_\theta}(s_0)$ è dato da:

$$\nabla J(\theta) = K \sum_{s} \mu(s) \sum_{a} q_{\pi_\theta}(s, a) \nabla \pi_\theta(a|s) \tag{4.40}$$

Dove la costante K è la lunghezza media degli episodi e $\mu(s)$ è la policy[1] utilizzata π_θ. Per poter campionare ed eseguire una stima si scriverà il gradiente come un valore atteso:

$$\nabla J(\theta) \alpha \sum_{s} \mu(s) \sum_{a} q_{\pi_\theta}(s, a) \nabla \pi_\theta(a|s) \tag{4.41}$$

$$\mathbb{E}_\pi[\sum_{a} q_{\pi_\theta}(s, a) \nabla \pi_\theta(a|s)] \tag{4.42}$$

K viene assorbito dal passo di apprendimento α. Ad ogni passo si andrà a vedere il valore di ciascuna azione e il valore del gradiente della policy utilizzata per quell'azione per quello stato, li si moltiplica e infine si eseguirà la somma su tutte le azioni. Si otterrà la regola di aggiornamento policy gradient, cioè un algoritmo che porta ad aggiornare il gradiente nella direzione che coinvolge tutte le azioni.
I metodi che apprendono direttamente dalla policy utilizzando il gradiente, vengono definiti policy gradient. I vantaggi che si ottengono sono:

- Questi metodi possono essere utilizzati in spazi di stato continui.
- La policy può essere una funzione molto più semplice da approssimare.
- Come parametrizzare la policy, ovvero la scelta delle h(a,s,θ). Questa scelta permette di inserire una conoscenza ottenuta a priori sul problema.
- Risolvono il problema di discontinuità sul valore delle azioni dato da una assenza di soluzione di continuità. Esempio: azione che vale 10 e una che va alle 9 e la policy sceglie quella migliore al 99% e l'altra all'1%. Improvvisamente il 9 diventa 11. Una policy senza soluzione

[1] si parla di on-policy (policy online).

di continuità passerà dalla scelta del 99% sull'azione che vale 10 e 1% sull'azione che vale 9, a 99% l'azione che ora vale 11 e 1% l'azione che vale 10, in maniera "immediata" creando un punto di discontinuità dove sarà complicato calcolare il gradiente, rendendo di conseguenza difficile la convergenza. Invece con la parametrizzazione dell'apprendimento diretto della policy, ho possibilità di far scendere e alzare le probabilità dando continuità. Questo da proprietà di convergenza molto migliori rispetto ai metodi basati sul valore.

4.5 NOTE DI FINE CAPITOLO

In questa breve sezione si vogliono definire altri termini che vengono trovati nell'ambito dell'apprendimento per rinforzo.

- model-free: quando non si ha il modello, cioè quando non si ha la distribuzione di probabilità p della transizione dallo stato s a quello successivo nota a priori.
- model-based: in maniera opposta, quando si dispone del modello si utilizzano algoritmi basati sul modello.
- off-policy: si utilizzano più policy differenti per l'apprendimento.
- on-policy: si utilizza la stessa policy per l'apprendimento.
- metodi actor-critic: metodi che apprendono sia dalla funzione valore che dalla policy. L'attore è la policy, quella che esegue le scelte, mentre la critica è la funzione valore usata per quantificare quanto le scelte della policy siano valide.

Capitolo 5

RETI NEURALI

Le reti neurali, come altre invenzioni dell'uomo, sono state concepite basandosi su un sistema biologico. In questo caso ci si è chiesti se si potesse replicare artificialmente li elementi e i processi che compongono il cervello umano, complessa struttura neurobiologica composta da un mattone elementare che caratterizza tutte le strutture celebrali, una cellula denominata neurone. I neuroni sono composti da un corpo detto soma e da due tipi di diramazioni: i dentriti e l'assone. Nel cervello umano sono presenti tipicamente oltre 100 miliardi di neuroni, ciascuno interconnesso a circa altri 10.000. In questo modello biologico l'assone riceve l'input dagli altri neuroni i dentriti trasmettono le informazioni alle altre celle e il soma decide quando e come trasmettere sulla base di vari criteri. Nelle interconnessioni ha luogo la sinapsi, un processo elettrochimico atto a rinforzare o inibire l'interazione cellulare. Come avviene tutto questo? Nel neurone biologico le correnti elettriche cominciano a caricare il nucleo, quando si supera una certa soglia, viene emesso un certo segnale che arriverà agli altri neuroni dove ripeteranno la stessa operazione ricorsivamente. Mappando il cervello si è scoperto che questi neuroni sono spesso organizzati in strati disposti consecutivamente. Ora, nel cercare di costruire un neurone artificiale si è arrivati nel 1957 all'invenzione del percettrone (perceptron, fig. 5.1), basato su una unità logica di soglia (treshold logic unit, TLU). Input e output ora sono numeri e ogni input è associato a un peso. La TLU computa una somma pesata dei suoi input creando un vettore $z = w_1x_1 + w_2x_2 + \ldots.. + w_nx_n + b$ alle quali viene applicata una funzione f detta funzione di attivazione. Tale f serve per calcolare l'output del percettrone e quindi servirà a simulare l'operazione, descritta in precedenza, che esegue il neurone biologico.

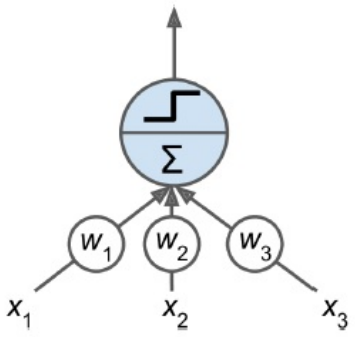

Figura 5.1: percettrone [13]

Uno o più neuroni formano uno strato (layer). Con gli starti di input,output e uno nascosto si costruisce una rete neurale, mentre con più strati nascosti si parlerà di reti neurali profonde (deep neural network) e quindi di apprendimento profondo (deep learning). Quando tutti i neuroni di uno strato sono connessi a ogni neurone dello strato precedente allora si avrà una rete neurale con strati completamente collegati (fully connected layer, fc). Tutto questo viene rappresentato in figura 5.2.

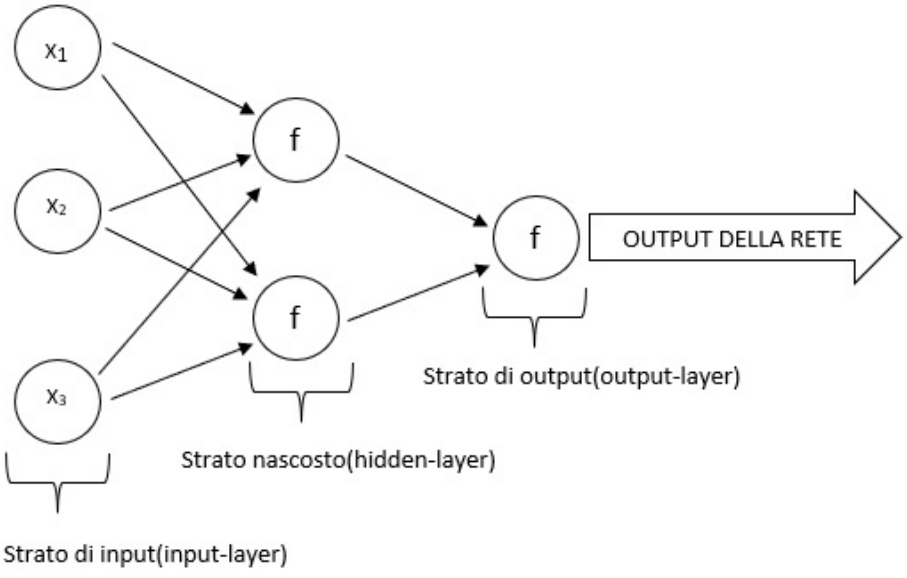

Figura 5.2: Schema rete neurale

Si prenda come esempio di nuovo la figura 5.2, una semplicissima rete neurale fc composta da un solo strato nascosto. Si andrà a descrivere quello che accade nel primo neurone dello strato nascosto:

$$f(w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3 + b_1^2) \tag{5.1}$$

Dove:

- x_n: x rappresenta l'input iniziale dove n indica il numero dell'input dello strato di input. Gli input x_n possono essere rappresentati tramite il vettore $\vec{x}$.
- w_{ij}^k: peso w che andrà nella matrice dei pesi W. Il primo indice i indica che sarà l'input dell' i-esimo neurone dello strato k-esimo, mentre il secondo indice j indica a quale input appartiene il peso.
- b_i^k: si ha una distorsione (bias) per ogni i-esimo neurone che compone lo strato k-esimo. I neuroni di input non hanno distorsioni.

Quindi:

$$\begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} z_1^2 \\ z_2^2 \end{bmatrix} \tag{5.2}$$

a z_1^2 e z_2^2 verranno sommate le rispettive distorsioni b_1^2 e b_2^2. Si può scrivere anche:

$$\begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix} = w^2 x + b^2 = z^2 \tag{5.3}$$

w^2 ha 6 parametri. Nello strato di output si avrà:

$$f(w_{11}^3 o_1 + w_{12}^3 o_2 + b^3) \tag{5.4}$$

dove o_n è l'output del neurone n-esimo dello strato precedente k-esimo-1 che diventa l'input dello strato k-esimo. w^3 ha 2 parametri. Infine la funzione di output F sarà uguale a:

$$F(x; w, b) = f(w^3 f(w^2 x + b^2) + b^3) = f(w^3 f(z^2) + b^3) = f(z^3) \tag{5.5}$$

Come viene addestrata la rete?

- Tramite apprendimento supervisionato: alla rete viene dato in ingresso un bacino di N esempi (set di dati costruito) che saranno costituiti dagli input $x^1, ..., x^N$. Di questi input, si conoscono già gli output associati $y(x^1), ..., y(x^N)$, **etichette** che indicano la risposta corretta degli input.

- Tramite apprendimento non supervisionato: alla rete viene dato impasto una serie di esempi (set-di dati costruito) che saranno costituiti dagli input $x^1, ..., x^N$. però, di questi esempi non si conoscono gli output associati, quindi sarà la stessa rete che dovrà trovare le etichette $y(x^1), ..., y(x^N)$.

L'apprendimento supervisionato è molto utilizzato. A fronte di questi input la rete neurale produrrà un output del tipo visto nella eq. (5.5). Quindi, si dovrà trovare un modo per misurare la "distanza" tra l'eq. (5.5) in uscita dalla rete e gli output (etichette) di questi esempi. Si definirà una funzione di costo C parametrizzata attraverso i parametri peso w e distorsione b:

$$C(w,b) = \frac{1}{N}\sum_x \frac{1}{2}\left|y(x) - F(x;w,b)\right|^2 \tag{5.6}$$

Dove, come scritto in precedenza nel primo dei due punti che rispondevano alla domanda "Come viene addestrata la rete?", N è il numero di esempi del set di dati costruito, x sono gli input degli N esempi, y(x) le etichette che indicano la risposta corretta degli input x degli esempi e $F(x;w,b)$ è l'output prodotto dagli input x degli esempi. Si avrà un problema di ottimizzazione della funzione di costo C, che si vorrà risolvere andando a minimizzarla utilizzando il gradiente:

$$\nabla C(w,b) = \frac{1}{N}\sum_x \frac{1}{2}\nabla\left|y(x) - F(x;w,b)\right|^2 \tag{5.7}$$

Per approssimare, si dovrà avere una media aritmetica su un numero di esempi M più piccolo di N. Il numero di esempi $x^1, ..., x^M$ si definisce mini-batch. Quindi si avrà:

$$\nabla C(w,b) = \frac{1}{M}\sum_{m=1}^{M} \frac{1}{2}\nabla\left|y(x^m) - F(x^m;w,b)\right|^2 \tag{5.8}$$

Gli M esempi vengono scelti a caso, quindi si parlerà di discesa stocastica del gradiente (SDG).

5.1 ALGORITMO DI APPRENDIMENTO

Le reti neurali utilizzano l'algoritmo SDG per apprendere. Si andrà ad illustrare il suo funzionamento:

1. Si mescola il set di allenamento composto dagli N esempi $x^1, ..., x^N$.
2. Si forma il mini-batch $x^1, ..., x^M$.
3. Per ogni x^m si calcola:

$$\nabla_w \left|y(x^m) - F(x^m; w, b)\right|^2 \tag{5.9}$$

$$\nabla_b \left|y(x^m) - F(x^m; w, b)\right|^2 \tag{5.10}$$

4. Dal punto 3 si ottiene:

$$\sum_{m=1}^{M} \nabla_w C_{x^m}(w, b) \tag{5.11}$$

$$\sum_{m=1}^{M} \nabla_b C_{x^m}(w, b) \tag{5.12}$$

5. Ora si possono definire le regole di aggiornamento dei parametri w e b. w^0 e b^o sono i parametri attuali mentre quelli con l'apice 1 sono i parametri aggiornati. η è il passo di apprendimento:

$$w^1 = w^0 + \eta \frac{1}{M} \sum_{m=1}^{M} \nabla_w C_{x^m}(w^0, b^0) \tag{5.13}$$

$$b^1 = b^0 + \eta \frac{1}{M} \sum_{m=1}^{M} \nabla_b C_{x^m}(w^0, b^0) \tag{5.14}$$

6. Si ritorna al punto 2 e si ripete.
7. Quando terminano i $\frac{N}{M}$ mini-batch si dice che è conclusa un'epoca (epoch), quindi si torna in 1 e si inizia una nuova epoca.

Infine, il gradiente viene calcolato dall'algoritmo backpropagation.

5.2 RETI NEURALI CONVOLUZIONALI

Le reti neurali convoluzionali[1] (convolutional neural networks, CNNs) hanno origine dagli studi condotti sulla corteccia visiva del cervello. Inizialmente il problema che veniva posto a questa CNN era il riconoscimento delle immagini, campo in cui grazie alla tecnologia odierna si è riusciti ad ottenere risultati molto importanti. Al giorno d'oggi, le CNNs vengono utilizzate anche per il perseguimento di obiettivi (task) di altro tipo. Ma perché si è deciso di puntare sulle CNN e non su delle reti neurali completamente connesse (fc)? Le fc presentano due punti critici: stessa architettura per tutti i tipi di dati e tutti i problemi, il numero di parametri è molto elevato e questo rappresenta un ostacolo nella costruzioni di reti neurali. Le CNNs sono la risposta a queste criticità. Una CNN ha:

- architettura precisa per dati con una struttura spaziale (o temporale, perché il tempo è uno spazio di dimensione 1), questo perché si vuole tenere conto della posizione di certe quantità che vengono rilevate.
- connessioni sparse, cioè tra uno strato e il successivo il numero delle connessioni è molto piccolo, quindi di conseguenza si avrà un numero di parametri piccolo.
- condivisione dei parametri, cioè ci saranno dei gruppi di connessioni che avranno li stessi pesi e di conseguenza anche questo ridurrà il numero di parametri.

Rispetto a una fc i dati vengono organizzati in matrici e gli input pesati formano le mappe delle caratteristiche (feature map). Le CNN sono composte da strati di convoluzione, strati di pooling e filtri. Per spiegare il loro funzionamento si userà come esempio l'utilizzo di tali reti nel riconoscimento delle immagini.

5.2.1 STRATO DI CONVOLUZIONE

L'elemento costitutivo più importante di una CNN è lo strato di convoluzione. Nel caso del riconoscimento delle immagini ogni pixel dell'immagine corrisponderà a un neurone. Ogni neurone avrà un valore e, per quanto detto in precedenza, questi valori faranno parte di una matrice. Ad esempio per un'immagine in full-HD si avrà una matrice 1920x1080.

[1] operazione matematica di convoluzione

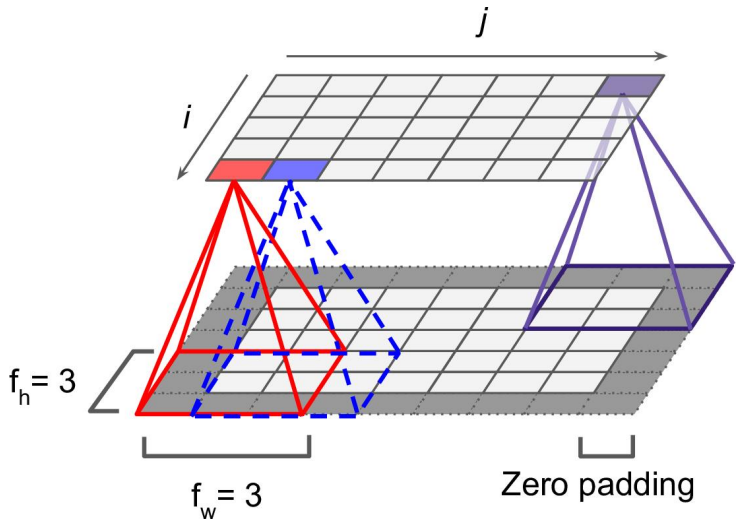

Figura 5.3: strato di convoluzione di una CNN[13]

Come si è detto all'inizio non ci saranno tutti i collegamenti di una fc, quindi i neuroni del primo strato non sono connessi a ogni singolo pixel dell'immagine data in input ma solo ai pixel che fanno parte di una finestra, chiamata campo ricettivo, di determinate dimensioni. A sua volta, ogni neurone nel secondo strato di convoluzione è connesso solo ai neuroni che faranno parte anch'essi di un altro campo ricettivo nel primo strato di convoluzione, e così via. Perciò, un neurone situato nella posizione (riga i, colonna j) di un dato strato è connesso all'output dei neuroni dello strato precedente situati nella posizione: riga da i a $i + f_h$–1, colonna da j a $j + f_w - 1$ dove f_h e f_w sono altezza e larghezza del campo ricettivo (figura 5.3). Per uno strato che ha la stessa altezza e larghezza dello strato precedente è comune aggiungere zeri attorno agli input. Questa cosa viene chiamata zero padding (figura 5.3). È anche possibile connettere all'input di uno strato superiore, uno strato (precedente) di dimensioni più grandi (fig. 5.4). Lo spostamento da un campo ricettivo a quello successivo viene chiamato stride. Un neurone situato nella posizione (riga i colonna j) dello strato superiore è connesso agli output dei neuroni dello strato precedente situati nella posizione: riga da $i \times s_h$ a $i \times s_h + f_h - 1$, colonna da $j \times s_w$ a $j \times s_w + f_w - 1$, dove s_h e s_w sono gli strides verticali e orizzontali (figura 5.4).

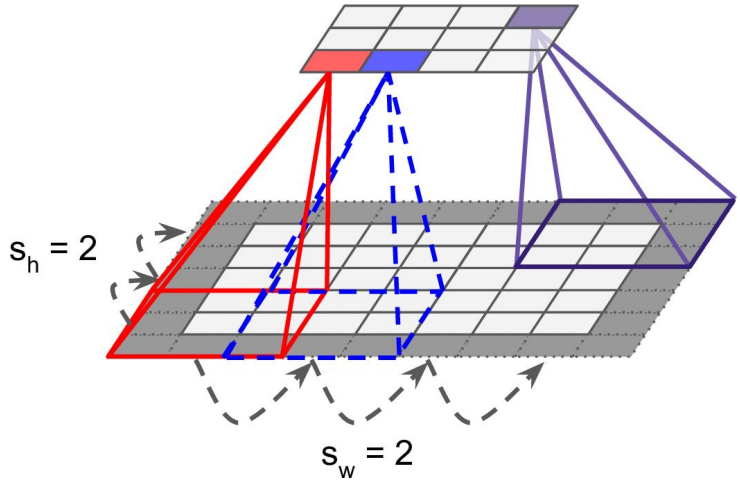

Figura 5.4: connessione a uno strato di convoluzione più piccolo[13]

Questa architettura consente di far concentrare la rete sulle caratteristiche dell'immagine in input ed estrarle per creare le mappe delle caratteristiche che impilate una sopra all'altra comporranno lo strato di convoluzione (fig. 5.6).

5.2.2 FILTRI

I neuroni all'interno del campo ricettivo vengono pesati tramite una matrice di pesi W. La matrice dei pesi costituisce il filtro o kernel di convoluzione. Gli output che daranno i filtri saranno le caratteristiche estratte utilizzate per creare le mappe delle caratteristiche. Come operano i filtri? Consideriamo l'immagine di figura 5.5 come input della CNN. Quindi, dopo aver ricevuto in input l'immagine in bianco e nero, la rete neurale comincerà ad applicare un kernel a un campo ricettivo avente (ad esempio) dimensioni 5×5. Si supponga che la rete per il primo strato di convoluzione crei due mappe di caratteristiche: una in cui si "esalteranno" le linee verticali bianche (tutto il reso sarà sfocato), mentre nell'altra verranno "esaltate" le linee orizzontali bianche (tutto il reso sarà sfocato). Come dovranno essere le matrici dei pesi? Il primo kernel sarà costituito da tutti 0 ad eccezione della colonna centrale che sarà riempita con il valore 1. Così facendo, i neuroni usando questi pesi ignoreranno tutto nel loro campo ricettivo eccetto per la linea verticale centrale. Invece, il secondo kernel sarà costituito da tutti 0 ad eccezione della riga centrale che sarà riempita con il valore 1. Così facendo, i neuroni usando questi pesi ignoreranno tutto nel loro campo ricettivo eccetto per la linea orizzontale centrale.

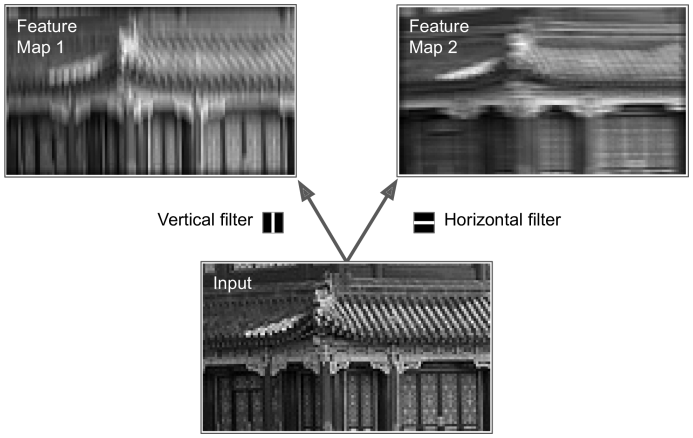

Figura 5.5: Esempio applicazione dei filtri su un'immagine[13]

Durante l'addestramento della CNN, sempre attraverso gli algoritmi visti nella sezione 5.1, lo strato convoluzionale imparerà automaticamente il filtro che risulterà più utile per il suo obiettivo (task) assegnatoli. Tuttavia, una CNN è in grado di applicare kernel multipli allo stesso campo ricettivo ottenendo quindi in output più mappe impilate una sopra l'altra (fig. 5.6). Si sta sempre parlando di una rete neurale, perciò come saranno connessi i neuroni? Le connessioni vengono sviluppate attraverso le mappe delle caratteristiche. Un neurone situato nella riga i, colonna j della mappa k in un dato strato di convoluzione l è connesso agli output dei neuroni nel precedente strato l–1, situato nella riga: da $i \times s_h$ a $i \times s_h + f_h$–1 e colonna: da $j \times s_w$ a $j \times s_w + f_w$–1 attraverso tutte le mappe dello strato $l - 1$. Più semplicemente, tutti i neuroni situati nella stessa riga i e colonna j ma in mappe delle caratteristiche differenti, sono connessi agli output degli stessi identici neuroni dello strato precedente (figura 5.6). Infine, **l'operazione di convoluzione** non è altro che una media pesata dei neuroni all'interno del campo ricettivo, dove l'equazione sarà:

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f'_n-1} x_{i',j',k'} w_{u,v,k',k} \tag{5.15}$$

$$con = \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} \tag{5.16}$$

- $z_{i,j,k}$ è l'output del neurone situato nella riga i,colonna j nella mappa della caratteristica k dello strato di convoluzionale (strato l).

- s_h e s_w sono gli srtides verticali e orizzontali, f_h e f_w sono l'altezza e la larghezza del campo ricettivo, e f'_n il numero della mappa della caratteristica dello strato precedente (strato $l-1$).

- $x_{i',j',k'}$ è l'output del neurone situato nello strato l–1, riga i', colonna j', mappa della caratteristica k' (o canale k' se lo strato precedente è lo strato di l'input).

- b_k è la distorsione per la mappa della caratteristica k (nello strato l). Si può pensare ad esso come una manopola che ottimizza la luminosità complessiva della mappa k.

- wu, v, k', k è la connessione peso tra ogni neurone nella mappa della caratteristica k dello strato l e il suo input situato nella riga u, colonna v (relativo la campo percettivo del neurone) e mappa della caratteristica k'.

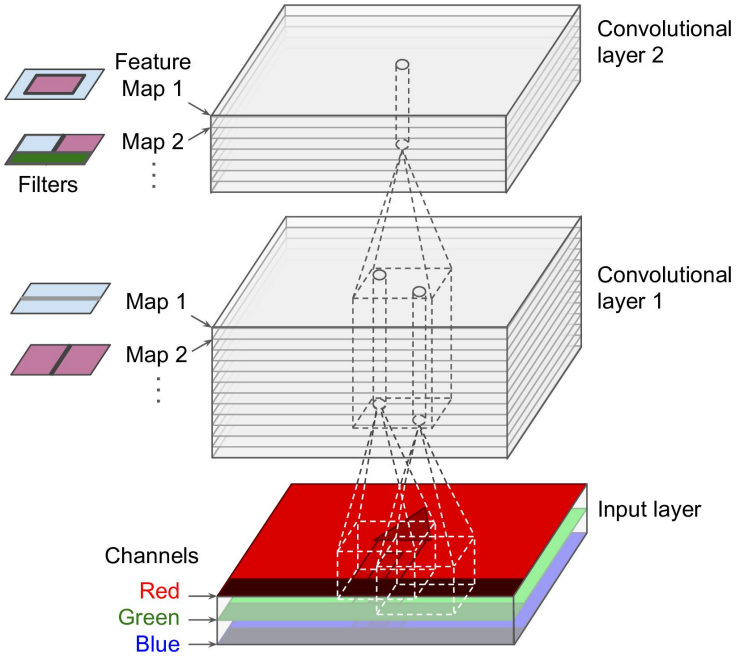

Figura 5.6: Esempio CNN con più strati di convoluzione[13]

5.2.3 STRATO DI POOLING

È comune trovare nelle CNN un blocco chiamato strato di pooling. L'obiettivo di questo strato è di sotto-campionare l'immagine in input in modo da

ridurre il numero di parametri e quindi di conseguenza il carico computazionale. Come nello strato di convoluzione, ogni neurone in un pooling layer è connesso agli output dei neuroni appartenenti al campo ricettivo dello strato precedente. Si dovrà definire quindi le dimensioni del campo ricettivo sempre attraverso f_h, f_w, s_h e s_w. Tuttavia, un neurone appartenente allo strato di pooling non viene pesato. Nell'operazione di pooling viene fatta una aggregazione degli input utilizzando una funzione come il massimo o la media. Infine, nel pooling i campi percettivi non si sovrappongono come nella convoluzione (figura 5.7).

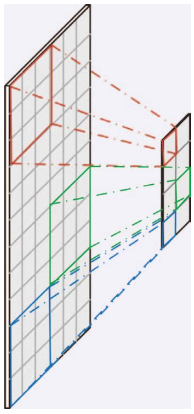

Figura 5.7: Operazione di pooling[6]

5.2.4 ARCHITETTURE CNN

Tipicamente le architetture CNN fanno susseguire: strato convoluzionale (ognuno generalmente seguito da uno statro avente la funzione di attivazione ReLU applicata, ad oggi, quasi sempre in tutte le reti neurali), poi strato di pooling, poi ancora strato convoluzionale, poi ancora strato di pooling e così via. Utilizzando sempre un'immagine come esempio, attraversando la rete, diventerà via via sempre più piccola. Alla fine sarà aggiunta una rete neurale feedfoward (reti neurali dove le connessioni tra neuroni non formano cicli), composta da alcuni strati fc, con lo strato finale che darà l'output (figura 5.8).

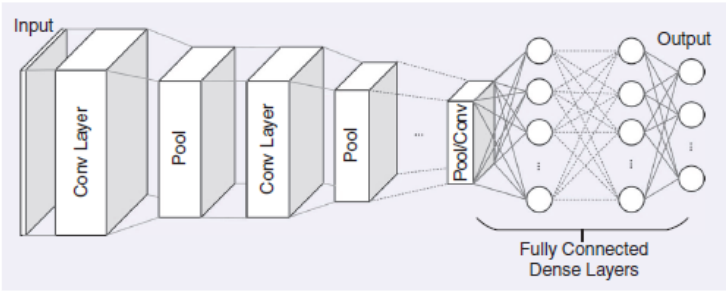

Figura 5.8: Architettura CNN[6]

Se dopo questa architettura se ne mette un'altra a specchio che riceverà in input gli output della prima ed eseguirà le operazioni di deconvoluzione e di unpooling avremo una rete auto-encoder. La prima rete codificherà la seconda decodificherà (figura 5.9).

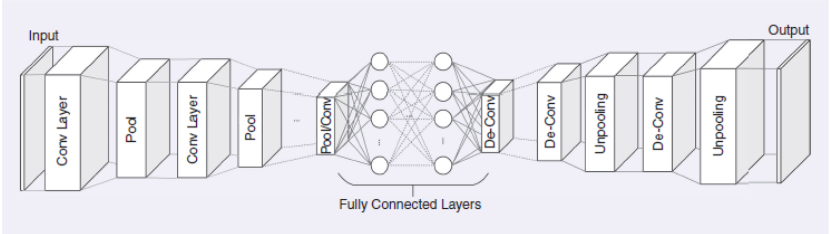

Figura 5.9: Rete CNN auto-encoder[6]

5.3 RETI NEURALI A GRAFO

Ma se la struttura dati, che dovrebbe ricevere la rete neurale in input, al contrario dei pixel di un'immagine non appartenesse a uno spazio euclideo (figura 5.10)? Come si riuscirebbe a definire un campo ricettivo con i conseguenti filtri da applicare? Questi sono i motivi per cui si studiano le reti neurali basate su strutture a grafo (GNN), reti dal grande potere espressivo.

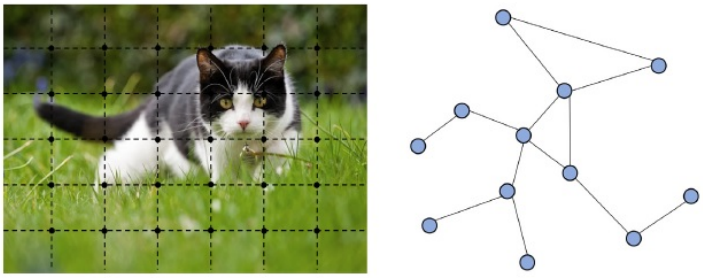

Figura 5.10: a destra immagine in uno spazio euclideo, a sinistra grafo in uno spazio non euclideo[10]

Prima di procedere, è istintivo porsi il seguente quesito: ma cos'è un grafo? Dalla teoria dei grafi si andranno a estrapolare le seguenti definizioni:

Definizione 1 *Un grafo $G = (V, E)$ è composto da un insieme finito di nodi (o vertici) V e da un insieme di archi $E \subset V \times V$ che connettono coppie di nodi. Due nodi connessi da un arco sono detti adiacenti. Dato $e \in E$, esistono v_i e v_j tali che $e = (v_i, v_j)$.*

Definizione 2 *Dati V ed E, sia $w : E \to \mathbb{R}$ una funzione che associa un valore (o costo) ad ogni arco. Il grafo $G = (V, E, w)$ è un grafo pesato (o iper-grafo).*

Definizione 3 *la matrice di adiacenza di un grafo G è una matrice simmetrica A di dimensione $n \times n$, in cui si rappresentano le relazioni di adiacenza nella rete del grafo:*

$$[A(G)]_{i,j} = a_{i,j} = \begin{cases} 1 & se(v_i, v_j) \in E \\ 0 & altrimenti \end{cases} \tag{5.17}$$

Gli archi vengono chiamati anche collegamenti o edges. Una GNN viene utilizzata in due tipi di contesti: dati che hanno una struttura relazionale esplicita come ad esempio le molecole (figura 5.11) e dati la cui struttura razionale è implicita o assente come ad esempio un testo (figura 5.12).

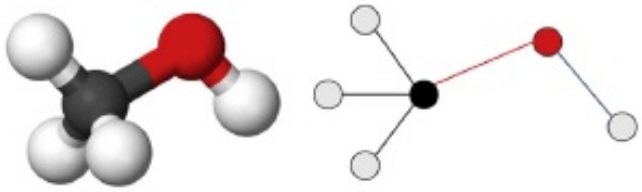

Figura 5.11: molecola[10]

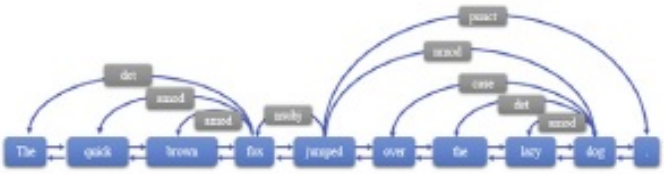

Figura 5.12: testo[10]

Quindi dopo aver definito la struttura del grafo nel contesto in cui si vorrà applicare la GNN, si dovrà specificare il tipo di grafo. I grafi sono tipicamente suddivisi in:

- Grafi diretti/indiretti:

  Definizione 4 *Un grafo G = (V, E) è un grafo orientato (grafo diretto o digrafo) se E è formato da coppie ordinate di nodi. L'arco* (v_i, v_j) *è un arco da* v_i *a* v_j*, si definisce* v_j *come la testa dell'arco e* v_i *come la coda dell'arco.*

  Se questa definizione non viene rispettata allora il grafo si dice indiretto.

- Grafi omogenei/eterogenei: negli omogenei nodi e collegamenti sono della stessa tipologia. Il contrario negli eterogenei.

- Grafi statici/dinamici: se gli input o la tipologia del grafo cambiano nel tempo allora il grafo è dinamico, al contrario si dice statico.

Fatto questo, si arriva all'aggiunta e all'utilizzo di quelli che vengono chiamati moduli computazionali. Questi moduli servono per ottenere in output l'incorporamento: del nodo, del collegamento e della struttura del grafo. Un

incorporamento (embedding) è una operazione in cui è possibile trasportare vettori di grandi dimensioni all'interno di uno spazio, detto spazio di incorporamento, avente dimensione piccola. Questa operazione viene utilizzata per facilitare l'apprendimento su input di grandi dimensioni. Idealmente, un incorporamento cattura parte dell'input posizionando input simili vicini, nello spazio di incorporamento. Un incorporamento può essere appreso e riutilizzato tra i vari modelli. Alcuni dei moduli più utilizzati sono:

- modulo di propagazione: usato per propagare l'informazione tra i nodi. In questa tipologia di moduli viene utilizzato ad esempio l'operatore convoluzionale che ha lo scopo di aggregare l'informazione proveniente dai nodi adiacenti (neighbors). L'operatore di convoluzione è il più utilizzato e l'idea principale che gli sta dietro è quella di generalizzare la convoluzione da un altro dominio al dominio dei grafi.

- modulo di campionamento: le GNN aggregano per ogni nodo l'informazione proveniente dalle sue adiacenze dello strato di GNN precedente. Parlando di apprendimento profondo e quindi da più strati di GNN, queste adiacenze tenderanno a crescere in maniera esponenziale. Una soluzione efficiente ed efficace per ridurre questo problema di dimensionalità, è appunto il campionamento.

- modulo di pooling: ha lo stesso scopo dello strato di pooling visto nella sezione 5.2.3.

Per l'apprendimento della GNN si usa la funzione di costo (loss). Viene definita in base all'obiettivo (task) e al set di allenamento. Gli obiettivi possono essere posti su tre tipi di livelli:

- livello del nodo: gli obiettivi si concentrano nei nodi come, la classificazione dei nodi, la predizione sui valori dei nodi e il creare cluster al cui interno si raggruppano nodi simili tra loro.

- livello del collegamento: li obiettivi si concentrano nei collegamenti, come la classificazione dei collegamenti se si conosce il modello o la predizione sulla possibile esistenza di un collegamento tra due nodi dati.

- livello del grafo: gli obiettivi si concentrano sul grafo come, la classificazione del grafo o la corrispondenza dei grafi. Per porre questi tipi di obiettivi bisogna apprendere la rappresentazione del grafo e ciò necessita il modello.

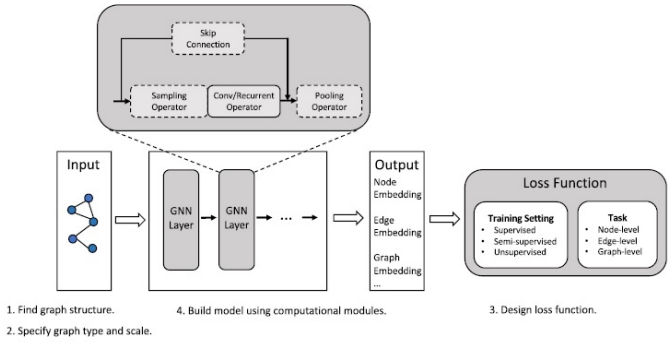

Figura 5.13: Architettura generica di una GNN[10]

5.4 NOTE DI FINE CAPITOLO

In questa breve sezione si vogliono definire altri termini che vengono trovati nell'ambito dell'apprendimento supervisionato.

- iper-parametri: gli iper-parametri sono variabili esterne alla rete che consentono di controllare il processo di allenamento del modello supervisionato. Sono regolabili e spesso impostati manualmente dall'utilizzatore in modo sperimentale e iterativo (perchè non riassumibili da una funzione). Ad esempio, per le reti neurali, il numero di livelli nascosti e il numero di nodi in ogni livello vengono considerati iper-parametri. Le prestazioni del modello supervisionato dipendono molto dalla scelta degli iper-parametri.

- over-fitting: l'over-fitting avviene quando una rete neurale, invece di imparare a generalizzare sui dati di allenamento (e quindi comprendere ciò che sta facendo), si limita ad imparare a memoria tali dati, avendo quindi degli scarsi risultati quando si tratta di vedere i suoi progressi sui dati di test. Ad esempio, se si è costruita una rete neurale in grado di riconoscere il numero uno, se in over-fitting riconoscerà soltanto l'uno di destra e non entrambi (figura 5.14).

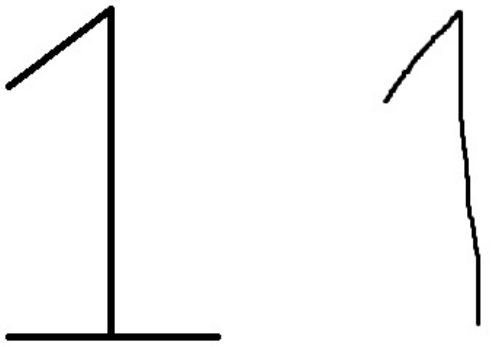

Figura 5.14: Esempio over-fitting

Capitolo 6

APPLICAZIONI

Per l'applicazione dei metodi di machine learning visti fino ad ora, nella progettazione di circuiti integrati, si userà come riferimento [7]. In [7] lo scopo è usare il machine learning per sostituire la parte umana nell'operazione di piazzamento delle macro celle. Si parte dalla netlist creata dall'operazione di sintesi. Si definiscono le macro celle che saranno poi posizionate attraverso l'apprendimento per rinforzo profondo (combinazione tra RL e NN) e i raggruppamenti (clustering) delle celle standard, i quali vengono posizionati nello stato finale con un metodo force-directed. Le macro celle vengono indicizzate, tramite un'identità id, seguendo un ordine decrescente per dimensioni. L'agente sceglierà la macro da piazzare seguendo questo id.

6.1 APPRENDIMENTO PER RINFORZO

Per il processo decisionale di Markov (figura 6.1), stato, azione e ricompensa vengono definiti nel seguente modo:

- Stato: lo stato contiene informazioni codificate riguardo: il piazzamento parziale delle macro celle avvenuto o che deve ancora avvenire includendo la matrice di adiacenza corrispondente alla netlist, le caratteristiche dei nodi (larghezza, altezza, tipo), le caratteristiche dei collegamenti (numero di connessioni), il nodo preso in considerazione (macro cella) per il piazzamento e i meta-dati della netlist (allocazioni di instradamento del segnale, numero totale di fili, macro e raggruppamenti di celle standard). È stato settato uno stato iniziale s_0 avente un chip canvas vuoto e uno stato finale s_T che corrisponde a una netlist completamente piazzata (includente il raggruppamento di celle standard).

- Azione: le azioni sono tutte le possibili posizioni (celle della griglia 128x128 del chip canvas) in cui la macro corrente può essere collocata senza violare alcun vincolo. Ai fini dell'ottimizzazione delle policy, hanno convertito il canvas in una griglia m x n. In questo modo, per ogni dato stato, lo spazio delle azioni è la distribuzione di probabilità (l'output della rete di policy) del posizionamento della macro corrente sulla griglia m x n. L'azione viene quindi campionata da questa distribuzione di probabilità. Lo spazio delle azioni è composto da tutti i piazzamenti possibili della macro corrente.

- Ricompensa: le ricompense sono 0 per tutte le azioni tranne che per l'ultima, dove la ricompensa è una somma negativa ponderata della lunghezza dei fli metallici, della congestione e della densità.

$$R_{p,g} = -Wirelenght(p,g) - \lambda Congestion(p,g) - \gamma Density(p,g) \quad (6.1)$$

Viene utilizzato il proximal policy optimization (PPO) che è un metodo policy gradient. Si hanno la rete della policy e del valore perché la PPO in una rete neurale viene implementata come un metodo actor-critic. Si sta parlando di policy gradient, quindi la funzione obiettivo viene definita come segue:

$$J(\theta, G) = \frac{1}{K} \sum_{g=G} \mathbb{E}_{g,p=\pi_\theta}[R_{p,g}]$$

Dove $J(\theta, G)$ per la rete neurale è la funzione di costo. L'agente è parametrizzato tramite θ. Il dataset della netlist di dimensione K è denotata tramite G, dove ogni singola netlist è rappresentata come g. $R_{p,g}$ è la ricompensa dell'episodio di un piazzamento p estratto dalla rete di policy applicata alla netlist g. $\mathbb{E}_{g,p=\pi_\theta}[R_{p,g}]$ è il valore atteso della ricompensa, data una netlist g e piazzamento p ricavati dalla policy π_θ.

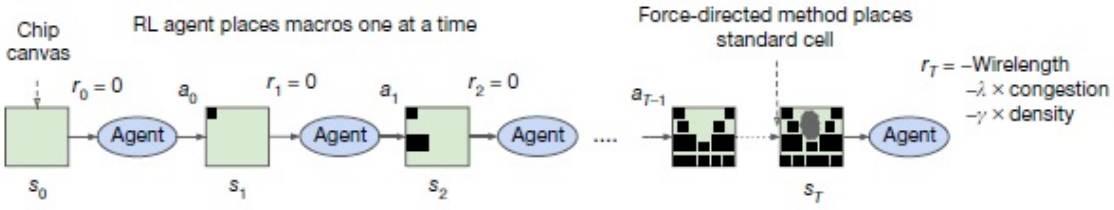

Figura 6.1: Processo decisionale di Markov[7]

6.2 ARCHITETTURA RETE NEURALE

Si andrà a descrivere la costruzione dell'architettura della rete neurale utilizzata (figura 6.2). Per processare i dati in input è stata sviluppata una rete neurale a grafo convoluzionale (quindi con moduli di propagazione convoluzionali) basata sui collegamenti (Edge-based GCN, va a lavorare sulle adiacenze e i collegamenti di tali adiacenze). Però prima, si deve individuare la struttura a grafo, in questo caso un iper-grafo, della netlist. Questo iper-grafo lo si converte in una matrice di adiacenza. La conversione viene eseguita seguendo questa regola: per ogni coppia di nodi nella netlist clusterizzata (sia macro che celle standard del cluster), si genera un collegamento nella matrice di adiacenza definendo il peso associato ad esso con una determinata regola. Se la distanza di registro tra i due nodi è maggiore di 4, non viene creato alcun collegamento. Altrimenti, si applica un peso che decresce esponenzialmente al crescere della distanza, iniziando con 1 se la distanza è 0 e dimezzando per ogni unità di distanza aggiuntiva. Alla GCN vengo dati in input la matrice di adiacenza della netlist e le caratteristiche della macro cella. Il ruolo della Edge-GNN è quello incorporare informazioni sul tipo e sulla connettività di un nodo. L'operazione di incorporamento del grafo serve a creare una **rappresentazione vettoriale** (da questo punto in poi si userà RV) dei dati in input. Queste RV numeriche sono le caratteristiche utilizzate per la classificazione. Nello specifico viene creata una RV iniziale di ogni nodo concatenando le sue caratteristiche, tra cui il tipo di nodo, la larghezza, l'altezza e le coordinate x e y e la sua connettività con gli altri nodi. La Edge-GCN darà in output l'incorporamento dei collegamenti (l'incorporamento avviene tramite l'applicazione di una funzione apposita) della netlist parzialmente piazzata fino a quel momento e l'incorporamento della macro cella che dovrà essere piazzata dall'agente. Inoltre, viene creata una rete fc feed forward per l'incorporamento dei meta-dati riguardanti la netlist. Questi meta-dati includono: tecnologia del semiconduttore (capacità di routing verticale e orizzontale), il numero totale dei nets (edges), macro e raggruppamenti delle celle standard, dimensione del chip canvas (suddiviso in una griglia con un massimo di 128 colonne e 128 righe). Successivamente, l'output della Edge-GCN e l'incorporamento dei meta-data della netlist vengono concatenati attraverso una fc feedforward per dare in output l'incorporamento dello stato. Quest'ultimo viene dato in pasto ad un'altra fc feedforward il cui output è la RV finale dello stato utilizzata come input:

- per la rete della policy composta da cinque strati convoluzione aventi campo ricettivo di dimensioni 3x3 e stride 2, composti corrispettiva-

mente da 16,8,4,1 mappe delle caratteristiche. Questa rete genera la distribuzione di probabilità sulle azioni.

- per la rete del valore composta da una rete feed-forward. Darà la predizione sul valore dello stato in input.

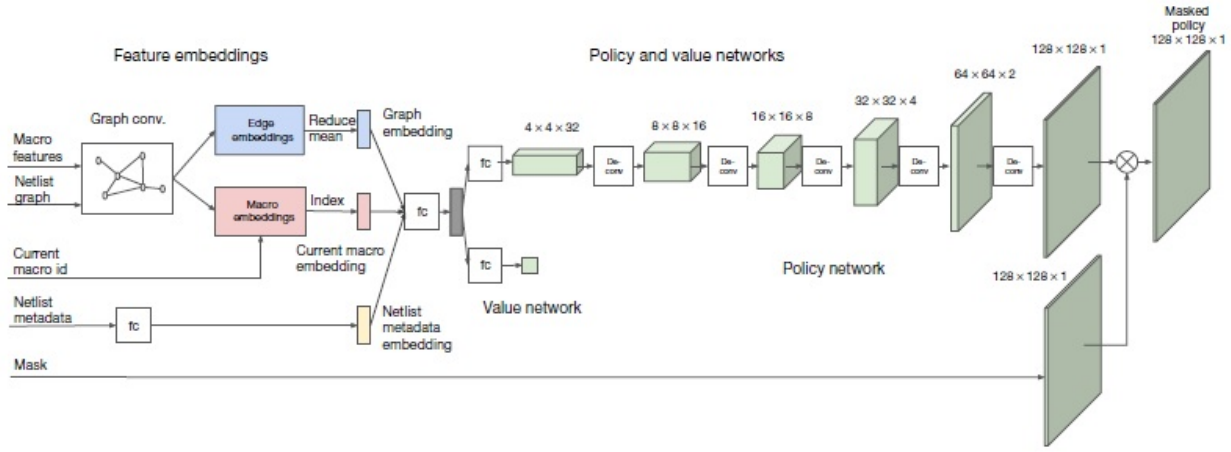

Figura 6.2: Architettura della rete neurale[7]

6.3 RISULTATI

Come si è visto nel capitolo 5, si è soliti costruire un set di dati composto da esempi su cui si fa allenare la rete. Inoltre, su tali reti viene eseguita l'operazione di messa a punto che verrà spiegata di seguito. Le reti neurali profonde sono strutture stratificate e hanno molti iper-parametri "sintonizzabili". Il ruolo degli strati iniziali è quello di estrarre caratteristiche generiche mentre quelli successivi estraggono caratteristiche di ordine superiore, concentrandosi maggiormente sull'obiettivo da svolgere. Attraverso la sintonizzazione di questi iper-parametri si andrà selettivamente ad eseguire la **messa a punto** delle RV di queste caratteristiche di ordine superiore, appartenenti ad alcuni strati, per renderle più rilevanti nel raggiungimento dell'obiettivo (task). La messa a punto viene eseguita durante questo raggiungimento. Per allenare la rete della policy, in [7] si è costruito un set di dati (un ampio set di dati fornisce robustezza al problrma dell'over-fitting) composto da esempi di chip, dove l'etichetta y(x) sarà la ricompensa (lunghezza dei fili e congestione) per il piazzamento del singolo esempio. Il modello supervisionato è stato utilizzato poi, per generare il piazzamento di Ariane. Ariane è un processore open

source a 64 bit che implementa il set di istruzioni RISC-V. Per capire l'importanza dell'addestramento e della messa a punto, nella figura 6.3 viene fatto eseguire tale processo utilizzando: una rete della policy senza allenamento (linea blu, non essendo stata allenata comincerà con pesi randomizzati) e una rete della policy allenata e messa a punto (linea verde). Si nota come la rete non allenata ci mette molto più tempo per andare a convergenza, ottenendo inoltre un costo (in termini di piazzamento) maggiore.

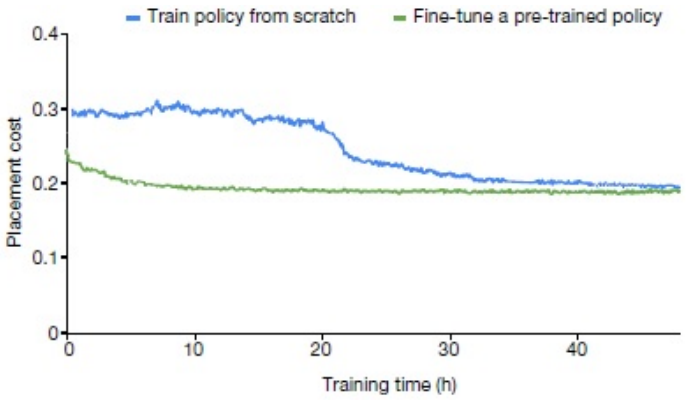

Figura 6.3: [7]

La linea verde viene fisicamente rappresentata nella figura 6.4 dove: nell'immagine a sinistra si ha il primo piazzamento fatto, **in meno di un secondo**, dalla rete della policy allenata, mentre nell'immagine destra il piazzamento eseguito dalla stessa rete della policy messa a punto andata a convergenza.

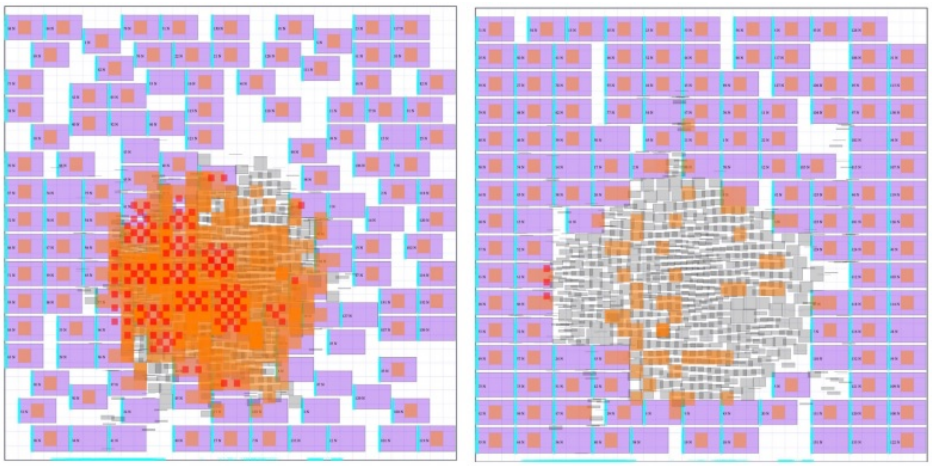

Figura 6.4: Piazzamento macro celle e raggruppamenti di celle standard di Ariane[7]

Oltre ad Ariane, il modello supervisionato è stato testato anche sulla progettazione di una TPU. Una tensor processing unit (TPU) è un acceleratore di intelligenza artificiale (IA) costituito da un circuito ASIC, per applicazioni specifiche nel campo delle reti neurali. Si sono generati i piazzamenti per 5 blocchi (fig. 6.6) di TPU e poi, nella tabella di figura 6.5, li si è confrontati con le TPU, della generazione precedente, progettate da personale umano.

Name	Method	Timing		Total area (μm^2)	Total power (W)	Wirelength (m)	Congestion	
		WNS (ps)	TNS (ns)				H (%)	V (%)
Block 1								
	Manual	136	47.6	1,680,790	3.74	51.12	0.13	0.03
	Our method	84	23.3	1,681,767	3.59	51.29	0.34	0.03
Block 2								
	Manual	75	98.1	830,470	3.56	62.92	0.23	0.04
	Our method	59	170	694,757	3.13	59.11	0.45	0.03
Block 3								
	Manual	18	0.2	869,779	1.42	20.74	0.22	0.07
	Our method	11	2.2	868,101	1.38	20.80	0.04	0.04
Block 4								
	Manual	58	17.9	947,766	2.17	29.16	0.00	0.01
	Our method	52	0.7	942,867	2.21	28.50	0.03	0.02
Block 5								
	Manual	107	97.2	1,480,881	3.23	37.99	0.00	0.01
	Ours	68	141.0	1,472,302	3.28	36.59	0.01	0.03

Figura 6.5: Tabella confronto manuale, modello supervisionato. Nei dati più basso è meglio[7]

Dove li slack negativi totali (TNS), il peggior slack negativo (WNS) e la congestione del routing: orizzontale H, verticale V, sono dei vincoli imposti, mentre la lunghezza dei fili metallici (wirelenght), potenza (power) e area occupata (area), sono le misure da ottimizzare. Sia il metodo supervisionato che il personale specializzato generano piazzamenti validi, che soddisfano i vincoli di slack e congestione. Tuttavia, il metodo supervisionato supera o eguaglia i piazzamenti "manuali" per quanto riguarda la potenza, l'area occupata e la lunghezza dei fili.

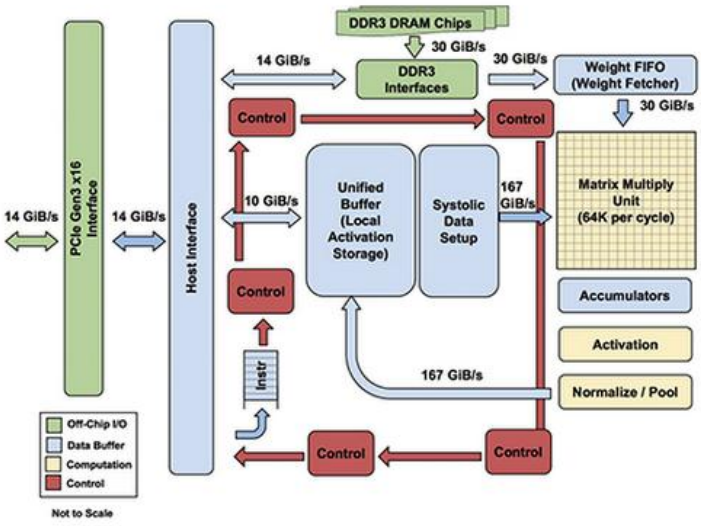

Figura 6.6: Blocco TPU[11]

Capitolo 7

CONCLUSIONI

Lo scopo della tesi era esplorare il machine learning con il fine di poterlo utilizzare nella progettazione dei circuiti integrati. Perciò, si è andati prima a cercare il perché dell'utilizzo di tali tecniche, riscontrando un problema nell'innalzamento dei costi, dovuto all'aumento del personale ingegneri (figura 3.2), e nell'allungamento dei tempi (figura 3.3) per la progettazione di un circuito integrato, andando a vedere nella sezione 3.2 dove è possibile attualmente avvalersi del machine learning. Quindi si è visto che tipo di algoritmi appartengono al machine learning e come funzionano attraverso i capitoli riguardanti l'apprendimento per rinforzo e reti neurali, andando infine a vederne l'applicazione (nel capitolo 6 si utilizzano tali metodi per la fase di piazzamento delle macro celle di un circuito integrato) e i risultati raggiunti. Specificatamente, nella sezione 6.3 si è detto che il metodo supervisionato sviluppato in [7] supera o eguaglia i piazzamenti "manuali" però con la differenza che gli algoritmi di machine learning, eseguiti sull'architettura "volta" di Nvidia (scheda grafica), **ci mettono 6 ore** ad arrivare a convergenza e generare un blocco di TPU che soddisfi i criteri di progettazione, mentre il personale specializzato **impiega mesi**. Il tempo impiegato per la generazione di un piazzamento valido per la manifattura è un punto fondamentale perché, secondo [14], si sta avendo un divario tra la legge di moore e la produttività degli IC causato dal tempo speso da parte degli ingegneri nelle operazioni di verifica (figura 3.3). Di conseguenza gli algoritmi di machine learning sono essenziali per ridurre questo divario. Infine, si arriva al problema dei costi esposto nel capitolo 3. **Il costo del personale ingegneri** (ed del loro eventuale addestramento) viene "sostituito" e quindi ridotto, con il semplice acquisto e mantenimento di tecnologia hardware (schede video) apposita. Quindi si sono raggiunti gli obiettivi di riduzione dei costi e dei tempi per la progettazione di un IC specificatamente nella fase di piazzamento, però come visto nel capitolo 3 i problemi presentati non riguardano

soltanto questo punto (figura 3.1). Perciò, si dovrà continuare nel tempo ad investire sempre di più in questo settore arrivando in un futuro prossimo ad avere delle intelligenze artificiali che interagiscano con l'uomo in tempo reale per aiutarlo a sviluppare, in tutte le fasi di progettazione e verifica, circuiti integrati molto complessi in maniera "semplice e veloce".

Bibliografia

1. Chiplet heterogeneous integration technology – Status and challenges. Tao Li, Jie Hou, Jinli Yan, Runlin Liu, Hui Yang and Zhigang Sun. Electronics, April 2020.
2. DARPA Rolls Out Electronics Resurengence Initiative (ERI). OUTREACH@DARPA.MIL, 9/13/2017.
3. Wilson Research Group functional verification study 2020. Siemens.com
4. https://www.youtube.com/watch?v=cIlwGFcDLhI Systemverilog Academy
5. https://it.mathworks.com/discovery/reinforcement-learning.html Mathworks
6. Deep Learning for consumer devices and services. Joseph Lemly, Shabab Bazrafkan, Peter Corcoran. IEE CONSUMER ELECTRONICS MAGAZINE April 2017.
7. A graph placement methodology for fast chip design. Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean. Nature, 9 June 2021.
8. DeepMind x UCL — Deep Learning Lecture Series 2021.
9. Corso di dottorato su machine learning e reti neurali, UniFi, Maurizio Parton. https://www.youtube.com/user/videomaumau
10. Graph neural networks: A review of methods and applications. Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyang Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun. Puplished by Elsevier B.V. on behalf of KeAi Communications Co. Ltd. January 2021.
11. https://cloud.google.com/blog/products/ai-machine-learning/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu
12. Updates of ITRS Design Cost and Power Models. Gary Smith. IEEE, 2014.
13. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.

Aurelien Geron. O'REILLY, 2019.
14. Why the design productivity gap never happened. Harry D. Foster. IEEE, 2013.