POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Machine Learning per la Predizione della Qualità di un Canale Wi-Fi

Relatori
Dr. Stefano Scanzio
Dr. Gianluca Cena

Candidato
Alberto Salvatore Colletto

Aprile 2023

Indice

Ringraziamenti

Quello che ho realizzato durante questi anni non sarebbe stato possibile senza l'affetto e l'appoggio di tante persone, che mi hanno aiutato e supportato in questo lungo percorso.

Senza i miei genitori, i miei fratelli e le mie cognatine, non avrei intrapreso questa strada. Mi avete dato un supporto enorme, sopratutto prima di prendere la decisione di partire. Senza questo supporto probabilmente non avrei avuto la forza di rischiare e di partire per Torino, per la paura di non riuscire a vivere da solo lontano da voi. Mi avete sempre aiutato a riprendermi nei momenti di sconforto e mi avete sempre dato la carica per ricominciare più forte e sereno di prima. Mi avete insegnato che la cosa più importante è impegnarsi al massimo per ottenere i risultati prefissati; magari le cose non andranno nel verso giusto, ma con la consapevolezza di avercela messa tutta. Avete fatto davvero tantissimi sacrifici per me, e non mi avete mai fatto pesare niente. Non vi potrò mai ringraziare abbastanza per tutto il supporto e l'amore che mi avete dato. Grazie alle mie nonnine, che nonostante la distanza mi fanno sentire sempre piccolino e viziato come un tempo. Alcuni amici vanno e vengono, ma di sicuro Angelo e Francesco resteranno sempre al mio fianco. Nonostante il tempo passi per tutti e nonostante siamo lontani da molto tempo, ogni volta che ci rivediamo sembra non sia passato nemmeno un giorno dall'ultima volta, passando il tempo a scherzare e a parlare del futuro tra un aperitivo al bar e un bel caffè all'Ufficio. Da voi ho imparato quanto importante sia la sincerità. Non importa quanto duri si possa essere, non importa quanto male possa fare, non bisogna mai nascondere la verità o quello che si pensa, soprattutto in amicizia.

Grazie a Giuseppe, con cui ho condiviso l'amore per il wrestling e per il cibo da piccolo e l'amore per gli anime e per il mondo videoludico oggi. Il gigante buono di cui tutti avremmo bisogno, sempre pronto ad ascoltare e a dare ottimi consigli, che spesso non ascolto.

Grazie a Giorgio e Denise, che ho rincontrato dopo tanto tempo. Da voi ho

imparato come qualunque situazione si possa alleggerire con una bella risata.

Grazie a Chiara, con cui ho condiviso praticamente tutta la scuola dalle elementari fino al quinto anno di liceo, e che ho ritrovato dopo anni anche qui a Torino. Sempre affiancati, sempre vicini, sempre pronta a darmi il tuo supporto. Mi hai insegnato che non si deve mai perdere la speranza, soprattutto in quelle situazioni sembrano impossibili.

Grazie a Sonia, la persona da cui ho ascoltato forse il messaggio vocale più lungo di sempre. Da te ho visto una dedizione nello studio e nel lavoro che ho visto solo in mio padre e in pochi altri, e nonostante tutto riesci sempre a trovare del tempo da condividere con me.

Grazie a Martina, che ci guardi da lassù da ormai troppo tempo. Avrei davvero voluto avere più tempo per parlare di tutti quei gruppi musicali che tanto ti contestavo, e che poi ho imparato ad amare. Tu mi hai insegnato cosa significa davvero lottare nella vita, in un modo così semplice e genuino che sembra quasi assurdo. Manchi tanto.

Grazie a Marta, Federica, Lorena e Jack, che mi avete accolto nel gruppo come se fossi amico vostro da sempre. Grazie per il supporto costante che mi date, e grazie soprattutto a Marta per aggiornarmi sulle novità del paesello.

Grazie a Giacomo, Alessandro, Silvia e Dario, con cui ho affrontato i primi anni torinesi nella stessa casa. Praticamente siete la mia seconda famiglia. Mi avete aiutato davvero tanto nei primi anni, in cui mi sentivo solo e senza un posto a cui appartenere.

Grazie a Matteo, Igor, Enrico, Marcello e Alessandra, con cui ho condiviso i momenti più belli e spensierati da quando sono a Torino.

Grazie al Prof. Scanzio e al Prof. Cena, che mi hanno seguito passo passo verso la realizzazione di questa tesi. Nonostante i mille impegni, siete sempre stati a disposizione per ogni mio dubbio.

Un grazie speciale va a te. La persona che più di tutti ha creduto in me. La persona che mi ha supportato di più. La persona che mi ha sopportato di più. La persona che mi ha fatto ridere più di tutti. La persona con cui ho pianto di più. La persona che mi ha consolato di più. La persona che mi dato una scossa quando ero sul punto di mollare. La persona con cui ho condiviso tutto. La persona che ha fatto di tutto per starmi accanto e che ora è qui con me. La persona che reputo la persona perfetta per me. La persona che amo follemente, e con cui voglio condividere il resto della mia vita. Grazie di esistere e di voler stare con me, Elisa.

Sommario

La qualità di una comunicazione wireless varia inevitabilmente nel tempo, per molte ragioni che spesso non si conoscono durante la fase di design di un sistema distribuito. Questo è un problema primario quando le tecnologie di rete senza fili vengono utilizzate per connettere dispositivi e sottosistemi in applicazioni industriali, dove sono richiesti requisiti quali alta affidabilità e traffico dati i cui ritardi devono essere bassi, limitati e deterministici. L'abilità di prevedere, con un certo margine di errore, la qualità del collegamento nel prossimo futuro può portare a grossi benefici in molti scenari applicativi. In particolar, questa conoscenza può permettere la regolazione proattiva dei parametri di comunicazione o delle caratteristiche del traffico trasmesso.

In questa tesi, verranno analizzate e comparate le prestazioni di due approcci, basati su una semplice media mobile e sull'uso delle reti neurali, rispettivamente, che realizzeranno delle previsioni per connessioni Wi-Fi. I risultati confermano che stime attendibili relative alla percentuale di pacchetti inviati con successo possano essere ottenute in condizioni realistiche mediante l'uso di tecniche di machine learning basate su reti neurali.

Capitolo 1

Introduzione

Negli ultimi anni, l'evoluzione tecnologica che ha subito il mondo dell'informatica, in particolar modo lo sviluppo delle connessioni Wi-Fi che, in ambienti domestici, offre delle prestazioni paragonabili a quelle fornite da una connessione Ethernet, ha fatto si che questa tecnologia possa essere utilizzata in contesti industriali. In particolare, essa permette la riduzione dei cablaggi e una gestione più flessibile del processo produttivo. In questo contesto però, dove la condivisione di informazioni tra i vari componenti di una industria deve avvenire senza errori o rallentamenti, il Wi-Fi risulta non essere ancora all'altezza, poiché il traffico scambiato con tecnologia wireless può subire interferenze o disturbi, ad esempio a causa di nodi Wi-Fi o operanti con altre tecnologie. Può succedere infatti che due o più dispositivi operanti nello stesso canale Wi-Fi si disturbino a vicenda, creando delle collisioni tra i frame inviati nella rete, che possono provocare rallentamenti consistenti. Anche se alcune tecniche di gestione dei frame aiutano la rete a limitare questo effetto, non ci sono modi per risolverlo del tutto, anche perché gli errori derivati da disturbi esterni non si possono eliminare poiché difficilmente individuabili, e spesso fuori dal controllo del progettista e del manutentore di rete. Una strategia possibile per mitigare il problema consiste nel trovare un metodo di analisi in grado di *prevedere* l'andamento del canale in certe finestre temporali, cercando così di prevenire situazioni in cui la comunicazione tra dispositivi risulti peggiore rispetto ai requisiti richiesti dalla rete industriale. Alcuni algoritmi che tentano di risolvere questa problematica si basano sul presupposto che conoscendo le statistiche sui frame precedentemente trasmessi sul canale si possano ottenere informazioni utili per prevedere la qualità

dello stesso nel futuro. Questa assunzione è il punto di partenza su cui si basa questo studio, in cui verrà analizzato il comportamento di alcuni canali Wi-Fi, usando alcuni algoritmi di Machine Learning (ML) ed in particolare tramite l'uso delle Reti Neurali Artificiali (Artificial Neural Networks, ANNs), che si stanno rivelando sempre più indispensabili nei problemi relativi alla previsione di diverse tipologie di serie temporali. L'impiego delle ANNs è stato già sperimentato in passato [1, 2, 3], e comparato con alcuni algoritmi euristici che tentano di prevedere la qualità di un canale Wi-Fi rielaborando alcune informazioni relative ai frame inviati, e si è visto come queste reti neurali presentino dei risultati generalmente migliori. Questi studi sono preliminari e sono stati effettuati su database di dimensione ridotta. L'obiettivo principale di questo studio sarà quello di testare le tecniche precedentemente analizzate e di implementare nuove proposte considerando una quantità di dati significativamente più elevata. Sono state analizzate architetture più complesse di rete neurale e tipologie di reti quali Long Short-Term Memory (LSTM), teoricamente più adatte nella previsione di serie temporali. Per analizzare e sperimentare le tecniche proposte, è stato sviluppato un software ex-novo in grado di rielaborare le statistiche relative ai frame inviati nella rete e di creare diverse topologie di rete neurale che verranno poi addestrate tramitei database raccolti in precedenza. In particolare, il software permette di calcolare il *tasso di successo di un canale*, dove per tasso di successo si intende una media che esprime il rapporto tra il numero di frame correttamente ricevuti da un dispositivo B e il totale dei frame inviati da un dispositivo A, attraverso un canale Wi-Fi specifico. Tramite l'uso di questi strumenti, verrà analizzato il comportamento delle varie reti neurali considerando diversi fattori: verrà dapprima visto il comportamento della rete considerando un tempo futuro di previsione della qualità del canale fisso, al quale verranno forniti in ingresso un numero crescente di feature, per verificare che la rete neurale riesca sempre ad assimilare informazioni utili per la previsione della qualità dello stesso, oppure si arrivi ad un punto in cui non sia più possibile migliorare le previsioni effettuate dal modello. Poi verrà analizzato il comportamento della stessa tramite l'uso dati acquisiti in altri canali Wi-Fi che non siano quelli di riferimento. In un successivo esperimento, la rete fornirà delle previsioni su diverse finestre temporali future, mantenendo questa volta fisso il numero di feature che la stessa utilizzerà nella fase di addestramento, cercando inoltre di capire se una topologia basata su singola rete che effettua previsioni su più finestre temporali future risulti più accurata rispetto alla implementazione di diverse reti neurali specializzate nell'analisi di

un determinato periodo. Infine, verranno eseguiti degli esperimenti considerando le reti di tipo LSTM, per comprendere quale tra le due topologie risulti più accurata nella previsione dei risultati.
La tesi verrà organizzata come segue:

- Capitolo 2, in cui verrà spiegato il funzionamento dello standard Wi-Fi e di altri standard relativi a tipologia di rete cablate e wireless, considerando la loro applicazione in contesti di tipo industriale.
- Capitolo 3, in cui verrà descritto il concetto di rete neurale, e verrà approfondito il funzionamento delle diverse topologie utilizzate in questo studio.
- Capitolo 4, in cui verrà analizzato il modo in cui i dati vengono acquisiti a partire dalla comunicazione tra due dispositivi attraverso un canale Wi-Fi specifico.
- Capitolo 5, in cui verranno definiti i metodi di rielaborazione dei dati raccolti dai vari canali e i metodi di confronto basati su algoritmi euristici.
- Capitolo 6, in cui verrà presentato il programma realizzato per effettuare le operazioni descritte nei capitoli precedenti, e che verrà utilizzato per raccogliere le statistiche dei modelli analizzati.
- Capitoli 7 e 8, in cui verranno descritti tutti gli esperimenti effettuati, verranno commentati i risultati ottenuti da ogni esperimento e verranno tratte le conclusioni relative a questa tesi.

Capitolo 2

Evoluzione e sviluppo delle reti Industriali

2.1 Introduzione

Dal 1997 lo standard IEEE 802.11 per le reti wireless (WLAN), la cosiddetta tecnologia Wi-Fi, ha visto una rapida diffusione in ambienti domestici, dando la possibilità di connettere più dispositivi contemporaneamente senza l'aggiunta di cablaggi ingombranti, ma sfruttando la comunicazione tra dispositivo e router tramite l'uso di onde elettromagnetiche. Per questa caratteristica, il Wi-Fi è stato introdotto anche in alcuni contesti industriali [4]. Poiché, però, nelle industrie la comunicazione tra i vari dispositivi deve avvenire in tempi ridotti rispetto ad un contesto domestico, il Wi-Fi presenta delle prestazioni scadenti se comparate con le prestazioni ottenute dalla tecnologia Ethernet, perché il traffico che viene scambiato tra i dispositivi può subire dei disturbi che possono causare un rallentamento o un arresto della connessione. Questi disturbi possono dipendere da interferenze esterne, difficilmente prevedibili dagli operatori di rete, oppure può dipendere da disturbi creati dai dispositivi nella stessa rete. Poiché la comunicazione Wi-Fi avviene in canali condivisi, può succedere che più dispositivi provino a comunicare nello stesso canale, creando internamente una coda di frame dovuta all'impossibilità di accedere al canale da parte del nodo. Partendo da questa premessa, questo capitolo spiega l'evoluzione delle diverse tecnologie utilizzate nel contesto industriale, descrivendo sia le soluzioni cablate che le soluzioni wireless, analizzando nel dettaglio le caratteristiche che le contraddistinguono. In particolare, verranno presi in esame

i seguenti parametri come metro di confronto tra le diverse tecnologie:

- Determinismo: condizione che garantisce che un messaggio scambiato tra due nodi di una rete venga inviato in un determinato periodo di tempo.
- Latenza: periodo di tempo tra l'invio del messaggio dal nodo A e la corretta ricezione al nodo B.
- Jitter: variazione di latenza su diversi scambi di messaggi.
- Sistemi Hard Real-Time: sistemi in cui ogni richiesta/risposta di rete deve essere consegnata/ricevuta prima di una scadenza, nell'ordine dei millisecondi o inferiore.
- Sistemi Soft Real-Time : stesso concetto, ma più flessibile, nel senso che le richieste non computate prima di quella scadenza non sono considerate critiche per la rete se avvengono sporadicamente.

2.2 Reti industriali cablate

Nonostante la rapida diffusione delle reti wireless nel contesto industriale, le reti cablate sono ancora la soluzione adottata. Tuttora, si stima che il 90% delle aziende usi un'architettura di questo tipo all'interno delle fabbriche. La motivazione principale è che le reti cablate risultano essere deterministiche. Dunque è possibile far comunicare o sincronizzare due o più dispositivi nella stessa rete locale, potendo così schedulare il traffico scambiato e sapendo con esattezza il tempo di elaborazioni dei dati nella rete. Inoltre, questa condizione è essenziale quando le fabbriche hanno bisogno di sistemi basati sul paradigma hard real-time, poiché è possibile definire in maniera univoca una deadline che i sistemi devono rispettare per gestire la comunicazione dei dispositivi. Il problema di questa tecnologia è che non esiste uno standard comune a tutti, ma esistono diverse soluzioni utilizzate da diverse industrie. Quando uno standard viene utilizzato, si ha la tendenza di mantenere questa configurazione per molti anni, senza apportare aggiornamenti o miglioramenti all'architettura proposta perché bisognerebbe ristrutturare l'ambiente di lavoro ad ogni innovazione, generando dei costi che le fabbriche non sono sempre pronte a sostenere. Di conseguenza, quando l'introduzione di queste innovazioni diventa indispensabile, spesso le industrie non riescono ad aggiornare in

modo graduale lo stato dell'arte del sistema complessivo, visto che le nuove tecnologie risultano spesso incompatibili con lo standard scelto in precedenza, creando un problema di scarsa interoperabilità dei dati trattati. Gli standard per le reti industriali cablate maggiormente usate si dividono in due tipologie distinte:

- *Bus di Campo*, detto anche *Fieldbus* (FB). In questo caso, il sistema viene considerato un sistema distribuito, in cui tutti i dispositivi sono considerati come nodi di una rete locale. Ognuno di essi è collegato fisicamente o logicamente agli altri, scambiando con la rete informazioni al livello fisico ed esso rielabora il traffico scambiato nel livello applicativo. Ancora molto diffusa soprattutto nelle vecchie industrie, i fieldbus presentano però molti problemi di compatibilità con le nuove tecnologie, derivati dalla presenza di tanti protocolli basati su questo sistema, e da una scarsa evoluzione dell'architettura.

- *Protocoli Real-Time Ethernet* (RTE). Il sistema viene sempre visto come distribuito, ma in questo caso le informazioni vengono scambiate al livello trasporto. In generale, la connessione Ethernet nella versione standard è una architettura non-deterministica, quindi non adatta a sistemi hard real-time. In ambito industriale, sono stati fatti molte studi per mitigare questo problema, ma con l'effetto di creare anche in questo caso una mole di protocolli incompatibili tra loro e con prestazioni molto differenti. È la soluzione maggiormente adottata attualmente, anche se in generale molto più costosa se comparata con i FBs.

In questo contesto, la sfida più importante che i ricercatori delle reti industriali stanno cercando di compiere è quella di creare un'architettura unica, che possa dunque creare uno standard unico usato da tutte le aziende nel mondo. Diverse sono le proposte, e ad oggi la soluzione candidata è Time Sensitive Networking (TSN).

2.3 Reti industriali wireless

Parallelamente alla ricerca di una rete cablata unificata, le reti wireless stanno subendo un'evoluzione sempre maggiore e molte industrie stanno iniziando ad introdurre il Wi-Fi e le tecnologie wireless in generale nell'architettura della rete locale. Anche se nel 2020 solo il 6% delle aziende utilizza un'infrastruttura basata

su rete wireless, il numero elevato di ricerche fa ben sperare per il futuro di questa tecnologia. Allo stato attuale, le reti wireless non riescono a soddisfare tutti i requisiti necessari per il processo produttivo di un'industria, perché rispetto alle reti cablate:

- Presentano una latenza più alta. Dunque un dispositivo in media riceve i dati da un trasmettitore con un ritardo maggiore.
- Non possono soddisfare richieste per sistemi hard real-time. Non sono dunque sistemi deterministici.
- Sono meno affidabili, nel senso che una comunicazione in una rete wireless ha più probabilità di perdere frame durante lo scambio di dati tra due dispositivi.

Queste sono alcune delle sfide che devono affrontare i sistemi basati su connettività wireless. Un altro obiettivo molto importante sta nel limitare il consumo di energia per le reti in cui i nodi sono difficilmente raggiungibili, senza aumentare la latenza della connessione stessa. Due classi importanti di reti industriali wireless sono le reti basate sul protocollo IEEE 802.15.4 e quelle relative al protocollo IEEE 802.11. Entrambe le tecnologie sono attualmente utilizzate per dispositivi relativi alla Industrial Internet of Things (IIoT), ma in contesti differenti. La prima soluzione viene impiegata nelle applicazioni in cui il risparmio di energia è fondamentale, mentre il Wi-Fi è tipicamente adottato in applicazioni in cui il flusso del traffico scambiato deve essere molto elevato. In questo studio verranno analizzati nel dettaglio queste due strategie, e verranno descritte le caratteristiche della connessione dati basata sul protocollo 5G, anch'essa possibile tecnologia candidata ad essere usata in contesto industriale.

2.3.1 TSCH basato sul protocollo IEEE 802.15.4

Come per le reti cablate, anche le reti wireless devono possedere un alto determinismo, una bassa latenza e un'ottima affidabilità del traffico scambiato per realizzare molte delle applicazioni utilizzate nelle industrie. Una delle reti wireless che viene usata nelle applicazioni di IIoT fa riferimento allo standard IEEE 802.15.4, protocollo che gestisce il traffico di una rete in locale a livello fisico e a livello MAC. Tra le caratteristiche che lo standard offre, la versione che è stata definita nel 2012 introduce un metodo operativo molto interessante, chiamato *Time Slotted Channel Hopping* (TSCH), che migliora notevolmente il tempo di trasmissione dei frame

nella rete, aumenta l'affidabilità del traffico scambiato e diminuisce il consumo energetico necessario per alimentare l'intera infrastruttura [5] [6]. Il funzionamento del TSCH è in certi aspetti simile al protocollo TDMA [7] ed è il seguente:

- la trasmissione viene suddivisa in slot temporali, ciascuno dei quali contiene un numero fisso di frame, chiamato *slotframe*. Una configurazione tipica potrebbe contenere 101 slotframe, ognuna facendo riferimento a 10 ms di comunicazione, questo per ogni slot temporale.

- Tutti i dispositivi della rete locale possono comunicare tra loro, ma possono farlo sfruttando uno slot temporale per volta. In questo modo si possono programmare in anticipo le trasmissioni, evitando collisioni nello stesso canale.

- Ogni slotframe può essere organizzato in maniera del tutto indipendente, in base alle informazioni che deve trasportare. Così si possono migliorare le prestazioni per quanto riguarda la latenza, l'affidabilità e il determinismo della trasmissione [8, 9]. Ad esempio, in protocolli passati, si utilizzava un meccanismo automatizzato basato su messaggi di acknowledge a livello MAC, chiamato *Automatic Repeat-reQuest* (ARQ), che gestiva la comunicazione tra due dispositivi confermando l'effettiva ricezione di un frame o, in caso di mancata ricezione dello stesso, si occupava della ritrasmissione del frame mancante. Questo meccanismo viene usato per aumentare l'affidabilità di una trasmissione, ma trattandosi di un meccanismo asincrono, la trasmissione subiva dei problemi di alta latenza, o comunque di latenza instabile. In questo standard, vi è la possibilità di riservare slot temporali che si occupino soltanto della ritrasmissione e della conferma di ricezione dei frame, diminuendo notevolmente il problema appena esposto [10].

- La comunicazione di dati tra i nodi della stessa rete viene gestita in canali diversi, scelti tramite una funzione pseudo-casuale nota a tutti i nodi. I frame vengono dunque trasmessi su frequenze sempre diverse, ottenendo dei canali generalmente meno occupati [11, 12]. Inoltre, vengono implementate delle tecniche di blacklisting, in cui alcuni canali vengono rimossi dall'insieme dei canali scelti per il traffico dei dati poiché risultano poco efficienti o non adatti alla gestione di questa tipologia di frame.

- I nodi della rete in cui non viene programmata una trasmissione, cambiano il proprio stato diventando dei nodi dormienti. Facendo così, l'energia usata in tutta la rete è notevolmente ridotta.

Il TSCH ha subito un'ulteriore evoluzione con l'introduzione di IPv6 nella architettura IEEE 802.15.4e (6TiSCH), permettendo di fatto l'implementazione del nuovo protocollo nei sensori delle reti industriali wireless. Questa nuova adozione dà la possibilità a tutti i nodi di una rete industriale di possedere un unico indirizzo di rete che lo identifichi univocamente dentro Internet, consentendo una più facile integrazione di dispositivi IIoT nell'architettura dell'industria stessa [13]. Questo standard permette anche di gestire in maniera più semplice i percorsi dei dispositivi, ovvero tutti i collegamenti in cui vengono scambiati i dati nella rete, tramite l'uso del protocollo IPv6 di routing per reti a bassa potenza e perdita (IPv6 Routing Protocol for Low-Power and Lossy Networks, RPL). L'obiettivo di questo protocollo è quello di gestire i vari collegamenti con i dispositivi della rete. Definisce per ogni nodo radice, ovvero quei nodi che fanno da tramite tra la rete locale e Internet, un grafo chiamato DODAG (Destination Oriented Directed Acyclic Graph), che altro non è che un percorso che collega i nodi radice con tutti gli altri nodi della rete locale, cercando di ottimizzare il più possibile la velocità di trasmissione dei frame. Ogni nodo ha un nodo radice preferito, cioè un nodo radice più vicino con cui comunicare, ma si possono presentare situazioni in cui questo collegamento sia compromesso. Dunque il protocollo RPL salva in una lista appropriata tutti i possibili nodi radice raggiungibili da quel nodo, così da garantire dei percorsi alternativi per la trasmissione dei dati. La possibilità di avere più percorsi disponibili per connettere i nodi di una rete potrebbe migliorare la qualità della rete wireless, diminuendo il consumo di energia di tutta l'architettura [14, 15, 16, 17, 18] e aumentandone il determinismo e l'affidabilità [19, 20, 21]. È al momento un argomento molto promettente per lo sviluppo delle reti industriali. Il protocollo RPL, oltre a gestire in maniera opportuna la topologia della rete, ha un altro vantaggio: aumentare la copertura della rete stessa, ovvero aumenta la distanza massima a cui due dispositivi possono comunicare. Anche se la comunicazione tra due nodi direttamente connessi può avvenire solo se i dispositivi in questione sono molto vicini tra loro, la massima distanza è di gran lunga maggiore se si considerano connessioni di tipo indiretto, in cui tra il nodo mittente e il nodo destinatario ci sono diversi nodi intermedi che fanno da tramite per il trasferimento dei frame.

2.3.2 IEEE 802.11

Lo standard IEEE 802.11 viene preferito rispetto allo standard basato sul metodo TSCH quando sono richieste connessioni a bassa latenza e con una portata di informazioni maggiore. Il Wi-Fi ha raggiunto notevoli traguardi durante il proprio sviluppo, soprattutto per quanto riguarda la portata che lo standard garantisce, passando da un data rate di circa 54 Mbps (Megabyte al secondo) agli inizi del 2000, fino ad arrivare ad un massimo di 7 Gbps in una delle ultime release dello standard nel 2013, IEEE 802.11ac, anche se a livello applicativo non viene sfruttato del tutto, soprattutto in contesto industriale, dove le informazioni vengono gestite poco alla volta, con operazioni spesso cicliche [22]. A differenza del protocollo IEEE 802.15.4, in cui ogni dispositivo della rete può essere usato come ponte per la comunicazione di due nodi non direttamente collegati, nelle reti wireless basate sul Wi-Fi viene usata una topologia molto diversa, chiamata *topologia a stella*, in cui un access point (AP) coordina il traffico della rete direttamente con l'insieme dei dispositivi connessi ad esso. In questo contesto, la copertura massima non dipende più dai singoli nodi collegati nella rete, ma dal numero di APs presenti e da come questi sono disposti nell'ambiente (per esempio collegare due access point nello stesso punto non aumenta la copertura del segnale Wi-Fi) [23]. Anche in questo caso il controllo sull'effettiva ricezione di un frame e la gestione della ritrasmissione del frame mancante è affidata al meccanismo ARQ descritto nella sezione precedente, aumentando l'affidabilità del traffico dati, ma peggiorando notevolmente le prestazioni se si parla di determinismo e di latenza. Dunque, in sistemi hard real-time, lo standard IEEE 802.11 non è ancora considerato una buona alternativa alle reti cablate, anche se molte ricerche cercano di ridurre la possibilità che un frame venga perso durante una comunicazione, in alcuni casi cercando di migliorare gli algoritmi che selezionano un tasso di trasmissione migliore per un specifico canale [24], in altri cercando di replicare i frame inviati dal trasmettitore in canali diversi, in modo da aumentare ancora di più l'affidabilità e ridurre la latenza della trasmissione [25, 26, 27, 28]. Per ridurre ulteriormente la latenza e migliorare il determinismo delle comunicazioni, è stato introdotto il concetto di Qualità del Servizio (Quality of Service, QoS) nel 2005. Con il termine QoS si vuole esprimere il concetto di caratterizzazione del servizio offerto da un protocollo, ovvero descrivere diversi livelli di qualità che uno standard deve rispettare. Per quanto riguarda il protocollo IEEE 802.11e, si è pensato di suddividere la tipologia di traffico in quattro classi di servizio, ognuna

con una priorità diversa dalle altre, cercando così di avvantaggiare quella tipologia di traffico che deve soddisfare i requisiti di soft real-time, fornendogli una priorità maggiore [29]. Ulteriori miglioramenti possono essere ottenuti se si utilizzano dei meccanismi per programmare gli accessi di un canale, come ad esempio la tecnica di accesso multiplo a divisione di tempo (Time division multiple access, TDMA) [30]. Il problema in questo caso deriva dal fatto che per usare questa tecnica, i nodi coinvolti nella comunicazione devono essere perfettamente sincronizzati [31, 32]. Un metodo simile al precedente prende il nome di accesso controllato e ibrido al canale (hybrid controlled channel access, HCCA), in cui l'access point programma l'ordine delle trasmissioni nella rete, cercando di ottenere un determinismo molto forte. Questo determinismo però è solo apparente, poiché due APs appartenenti a due organizzazioni diverse possono programmare il traffico dei propri dispositivi allo stesso instante temporale, creando delle collisioni tra i diversi frame inviati nelle diverse reti. In più, altre tipologie di infrastrutture wireless (quali Bluetooth, WSN, ecc.) possono disturbare questa comunicazione, di fatto bloccando l'accesso al canale scelto dal AP [33]. Nel 2009, è stato introdotto un aggiornamento dello standard, noto come IEEE 802.11n, anche se le migliorie introdotte dal nuovo protocollo non impattano direttamente il contesto industriale. Con il metodo di multipli-input e multipli-output (MIMO) e le migliorie apportate al livello fisico si è ottenuto come risultato un aumento generale del throughput a livello applicativo e una leggera diminuzione della latenza di comunicazione. Sempre con IEEE 802.11n, sono stati introdotti due nuovi metodi di aggregazione dei dati, chiamati *A-MSDU* e *A-MPDU*, che permettono di aggregare diversi tipi di frame nello stesso messaggio. Mentre A-MSDU crea dei problemi di affidabilità della comunicazione poiché la gestione di un messaggio aggregato richiede molto più tempo comparato alle trasmissioni normali, A-MPDU gestisce ogni singolo pacchetto aggregato con un meccanismo di checksum appropriato, senza compromettere l'affidabilità della trasmissione. Un metodo molto utile nel contesto industriale, introdotto dallo standard IEEE 802.11ac nel 2013, è il multi-utente MIMO (MU-MIMO). Adesso è possibile servirsi di algoritmi di scheduling, in cui il traffico industriale viene gestito in maniera più accurata, permettendo in molti più casi che i vari messaggi arrivino a destinazione rispettando le scadenze previste. Questa gestione del traffico può però essere condotta solo per le comunicazioni in downlink, cioè per quelle comunicazioni che partono da un AP e arrivano ai nodi della rete locale, in generale non il caso più frequente [34]. Con l'introduzione dell'ultimo standard IEEE 802.11ax,

pubblicizzato nel mercato con il nome di Wi-Fi 6, si ci aspettano dei miglioramenti notevoli nell'applicazione del protocollo nel contesto industriale.

2.3.3 Protocollo 5G

Come accennato in precedenza, un'altra tipologia di connettività wireless sta entrando a stretto contatto con lo sviluppo delle reti in contesto industriale. Sviluppato dal gruppo di sviluppo 3GPP, la tecnologia 5G sta rappresentando una vera e propria rivoluzione del concetto di connettività wireless [35]. A partire dalla prima release nel 1992, questo protocollo è stato sempre teorizzato su carta, ma ha visto la prima vera implementazione solo nel 2019 con la release 16, essendo l'anno in cui le prime attenne 5G sono state installate sul territorio europeo. Al momento, l'ultima versione disponibile dello standard deriva dalla descrizione fornita dalla release 17, che include aggiornamenti sulle politiche di QoS, in particolare l'aggiunta di tecniche di *network slicing* [36], lo scambio di dati periodici e deterministici tramite l'uso di tecniche di scheduling e di prelazione, un maggior supporto per la gestione di traffico misto e una descrizione delle basi per attuare una sincronizzazione per ottenere comunicazioni deterministiche. Con questi update, il 5G è in grado di raggiungere notevoli valori di affidabilità (poiché è previsto un errore o una mancata ricezione di un pacchetto ogni 100.000 trasmissioni) e un data rate relativamente alto (con picchi in downlink che possono raggiungere i 3 Gbps, inferiore al Wi-Fi in generale, ma relative ad una comunicazione che avviene al livello applicativo). Purtroppo, non ci sono molti documenti che descrivano il funzionamento del 5G nell'ambiente industriale con le nuove release, ma molti dei documenti presenti fanno riferimento all'applicazione dello standard facendo riferimento a release precedenti, che risultano obsolete .
Uno studio pubblicato nel 2018 [37], mostra un concetto preliminare che descrive l'integrazione della rete 5G nell'infrastrutture basate su RTE, presentando 3 scenari possibili, in cui viene discussa anche la gestione delle politiche di QoS da implementare per ognuna degli scenari descritti.

Gli stessi autori nel 2019 approfondiscono il tema, mostrando un esempio concreto di integrazione del 5G in un sistema basato su PROFINET (Protocollo di tipo Industrial Ethernet), modellando la rete 5G come un dispositivo di commutazione di rete, discutendo anche su come realizzare sistemi ibridi basati sulla comunicazione in real-time dei dispositivi connessi nella rete [38].

Riassumendo, ci sono molte aspettative su come il 5G possa fornire un supporto utile ad evolvere ancora di più le comunicazioni in ambito industriale. Come già descritto, molte delle ricerche condotte si focalizza sull'integrazione della rete in contesti che consentirebbero uso di requisiti e di applicazioni industriali sempre diversi ed innovativi, dipendenti non solo dalle politiche di QoS, ma anche dalla gestione di reti sempre più complesse.

Capitolo 3

Rete Neurale

3.1 Introduzione

Come suggerisce il nome, una Rete Neurale Artificiale (ANN, o più semplice rete neurale, NN) è una rappresentazione digitale dell'organo più complesso ed elaborato del corpo umano, il cervello. Come il cervello è composto da un insieme di neuroni, che elaborano alcuni input e trasmettono output alle altre unità, una ANN è strutturata in modo da emulare questo comportamento, cercando di valutare dati, immagini e pattern temporali per estrapolare pezzi utili di informazione [39].
L'utilizzo di una rete neurale è composto da due processi principali:

1. Fase di apprendimento.

2. Fase di previsione o predizione.

La più critica è la fase di apprendimento, in cui la rete neurale impara a prendere decisioni basate su delle informazioni ricevute in ingresso. In questo processo il sistema riceve un vasto database, suddiviso in input ed target. Gli input vengono analizzati per comprendere i modelli generali e generare un output appropriato, il target è l'output desiderato; quindi l'algoritmo di apprendimento della rete neurale capisce se l'output è sufficientemente vicino o meno al valore del target e corregge alcuni parametri dei neuroni per la successiva previsione degli input simili. Questo processo viene effettuato molte volte, in modo da ottenere una precisione maggiore.

3.2 Architettura ad alto livello

Le reti neurali artificiali sono comunemente composte da 3 o più livelli, ciascuno composto da un insieme di *neuroni artificiali*. I neuroni sono interconnessi tra loro attraverso collegamenti. Il database viene usato a seguito di alcune rielaborazioni per alimentare le unità di input, quindi elaborato e inviato allo strato nascosto (possibilmente più di una volta) e infine fornito alle unità di output per produrre gli output finali. Ogni arco della rete ha un peso corrispondente, un numero naturale che rappresenta il grado di influenza di un'unità su un'altra. Ci sono diverse architetture, che saranno discusse nella Sezione 3.5.

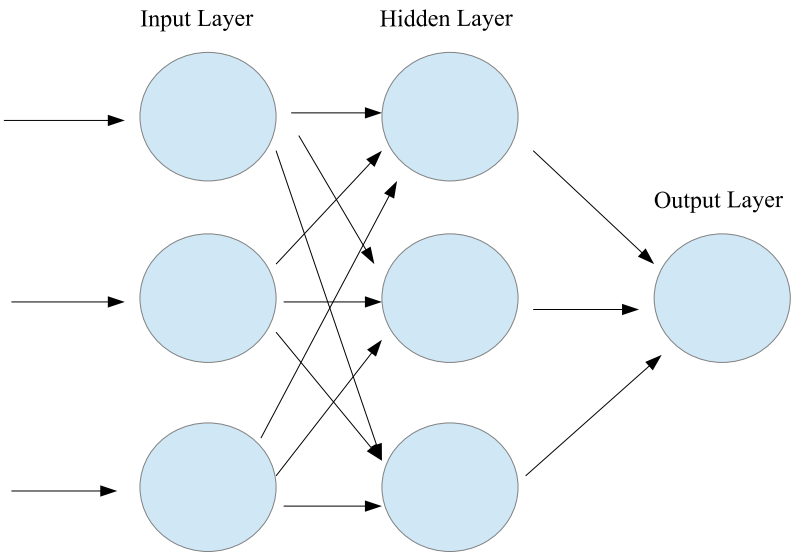


Figura 3.1: Topologia di una rete neurale artificiale

3.3 Neurone artificiale

Un *neurone artificiale* è il componente principale di una rete neurale, che è organizzata in una serie di *strati* (vedi Fig. 3.1). Considerando il neurone k, collegato a

m neuroni dello strato precedente, l'output $\hat{y}_k$ di quel neurone può essere calcolato come segue:

$$\hat{y}_k = f(net_k) \tag{3.1}$$

dove $f(net_k)$ è chiamato *funzione di attivazione* (maggiori dettagli nella prossima sezione) e net_k, chiamato *net input*, è uguale a:

$$net_k = \sum_{i=1}^{m} w_{ik} x_i + w_{0k} \tag{3.2}$$

dove x_i è l'uscita del nodo del livello precedente, w_{ik} è il peso associato al collegamento tra il nodo al livello precedente e il nodo stesso, e w_{0k} rappresenta un termine bias, utile quando la previsione necessita di qualche aggiustamento lineare.

3.4 Funzioni di attivazione

La funzione $f(\cdot)$ introdotta in (3.1) è chiamata *funzione di attivazione*. Prende net_k come input (tutti i contributi dei neuroni del livello precedente) e applica una trasformazione per generare l'output di quel neurone. Le funzioni di attivazione possono essere essenzialmente suddivise in due gruppi principali:

- funzioni di attivazione lineari.
- funzioni di attivazione non lineari.

Tra le molteplici funzioni di attivazione disponibili in letteratura, le più utilizzate e comuni sono:

- ReLu. È forse la funzione di attivazione più popolare utilizzata nelle ANN. La sua equazione è:
$$y(x) = max(0, x) \tag{3.3}$$
- Sigmoide. È una funzione non lineare che prende come input qualsiasi valore reale e restituisce valori nell'intervallo $[0, 1]$, estremi inclusi:
$$y(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$
dove e è il numero di Eulero.

- Tanh. È una funzione molto simile alla funzione sigmoide. Trasforma qualsiasi valore numerico ricevuto in input in un valore compreso nell'intervallo $[-1, 1]$, estremi inclusi.

$$y(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.5}$$

- Identità. È una funzione che restituisce il valore ricevuto come input, moltiplicato per una costante. Fa parte di una famiglia più ampia chiamata funzioni di attivazione lineare, le cui funzioni possono essere espresse come:

$$y(x) = cx \tag{3.6}$$

  dove c denota la pendenza della retta. Per la funzione identità, c assume il valore 1

3.5 Topologia

Vediamo più nel dettaglio come è organizzata una Rete Neurale Artificiale. Come discusso nella Sezione 3.3, i neuroni sono raggruppati in insiemi chiamati *strati* oppure più comunemente *layers*, ognuno dei quali è connesso tramite collegamenti pesati ad altri layer. Questi pesi rappresentano l'influenza che un nodo del livello $i - th$ ha su un nodo dello strato successivo. Più questo valore è alto, più quel nodo considererà come 'attendibile' informazione ricevuta dal nodo precedente. Le topologie più utilizzate nelle reti neurali contengono tre o più livelli:

- Un *livello di input.*
- Uno o più *livelli nascosti.*
- Un *livello di output.*

L'architettura riceve gli input attraverso dei database opportunamente pre-processati, ma non può gestire tutte le informazioni contemporaneamente, perché i neuroni a livello input sono di solito inferiori ai record contenuti nel dataset. Di conseguenza, considerando un dataset T di dimensione $N_{Features} * N_{Records}$, dove $N_{Records}$ è il numero di record nel DB e $N_{Features}$ è la larghezza di ogni record (il numero di elementi al suo interno), il layer di input ha un numero di neuroni pari a $N_{Features}$ e

gestisce gli input per ogni record. Quindi può essere rappresentato come un vettore X_t come segue:

$$X_t = (x_0, x_1, ..., x_{N_{features}-1})_t \tag{3.7}$$

Normalmente, sono necessari meccanismi di transformazione/normalizzazione per gestire in modo appropriato quegli input e impostare i neuroni del livello di input. Tali informazioni vengono quindi inviate al livello successivo, utilizzando (3.1) e (3.2); quindi, considerando un neurone k dello strato nascosto:

1. il neurone k riceve tutti i $\hat{y_k}$ contributi di tutti i neuroni del livello input.
2. Una funzione di attivazione viene applicata a k.
3. Il risultato viene inviato ai neuroni connessi al livello successivo, che può essere un altro livello nascosto o quello di output.
4. Itera fino a quando tutti i neuroni dello strato nascosto sono calcolati [1].

La larghezza di ogni strato nascosto può essere arbitraria. ma se si utilizza una larghezza ridotta, alcuni pattern interessanti possono andare perduti, ottenendo un risultato non ottimale. Al contrario, se la larghezza è troppo elevata, si può incorrere in un fenomeno denominato *Overfitting*, che sarà descritto nella Sezione 3.6.7. Il livello nascosto è solitamente rappresentato come segue:

$$H_t = (h_0, h_1, ..., h_{m-1})_t \tag{3.8}$$

dove m è la larghezza del layer considerato. Infine, i dati vengono ricevuti nel livello di output, che calcola la previsione finale. Può essere un singolo valore o un vettore di valori a seconda del numero di etichette rappresentate:

$$\hat{Y_t} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_{N_{uscite}})_T \tag{3.9}$$

dove $N_{outputs}$ indica il numero di valori di output generati e t identifica il valore di output generato seguendo il vettore di input identificato dall'indice t.

Quanto segue spiega la terminologia usata per descrivere la forma e la capacità di una rete neurale:

[1] A seconda della topologia usata, questo meccanismo può variare, per esempio, può esistere più di uno strato nascosto o i neuroni sono connessi in modo diverso; maggiori dettagli nella sezione successiva.

- dimensione: numero di neuroni nella rete.
- larghezza: numero di neuroni in uno specifico livello.
- profondità: il numero di livelli in una rete neurale[2].
- capacità: il tipo o la struttura delle funzioni che una configurazione di rete può apprendere.
- architettura: l'organizzazione specifica degli strati e dei neuroni nella rete.

Dalla letteratura esistono molti tipi di topologie; per questo studio ne verranno prese in considerazione due: reti neurali feed-forward e reti neurali ricorrenti, con maggiore enfasi sulle reti LSTM.

3.5.1 Reti neurali feed-forward

La rete neurale feed-forward (FFNN) è il tipo più semplice di rete neurale artificiale progettata. Ogni strato è composto da un insieme di neuroni, collegati solo ai neuroni dello strato successivo. Se ogni neurone è connesso a tutti i neuroni dello strato successivo, si dice che l'ANN è *completamente connesso*. Per costruzione, questa topologia non contiene loop, quindi i dati passano dal livello di input al livello di output, attraverso i vari strati nascosti solo in una direzione, senza mai ripercorrere archi già attraversati.

Una rete neurale feed-forward può essere implementata nella sua forma più semplice come un singolo strato con un neurone in grado di apprendere solo modelli linearmente separabili. L'idea di base è di approssimare la funzione continua con una serie di rettangoli affiancati, dove un neurone rappresenta concettualmente un rettangolo. Pertanto, l'aggiunta di più neuroni si traduce in un'approssimazione più accurata della funzione, ma ciò ha lo svantaggio di aggiungere una maggiore complessità computazionale durante la fase di apprendimento.

3.5.2 Reti neurali ricorrenti

Le reti neurali ricorrenti (RNN), a differenza delle reti feed-forward, possono inoltrare le informazioni anche nella direzione opposta, creando alcuni loop nella topologia.

[2]Di solito nel conteggio non consideriamo il livello di input.

In questo modo alcune informazioni che la rete considera importanti, possono essere fornite nuovamente in input a quest'ultima. In questo modo, si ha per la prima volta il concetto di *memoria*, che può essere efficace quando si studia un processo basato su serie temporali, ottenendo in generale prestazioni migliori rispetto ad una architettura di tipo feed-forward. Tuttavia, può accadere che la previsione data da questa architettura non sia ottimale, o in generale, la rete non converga ad un valore ottimo (maggiori dettagli nella Sezione (3.6.6)). In particolare, unità speciali chiamate "unità di memoria" memorizzano informazioni importanti dal recente che influenzano l'ingresso e l'uscita correnti.

3.5.3 Reti neurali Long Short-Term Memory

È stato detto all'inizio di questo capitolo che una rete neurale parte dal presupposto di emulare il comportamento del cervello umano. Dunque il cervello riesce ad associare immagini, eventi o a risolvere problemi attraverso l'esperienza che esso acquisisce e dalle informazioni che riesce a collezionare nel tempo. Ci sono tipologie di problemi però, in cui conoscere le informazioni in se non è sufficiente, ma bisogna anche conoscere l'ordine in cui queste informazioni vengono elaborate. In altri termini, abbiamo bisogno di *memorizzare* la sequenzialità di certe informazioni. Una rete neurale Long Short-Term Memory (LSTM) è una particolare tipologia di rete neurale ricorrente che emula questo comportamento. La struttura generale di questa rete è identica ad una RNN, quindi un'architettura che presenta dei loop nella topologia della stessa, e che dunque può usare alcune informazioni da dati precedentemente analizzati. La novità di questo modello sta nella struttura dei singoli neuroni che la compongono. Infatti il neurone, oltre ai componenti descritti nella Sezione 3.3, possiede dei componenti chiamati *celle di stato*. Questi componenti emulano il concetto di memoria all'interno della rete. Il suo funzionamento è il seguente: memorizzare temporaneamente l'informazione relativa ad un dato precedentemente acquisito all'interno del neurone finché esso risulta utile alla regolazione dei pesi nella rete. All'interno di un singolo neurone possiamo avere più celle di stato, ognuna delle quali fa riferimento a un dato precedentemente acquisito. Le reti LSTM sono molto utilizzate in problemi di previsione del testo, poiché riescono ad associare una singola parola con alcune parole o frasi che la precedono.

3.6 Fase di Apprendimento

L'apprendimento o più specificamente la fase di Learning è il processo che ha come obiettivo quello di settare i parametri interni di una rete neurale (ad esempio, pesi e bias). Facendo ciò, si *addestra* la rete neurale, così che sia in grado di ottenere ottimi risultati nella fase di previsione.

L'idea di questa fase è la seguente: confrontare l'output desiderato con quello ottenuto nella rete, cercando di correggere l'eventuale errore e trovare un risultato più vicino possibile a quello desiderato. Quindi, cercare di trovare una relazione valida tra gli input e i target considerati.

Un esempio pratico di questa fase può essere il seguente: un bambino che vede per la prima volta un fuoco di un fornello è incuriosito da questo strano oggetto; inizialmente non capisce che può essere pericoloso, ma quando tocca il fornello, capisce che una fiamma gli farà male. Di conseguenza, non toccherà più la fiamma. Allo stesso modo, questa fase insegna alla rete determinanti informazioni circa la natura del problema da esaminare, e associando sempre più informazioni all'evento dello studio, riesce ad applicare una relazioni tra i dati ricevuti in ingresso, e a trarre le dovute conseguenze.

Questi dati in ingresso e i target desiderati sono record di un dataset, definito come segue:

$$D = \{ (X_t, Y_{true_t}) \mid t \in (1, ..., s) \} \tag{3.10}$$

dove D rappresenta il dataset, s è il numero di tuple all'interno del dataset, X_t e Y_t sono le tuple definite in (3.7) e (3.9), rispettivamente. Il dataset D viene ulteriolmente diviso in due subset:

- *TRA*: il *training set* che viene usato durante la fase di apprendimento
- *TES*: il *test set* che viene usato nella fase di predizione per valutare la bontà della rete neurale appena addestrata.

Per ottenere dei buoni risultati durante il training, si utilizza di solito l'80% del dataset D come subset TRA ed il restante 20% viene allocato per il subset TES, anche se le proporzioni possono variare dipendentemente dalla dimensione dello stesso.
La fase di apprendimento si svilluppa in questo modo: il training set viene fornito

alla rete neurale, che analizza i dati in diverse iterazioni chiamate *epoche*, dove il termine epoca identifica il numero di volte in cui il training set è fornito alla ANN per effettuare l'addestramento della stessa. Ad ogni epoca, la rete ha come obiettivo quello di modificare i pesi e i bias presenti per minimizzare una *funzione di costo*. Generalmente questa funzione di costo è una funzione che calcola la differenza tra il risultato ottenuto nella rete e i target del test set. Più questa differenza è minimina, più la rete produce risultati buoni. Dopo qualche epoca, il risultato della nostra architettura dovrebbe rimanere più o meno invariato, raggiungendo uno stato di *convergenza*.
In base al problema che si affronta e al tipo di funzione di attivazione utilizzata, esistono diverse funzioni di costo che possono ottenere risultati migliori rispetto ad altre. Le più comuni sono le sequenti:

- Errore Quadratico Medio (MSE). Questa funzione compie la media della differenza elevata al quadrato tra la predizione e la label.

$$MSE = \frac{1}{f}\sum_{i=1}^{f}(y_{pred}^{(i)} - y_{true}^{(i)})^2 \tag{3.11}$$

- Errore Assoluto Medio (MAE). Compie la media del valore assoluto tra la predizione e la label.

$$MAE = \frac{1}{f}\sum_{i=1}^{f}|y_{pred}^{(i)} - y_{true}^{(i)}| \tag{3.12}$$

- cross-entropy (CSE). Compie il prodotto tra il label e il logaritmo della predizione per ogni label presente:

$$CSE = -\sum_{i=1}^{n} y_{true}^{(i)} log(y_{pred}^{(i)}) \tag{3.13}$$

Per la natura di queste funzioni di costo, quest'ultime non possono essere usate per tutte le tipologie di problemi, ma dipendono molto dal dominio in cui stiamo lavorando. Inoltre, esse hanno una stretta correlazione con la funzione di attivazione scelta.

Ad esempio, per problemi di classificazione binaria, ovvero quando il risultato di un'analisi può essere solo vero o falso, utilizzare come funzione di costo MAE o MSE non ci dà abbastanza informazioni utili a migliorare la nostra rete. Infatti la

configurazione più diffusa è usare la cross-entropy come funzione di costo e solitamente la sigmoide o la funzione tangente come funzione di attivazione. Al contrario, per problemi di regressione come quello affrontato in questa tesi, ovvero quelli in cui il target in questione è un valore numerico definito in un certo dominio, ha più senso utilizzare l'errore assoluto medio o l'errore quadratico medio come funzioni di costo.[40].
Le funzioni di costo sopra definite si riferiscono all'errore associato a una singola previsione, mentre la funzione di costo reale utilizzata per misurare la perdita complessiva su un set di dati è definita come:

$$J(T, \theta) = \frac{1}{t} \sum_{i=1}^{t} loss(y^i_{pred}, y^i_{true}) \tag{3.14}$$

dove $J(T, \theta)$ indica l'errore totale compiuto sul dataset T, considerando un set di pesi θ, $loss$ indica una funzione di costo considerate in precedenza, mentre t indica la dimensione del dataset T. Un valore alto di $J(T, \theta)$ indica una imprecisione marcata sulla predizione del modello, quindi l'obiettivo è quello di minimizzare questo valore, cercando di capire come i pesi (ed eventuali bias) devono essere modificati.
La fase di apprendimento può essere suddivisa in tre parti:

1. la fase di forward, in cui il modello genera le predizioni per ogni iterazione del dataset. Ad ogni iterazione, viene esaminato solo una piccola porzione di dati; questa dimensione viene definito da una variabile esterna al modello detta *batch size*.

2. la fase di valutazione, in cui tutti questi dati contribuiscono al calcolo della funzione di costo.

3. la fase di backward, dove sono modificati i pesi e i bias presenti nella rete. In particolare, questa fase viene implementata usando due algoritmi, chiamati *backpropagation* e *gradient descent*. Il primo si occupa di stimare il contributo o la penalità che ogni peso da alla funzione di costo; il secondo, invece, si occupa di come modificare questi pesi, per cercare di minimizzare la funzione di costo nello step successivo.

Queste tre fasi si ripetono per ogni epoca definita in precedenza. Dopo un numero indefinito di epoche, si arriverà ad un punto in cui la funzione di costo *converge* ad

un valore limite. In questo caso, sarebbe ideale fermare il training, poichè abbiamo raggiunto il minimo valore possibile per la funzione di costo. Ma è difficile stabilire a priori il numero esatto di epoche necessario per raggiungere questa convergenza, quindi si cerca di stabilirlo sperimentalmente.

3.6.1 Fase di Forward

È la prima fase dell'algoritmo di apprendimento. Come spiegato in precedenza, l'obiettivo di questa fase è fornire una prima previsione del problema $\hat{Y}_t$, partendo da un pattern di input X_t, di dimensione uguale a $batchsize$.
Nel caso di una rete neurale feed-forward completamente connessa, X_t (3.7) viene inviato al livello di input, che propaga i valori al primo livello nascosto. Quindi ogni unità nascosta calcola il proprio output usando (3.1) e inoltra i propri valori alle unità del livello successivo. Quindi il processo di calcolo e propagazione viene ripetuto fino ai neuroni dello strato di output, ottenendo una prima previsione $\hat{Y}_t$ (3.9). Per brevità, i passaggi di cui sopra sono riassunti nella funzione $forward\_phase(\cdot)$:

$$\hat{Y} = forward\_phase(X_t, \theta) \tag{3.15}$$

dove θ denota un insieme specifico di pesi e bias.

3.6.2 Fase di Backward

Dopo questo passaggio, questa previsione viene gestita nella fase di Backward, che è la fase più critica del processo. L'obiettivo è quello di confrontare la previsione appena trovata con l'output desiderato (con il target fornito per quel particolare batch in ingresso), e minimizzare la differenza tra le due informazioni. Come metro di paragone, viene utilizzata una funzione di costo, definita all'inizio della Sezione 3.6, scelta prima dell'inizio della fase di apprendimento. Come abbiamo anticipato nella sezione precedente, una funzione di costo o funzione di perdita, ci aiuta a determinare la differenza tra due valori, dando un peso a questa differenza; in questo caso specifico ci aiuta a capire quanto due previsioni siano vicine tra di loro. Per comprendere meglio questa vicinanza, ci viene in aiuto uno strumento matematico molto comodo, il *gradiente* di una funzione multivariabile. Il gradiente di una funzione a più variabili ci dà la pendenza di questa funzione, fissando una direzione. In questo caso ci dice come la funzione di costo utilizzata crescerebbe fissando la previsione target. Poichè il nostro obiettivo è quello di minimizzare la funzione di

costo, la strategia è la seguente: dopo aver calcolato il gradiente, questo algoritmo tenterà di modificare i pesi e gli eventuali bias nella direzione opposta al gradiente, cercando quindi nella prossima iterazione, di minimizzare la distanza tra l'output desiderato e la previsione fornita dal modello.
La gestione del calcolo del gradiente della funzione di costo viene affidata da un algoritmo detto *backpropagation*, mentre l'aggiornamento dei pesi rispetto al gradiente della stessa, viene gestito da un algoritmo detto *discesa del gradiente* (maggiori informazioni nella Sezione 3.6.3).

3.6.3 Backpropagation

Bisogna fare una precisazione importante. il calcolo del gradiente rispetto a una funzione di costo data, non dipende soltanto dal target scelto e dalla previsione effettuata, ma anche da tutti i pesi usati nei livelli precendenti. Questa caratteristica è fondamentale nell'addestramento della rete, poichè è possibile calcolare tutti i gradienti, fissando un peso specifico, e aggiornare di conseguenza tutti i pesi che potrebbero contribuire al miglioramento della previsione. Quindi quello che fa l'algoritmo di backpropagation è di calcolare i gradienti della funzione di costo a partire dai neuroni del livello output e andare a ritroso nei livelli precedenti calcolando tali gradienti per ogni singolo peso. Queste informazioni verrano poi usate dall'algoritmo di discesa del gradiente per aggiornare i vari pesi

Discesa del gradiente

È un algoritmo di ottimizzazione utilizzato dalle reti neurali per minimizzare una funzione di costo data; in pratica consente di trovare un minimo locale o globale. In questo modo è possibile per la rete trovare una convergenza, anche se non ottima, cioè corrispondente ad un minimo locale. L'algoritmo aggiorna i pesi e i bias alla fine di ogni iterazione secondo questa formula:

$$\theta_{i+1} = \theta_i - \alpha \frac{\partial J(\theta_i)}{\partial \theta_i} \tag{3.16}$$

dove θ_i denota collettivamente uno specifico insieme di pesi al tempo i, e α è un coefficiente chiamato *tasso di apprendimento*. Per $i = 0$, θ viene inizializzato con valori casuali. Poiché il nostro obiettivo è quello di trovare la direzione della funzione di costo in cui la funzione scende più rapidamente, e considerando che il gradiente ci dà l'informazione opposta, quest'ultima viene sottratto al valore precedente del

peso o del bias. Il tasso di apprendimento è un fattore che determina il contributo del gradiente ad ogni iterazione, che calcola quanto velocemente o lentamente il modello evolverà verso i pesi ottimali (maggiori informazioni nella Sezione 3.6.4). Applichiamo ripetutamente (3.16) fino a quando non vengono soddisfatti i criteri di terminazione, ad esempio, il valore della funzione di costo entra in un stato di convergenza, questo valore supera un valore minimo che ci siamo prestabiliti oppure abbiamo superato il numero di epoche.
Se il tasso di apprendimento non fosse presente, avremmo delle situazioni in cui si troverebbero una funzione di costo ottima per la fase di apprendimento, ma con un'alta probabilità di ottenere una configurazione pessima nella fase di predizione (minimo locale). In questo modo, la funzione di costo viene migliorata gradualmente, con maggiori possibilità di trovare una buona configurazione dei pesi per la fase di predizione. Le convenzioni utilizzate sono:

1. apice k denota il k-esimo livello di una rete. Il livello di input è considerato il livello 0-esimo.

2. I pedici costituiti da un singolo valore denotano l'indice di un neurone all'interno del suo strato

3. I pedici associati ai pesi sono costituiti da due numeri che denotano gli identificatori dei neuroni coinvolti nella connessione. Per semplicità, si farà riferimento ad una architettura di tipo feed-forward, dove i collegamenti tra un livello di neuroni e il successivo avvengono solo in una direzione (assenza di loop).

Come anticipato, l'obiettivo è aggiornare ogni peso in modo che ogni previsione sia più vicina, dopo ogni aggiornamento, ai propri valori target. La grandezza di un aggiornamento del peso è dettata dal gradiente della funzione di costo rispetto a quel peso $\frac{\partial E(T,\theta)}{\partial w_{ij}^k}$, che concettualmente ciò che fa è dirci quale effetto ha un piccolo cambiamento di peso sulla funzione di costo complessiva.
Dal calcolo, sappiamo che la derivata della somma di più funzioni è equivalente alla somma delle loro derivate. Poiché la funzione di costo, secondo la definizione (3.14), è la media dei singoli errori commessi su un dataset T, il suo gradiente rispetto al peso w_{ij}^k diventa:

$$\frac{\partial E(T,\theta)}{\partial w_{ij}^k} = \frac{1}{S}\sum_{u=1}^{S}\frac{\partial E_u(y_{pred}^u, y_{true}^u)}{\partial w_{ij}^k} \tag{3.17}$$

Pertanto, invece di risolvere interamente (3.14), possiamo concentrarci sui passaggi matematici coinvolti nel calcolo della derivata dell'errore associato a un singolo input. Quindi, per ottenere il gradiente complessivo, dobbiamo semplicemente fare la media dei singoli gradienti più piccoli.

Applicando la regola della catena su $\frac{\partial E_u}{\partial w_{ij}^k}$ otteniamo:

$$\frac{\partial E_u}{\partial w_{ij}^k} = \frac{\partial E_u}{\partial out_j^k} \frac{\partial out_j^k}{\partial net_j^k} \frac{\partial net_j^k}{\partial w_{ij}^k} \tag{3.18}$$

I termini $\frac{\partial E_u}{\partial out_j^k}$ e $\frac{\partial out_j^k}{\partial net_j^k}$ vengono spesso semplificati come:

$$\delta_j^k = \frac{\partial E_u}{\partial out_j^k} \frac{\partial out_i^k}{\partial net_j^k} = \frac{\partial E_u}{\partial net_j^k} \tag{3.19}$$

mentre $\frac{\partial net_j^k}{\partial w_{ij}^k}$ può essere riscritto come:

$$\frac{\partial net_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k} (\sum_{l=0}^{r_{k-1}} w_{lj}^k out_l^{k-1}) = out_i^{k-1} \tag{3.20}$$

dove r_{k-1} indica il numero di neuroni presenti nel layer $k-1$ e δ_j è detto *termine di errore*.
Dunque, (3.18) può essere riassunto come:

$$\frac{\partial E_u}{\partial w_{ij}^k} = \delta_j^k out_i^{k-1} \tag{3.21}$$

L'errore δ_j^k ha una diversa definizione, che cambia se il gradiente dipenda o meno da un peso collegameto ad un neurone del livello output. Nel primo caso, δ_j^L vale:

$$\delta_j^L = \frac{\partial E_u}{\partial net_j^L} = loss'(act_j^L(net_j^L), y_{true}^i) \cdot act'^L_j(net_j^L) \tag{3.22}$$

dove L indica il livello output e act_j è la funzione di attivazione del neurone $j-th$.
Mettendo tutto insieme, la derivata parziale di E rispetto al peso nel layer di output w_{ij}^L e:

$$\frac{\partial E_u}{\partial w_{ij}^L} = loss'(act_j^L(net_j^L), y_{true}^u) \cdot act'^L_j(net_j^L) * out_i^{k-1} \tag{3.23}$$

Considerando i neuroni del livello nascosto, $1 \leq k < L$, δ_j^k diventa:

$$\delta_j^k = \frac{\partial E}{\partial net_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial net_l^{k+1}} \frac{\partial net_l^{k+1}}{\partial net_j^k} \tag{3.24}$$

dove r^{k+1} indica il numero di nodi del prossimo livello. Si fa notare che l inizia da 1 perchè il bias out_0^k corrispondente a w_{0j}^{k+1} è costante, poiché non dipende dagli output precedenti. Il primo termine dell'equazione è, per definizione, l'errore del prossimo livello, δ_l^{k+1}

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} \frac{\partial net_l^{k+1}}{\partial net_j^k} \tag{3.25}$$

Una formula alternativa a (3.2) è:

$$net_j^k = \sum_{i=0}^{r_{k-1}} w_{ij}^k g(net_i^{k-1}) \tag{3.26}$$

e sostituendo questa espressione a (3.25), si ottiene:

$$\frac{\partial net_l^{k+1}}{\partial net_j^k} = w_{jl}^{k+1} g'(net_j^k) \tag{3.27}$$

L'espressione finale di δ_j^k è:

$$\delta_j^k = g'(net_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} \tag{3.28}$$

Mettendo tutto insieme la derivata parziale di E rispetti al peso w_{ij}^k nei livelli nascosti $1 \leq k < L$ è:

$$\frac{\partial E}{\partial w_{ji}^k} = \delta_j^k out_j^{k-1} = out_j^{k-1} g'(net_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} \tag{3.29}$$

Il termine di errore associato ad un peso nel livello k dipende dal termine di errore dei pesi nel livello k+1 (vedi (3.25)). La procedura standard è quindi quella di calcolare i termini di errore a partire dallo strato di output e propagarli all'indietro, aggiornandoli quando necessario, strato dopo strato fino a raggiungere il primo strato. Questo processo è ciò che rende la backpropagation efficiente nel calcolo dei gradienti e adatta per essere utilizzata insieme ad algoritmi di ottimizzazione basati sul gradiente come la discesa del gradiente.
Per calcolare il gradiente della funzione di costo rispetto a w_{ij}^k facciamo semplicemente la media delle singole derivate calcolate sull'intero set di dati

$$\frac{\partial E(T, \theta)}{\partial w_{ij}^k} = \frac{1}{S} \sum_{u=1}^{S} \frac{\partial E_u}{\partial w_{ij}^k} \tag{3.30}$$

dove S è la dimensione del set di dati.
Per ricapitolare, una volta che i gradienti sono stati calcolati tramite backpropagation, i pesi vengono aggiornati tramite la discesa del gradiente utilizzando questa formula:

$$w_{ij}^k = w_{ij}^k - \alpha \frac{\partial E(T, \theta)}{\partial w_{ij}^k} \tag{3.31}$$

dove α è anche in questo caso il tasso di apprendimento.
Ciò che abbiamo descritto finora è la discesa del gradiente standard, nota anche come discesa del *gradiente vanilla*, in cui il gradiente di ciascun aggiornamento viene calcolato su tutti gli esempi contenuti in un set di dati di apprendimento. Questo processo può richiedere molto tempo nelle applicazioni su larga scala [41, 42, 43], dove i set di dati di apprendimento possono contenere milioni di esempi. Una strategia molto comune per contrastare questo problema è prendere in considerazione l'aggiornamento dei pesi dopo ogni x elementi, dove x è noto anche come *batch size*. È un iperparametro del modello che definisce il numero di modelli da considerare per ogni aggiornamento. Il suo valore è in genere una potenza di 2 e può variare da $0 < batchsize < size(T)$, dove $size(T)$ indica la dimensione di un set di dati T. In letteratura si trovano comunemente due varianti dell'algoritmo di discesa del gradiente primario, che differiscono solo per il valore della dimensione del batch:

- discesa del gradiente stocastico (SGD). Aggiorna i parametri del modello per ogni modello di input visualizzato durante l'apprendimento (dimensione batch = 1). Tende ad essere molto più veloce dal punto di vista del calcolo, ma può causare forti fluttuazioni dei valori della funzione di costo nel corso di un'istanza di apprendimento perché il gradiente valutato su un singolo input è un'approssimazione rumorosa di quello completo.

- la discesa del gradiente mini-batch è a metà strada tra la discesa del gradiente vanilla e SGD. Esegue un aggiornamento dopo ogni mini-batch di n campioni di apprendimento visualizzati, dove n è compreso tra 1 e la dimensione del set di dati di apprendimento (escluso). Questa variante offre una migliore efficienza rispetto alla versione vanilla in termini di memoria e velocità e riduce la quantità di rumore riscontrata in SGD.

Le principali complessità relative alla configurazione associate all'algoritmo di discesa del gradiente sono:

- scegliere un tasso di apprendimento adeguato è un processo che può risultare complesso. Un tasso di apprendimento troppo basso porta a una convergenza lenta, mentre un tasso di apprendimento troppo alto fa sì che la rete neurale possa non raggiungere mai la convergenza

- lo stesso tasso di apprendimento viene applicato a tutti i pesi e bias. A volte vorremmo applicare tassi di apprendimento diversi se il nostro set di dati contiene dati sparsi e funzionalità con frequenze molto diverse

- trovare un minimo locale o globale che generalizzi abbastanza bene il nostro problema. Una funzione obiettivo associata a insiemi di dati della vita reale ha, nella maggior parte dei casi, superfici molto complesse con molti punti di minimo diversi. Trovare un minimo globale a volte può essere irrealizzabile.

Ricapitolando, la discesa del gradiente è un algoritmo di ottimizzazione per trovare il minimo di una funzione multivariabile. Nel nostro caso, la funzione che vogliamo ottimizzare, cioè la funzione obiettivo, dipende dai parametri del modello. Applicando ripetutamente (3.31) ai pesi e ai bias, permette in molti casi di ottenere un insieme di valori ottimali per i parametri in modo che l'errore complessivo associato al nostro modello sia il più piccolo possibile.

3.6.4 Tasso di apprendimento

Il Tasso di apprendimento, chiamato anche dimensione del passo, è uno scalare positivo, spesso compreso tra 0 e 1. Il suo scopo è quello di ridimensionare l'entità di un aggiornamento del peso, che si trova all'interno dell'algoritmo di discesa del gradiente. Intuitivamente, determina quanto è grande la modifica dei parametri ANN nella direzione di un minimo locale. È un iperparametro vitale che influisce sulla capacità di una rete neurale di raggiungere la convergenza e, in tal caso, a quale velocità. Questo parametro può essere regolato manualmente dal professionista o regolato automaticamente da un ottimizzatore durante la fase di apprendimento. In generale, dovremmo impostare il tasso di apprendimento ad un valore che non sia né troppo alto né troppo basso per diversi motivi:

- con un tasso di apprendimento basso può far convergere la rete troppo lentamente e rimanere bloccata in un minimo non ottimale.

- con un alto tasso di apprendimento può causare un overshoot della rete, portando ad un aumento della perdita complessiva.

Sebbene non sia possibile conoscere con certezza a priori il tasso di apprendimento ottimale, sono disponibili alcune tecniche ben note per determinare un tasso di apprendimento ragionevole. Sono costruiti sull'idea ricorrente di modificare dinamicamente il tasso di apprendimento man mano che una ANN progredisce con l'addestramento invece di mantenerlo costante per tutto il processo di formazione. Esempi di queste tecniche sono: "learning rate schedules" e "adaptive learning rate". Il primo riduce il tasso di apprendimento secondo un programma predefinito, ad esempio, metà del tasso di apprendimento dopo ogni epoca. Questa tecnica ha mostrato, in alcuni contesti applicativi, prestazioni migliori rispetto all'utilizzo di un tasso di apprendimento fisso. Tuttavia, richiede la definizione di un programma di discesa del tasso di apprendimento in anticipo, il che può essere problematico poiché è difficile sapere quali siano le impostazioni migliori da adottare. Inoltre, manca di flessibilità e adattabilità poiché molto probabilmente ci viene richiesto di definire un nuovo programma per un diverso tipo di problema. Gli algoritmi del tasso di apprendimento adattivo aggiornano il tasso di apprendimento in fase di esecuzione sulla base di metodi euristici, come l'adattamento del suo valore in base ai valori dei gradienti passati. Il loro principale vantaggio rispetto ad altre tecniche è che richiedono meno lavoro per la messa a punto degli iperparametri.

3.6.5 Momentum

Momentum è un'altra tecnica ampiamente utilizzata per aiutare una rete neurale a trovare un set ottimale di parametri del modello. È un concetto preso dalla fisica, che esprime la capacità di un oggetto in movimento di continuare la sua traiettoria anche quando gli viene applicata una forza esterna contraria. La stessa idea di base è stata tradotta per l'apprendimento automatico. A (3.31) è stato aggiunto un termine chiamato momentum che tiene conto della direzione generale degli aggiornamenti precedenti e minimizza l'effetto dei cambiamenti opposti. Quindi (3.31) diventa:

$$\Delta w_{ij}^{k(t)} = \alpha \frac{\partial E(T,\theta)}{\partial w_{ij}^{k(t)}} + m\Delta w_{ij}^{k(t-1)} \tag{3.32}$$

$$w_{ij}^{k(t)} = w_{ij}^{k(t)} - \Delta w_{ij}^{k(t)} \tag{3.33}$$

dove m, detto momentum factor, denota un coefficiente, t indica un aggiornamento associato al peso, e $\Delta w_{ij}^{k(t-1)}$ è il vettore di aggiornamento al tempo $t-1$. È pratica comune utilizzare valori di momentum vicini a 0,9, ma può assumere qualsiasi valore compreso tra 0 e 1.
Con la direzionalità data dal momentum, la rete neurale ha maggiori possibilità di non rimanere bloccata in un minimo locale, evitando di oscillare attorno ad esso. Esso infatti costringe gli aggiornamenti dei pesi di una rete neurale a seguire la direzione generale degli ultimi aggiornamenti, riducendo l'effetto dei cambiamenti opposti. In termini pratici, questo viene implementato salvandoil valore dei gradienti passati. Come possiamo vedere da (3.33), la quantità di moto aumenta quando il gradiente continua a puntare nella stessa direzione mentre smorza l'effetto dei cambiamenti opposti. Di conseguenza, la rete neurale ottiene una convergenza più rapida e diminuisce le oscillazioni nell'aggiornamento dei pesi.

3.6.6 Vanishing e exploding gradients

Secondo (3.31), ciascuno dei pesi di una rete neurale riceve un aggiornamento proporzionale al gradiente della funzione di costo rispetto al peso corrente. In alcuni casi, il gradiente può essere così piccolo da impedire al peso di cambiare valore. Nel peggiore dei casi, ciò può interrompere completamente l'apprendimento di una rete neurale. Una causa di questo problema sono le funzioni di attivazione. Ad esempio, consideriamo la funzione tangente iperbolica. Il suo gradiente ha valori nell'intervallo (0,1) e, come sappiamo, la backpropagation calcola i gradienti usando la regola della catena. Questo ha l'effetto di moltiplicare n di questi piccoli numeri per calcolare i gradienti dei primi strati in una rete a n strati, il che significa che il gradiente diminuisce esponenzialmente man mano che ci avviciniamo ai neuroni nei primi strati. Il problema presentato sopra è chiamato *gradienti sfumati.*
Un problema simile chiamato *gradiente che esplode* si verifica quando le funzioni di attivazione hanno derivate che possono assumere valori estremamente grandi, risultando in grandi gradienti. Questo, a sua volta, provoca grandi aggiornamenti nei pesi, rendendo instabile la rete neurale.

Diverse tecniche possono essere utilizzate per mitigare i problemi sopra menzionati:

- riducendo il numero di livelli che compongono una rete neurale

- usando una tecnica chiamata gradient clipping per prevenire l'esplosione dei gradienti. L'idea è di scalare i gradienti in modo che la loro norma non superi un dato valore.

- usa funzioni di attivazione che soffrono meno del problema del gradiente di fuga, come ReLu [44].

3.6.7 Underfitting e Overfitting

Introduciamo brevemente due dei problemi più comuni che influenzano negativamente un modello: underfitting e overfitting. L'underfitting si riferisce a un modello che non riesce ad apprendere dai dati di apprendimento e che, di conseguenza, ottiene errori molto elevati sia in fase di apprendimento che in fase di previsione. È facile da rilevare poiché i modelli sottodimensionati hanno metriche di prestazioni scadenti, ad esempio previsioni imprecise. Nel contesto delle reti neurali, la generalizzazione si riferisce alla capacità del modello di produrre buone previsioni, cioè ragionevolmente vicine all'output desiderato, a insiemi di input che non ha mai visto. Ciò significa che l'attuale configurazione della rete neurale non è riuscita a catturare alcun modello rilevante dai dati di apprendimento. D'altra parte, un modello che soffre di overfitting corrisponde quasi perfettamente ai dati di apprendimento mentre lo fa in modo inadeguato su dati che non ha mai visto prima. Il motivo principale per cui un modello è in overfitting è a causa del rumore raccolto dal set di dati di apprendimento. I rumori sono quelle caratteristiche che compaiono nel set di dati di apprendimento che sono, ad esempio, etichettate in modo errato o che compaiono raramente in altri set di dati simili. Ciò fa sì che la rete neurale acquisisca informazioni distorte o errate, generando previsioni meno accurate quando si tratta di dati del mondo reale [45]. Sono disponibili molteplici strategie che possono essere adottate per ridurre l'effetto dell'overfitting:

- arresto anticipato: è una strategia che consiste nell'arrestare l'apprendimento quando raggiunge un punto in cui la metrica delle prestazioni valutata sul set di dati di test smette di migliorare. Quindi, semplicemente congela il modello prima che abbia la possibilità di eseguire un overfit.

- espandere il set di dati di apprendimento: le prestazioni di un modello possono essere notevolmente migliorate utilizzando un set di dati più ampio.

- dropout si basa sull'idea di eliminare temporaneamente alcuni nodi durante l'apprendimento per ridurre il fenomeno chiamato co-adattamento. In un certo senso, stiamo riducendo la capacità della nostra rete neurale, costringendola a creare connessioni "più forti" su quei collegamenti dove normalmente avrebbero poca influenza sulla previsione finale. Inoltre, il dropout facilita l'apprendimento della rete neurale, riducendo significativamente la complessità di calcolo del modello. È una strategia efficace da adottare per reti grandi o complesse.

Capitolo 4

Database

4.1 Introduzione

Questo capitolo descrive la configurazione e le tecniche utilizzate per acquisire i database utilizzati per l'addestramento e il test delle ANN e del modello euristico. Il *setup di base* per raggiungere questo scopo è il seguente: viene usato un PC commerciale, che ha il compito di inviare dei frame su due canali Wi-Fi separati, considerando il Wi-Fi di tipo 2.4 GHz. Questi due canali sono gestiti da due adattatori Wi-Fi disgiunti verso un punto di accesso, ad una distanza fissa e uguale rispetto agli adattatori. Le informazioni utili allo studio sono gli esiti di ogni trasmissione tra il PC e l'access point, utilizzando il Wi-Fi come canale di comunicazione. In particolare siamo interessati a capire se i frame inviati vengono effettivamente ricevuti oppure no, avendo una spaziatura uniforme e omogenea tra frame successivi. È stata utilizzata a tal proposito un'applicazione software ad hoc che salva gli identificatori di tutti i frame trasmessi sui canali e i relativi esiti in un database. È basata su un'architettura chiamata SDMAC [46, 47], una libreria che dà accesso al programma chiamante ad alcune operazioni di basso livello (livello MAC, per la precisione) eseguite dagli adattatori Wi-Fi che altrimenti sarebbero inaccessibili o difficilmente gestibili al livello utente. Viene utilizzata questa libreria non solo per gestire le trasmissioni di un frame e ricevere informazioni riguardo all'effettiva ricezione dello stesso, ma anche per disabilitare alcune funzionalità di gestione di tali frame, come ad esempio il meccanismo di ritrasmissione dei pacchetti, per garantire alcune caratteristiche che approfondiremo in seguito. L'applicazione principale esegue tre macro funzioni, eseguite nel seguente ordine:

1. ogni T_s secondi, l'applicazione invia una richiesta all'adattatore Wi-Fi per trasmettere un frame all'access point tramite un canale specifico. Questa operazione viene eseguita chiamando la funzione `SDMAC_DATA_req(·)`.

2. l'applicazione si blocca e attende indefinitamente finché non riceve una notifica sull'esito della trasmissione (*successo* o $fallimento$). Questa operazione viene eseguita chiamando la funzione `SDMAC_DATA_con(·)`.

3. l'applicazione scrive le statistiche di interesse relative alla richiesta appena effettuata in un database apposito.

Questi passaggi vengono ripetuti finché il database risulta completo.

4.1.1 Prima Fase: Invio e gestione dei frame

Come già accennato, per inviare un frame all'access point, l'applicazione chiama la funzione `SDMAC_DATA_req(·)`, che a sua volta, tutto ciò che fa è principalmente chiamare un endpoint (*sendto*) esposto dai raw socket POSIX API. Il frame passa attraverso il driver e viene temporaneamente salvato all'interno di una coda prima di essere inviato a destinazione dall'adattatore. Questa coda di frame è chiamata *ring buffer*, ed è una struttura dati all'interno degli adattatori Wi-Fi che preleva i frame ricevuti dal PC, cerca di riordinarli nell'ordine prestabilito e li rimanda verso l'access point. Come anticipato in precedenza, alcune funzioni di gestione dei pacchetti verranno disabilitate o modificate, in modo da garantire alcune proprietà utili allo studio. In particolare, andremo a parlare del meccanismo di backoff, la gestione della velocità di trasmissione dei frame e il processo di ritrasmissione degli stessi, che sono le funzionalità disattivate/modificate per questo studio.

Backoff

il backoff casuale è un meccanismo presente nello standard Wi-Fi che serve a ridurre al minimo la probabilità di collisione, dove per collisione si intende la possibilità che due o più dispositivi trasmettano un frame sulla rete nello stesso istante, o quando un dispositivo invia un frame quando il canale di trasmissione è occupato da altri frame. Quando ciò accade, il dispositivo che trova il canale occupato, aspetta un tempo casuale prima di tentare una nuova trasmissione, detto tempo di backoff. Per ottenere una latenza minore dei frame, e garantire una maggiore precisione

delle misurazioni nel tempo, il backoff casuale è stato disattivato. In questo modo, il frame verrà trasmesso subito dopo che il canale sarà considerato libero dalla stazione deputata ad effettuare la trasmissione.

Ritrasmissione del frame

È un meccanismo che si occupa di ritrasmettere pacchetti persi o danneggiati a causa di errori di trasmissione. Ad esempio, un frame viene considerato perso quando il ricevitore non riceve il pacchetto, o tutte le informazioni relative ad esso prima della scadenza di un timer, oppure il ricevitore non riesce ad informare il trasmettitore della corretta ricezione dello stesso prima della fine del timer attraverso uno specifico frame denominato di *acknowledgment*. In tutti questi casi, il frame viene considerato perso, e si attiva il meccanismo di ritrasmissione. Per quanto riguarda il nostro lavoro, abbiamo disabilitato questo meccanismo per i seguenti motivi:

- assicura che ogni pacchetto venga inviato esattamente ogni T_s secondi. Con le ritrasmissioni abilitate, i tentativi potrebbero essere inoltrati anche con molto ritardo.
- garantisce che nel ring buffer sia presente un solo pacchetto alla volta. In questo modo, essendo che l'applicazione raccoglie il tempo effettivo che il pacchetto impiega per completare la trasmissione, questa informazione non verrà influenzato dalla presenza di pacchetti multipli nella coda.

Disabilitandolo e impostando il parametro T_s ad un valore appropriato, garantiamo che le richieste siano sufficientemente equispaziate in modo che non possano esserci più di un pacchetto contemporaneamente nel ring buffer. In questo studio, è stato scelto il valore $T_s = 0.5$ *secondi*.

Velocità di trasmissione

L'ultimo parametro rilevante a cui abbiamo apportato una modifica è la velocità di trasmissione. Nei moderni adattatori Wi-Fi, la velocità di trasmissione viene regolata al volo da meccanismi adattivi, che decidono la velocità di trasmissione per ciascun frame. I driver dei nostri adattatori Wi-Fi utilizzano un algoritmo di adattamento della velocità chiamato Minstrel, che però ha senso utilizzare solo se il sistema applica dei meccansmi di ritrasmissione. [48, 49]. Poiché quest'ultimo è stato disabilitato per i suddetti motivi, non ha senso mantenere attiva questa

funzione. Di conseguenza, la velocità di trasmissione deve essere gestita lato applicazione (con SDMAC). Dagli studi sperimentali effettuati dagli autori di SDMAC, si è visto che avere un bit rate fisso e costante sia ottimale nel migliorare il determinismo di un sistema. Per questo motivo, la velocità di trasmissione è stata impostata a 54Mbps

4.1.2 Seconda fase: rilevazione della ricezione dei frame

La fase successiva si occupa di rilevare se una trasmissione ha avuto successo. Alla ricezione di un frame di tipo acknowledgment (successo) o alla scadenza di un timer (fallimento), il dispositivo genera un interrupt destinato ad informare il sistema operativo che è arrivato un frame. Appena ricevuto, il sistema operativo cercherà di gestire l'interrupt nel minor tempo possibile, rilevando l'esito della trasmissione e alcune statistiche ad essa associata, meglio definite nella sezione successiva. Queste informazioni vengono quindi propagate dal driver all'applicazione in SDMAC e infine salvate come singolo record nel database.

4.2 Configurazione sperimentale

Questa sezione presenterà le impostazioni dei dispositivi sperimentali utilizzati durante l'acquisizione del database. In questo studio, il setup di base presentato all'inizio del capitolo è stata duplicata. Di conseguenza, sono stati usati 2 PC commerciali, che trasmettono separatamente a due punti di accesso diversi, e distribuiti su quattro canali separati, due per PC. Riassumendo, l'applicazione ha acquisito il database:

- utilizzando due PC desktop (trasmettitore A e trasmettitore B) con installato un sistema operativo Linux Ubuntu (v.18.04 con kernel v.4.4.0) e dotato di un adattatore Wi-Fi dual-band TP-Link TL-WDN4800 gestito dal driver del dispositivo ath9k (v. 4.4.2-1).
- con il tempo di campionamento T_s fissato a 0.5 secondi su tutti i canali
- con la velocità di trasmissione impostata a 54Mbps
- con il processo di ritrasmissione, backoff e Minstrel disattivati

Il trasmettitore A invia due pacchetti ogni T_s secondi al punto di accesso A, uno sul canale 1 e l'altro sul canale 9. Allo stesso modo, il trasmettitore B trasmette al punto di accesso B sui canali 5 e 13.
Per ogni trasmissione effettuata, l'applicazione salva all'interno di un database un record con le seguenti informazioni:

- nome del canale (ch1, ch5, ch9 o ch13)
- numero progressivo di un pacchetto, che è univoco all'interno di un file.
- timestamp in ms secondo l'ora Unix
- Time Stamp Counter (TSC) al momento della trasmissione. Il TSC rappresenta il numero di cicli della CPU dalla sua accensione al momento dell'invio del pacchetto[1].
- TSC al momento della ricezione dell'ACK. Analogo al precedente ma calcolato al momento della ricezione dell'ACK o allo scadere del timeout dell'ACK.
- esito della trasmissione. 1 in caso di successo, 0 in caso contrario
- indicatore di potenza del segnale ricevuto (RSSI) dell'ACK. Misura la qualità dell'ACK ricevuto. Se il timer scade prima della ricezione dell'acknowledgement (noto come ACK timeout), il suo valore è indefinito, quindi inaffidabile.
- qualità del collegamento secondo iwconfig
- livello di segnale secondo iwconfig
- bit rate, fissato a 54 Mbps nel caso del database acquisito in questa tesi.

Per vari motivi (guasti, interferenze e disconnessioni) non è stato possibile campionare continuamente sullo stesso canale per tutta la durata dell'esperimento. Il database è perciò costituito da più file per ogni canale contenenti parti non contigue. A tal fine, gli esiti delle trasmissioni da ogni spezzone sono stati salvati in un file di testo separato avente come nome: `<channel>_<trunk>`, dove <channel> e

[1]Per funzionare correttamente, tutti i meccanismi di frequency scaling della CPU devono essere disattivati

<spezzone> sono segnaposti per il nome del canale e l'identificativo dello spezzone, rispettivamente.
Pertanto il database completo viene partizionato prima per canale e poi per spezzone. Sebbene solo una parte delle informazioni raccolte verrà utilizzata durante l'addestramento e il test dei predittori, il set di dati rappresenta una preziosa fonte di informazioni che può essere utilizzata per condurre altri esperimenti basati su questo lavoro. Le caratteristiche dei database raccolti sono riassunte in Tab. 4.1.

canale	F_0 (MHz)	# file	# record/file	# record	# giorni	nome codice
1	2412	121	204354.78	24726928	143.09	db_ch1
5	2432	128	213075.19	27273624	157.83	db_ch5
9	2452	74	375268.49	27769868	160.70	db_ch9
13	2472	74	375156.01	27761545	160.66	db_ch13

Tabella 4.1: Alcune metriche sul database raccolto.

D'ora in poi per riferirci al database acquisito su un determinato canale, verrà preso come riferimento la corrispondente sigla presente nella colonna "nome in codice".
In sintesi, i dati sperimentali sono stati acquisiti in una vasta campagna di acquisizione basata su dispositivi reali, che è durata in generale tra i 140 e i 160 giorni. Per gestire l'applicazione è stata riproposta un'applicazione esistente, basata sull'architettura SDMAC, per trasmettere periodicamente con un periodo fisso un frame su quattro distinti canali Wi-Fi operanti nella banda 2.4 GHz. Sono state quindi raccolte e salvate le varie statistiche relative a ciascuna trasmissione, come il suo esito, in un database. Come anticipato nella sezione 3.6, in modo da poter addestrare la

Canale	Intervallo DBs	#Record	Giorni
13	[0 - 22]	6723595	38.909
1	[0 - 22]	3913437	22.6472
5	[0 - 44]	6809054	39.404
9	[0 - 22]	6727267	38.930

Tabella 4.2: Alcune metriche sul database di previsione

rete neurale, un'operazione molto importante è quella di suddividere il database in due gruppi, il training set e il test set, il primo usato nella fase di apprendimento e il secondo durante la fase di previsione. In Tab. 4.2 e 4.3 si può osservare il modo in cui il database di partenza è stato suddiviso utilizzando questo criterio. In particolare In Tab 4.3, si può notare come, per ogni canale, esistano quattro

configurazioni diverse. Questa scelta è stata fatta per verificare se la rete neurale riesce a migliorare la previsione sul dataset di test all'aumentare dei dati in ingresso.

Canale	Intervallo DBs	#Record	Giorni
13	[23 - 29]	6332847	36.648
	[30 - 49]	4049253	23.433
	[50 - 67]	5612568	32.480
	[68 - 74]	5043282	29.185
1	[23 - 62]	5331418	30.853
	[63 - 91]	5076988	29.380
	[92 - 111]	5534510	32.028
	[112 - 121]	4870575	28.186
5	[44 - 78]	5246181	30.359
	[79 - 108]	5330209	30.846
	[109 - 117]	5183580	29.997
	[118 - 128]	4704600	27.225
9	[23 - 29]	6332659	36.647
	[30 - 49]	4048837	23.430
	[50 - 65]	3555990	20.578
	[66 - 74]	7105115	41.117

Tabella 4.3: Alcune metriche sul database di apprendimento

Di conseguenza, tutti i gruppi definiti non verranno forniti alla rete neurale singolarmente, ma verranno aggiunti al gruppo di database precedente. Quando faremo riferimento ai database di training nei capitoli successivi, verrà utilizzata la seguente notazione: `db_train = chx_y`, dove x indica il canale a cui fanno riferimento, mentre y indica il numero di gruppi che verranno forniti alla rete neurale. Per esempio, se `db_train = ch1_3`, vuole dire che vengono considerati i primi 3 database di training del canale 1.

Capitolo 5

Predizione del canale

5.1 Introduzione

Per analizzare al meglio le proprietà che caratterizzano i canali usati dagli adattatori Wi-Fi, esistono svariati metodi e tecniche per raccogliere queste informazioni. Questi metodi sono generalmente suddivisi in due categorie [50, 51]:

- Schemi per la predizione di un canale che usano come base di partenza alcuni strumenti statistici tradizionali.
- Schemi di predizione che usano algoritmi di machine learning.

Per questo studio, si è scelto di usare metodi relativi al secondo gruppo. L'obiettivo è di creare un modello predittivo che sia in grado di stabilire con una certa precisione la qualità di connessione di un canale Wi-Fi tra due dispositivi, dove per qualità di connessione, si intende la probabilità che un frame, inviato nel suddetto canale da un dispositivo A, riesca ad arrivare senza errori ed entro i tempi limite al dispositivo B, e che il dispositivo B riesca a notificare correttamente l'avvenuta consegna del frame. Per ottenere questo risultato, si è pensato di creare una rete neurale che sia in grado di prevedere questi canali. Per l'addestramento sono state utilizzate le informazioni relative alle trasmissioni avvenute nel passato e nel presente. Per comprendere meglio come queste informazioni vengano collezionate, riportiamo un esempio pratico.

Consideriamo un istante temporale t, e consideriamo un frame f_t inviato all'istante t da un dispositivo A, ma non ancora ricevuto dal dispositivo B. In generale, possiamo affermare che:

- I frame precedenti a f_t, con istante temporale $t_{temp} < t$, sono già stati inviati dal dispositivo A. Di conseguenza sappiamo già se sono stati ricevuti correttamente o meno dal dispositivo B. Questa informazione viene codificata con:

  1. "1" se il pacchetto è stato ricevuto correttamente. La corretta ricezione del frame è registrata dal dispositivo A quando viene ricevuto il rispettivo frame di Acknowledgment.
  2. "0" altrimenti.

  chiameremo questi frame f_{past}.

- I frame successivi a f_t, con istanti temporali $t_{temp} > t$, non sono ancora stati inviati dal dispositivo A; quindi sono quei frame in cui il modello dovrà cercare di predirne l'esito. Chiameremo questi frame f_{next}, in cui è incluso anche f_t, poiché non sappiamo ancora l'esito relativo alla sua trasmissione.

- Definiamo la variabile N_{past}, che identifica il numero di frame f_{past} che andremo a considerare, la variabile N_{next}, che identifica il numero di frame f_{next} di cui vogliamo conoscere l'esito e la variabile N_{step}, che sarà spiegata in seguito. Tutte queste variabili, fanno riferimento all'istante t.

L'idea generale per cercare di comprendere l'andamento del canale, quindi di prevedere la qualità di trasmissione dei frame f_{next} è di usare gli esiti di N_{past} frame f_{past}, rielaborare questi esiti tramite alcune funzioni matematiche, ed usarle come riferimento per prevedere gli esiti di N_{next} frame f_{next}. Poiché per la rete neurale sarebbe molto difficile gestire tutti gli esiti dei frame inviati singolarmente, si è attuata una strategia per poter aggregare questi risultati, rendendoli così più accessibili per il modello stesso, potendo ottenere così un *tasso di successo* in un intervallo di tempo multiplo di N_{step}, definito poco fa. Il tasso di successo esprime il rapporto tra i frame correttamente trasmessi e ricevuto dal dispositivo B e il numero di frame totali.

5.2 La finestra mobile

In questa sezione verrà esposto con maggior dettaglio il modo in cui vengono calcolati i tassi di successo, che verranno poi usati come parametri d'ingresso della rete neurale.
Per semplicità, consideriamo le seguenti variabili:

- $N_{past} = 9$, ovvero verranno considerati i primi nove frame inviati e processati, appartenenti quindi a f_{past}.
- $N_{next} = 6$, ovvero i 6 frame successivi, e di cui non abbiamo ancora una rilevazione, appartenenti quindi a f_{next}.
- $N_{step} = 3$ che indica il numero di elementi appartenenti ad un gruppo di frame di f_{past}. Il numero di gruppi da creare dipendono dal rapporto $\frac{N_{past}}{N_{step}} = N_{features}$; di conseguenza N_{past} deve esser un multiplo di N_{step}.
- $window$, un array di $\frac{N_{past}}{N_{step}} + 1 = 4$ elementi, che viene usato per salvare i vari tassi di successo.

Consideriamo la situazione in cui sono stati inviati 9 frame, e si voglia elaborare i tassi di successo sia dei frame inviati che dei successivi 5 frame. Siamo quindi nella situazione in cui il decimo frame sta per essere inviato dal dispositivo A. Per creare le etichette rappresentati i tassi di successi dei frame passati, il ragionamento è il seguente: divido i 9 frame precedenti in tre gruppi da tre frame; nel gruppo formato dai frame 7, 8 e 9, calcolo il tasso di successo di ricezione $success_1$ per i primi tre frame con $t = 10$ come:

$$success_1 = \frac{1}{N_{step}} \sum_{i=7}^{N=9} f_{t-i+1} \tag{5.1}$$

dove f_i è uguale a 1 se il frame $i-esimo$ è stato ricevuto correttamente dal dispositivo B, e 0 se invece non è stato ricevuto. Questo valore ci dà una prima etichetta, che descrive la qualità del canale Wi-Fi nei primi 1.5 secondi prima dell'invio del decimo frame. Per il calcolo della seconda etichetta, non si tengono in considerazione solo i frame 4, 5 e 6, ma si usano anche gli esiti forniti dai frame precedenti. Di conseguenza, il secondo tasso di successo sarà uguale a:

$$success_2 = \frac{1}{2N_{step}} \sum_{i=4}^{N=9} f_{t-i+1} \tag{5.2}$$

In questo caso, il tasso di successo trovato ci dice invece la qualità del canale nei primi 3 secondi prima dell'invio del decimo pacchetto. In generale, possiamo calcolare il tasso di successo dell'etichetta $j - esima$ come segue:

$$success_j = \frac{1}{jN_{step}} \sum_{i=k}^{N_{past}} f_{t-i+1} \tag{5.3}$$

dove k varia per ogni etichetta e vale $N_{past} - jN_{step}$ per ogni gruppo j considerato. Quando tutti i gruppi vengono esplorati e i tassi cumulativi di successo vengono calcolati per ognuno di essi, abbiamo un nuovo record di etichette, che si aggiunge al database che verrà usato come base di partenza per il modello della rete neurale. A questo punto, per calcolare le etichette relative all'undicesimo pacchetto, o più in generale al frame f_{t+1} quello che bisogna fare è "andare avanti di un frame", cioè considero l'undicesimo frame come quello che sta per essere inviato dal dispositivo A, e considero invece tutti i $N_{past} = 9$ frame precedenti per il calcolo dei nuovi tassi di successo (per intenderci, i frame presi in considerazione sono inclusi nell'intervallo [2, 10]). In questo modo si viene a creare una vera e propria *finestra mobile*, che simula l'ordine in cui i frame vengono inviati, e in cui si possono raccogliere le statistiche per i frame antecedenti a quello che si vuole considerare.

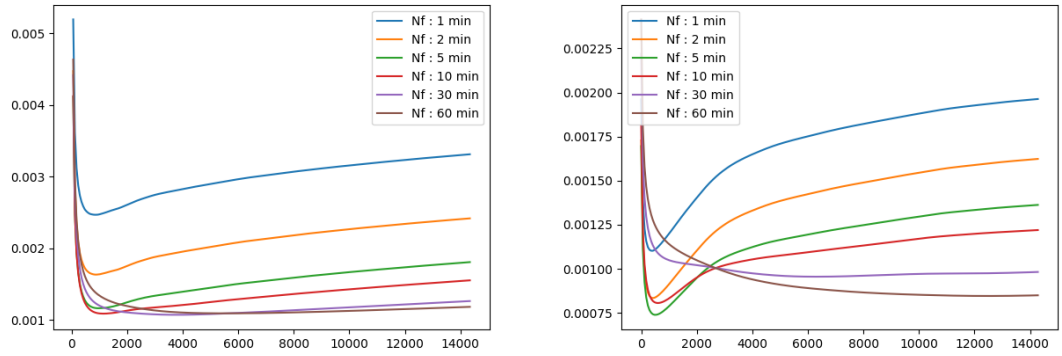
5.3 Metodo Euristico: metro di paragone per la rete neurale

In questa sezione, viene approfondito il metodo euristico utilizzato per comparare le prestazioni della rete neurale dopo che è stata addestrata con il database creato nella sezione precedente. Questo metodo si basa sulla seguente premessa: l'andamento di un canale nell'immediato futuro è molto simile alla qualità offerta dal canale nel passato. Questa assunzione implica che, trovando il tasso di successo del canale nel gruppo di frame antecedenti a un frame f_t, possiamo considerarlo almeno comparabile con il tasso di successo che si otterrebbe nei frame successivi. Per modellare questo concetto, l'idea non è più quella di calcolare dei tassi cumulativi, che ci danno una visione più ampia della situazione del canale in diversi periodi temporali, ma calcolare un unico tasso di successo per ogni frame considerato. Consideriamo la situazione precedente: il decimo frame sta per essere inviato

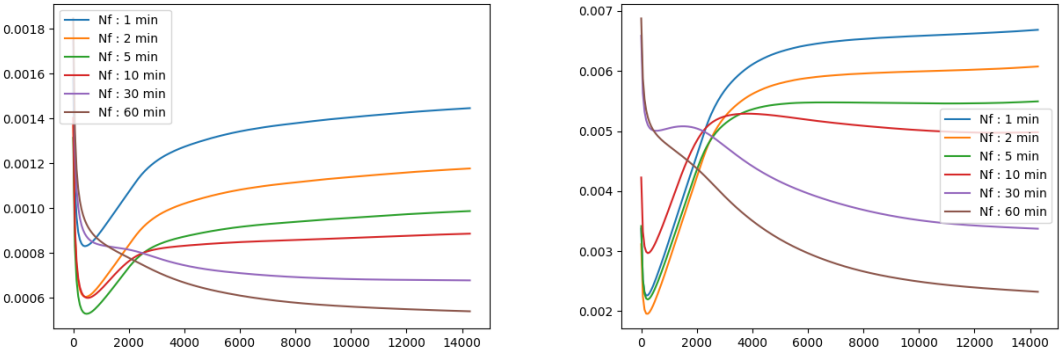
dal dispositivo A. Quello che si andrebbe a calcolare non sono più i tre tassi cumulativi, ma un unico tasso di successo in cui tutti i 9 frame precedenti o solo una parte verranno considerati nella media totale. In altri termini, possiamo esprimere la previsione dell'euristica come segue:

$$success_{10}^{AVG} = \frac{1}{N_{past}^{AVG}} \sum_{i=1}^{N_{past}^{AVG}} f_{t-i+1} \tag{5.4}$$

dove N_{past}^{AVG} è un valore minore o uguale a N_{past} considerato in precedenza. Il valore in N_{past}^{AVG} è stato scelto per cercare di trovare l'intervallo di frame ottimale in cui il metodo euristico dia in media risultati migliori. Per ogni valore compreso tra 2 e il valore di N_{past} scelto per la modellazione della rete neurale, vengono calcolati tutti i tassi di successo di ogni frame, e viene poi calcolato l'Errore Quadratico Medio (MSE, Eq. 3.11) tra tutte le previsioni generate dal modello euristico e la qualità reale del canale in quel lasso temporale. Per ogni canale, e per ogni periodo temporale che vogliamo prevedere, viene poi scelto il valore di N_{past}^{AVG} in cui MSE risulta essere il più basso. La Figura 5.3 riporta il valore di MSE ottenuto al variare del parametro N_{past}^{AVG} per ogni canale sotto analisi.
Intuitivamente si può notare che per valori molto bassi di N_{past}^{AVG}, si ottiene un errore quadratico medio abbastanza alto, questo perché il numero di frame per etichetta è troppo basso per poter prevedere l'andamento del canale, soprattutto se l'obiettivo è prevedere andamenti di 30 o di 60 minuti. Più N_{past}^{AVG} aumenta, più il metodo euristico inizia ad ottenere risultati migliori, ottenendo errori sempre più piccoli. Questo andamento ci dà un'indicazione su come l'assunzione fatta all'inizio di questa sezione non sia del tutto erronea. In base al canale e in base anche al target temporale di riferimento, si arriva ad ottenere un valore di N_{past}^{AVG} in cui l'errore è minimo. Qui di seguito vengono visualizzati i risultati, con i relativi target e valore di N_{past}^{AVG} corrispondente, che sono stati scelti per la comparazione con i risultati ottenuti con la rete neurale.



(a) Andamento MSE per il canale 13 e per il canale 1



(b) Andamento MSE per il canale 5 e per il canale 9

Figura 5.1: MSE dei canali al variare di N_{past}^{AVG}

5.4 Metodi di confronto usati

Il metodo euristico ci dà un primo strumento di analisi che ci permette di confrontare le previsioni ottenute dalle reti neurali che andremo a testare. In aggiunta verranno forniti altri strumenti statistici ed alcune funzioni matematiche utili ad analizzare il modello. In particolare, andremo ad usare:

- Errore Quadratico Medio, definito in Eq. 3.11
- Errore Assoluto Medio, definito in Eq. 3.12
- 95 percentile relativo all'Errore Assoluto, una funzione che esprime il seguente concetto: il valore ottenuto rappresenta l'errore assoluto più alto che si è ottenuto nel 95% dei casi.

Canale	Target [min]	MSE [$\cdot 10^{-3}$]	N_{past}^{AVG}
13	1	2,470	900
	2	1,635	960
	5	1,165	1020
	10	1,090	1200
	30	1,073	3840
	60	1,094	5580
9	1	2,260	300
	2	1,955	360
	5	2,194	360
	10	2,966	360
	30	3,373	14400
	60	2,322	14400
5	1	0,830	540
	2	0,604	540
	5	0,528	600
	10	0,599	660
	30	0,677	14400
	60	0,539	14400
1	1	1,102	540
	2	0,833	540
	5	0,738	600
	10	0,805	720
	30	0,955	6420
	60	0,845	12600

Tabella 5.1

- 90 percentile relativo all'Errore Assoluto, che esprime lo stesso concetto, ma applicato al 90%.
- Percentuale di vittoria del metodo basato su rete neurale rispetto al metodo euristico.

Capitolo 6

Software

6.1 Introduzione

Questo capitolo descrive la struttura del software in `Python` creato per lo studio in questione. Prima di approfondire nel dettaglio tutte le classi, verranno elecante e definite tutti i moduli creati, in modo da avere una visione ad ampio spettro del programma:

- Modulo *WiFiPredictor*, modulo wrapper che gestisce l'avvio di tutti i moduli. In particolare, in base ai parametri passati da linea di comando, è in grado di eseguire solo un determinato modulo del software per volta. Questa caratteristica è stata scelta per due ragioni fondamentali:

    1. In questo modo possiamo scegliere in maniera indipendente quale modulo usare senza per forza passare per tutti gli altri.
    2. Per problemi di memoria, legati principalmente alla macchina usata e alle librerie fornite per la gestione dei dataset nella memoria centrale. In particolare, se il programma viene eseguito più di una volta e vengono create delle nuove istanze delle variabili che gestiscono i dataset, parte della memoria utilizzata in precedenza non viene liberata correttamente, creando così il rischio che il programma termini in maniera anomala.

    Per questi motivi, si è pensato di usare un meccanismo dove, tramite script in bash, il programma viene richiamato più volte, assicurandosi però che la memoria venga prima liberata del tutto.

- Modulo *Frontend*, che si occupa della precomputazione dei dataset da fornire in ingresso alla rete neurale. Questi dataset verranno utilizzati sia nella fase di apprendimento che nella fase di previsione dei risultati, seguendo gli algoritmi descritti nella Sezione 5.

- Modulo *Heuristic*, che si occupa della precomputazione dei dataset da usare come metro di paragone per confrontare i risultati ottenuti con la rete neurale. È stato usato l'algoritmo descritto nella Sezione 5.3

- Modulo *Control*, che si occupa di controllare la coerenza dei dataset appena creati. In particolare, per ogni record del dataset generato nella modulo Frontend, viene controllato se il target generato risulti uguale al target presente nel record del dataset generato nella modulo Heuristic.

- Modulo *TrainModel*, che si occupa della gestione delle funzioni relative alla fase di apprendimento del modello, dove viene inizializzata la rete neurale che poi verrà addestrata da alcune funzioni fornite dal modulo stesso. Lo stesso modulo si occupa della fase di previsione e di raccolta delle statistiche del modello.

- Modulo *Datagenerator*, modulo ausiliario di TrainModel, che si occupa in particolare della creazione e del trasferimento degli input nella rete neurale, distribuendo il dataset di ingresso in gruppi di record di dimensione $dim = batch\_size$, come descritto nella Sezione 3.6.1. Inoltre, ci permette di gestire, quando è richiesto, lo shuffle dei dati da fornire alla rete.

- Modulo *Stats_choice*, che si occupa della creazione di gruppi di dati, a partire da tutti i dataset creati nel modulo Frontend, che verranno usati per la sperimentazione del modello. In particolare, per ogni canale che viene controllato, si è suddiviso il dataset in gruppi equispaziati, dove ogni gruppo contiene le informazioni raccolte in un mese e mezzo del canale stesso. Questa suddivisione è stata scelta per controllare se, all'aumentare dei dati forniti alla rete, il modello riesce a trarne beneficio o meno e, in caso affermativo, se c'è un numero di dati limite in cui il modello entra in uno stato di convergenza.

- Modulo *ITL.conf*, file di configurazione che ci permette di inizializzare tutte le variabili utili per lo svolgimento del programma.

Per una migliore comprensione del capitolo, verrà descritta la struttura del modulo ITL.conf, definendo così le variabili e le definizioni necessarie per poter descrivere gli altri moduli.

6.2 Modulo ITL.conf

Come anticipato, questo modulo ci aiuta nella definizione delle variabili che entrano in gioco nei moduli successivi. In particolare, è utile nel definire i database necessari, e ci aiuta a definire il setup di partenza su cui si intende sperimentare la rete. Di seguito, verrano suddivise ed elencate tutte le variabili definite all'interno del modulo, così da poterci riferire direttamente ad esse per tutto il capitolo:

- Variabili Directory: ognuna delle variabili appartenenti a questo gruppo, identifica in modo univoco un percorso all'interno del PC utilizzato, dove verranno caricate/salvate tutte le informazioni necessarie. A questo gruppo, appartengono le seguenti variabili:

  1. HOME, variabile che indica il percorso di partenza da cui tutte le directory fanno riferimento.
  2. DB_DIR, percorso che conduce nella directory dove sono salvati tutte le informazioni relative alle trasmissione di singoli frame verso il punto di accesso di riferimento, come illustrato nella Sezione 4.
  3. PATTERN_DIR, directory che verrà usato per salvare tutti i dataset elaborati a partire dal database di partenza, usando i metodi illustrati nella Sezione 5.
  4. PATTERN_DIR_HEU, directory che verrà usato per salvare tutti i dataset elaborati a partire dal dataset di partenza, usando però il metodo euristico scelto come metro di paragone spiegato nella Sezione 5.3.

- Variabili Dataset: questo gruppo si occupa di tener traccia della nomenclatura dei file da cercare o da generare in memoria. Ogni variabile definita in questo gruppo è un array di stringhe, ognuna delle quali, ha la seguente struttura:

  $$\text{“}chx\_y\text{”} \tag{6.1}$$

  dove x indica il numero del canale a cui si fa riferimento e y è un identificativo, utile a selezionare propriamente la porzione di dataset da caricare/salvare in

memoria. Inoltre, ognuno di questi gruppi è formato da una serie di dataset che fanno riferimento a circa un mese e mezzo di acquisizione dei dati. Ogni variabile è definita come:

$$db\_x\_y \tag{6.2}$$

dove x rappresenta anche in questo caso il numero del canale selezionato e y rappresenta a quale dei 4 gruppi stiamo facendo riferimento. Oltre a questi gruppi, è stato definito un gruppo di dataset di partenza, chiamati *train_temp* e *test_temp*, che identificano il setup di partenza di apprendimento e di previsione su cui è partito questo studio, e su cui vengono sperimentati tutti i cambiamenti che si voglio apportare al modello.

- Variabili Frontend: questo gruppo ci aiuta ad identificare tutte le variabili necessarie per la corretta inizializzazione del modulo Frontend. È composto dalle seguenti variabili:

  - *in_dir*, che indica il percorso DB_DIR.
  - $in\_dir\_file\_ext$, che indica l'estensione dei dataset salvati in *in_dir*. In questo studio, essendo che il dataset viene acquisito usando il linguaggio C, verranno salvati come file .dat.
  - *out_dir*, che indica il percorso PATTERN_DIR.
  - $out\_dir\_file\_ext$, che indica l'estensione dei dataset salvati in *out_dir*. In questo studio, essendo che il dataset viene generato usando delle librerie fornite dal linguaggio Python, verranno salvati come file .feather.
  - N_{step}, che indica la dimensione su cui vogliamo calcolare il tasso di successo, relativo alla consegna o meno del pacchetto.
  - N_{past}, che indica il numero di feature totali considerate per calcolare tutte le percentuali di successo; di conseguenza il rapporto N_{past}/N_{step} indicherà il numero di label per ogni record del nuovo db.
  - N_{next}, che indica la dimensione della finestra su cui sarà calcolato il target.
  - *list* indica tutti i nomi dei dataset, definiti in 6.1, utilizzati dunque sia per caricare i dataset .dat, sia per salvare i dataset .feather, usati come rappresentazione intermedia.

- Variabili Train: questo gruppo ci aiuta ad identificare tutte le variabili necessarie per la corretta inizializzazione del modulo TrainModel. È composto dalle seguenti variabili:

  - *db_in*, che indica il percorso PATTERN_DIR.
  - *db_heu*, che indica il percorso PATTERN_DIR_HEU.
  - *train_dbx*, dove x indica un numero progressivo compreso tra 1 e 10. Ogni variabile contiene i dataset a partire da un canale scelto, suddivisi in come descritto nel Capitolo 4, e verranno usati in maniera progressiva per addestrare la rete neurale. In questo caso particolare, i primi 4 *train_db* contengo i gruppi di quel determinato canale, mentre gli altri 6 contengono 2 gruppi per ogni canale, poichè si è voluto sperimentare quanto possa influire la conoscenza di un altro canale per la comprensione dello stesso.
  - *test_db*, identifica il gruppo scelto per verificare la correttezza del modello durante la fase di previsione. In questo caso, il gruppo scelto è unico e fisso per ogni canale scelto, così da garantire che i risultati siano comparabili con i risultati ottenuti con la computazione euristica.
  - *train* e *test* sono invece i dataset scelti come configurazione di partenza del modello. In queste variabili sono presenti molte meno informazioni (solo i db 0, 1, 2, 4 per il testing e 5, 6, 7, 8, 9 per il training, tutti riferiti al canale 13). Questa configurazione viene utilizzata nella fase iniziale di questo studio, per analizzare tutti i parametri della rete neurale, scegliendo quelli che presentano i risultati più promettenti.
  - *ext*, estensione dei file salvati in db_{in} (.feather).
  - *heu_ext*, estensione dei file salvati in db_{heu} (_heu.feather).
  - *stats_dir*, percorso della directory dove verranno salvate tutte le statistiche relative alla fase di previsione raccolta dal programma.

- Variabili Euristiche: questo gruppo ci aiuta ad identificare tutte le variabili necessarie per la corretta inizializzazione del modulo Heuristic. È composto dalle seguenti variabili:

  - *N_next* e *N_past*, già definiti sopra.

  - N_past_used, parametro usato per definire l'effettiva dimensione dei record da usare nel database di acqusizione per applicare il metodo euristico spiegato nel Capitolo 5.3 e creare gli input da usare come metro di paragone con i dati generati dalla rete neurale.
  - in_dir, come db_dir.
  - in_dir_ext, come ext.
  - out_dir, come db_heu.
  - out_dir_ext.
  - N_past_files.
  - N_past_dir.
  - $list$.
  - ch.
  - $targets$.

- Variabili Control: questo gruppo ci aiuta ad identificare tutte le variabili necessarie per la corretta inizializzazione del modulo Control. Tutte le variabili usate in questo gruppo sono già dichiarate nei gruppi precedenti.

6.2.1 Frontend

In questo modulo viene effettuata la gestione e la creazione della finestra mobile descritta nel capitolo 5. Sono le variabili e le funzioni utilizzate per lo svilluppo del modulo:

- N_{step}, N_{next} e N_{past}, definite nella Sezione precedente.
- win_loss_list, un array di dimensione $N_{past} + N_{next}$ dove vengono prelevate tutte le risposte di ogni singolo pacchetto (pacchetto ricevuto = 1, pacchetto non ricevuto = 0)
- $window$, un array di dimensione $N_{past}/N_{step} + 1$ dove verrano calcolati rispettivamente i N_{step} label e il target di riferimento.
- $feather\_table$, una matrice di dimensione $size[window] * 100000$ che verrà usata per salvare in locale la finestra mobile. Per questioni relative alla

gestione della memoria del PC e delle prestazioni per il training della rete, si è deciso di effettuare una precomputazione delle feature, da utillizzare in seguito.

- *success*, un booleano che indica se un pacchetto x è stato ricevuto correttamente o meno. questa variabile non viene usata nel programma, ma viene definita per una più facile comprensione del testo.

- funzione $feather\_creation$, funzione che si occupa della gestione dei database descritti nel Capitolo 4 e della creazione e del salvataggio della finestra mobile da usare durante la fase di apprendimento della rete neurale.

- funzione *perc_success*, funzione chiamata dalla funzione $feather\_creation$, che ha il compito di calcolare la totalità della finestra. Per ogni record i, dopo che la funzione $feather\_creation$ preleva gli N_past elementi precedenti e gli N_next successivi, la funzione *perc_success* viene chiamata e calcola tutti i $\frac{N\_past}{N\_step}$ tassi di successo progressivi anteriori a i, con i escluso, e il tasso di successo relativo al target/ai target da confrontare, successivi ad i, con i incluso.

A partire dai dati del database descritto nel Capitolo 4, il programma inizia il calcolo della finestra mobile al record $i = N\_past$ e salva in win_loss_list $N\_past + N\_next$ instanze di *success*, esattamente N_past precedenti a i e N_next successive ad esso. Fatto ciò, il programma calcola separatamente i progressivi $\frac{N\_past}{N\_step}$ tassi di successo e calcola il tasso di successo delle N_next feature successive. Questi input e il target vengono poi salvati in *window*, e aggiunti a $feather\_table$. Questo processo viene iterato finchè tutti i record meno N_next verranno esplorati.

6.3 Euristica

In questo modulo viene effettuato invece il calcolo delle feature basate sull'euristica che verrà usata come metro di paragone per analizzare i risultati della rete neurale. Lo sviluppo di questo algoritmo è simile a quello descritto nella Sezione 6.2.1, ma invece di calcolare N_{step} tassi di successi per la creazione degli input delle rete, viene calcolato un unico tasso di successo, calcolando una semplice media tra gli N_past_used esiti di trasmissione dei pacchetti (Sezione 5.3, che ricordiamo essere un valore minore o uguale a N_past usato per il calcolo della finestra mobile, che

ci garantisce il miglior risultato che l'euristica può ottenere per poter poi valutare i risultati ottenuti nella fase di previsione della rete neurale. Anche se vengono usati N_past_used dati per ogni input dal database di acquisizione, il calcolo delle feature inizia dal record $i = N\_past$, come nel modulo Frontend. Questo per garantire che ci sia coerenza tra i target generati nei due moduli.

6.4 TrainModel

La sezione più importante del programma creato, si occupa della gestione e dello sviluppo della fase di apprendimento (Sezione 3.6), e della gestione dei risultati ottenuti nella fase di previsione. Andiamo ad analizzare le funzioni principali e tutte le librerie utilizzate al fine dello sviluppo di questo modulo:

- funzione *train*, che si occupa della creazione della rete neurale da analizzare e gestisce tutta la fase di apprendimento del modello.
- funzione *predict*, che gestisce invece tutta la fase di previsione dei risultati. Inoltre, genera le tebelle necessarie per la raccolta delle statistiche.
- funzione *stats*, che si occupa del calcolo delle funzioni matematiche descritte nella Sezione 5.4, oltre a caricare in memoria tutte le informazioni relative al metodo euristico.

Per poter gestire in maniera ottimale tutta la mole di dati che la rete neurale deve processare, questo modulo lavora in concomitanza con alcune funzionalità offerte dal framework *keras*. In particolare, le funzioni offerte da keras ci danno strumenti utili a gestire e a modellare tutti i parametri necessari alla realizzazione delle reti neurali che si vogliono testare. Il modulo *Numpy* e più nello specifico *Pandas* invece, aiutano a gestire l'acquisizione dei database in ingresso, gestendo in maniera più immediata le matrici che verranno processate e poi inviate in ingresso al modello. Infine il modulo *Datagenerator* ci da funzionalità utili sia per la divisione del dataset in ingresso in batch di dimensioni ridotte, sia per implementare in maniera efficente lo shuffle di questi batch, quando richiesto.

Capitolo 7

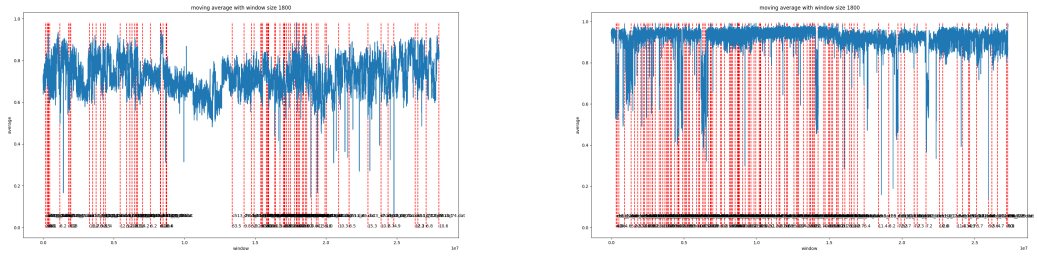
Esperimenti e Risultati

7.1 Introduzione

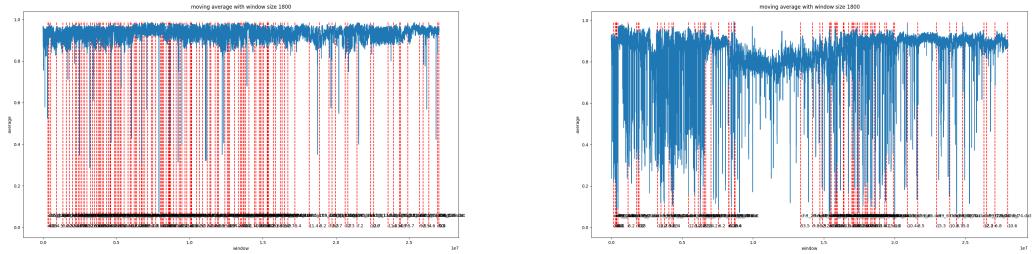
Questo capitolo andrà a descrivere le caratteristiche di tutti i canali che sono stati utilizzati per questo studio, cercando di effettuare una prima previsione empirica basata su database di piccole dimensioni sui risultati che si potrebbero ottenere dalla rete neurale, analizzerà tutti gli esperimenti svolti sulle varie architetture sviluppate, e mostrerà i risultati di tali esperimenti utilizzando le metriche e le funzioni di costo descritte nella Sezione 5.4.
Nel capitolo 4, è stato analizzato il metodo di acquisizione dei dati relativi ai frame inviati da due PC commerciali e ricevuti da un access point, passando attraverso quattro canali diversi. Per ognuno di questi canali, sono state collezionate le informazioni corrispondenti a circa 8 mesi di comunicazione tra i due end point[1]. Da questi dati, è possibile calcolarne la qualità durante questo lasso temporale.
In particolare, esprimendo un passo $window\_size$ intero e maggiore di 2, si può esprimere il tasso di successo del canale all'istante in cui $window\_size$ è riferita per tutta la durata dell'acquisizione, cioè dividendo tutti i frame in gruppi formati da $window\_size$ elementi, si può calcolare il tasso di successo per ognuno di questi gruppi.
In Fig. 7.1, si può osservare l'evoluzione temporale dei tassi di successo dei canali 1, 13, 5 e 9, considerando $window\_size = 1800$ (ogni gruppo dove il tasso di successo

[1]Nella pratica i dati raccolti fanno riferimento ad un periodo temporale effettivo compreso tra i 140 e i 160 giorni, perchè ci sono stati istanti in cui l'acquisizione è stata fermata

viene calcolato è formato da 1800 frame; di conseguenza i dati relativi al canale vengono suddivisi in gruppi da 15 minuti, per il tempo di ritrasmissione scelto nella Sezione 4.1.1). Sull'asse delle ascisse viene rappresentato l'istante temporale a cui



(a) Status canale 13 e canale 1



(b) Status canale 5 e canale 9

Figura 7.1: Tassi di successo relativi ai quattro canali analizzati

si va riferimento e sull'asse delle ordinate viene espresso il tasso di successo, che per definizione è un valore compreso tra 0 e 1. In blu viene espresso il tasso di successo che il canale ha all'istante t. In Fig. 7.1a, si osserva la qualità del canale 13 e del canale 1. Questi due canali, verranno esaminati in tutti gli esperimenti effettuati in questo studio. Si può osservare una prima differenza sostanziale tra i due canali:

- il canale 13 presenta un tasso di successo generalmente discreto in tutto il periodo di acquisizione analizzato (tasso di successo compreso tra 0.8 e 0.65), senza presentare oscillazioni troppo brusche. Questo grafico rappresenta una situazione in cui il canale è utilizzato da altri dispositivi nella rete. Di conseguenza, le reti neurali sotto esame dovrebbero essere in grado di prevedere con un basso errore l'evoluzione della qualità del canale.

- Il canale 1, invece, presenta un tasso di successo migliore rispetto al canale 13 in quasi tutto il periodo (tasso di successo che talvolta supera 0.95), ma

sono presenti piccoli intervalli temporali in cui il tasso di successo crolla velocemente, raggiungendo valori a volte inferiori a 0.4. In questo scenario, le reti neurali dovrebbero affrontare più difficoltà nell'elaborare le informazioni fornite dal canale, per via di questi bruschi cambiamenti, difficilmente prevedibili dal modello, rendendo i risultati generalmente peggiori se comparati al canale 13.

In Fig. 7.1b, sono invece visualizzati le funzioni del tasso di successo riferiti al canale 5 e al canale 9. I dati forniti da questi due canali sono stati utilizzati soltanto nella prima fase di sperimentazione di questo studio, relativo alla rete neurale feed-forward descritta nella Sezione 3.5.1. Questa scelta è stata fatta per due motivi:

- I canali 5 e 9 rappresentano delle situazioni limite, dove il tasso di successo si presenta in un caso ottimo con un'evoluzione molto regolare e nell'altro pessimo con un'evoluzione molto irregolare; sono dunque scenari che si presentano molto di rado nelle reti industriali.

- Per l'analisi di una rete LSTM, i tempi di addestramento sono molto più alti rispetto ai tempi di addestramento di una rete feed-forward. Di conseguenza, sono stati scelti i due canali che rappresentano le situazioni più comuni.

7.2 Esperimenti effettuati

In questa sezione, verranno elencati tutti gli esperimenti che sono stati condotti in questo studio, spiegando le motivazioni per le quali sono stati effettuati. Per ognuno di questi esperimenti, verranno elencati tutti i parametri utilizzati per la modellazione della rete neurale, le caratteristiche e le configurazioni dei database utilizzati sia per addestrare la rete sia per la fase di verifica della stessa. I parametri che verranno presi sotto esame sono i seguenti:

- `batch_size`, parametro che identifica il numero di elementi forniti in ingresso alla rete durante la fase di addestramento. Per elemento in questo caso si intende il vettore di feature definito in Eq. 3.7.

- `learning_rate`, che descrive il tasso di apprendimento di partenza usato durante la fase di apprendimento. I valori tipici usati sono `0.1, 0.01 o 0.001`.

- `learning_rate_descend`, che descrive invece l'andamento del tasso di apprendimento, ovvero come il tasso di apprendimento varia alla fine di ogni epoca. In questo studio, le discese che sono state analizzate sono:

  1. discesa quadratica: il tasso di apprendimento all'epoca e assume il seguente valore:

     $$learning\_rate(e) = \frac{learning\_rate}{2^e} \tag{7.1}$$

     dove $learning\_rate$ indica il tasso di apprendimento iniziale.

  2. discesa lineare: il tasso di apprendimento all'epoca e assume il seguente valore:

     $$learning\_rate(e) = \frac{learning\_rate}{c \cdot e} \tag{7.2}$$

     dove c indica una costante compresa tra 0 e 1.

  3. discesa sinusoidale o discesa a spirale: il valore del tasso di apprendimento, al variare del numero di epoche, vale:

     $$learning\_rate(e) = learning\_rate \cdot \frac{\cos \frac{\pi}{11} \cdot e}{e} \tag{7.3}$$

     Questa particolare discesa non si comporta come le due precedenti, ma presenta punti in cui il valore del tasso di apprendimento risulta più alto del valore all'epoca precedente. Questo perché una piccola risalita di questo tasso permette in certi casi di uscire da un minimo locale.

- `cell_size`, valore che descrive la dimensione dello strato nascosto della rete neurale, ovvero il numero di neuroni artificiali da cui esso è composto. I valori testati sono `32, 64, 128 e 256`.

- `output_cell_size`, che indica il numero di neuroni dello strato di uscita. In altre parole, il numero di target che il modello è in grado di analizzare. In questo studio, abbiamo analizzato reti con 1 uscita o con 6, per verificare se un modello con 6 uscite possa risultare migliore rispetto ad una rete che si focalizza su un'unica uscita.

- `hidden_layer_type`, variabile che identifica la tipologia di rete neurale creata, ovvero se si vuole analizzare una rete di tipo feed-forward (Sezione 3.5.1) o una rete di tipo LSTM (Sezione 3.5.3).

- `shuffle`, valore logico che indica se il database in ingresso verrà ricevuto in modo casuale oppure no. Questa operazione viene effettuata soprattutto nel modello feed-forward perché in questo modo è molto più facile per la rete neurale uscire da un minimo locale, trovando in generale risultati migliori.

- `shuffle_per_batch`, valore logico che indica invece se lo shuffle viene effettuato solo tra i batch già generati, mantenendo così una correlazione temporale delle feature all'interno del singolo batch. Questo particolare shuffle è stato implementato soltanto per la rete LSTM, così da poter sfruttare al meglio le proprità di memoria offerte dalla stessa.

- `train_dbs`, che indica i database usati in ingresso durante la fase di apprendimento (Tab. 4.3).

- `test_dbs`, che indica i database usati in ingresso durante la fase di previsione (Tab. 4.2).

Prima di parlare dei singoli esperimenti effettuati usando i dati raccolti nella Sezione 4, poi rielaborati seguendo il criterio descritto nella Sezione 5, sono stati effettuati degli esperimenti preliminari, in cui sono state testate le varie configurazioni descritte sopra, partendo da un set di addestramento e da un set di previsione di partenza, anticipato nella Sezione 6.2. In particolare, per ogni configurazione testata, il procedimento di selezione è stato il seguente:

1. si esegue il programma, andando a raccogliere le statistiche per quella particolare configurazione. Questa operazione si effettua cinque volte.

2. vengono calcolate la media e la varianza di tutte le funzioni di errore raccolte, e si sceglie la configurazione con errori minori, dando priorità alle misurazioni a varianza minore.

In Tab. 7.1 e 7.2 sono rappresentate tutte le configurazioni analizzate e tutte le statistiche raccolte per le configurazioni più promettenti seguendo il criterio appena

descritto[2].

Queste statistiche fanno riferimento ad una rete neurale di tipo feed-forward con

Configurazione	batch_size	LR	LRD	cell_size	shuffle
1	256	0.01	quadratica	64	Vero
2	256	0.01	lineare	64	Vero
3	256	0.001	quadratica	64	Vero
4	256	0.001	lineare	64	Vero
5	256	0.0001	quadratica	64	Vero
6	256	0.0001	lineare	64	Vero
7	256	0.01	quadratica	128	Vero
8	256	0.01	lineare	128	Vero
9	256	0.001	quadratica	128	Vero
10	256	0.001	lineare	128	Vero
11	256	0.0001	quadratica	128	Vero
12	256	0.0001	lineare	128	Vero
13	256	0.01	quadratica	128 x 64	Vero
14	256	0.01	lineare	128 x 64	Vero
15	256	0.01	quadratica	256 x 128	Vero
16	256	0.01	lineare	256 x 128	Vero
17	512	0.01	quadratica	256 x 128	Vero
18	512	0.01	lineare	256 x 128	Vero

Tabella 7.1: Configurazioni testate

Conf	MAE [%]	Var MAE	MSE [$\cdot 10^{-3}$]	Var MSE [$\cdot 10^{-5}$]	Win vs. Heu [%]	Var Win vs. Heu
1	2.030	6.117	0.689	2.514	55.567	0.635
7	2.024	0.375	0.686	0.735	56.170	0.155
13	2.033	2.387	0.691	0.318	55.815	0.305
14	2.028	2.836	0.688	2.963	56.166	0.172

Tabella 7.2: Raccolta statistiche configurazioni migliori

un'unica uscita. In particolare, come target di riferimento, è stato usato il tasso di successo relativo ad $N_{next} = 3600$, ovvero è stata scelta una finestra temporale di 30 minuti.

[2]Non sono state riportate le statistiche di tutte le configurazioni perché presentano valori troppo alti di varianza, dunque sono stati scartati

7.2.1 Esperimento 1: rete FFNN con singola uscita, database crescenti

Il primo esperimento è stato effettuato usando i dati di tutti i canali Wi-Fi. L'idea generale dell'esperimento è la seguente: per ogni canale, vengono divisi i dati raccolti in 5 database, come descritto in Tab. 4.3 e 4.2. Dunque si hanno a disposizione 4 database usati per la fase di addestramento e un database fisso, usato per la fase di previsione. Il database di test sarà sempre fisso, in modo che i risultati delle previsioni siano comparabili. Dopo aver scelto la configurazione della rete, il modello viene addestrato 4 volte per ogni canale. Dopo ogni esecuzione, vengono dapprima raccolte le statistiche descritte nella Sezione 5.4, per poi aggiornare i database in ingresso da fornire alla rete per il prossimo addestramento, aggiungendo il prossimo gruppo in lista. Questo esperimento permette di verificare se la rete neurale trova sempre dei benefici nel ricevere nuovi dati da analizzare, oppure se si arriva ad una saturazione della rete neurale, non riuscendo così ad apprendere nuove relazioni.

7.2.2 Esperimento 2: rete FFNN con singola uscita, database di training misti

Questo esperimento ha come obiettivo quello di stabilire se la rete neurale riesce a migliorare le previsioni della qualità di un determinato canale analizzando le informazioni relative ad altri canali. L'esperimento è stato condotto seguendo questi step: prima di configurare la rete neurale, i database di training dei vari canali vengono suddivisi in diversi gruppi, come descritto in Tab. 4.3. Dopo questa suddivisione, viene configurata le rete neurale, e si sceglie il canale di riferimento e il target su cui il modello andrà a effettuare le previsioni. A questo punto, l'esperimento parte, eseguendo l'addestramento per un totale di sette volte. Nella prima esecuzione, vengono forniti in ingresso tutti e quattro i database di training relativi al canale scelto. Dopo ogni esecuzione, vengono raccolte le statistiche, e vengono aggiunti in ingresso due database di training relativi a canali diversi a quello scelto. Questo esperimento ha coinvolto tutti i canali Wi-Fi sotto esame, ma sono stati scelti come canali di riferimento solo il canale 13 e il canale 1, per gli stessi motivi descritti nella Sezione 7.1. Per chiarezza, i dati relativi ad altri

canali diversi da quello di riferimento si basano su un periodo temporale successivo ai database di testing e conformi con i database di training del canale scelto.

7.2.3 Esperimento 3: sei reti FFNN con singola uscita vs. una rete FFNN con sei uscite

Questo esperimento ha un duplice obiettivo: da una parte si vuole vedere come si comporta una rete neurale nella previsione del canale considerando diversi intervalli temporali; d'altra parte, si vuole sperimentare la capacità della rete neurale di prevedere il canale, avendo però molteplici target da analizzare, e per capire se la rete con più target risulta meno precisa di diverse reti che analizzano un singolo target. L'esperimento in questione è stato condotto come segue: il database di training è composto da tutti e quattro i gruppi considerati. Dunque ogni esperimento avrà a disposizione la stessa mole di dati durante la fase di addestramento. Viene creata dal programma una rete neurale feed-forward, con un solo neurone nello strato di output, che verrà addestrata ogni volta considerando un target diverso. I target considerati sono stati calcolati usando i seguenti numeri campioni: `N_next = 120, 240, 600, 1200, 3600 e 7200`, che fanno riferimento al tasso di successo relativo a 1, 2, 5, 10, 30 e 60 minuti, rispettivamente. Dopo aver completato la raccolta delle statistiche per tutti i target, si passa alla modellazione di una nuova rete neurale, inserendo in questo caso sei neuroni nello strato di output, ognuno dei quali riferiti ad un target differente. In questo scenario, il programma viene eseguito solo una volta, poiché la rete è in grado di prevedere l'andamento del canale per tutti i target citati. Per questo esperimento sono stati presi come riferimento i soli canali 13 e 1.

7.2.4 Esperimento 4: sei reti LSTM con singola uscita vs. una rete LSTM con sei uscite

Questo esperimento è concettualmente identico al precedente. Però in questo scenario viene analizzato il comportamento di una rete neurale di tipo LSTM. Dunque, oltre a confrontare le prestazioni tra 6 modelli a singola uscita e un modello a sei uscite, questa analisi permetterà di confrontare per la prima volta una rete neurale di tipo feed-forward con una rete LSTM, controllando così quali delle due reti

ottengono previsioni più accurate. Nella modellazione del modello, sono state apportate due modifiche molto importanti: il modo in cui lo shuffle dei database di training viene effettuato e la funzione di attivazione usata dalla rete. Mentre nella FFNN lo shuffle veniva effettuato sia sui dati di un singolo file sia nel modo in cui il modello caricava i file in memoria, lo shuffle nella LSTM viene gestito in questo modo: i file di ingresso vengono caricati in ordine, poi ogni record viene raggruppato in batch, ed infine l'ordine di questi batch viene reso casuale. In questo modo, ogni record all'interno del singolo batch è un record contiguo nel tempo rispetto agli altri, cercando di sfruttare le potenzialità della rete LSTM[3].

7.2.5 Esperimento 5: rete LSTM a memoria crescente

Questo esperimento è un pò diverso rispetto ai precedenti, e si basa sulle potenzialità che la rete LSTM può sfruttare. Come descritto nella Sezione 3.5.3, la peculiarità di una rete di tipo LSTM sta sull'uso della cella di stato, che salva le feature precedenti di una determinata sequenza. Nell'esperimento, il numero di celle di stato e dunque il numero di elementi di una sequenza che può essere memorizzato per volta è fisso ed uguale a tre. Infatti, come vedremo, non tutti i target hanno dei benefici ad usare lo stesso numero di celle. Per capire perché ciò accade, si pensi al seguente esempio: Se si considera una somma di due numeri, e si vuole che la rete neurale impari ad effettuare una somma, alla rete basta conoscere i due numeri e capire il significato del simbolo più. Se invece la stessa rete deve imparare ad effettuare la somma di dieci numeri, avrà bisogno di riconoscere una sequenza di dieci numeri seguiti dallo stesso simbolo. Quindi l'idea di base sarebbe la seguente: per riconoscere l'andamento di un canale in 1 minuto nel futuro, la rete LSTM avrà bisogno di pochi elementi nel passato per capire come esso si comporta. Se invece il periodo sotto esame è maggiore, ad esempio di un'ora, la rete avrà bisogno di sequenze più lunghe, e dunque di avere più celle di stato per ogni neurone. Seguendo questo ragionamento, l'esperimento è stato condotto come segue: usando la configurazione usata negli altri sperimenti, sono state create 6 reti neurali, ognuna delle quali sarà in grado di prevedere l'andamento di tutti i target. Ognuna di queste reti verrà addestrata sei volte, e ad ogni nuova esecuzione verrà

[3]In questo studio è stato analizzato anche il comportamento della rete LSTM senza effettuare lo shuffle, ma i risultati delle previsioni sono stati di scarsa accuratezza, quindi non sono stati riportati

cambiato il numero di sequenza dei dati in ingresso, con il numero di sequenza che varierà tra 1, 2, 3, 4, 5, 7 e 9. Questo esperimento verrà effettuato considerando soltanto il canale 13, poiché non si vuole confrontare la qualità dei vari canali, ma si vuole capire se la lunghezza di sequenza più essere un parametro fondamentale per adattare la previsione della rete neurale con periodi temporali più o meno lunghi.

7.3 Risultati

In questa sezione, verranno riportati ed analizzati i risultati relativi agli esperimenti descritti nelle sezioni precedenti. Per ogni esperimento, verranno specificate:

- i database di training usati nella fase di addestramento. Per quanto riguarda i database di testing, verranno usati sempre gli stessi database relativi al canale sotto esame, avendo così la possibilità di poter confrontare direttamente i risultati di tutti gli esperimenti.
- La tipologia di rete neurale che è stata scelta (FFNN o LSTM).
- La configurazione della rete neurale, ovvero i parametri analizzati in Tab. 7.1 e 7.2, andando a specificare di volta in volta altri parametri esterni usati dal modello.
- Tutti i canali coinvolti nell'esperimento.
- Tutti i target che la rete dovrà analizzare.

Esperimento 1

I parametri relativi alla rete neurale analizzata nel primo esperimento sono i seguenti:

- tasso di apprendimento iniziale = `0.01`
- funzione di discesa `quadratica` del tasso di apprendimento
- numero di elementi contenuti in un batch = `256`
- numero di strati nascosti della rete neurale = `1`
- numero di neuroni per strato nascosto = `128`

- numero di uscite analizzate dalla rete = `1`
- target di riferimento per l'esperimento = `3600` (30 minuti)
- canali considerati nella analisi = `1, 5, 9, 13`
- shuffle = `Vero`
- tipologia di rete analizzata = `feed-forward`
- funzione di attivazione = `ReLu`

Per quanto riguarda i database di training usati nelle varie esecuzioni, verrà preso come riferimento la suddivisione descritta nella Sezione 4.2, e verranno inseriti nella tabella come riferimento. In Tab. 7.3, 7.4, 7.5 e 7.6 sono rappresentate le statistiche raccolte relative al primo esperimento.

In queste tabelle, possiamo osservare due pattern diversi:

- Per quanto riguarda i canali 13 e 5, non si notano miglioramenti consistenti nei risultati. Questo andamento può dipendere dal fatto che i due canali presentano di base un comportamento abbastanza regolare e omogeneo durante tutto il periodo di acquisizione. Di conseguenza, dare alla rete neurale dei pattern già conosciuti in precedenza non consente alla rete di ottenere risultati migliori.
- Il comportamento è diverso per i canali 1 e 9, in quanto si nota un miglioramento maggiore delle previsioni del modello. Questo effetto può dipendere dalle irregolarità presenti nei due canali. Dunque la rete riceve dati abbastanza eterogenei.

Esperimento 2

I parametri relativi alla rete neurale analizzata nel secondo esperimento sono i seguenti:

- tasso di apprendimento iniziale = `0.01`
- funzione di discesa `quadratica` del tasso di apprendimento

Canale 13						
Target	DB_train	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs. Heuristic
30	ch13_1	2.276%	1.076	6.673%	5.049%	56.127%
	ch13_2	2.275%	1.418	7.404%	6.077%	55.752%
	ch13_3	2.286%	1.066	6.619%	5.065%	53.637%
	ch13_4	2.249%	1.049	6.632%	4.975%	55.645%

Tabella 7.3

Canale 1						
Target	DB_train	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs. Heuristic
30	ch1_1	2.276%	1.892	9.712%	5.823%	50.603%
	ch1_2	2.174%	1.785	9.292%	5.585%	54.581%
	ch1_3	2.220%	1.793	9.307%	5.653%	53.047%
	ch1_4	2.184%	1.726	9.049%	5.427%	52.754%

Tabella 7.4

Canale 5						
Target	DB_train	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
30	ch5_1	1.377%	0.573	3.726%	2.850%	57.415%
	ch5_2	1.370%	0.573	3.721%	2.851%	57.499%
	ch5_3	1.395%	0.565	3.663%	2.810%	55.570%
	ch5_4	1.382%	0.579	3.714%	2.844%	56.967%

Tabella 7.5

Canale 9						
Target	DB_train	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
30	ch9_1	6.234%	11.997	27.928%	19.346%	50.727%
	ch9_2	6.093%	11.827	28.095%	19.335%	53.449%
	ch9_3	5.995%	11.975	28.683%	19.043%	54.717%
	ch9_4	6.004%	11.742	28.358%	18.959%	55.462%

Tabella 7.6

- numero di elementi contenuti in un batch = `256`
- numero di strati nascosti della rete neurale = `1`

- numero di neuroni per strato nascosto = `128`
- numero di uscite analizzate dalla rete = `1`
- target di riferimento per l'esperimento = `3600` (30 minuti)
- canali canali coinvolti sia nella fase di addestramento che nella fase di previsione = `1, 13`
- canali coinvolti solo nella fase di addestramento nella analisi = `1, 5, 9, 13`
- shuffle = `Vero`
- tipologia di rete analizzata = `feed-forward`
- funzione di attivazione = `ReLu`

Per quanto riguarda i database di training usati nelle varie esecuzioni, verrà preso come riferimento la suddivisione descritta nella Sezione 4.2, e verranno specificati ad ogni riga della tabella relativa ai risultati di questo esperimento. In questo caso, verranno inseriti anche database di training relativi ad altri canali, come descritto nella Sezione 7.2.2. In Tab. 7.7 e 7.8 sono rappresentate le statistiche raccolte relative al secondo esperimento.

Canale 13					
DB_train	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
ch13_4	2.249%	1.049	6.632%	4.975%	55.645%
ch13_4+ch1_2	2.249%	1.043	6.575%	4.957%	55.583%
ch13_4+ch1_4	2.276%	1.053	6.547%	4.978%	54.416%
ch13_4+ch1_4+ch5_2	2.259%	1.039	6.536%	4.976%	54.413%
ch13_4+ch1_4+ch5_4	2.287%	1.055	6.613%	5.056%	53.211%
ch13_4+ch1_4+ch5_4+ch9_2	2.274%	1.064	6.576%	4.982%	55.282%
ch13_4+ch1_4+ch5_4+ch9_4	2.261%	1.057	6.594%	5.000%	55.098%

Tabella 7.7

Canale 1					
DB_train	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
ch1_4	2.184%	1.726	9.049%	5.427%	52.754%
ch1_4+ch13_2	2.143%	1.652	8.844%	5.394%	55.476%
ch1_4+ch13_4	2.173%	1.698	9.122%	5.560%	54.182%
ch1_4+ch13_4+ch5_2	2.165%	1.733	9.159%	5.457%	54.808%
ch1_4+ch13_4+ch5_4	2.143%	1.734	9.215%	5.477%	55.765%
ch1_4+ch13_4+ch5_4+ch9_2	2.156%	1.740	9.043%	5.373%	55.583%
ch1_4+ch13_4+ch5_4+ch9_4	2.233%	1.870	9.168%	5.453%	53.225%

Tabella 7.8

Anche in questo caso possiamo individuare due pattern differenti:

- nel canale 13 non si notano miglioramenti nella previsione del tasso di successo, tranne quando vengono aggiunte i database di training relativi al canale 5, e si notano dei piccoli peggioramenti quando viene aggiunto il canale 9. Essendo il canale 13 molto regolare ma con tassi di successo non molto elevati, il contributo fornito dai dati relativi al canale 5 ha aiutato il modello nella previsione, poiché presenta dei tassi di successo che il canale 13 non riesce a raggiungere. Al contrario, la forte irregolarità del canale 9 può essere la cause del peggioramento subito dal canale 13.

- Il canale 1 presenta un leggero miglioramento delle previsioni quando vengono aggiunti i dati relativi al canale 13, una situazione stazionaria quando il canale 5 viene aggiunto, e un lieve peggioramento quando è il canale 9 ad essere aggiunto nel database di training. Questo comportamento rispecchia un andamento diverso rispetto al canale 13, ma per la seguente motivazione: il canale 1, anche se irregolare, presenta molti periodi temporali in cui il canale risulta libero. Di conseguenza, i dati relativi al canale 5 non danno

informazioni diverse rispetto a quelle che il canale 1 ha già. Al contrario, il canale 13 ha molte più informazioni in cui lo stesso risulta occupato solo in parte.

Esperimento 3

I parametri relativi alla rete neurale analizzata nel terzo esperimento sono i seguenti:

- tasso di apprendimento iniziale = `0.01`
- funzione di discesa `quadratica` del tasso di apprendimento
- numero di elementi contenuti in un batch = `256`
- numero di strati nascosti della rete neurale = `1`
- numero di neuroni per strato nascosto = `128`
- numero di uscite analizzate dalla rete = `1 per la prima configurazione, 6 per la seconda`
- target di riferimento per l'esperimento = `120, 240, 600, 1200, 3600, 7200` (rispettivamente 1, 2, 5, 10, 30 e 60 minuti)
- database di training considerato = chx_4. Tutti i database di un determinato canale vengono considerati.
- canali considerati nella analisi = `1, 13`
- shuffle = `Vero`
- tipologia di rete analizzata = `feed-forward`
- funzione di attivazione = `ReLu`

In Tab. 7.9 e 7.11 sono rappresentate le previsioni effettuate da sei reti diverse, ognuna con un target di riferimento differente. In Tab. 7.10 e 7.12 invece, sono rappresentate le statistiche relative alla singola rete neurale, che analizza in uscita sei target differenti. Prima di confrontare le diverse architetture, si può notare che i target corrispondenti a 5 e a 10 minuti sono quelli in cui le previsioni risultano più accurate rispetto agli altri target analizzati in entrambi i modelli. Questo è un

ottimo risultato in prospettiva di nuovi studi relativi all'argomento, visto e considerato che sapere la qualità del canale in tempi relativamente piccoli dà la possibilità di sviluppare delle tecniche più mirate per regolare il flusso di informazioni in una rete industriale. Per quanto riguarda il confronto tra le due architetture, si può notare come la seconda abbia risultati comparabili con la prima, con addirittura dei piccoli miglioramenti osservati nel canale 1. Anche questo dato è importante, sopratutto perché la singola rete con 6 uscite è riuscita ad ottenere i risultati in molto meno tempo rispetto a sei reti con singola uscita, effettuando l'addestramento in un sesto del tempo. In aggiunta la rete con 6 uscite ha un'occupazione molto inferiore rispetto a 6 reti separate. Questo la rende più adatta ad essere integrata in dispositivi con ridotte capacità computazionali come spesso sono quelli in cui è installato un sistema di comunicazione wireless. Come studi futuri, si potrebbe pensare di analizzare direttamente un'architettura seguendo il secondo modello proposto.

Canale 13					
Target	MAE	MSE $[\cdot 10^{-3}]$	95%	90%	% Win vs.Heuristic
1	3.666%	2.159	9.128%	7.579%	53.656%
2	2.933%	1.418	7.404%	6.077%	54.733%
5	2.410%	1.023	6.334%	5.021%	56.962%
10	2.293%	0.998	6.403%	4.868%	59.364%
30	2.244%	1.042	6.607%	4.950%	55.946%
60	2.300%	1.075	6.855%	5.100%	53.582%

Tabella 7.9

Canale 13					
Target	MAE	MSE $[\cdot 10^{-3}]$	95%	90%	% Win vs.Heuristic
1	3.669%	2.163	9.128%	7.587%	53.643%
2	2.938%	1.423	7.400%	6.080%	54.670%
5	2.416%	1.032	6.354%	5.034%	56.830%
10	2.294%	1.005	6.426%	4.879%	59.386%
30	2.252%	1.045	6.603%	4.993%	55.334%
60	2.307%	1.082	6.859%	5.133%	52.958%

Tabella 7.10

Canale 1					
Target	MAE	MSE $[\cdot 10^{-3}]$	95%	90%	% Win vs.Heuristic
1	2.553%	1.397	7.235%	5.263%	54.271%
2	2.156%	1.141	6.531%	4.460%	55.778%
5	1.969%	1.164	6.692%	4.180%	57.240%
10	2.097%	1.430	7.933%	4.685%	56.370%
30	2.247%	1.841	9.414%	5.578%	51.774%
60	2.383%	2.018	9.713%	6.280%	52.000%

Tabella 7.11

Canale 1					
Target	MAE	MSE $[\cdot 10^{-3}]$	95%	90%	% Win vs.Heuristic
1	2.597%	1.452	7.350%	5.346%	53.365%
2	2.187%	1.185	6.629%	4.520%	54.925%
5	1.999%	1.197	6.821%	4.21%	56.599%
10	2.036%	1.417	7.834%	4.586%	58.964%
30	2.182%	1.824	9.329%	5.477%	54.896%
60	2.312%	2.092	10.277%	6.013%	54.436%

Tabella 7.12

Esperimento 4

I parametri relativi alla rete neurale analizzata nel quarto esperimento sono i seguenti:

- tasso di apprendimento iniziale = `0.01`
- funzione di discesa `quadratica` del tasso di apprendimento
- numero di elementi contenuti in un batch = `256`
- numero di strati nascosti della rete neurale = `1`
- numero di neuroni per strato nascosto = `128`
- numero di uscite analizzate dalla rete = `1 per la prima configurazione, 6 per la seconda`
- target di riferimento per l'esperimento = `120, 240, 600, 1200, 3600, 7200` (rispettivamente 1, 2, 5, 10, 30 e 60 minuti)
- canali considerati nella analisi = `13`
- numero di celle di stato per neurone = `6`
- shuffle = `Parziale`, file caricati in ordine, singolo batch temporalmente ordinato ma forniti alla rete casualmente
- tipologia di rete analizzata = `LSTM`
- funzione di attivazione = `sigmoide`

In Tab. 7.14 e 7.13 sono rappresentate le statistiche relative a sei reti a singola uscita e ad una singola rete con sei uscite, rispettivamente. In questo caso, il secondo modello risulta comparabile con il primo, tranne nella previsione dei target di 60 e 30 minuti, che risulta leggermente meno accurata. Un dato però negativo, deriva da un altro fattore: rispetto alla rete feed-forward, l'architettura LSTM risulta molto meno accurata in tutte le previsioni effettuate. Può dipendere da diversi fattori, come il numero di celle di stato che può risultare inadeguato per alcuni target presi come riferimento, oppure dalla funzione di attivazione utilizzata, oppure da un numero di batch che risulta troppo basso per determinare i legami temporali tra features differenti.

Canale 13						
Target	metodo	MAE	MSE $[\cdot 10^{-3}]$	95%	90%	% Win vs.Heuristic
1	LSTM	3.831%	2.414	9.591%	7.915%	50.434%
2	LSTM	3.108%	1.642	7.943%	6.458%	51.178%
5	LSTM	2.574%	1.201	6.804%	5.348%	55.230%
10	LSTM	2.470%	1.256	6.950%	5.197%	56.309%
30	LSTM	2.391%	1.154	6.930%	5.251%	50.826%
60	LSTM	2.477%	1.180	7.035%	5.342%	48.748%

Tabella 7.13

Canale 13						
Target	metodo	MAE	MSE $[\cdot 10^{-3}]$	95%	90%	% Win vs.Heuristic
1	LSTM	3.825%	2.391	9.585%	7.917%	50.510%
2	LSTM	3.105%	1.635	7.940%	6.453%	51.117%
5	LSTM	2.596%	1.226	6.845%	5.407%	52.748%
10	LSTM	2.474%	1.179	6.877%	5.167%	54.596%
30	LSTM	2.481%	1.175	6.735%	5.159%	48.446%
60	LSTM	2.616%	1.236	6.992%	5.332%	45.897%

Tabella 7.14

Esperimento 5

I parametri relativi alla rete neurale analizzata nel primo esperimento sono i seguenti:

- tasso di apprendimento iniziale = `0.01`
- funzione di discesa `quadratica` del tasso di apprendimento
- numero di elementi contenuti in un batch = `256`
- numero di strati nascosti della rete neurale = `1`
- numero di neuroni per strato nascosto = `64`
- numero di uscite analizzate dalla rete = `1 per il primo riferimento, 6 per il secondo`
- target di riferimento per l'esperimento = `120, 240, 600, 1200, 3600, 7200` (rispettivamente 1, 2, 5, 10, 30 e 60 minuti)

- canali considerati nella analisi = `13`
- shuffle = `Parziale`, file caricati in ordine, singolo batch temporalmente ordinato ma forniti alla rete casualmente
- tipologia di rete analizzata = `LSTM`
- funzione di attivazione = `sigmoide`

In questo caso, per ogni esperimento, verrà indicato il numero di sequenza utilizzato per ogni run del modello. I numeri di celle di stato scelto per ogni esperimento sono 1, 2, 3, 4, 5, 7, 9.

Si possono osservare tre comportamenti distinti: per quanto riguarda le previsioni relative alle finestre temporali di 1 e 2 minuti, esse ottengono risultati migliori quando vengono utilizzate un numero di celle pari a 1 e pari a 2. Man mano che questo parametro viene aumentato, i risultati ottenuti dalla rete peggiorando lievemente, fino ad ottenere risultati peggiori rispetto al metodo euristico di confronto. Per quanto riguarda i target fissati a 5 e a 10 minuti, il comportamento è analogo, ma il peggioramento delle statistiche si osserva quando il numero di celle di stato sale a 5. Infine, per i canali relativi a 30 e a 60 minuti, il comportamento delle previsioni sembra stazionario, cioè non si osservano miglioramenti o peggioramenti delle statistiche raccolte. Anche se i risultati analizzati non sembrano promettenti, l'andamento delle varie previsioni risulta concorde con l'ipotesi effettuata prima dell'esperimento.

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
1	1	3.756%	2.311	9.383%	7.760%	51.221%
	2	3.053%	1.573	7.714%	6.317%	51.220%
	5	2.583%	1.189	6.671%	5.374%	51.525%
	10	2.510%	1.162	6.642%	5.208%	51.472%
	30	2.571%	1.188	6.549%	5.164%	45.759%
	60	2.724%	1.242	6.750%	5.347%	43.585%

Tabella 7.15

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
2	1	3.751%	2.276	9.343%	7.746%	52.068%
	2	3.031%	1.530	7.659%	6.275%	52.470%
	5	2.529%	1.136	6.571%	5.256%	53.899%
	10	2.434%	1.109	6.567%	5.063%	54.716%
	30	2.499%	1.172	6.704%	5.172%	47.656%
	60	2.629%	1.243	7.026%	5.388%	45.955%

Tabella 7.16

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
3	1	3.782%	2.354	9.456%	7.825%	51.442%
	2	3.063%	1.577	7.777%	6.361%	51.653%
	5	2.547%	1.174	6.681%	5.294%	53.815%
	10	2.465%	1.143	6.691%	5.134%	54.839%
	30	2.519%	1.181	6.680%	5.162%	47.537%
	60	2.664%	1.243	6.854%	5.325%	45.140%

Tabella 7.17

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
4	1	3.806%	2.389	9.507%	7.867%	51.157%
	2	3.084%	1.629	7.852%	6.388%	51.940%
	5	2.572%	1.213	6.739%	5.342%	53.720%
	10	2.457%	1.158	6.705%	5.117%	54.652%
	30	2.484%	1.161	6.634%	5.103%	47.940%
	60	2.624%	1.227	6.871%	5.288%	45.144%

Tabella 7.18

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
5	1	3.796%	2.347	9.468%	7.845%	51.300%
	2	3.076%	1.593	7.806%	6.377%	51.828%
	5	2.572%	1.188	6.707%	5.339%	53.278%
	10	2.472%	1.149	6.689%	5.140%	54.347%
	30	2.543%	1.192	6.653%	5.184%	47.277%
	60	2.718%	1.285	6.928%	5.421%	44.492%

Tabella 7.19

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
7	1	3.866%	2.474	9.702%	7.993%	49.688%
	2	3.147%	1.711	8.075%	6.534%	50.290%
	5	2.629%	1.288	6.986%	5.472%	52.259%
	10	2.493%	1.218	6.982%	5.227%	53.914%
	30	2.467%	1.176	6.769%	5.150%	48.556%
	60	2.593%	1.227	7.000%	5.292%	45.787%

Tabella 7.20

Canale 13						
# celle di stato	Target	MAE	MSE [$\cdot 10^{-3}$]	95%	90%	% Win vs.Heuristic
9	1	3.903%	2.530	9.827%	8.085%	49.173%
	2	3.185%	1.764	8.235%	6.622%	49.629%
	5	2.663%	1.332	7.188%	5.564%	51.404%
	10	2.516%	1.250	7.131%	5.313%	53.248%
	30	2.468%	1.181	6.792%	5.155%	49.096%
	60	2.600%	1.230	7.011%	5.268%	45.888%

Tabella 7.21

Capitolo 8

Conclusioni

A partire dai risultati ottenuti nel corso di questo studio, si può affermare che per effettuare una previsione accurata della qualità di una comunicazione tra dispositivi connessi tramite una rete Wi-Fi, uno strumento come le ANN può rivelarsi di grande utilità. Anche se la ricerca in questo senso non è ancora matura, sono emersi risultati molto promettenti con l'uso di reti neurali di tipo feed-forward, in cui l'errore commesso su tutti gli esperimenti effettuati risulta essere minore rispetto all'uso di una tecnica ragionevole, basata su media mobile, con cui sono stati comparati i risultati ottenuti. Un'altra considerazione che emerge è relativa all'uso di una singola rete neurale, in grado di analizzare contemporaneamente fino a 6 finestre temporali differenti, ottenendo non solo risultati migliori rispetto al metodo basato su media mobile, ma addirittura comparabili con l'implementazione di 6 reti distinte. Questo era un risultato inaspettato prima dello studio, che potrà essere usato in futuro per realizzare modelli previsionali con un occupazione di memoria ridotta, quindi adatti ad essere utilizzati in dispositivi con ridotte capacità computazionali. Un'ulteriore analisi sperimentale è stata effettuata usando un'architettura basata sulla rete LSTM. In questo caso, i risultati ottenuti sono in generale meno accurati rispetto a quelli ottenuti dalla rete feed-forward, anche se in generale migliori rispetto a quelli relativi al metodo basato su media mobile. Anche se i risultati non sembrano incoraggianti, l'ultimo esperimento effettuato ha fornito dei risultati interessanti, poiché si evince una correlazione tra numero di celle di stato usate e la finestra temporale su cui viene calcolato il target di previsione. La correlazione in questione non è così marcata, ma questo risultato potrà essere un punto di partenza per nuove ricerche. Una diversa composizione della finestra

mobile, usata in ingresso nella fase di addestramento della rete neurale, potrebbe anch'esso essere un punto di partenza per un possibile studio futuro. Altre attività di ricerca futura includono l'uso di parametri della rete che non sono stati considerati in precedenza (ad es., qualità del segnale o latenze), o ancora l'analisi di topologie non ancora valutate.

Bibliografia

[1] S. Szott, K. Kosek-Szott, P. Gawłowicz, J. T. Gómez, B. Bellalta, A. Zubow, and F. Dressler, "Wi-Fi Meets ML: A Survey on Improving IEEE 802.11 Performance With Machine Learning," *IEEE Communications Surveys Tutorials*, vol. 24, no. 3, pp. 1843–1893, 2022.

[2] S. Scanzio, F. Xia, G. Cena, and A. Valenzano, "Predicting Wi-Fi link quality through artificial neural networks," *Internet Technology Letters*, vol. 5, no. 2, p. e326, 2022.

[3] S. Scanzio, G. Cena, C. Zunino, and A. Valenzano, "Machine Learning to Support Self-Configuration of Industrial Systems Interconnected over Wi-Fi," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2022.

[4] S. Scanzio, L. Wisniewski, and P. Gaj, "Heterogeneous and dependable networks in industry – A survey," *Computers in Industry*, vol. 125, p. 103388, 2021.

[5] S. Kharb and A. Singhrova, "A survey on network formation and scheduling algorithms for time slotted channel hopping in industrial networks," *J. Netw. Comput.*, pp. 59–87, 2019. http://dx.doi.org/10.1016/j.jnca.2018.11.004.

[6] M. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel, "Internet of Things Challenges and Opportunities," *Springer International Publishing*, p. 111–141, 2014. http://dx.doi.org/10.1007/978-3-319-04223-75.

[7] Q. Wang, K. Jaffrès-Runser, Y. Xu, J. Scharbarg, Z. An, and C. Fraboul, "TDMA versus CSMA/CA for wireless multihop communications: a stochastic

worst-case delay analysis," *IEEE Trans. Ind. Inform*, p. 877–887, 2017. http://dx.doi.org/10.1109/TII.

[8] S. Scanzio, M. Ghazi Vakili, G. Cena, C. Demartini, B. Montrucchio, A. Valenzano, and C. Zunino, "Wireless sensor networks and TSCH: a compromise between reliability, power consumption, and latency," *IEEE Access*, p. 167042–167058, 2020. http://dx.doi.org/10.1109/ACCESS.2020.3022434.

[9] G. Cena, C. G. Demartini, M. Ghazi Vakili, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluating and Modeling IEEE 802.15.4 TSCH Resilience against Wi-Fi Interference in New-Generation Highly-Dependable Wireless Sensor Networks," *Ad Hoc Networks*, vol. 106, p. 102199, 2020.

[10] A. Karaagac, I. Moerman, and J. Hoebeke, "Hybrid schedule management in 6TiSCH networks: the coexistence of determinism and flexibility," *IEEE Access*, p. 33941–33952, 2018. http://dx.doi.org/10.1109/ACCESS.2018.2849090.

[11] V. Kotsiou, G. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LABeL: linkbased adaptive BLacklisting technique for 6TiSCH wireless industrial networks," *In: ACM Int. Conf. on Modelling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM'17*, p. 25–33, 2017. http://dx.doi.org/10.

[12] P. Gomes, T. Watteyne, and B. Krishnamachari, "MABO-TSCH: multihop and blacklist-based optimized time synchronized channel hopping," *Trans. Emerg. Telecommun. Technol. 29 (7)*, 2018. http://dx.doi.org/10.1002/ett.3223.

[13] X. Vilajosana, T. Watteyne, M. Vucinic, T. Chang, Pister, and K.S.J., "6TiSCH: industrial performance for IPv6 internet-of-things networks," *Proc. IEEE 107 (6)*, p. 1153–1165, 2019. http://dx.doi.org/10.1109/JPROC.2019.2906404.

[14] O. Iova, F. Theoleyre, and T. Noel, "Using multiparent routing in RPL to increase the stability and the lifetime of the network," *Ad Hoc Netw.*, p. 45–62, 2015. http://dx.doi.org/10.1016/j.adhoc.2015.01.020.

[15] S. Scanzio, G. Cena, and A. Valenzano, "Enhanced Energy-Saving Mechanisms in TSCH Networks for the IIoT: The PRIL Approach," *IEEE Transactions on Industrial Informatics*, pp. 1–11, 2022.

[16] G. Cena, S. Scanzio, and A. Valenzano, "Ultra-Low Power Wireless Sensor Networks Based on Time Slotted Channel Hopping with Probabilistic Blacklisting," *Electronics*, vol. 11, no. 3, 2022.

[17] S. Scanzio, G. Cena, A. Valenzano, and C. Zunino, "Energy Saving in TSCH Networks by Means of Proactive Reduction of Idle Listening," in *Ad-Hoc, Mobile, and Wireless Networks* (L. A. Grieco, G. Boggia, G. Piro, Y. Jararweh, and C. Campolo, eds.), (Cham), pp. 131–144, Springer International Publishing, 2020.

[18] S. Scanzio, G. Cena, L. Seno, and A. Valenzano, "Robustness and Optimization of PRIL Techniques for Energy Saving in TSCH Networks," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2021.

[19] Scanzio, Stefano and Bitondo, Federico and Cena, Gianluca and Valenzano, Adriano, "CONSIP: Consistency Protocol for Hopping Function Exchange and Black listing in TSCH," in *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, pp. 1–8, 2022.

[20] B. Pavkovic, F. Theoleyre, and A. Duda, "Multipath opportunistic RPL routing over IEEE 802.15.4," *In: ACM Int. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM*, p. 179–186, 2011.

[21] R. Koutsiamanis, G. Papadopoulos, X. Fafoutis, Fiore, J.M.D., P. Thubert, and N. Montavont, "From best effort to deterministic packet delivery for wireless industrial IoT networks," *IEEE Trans. Ind. Inform. 14 (10)*, p. 4468–4480, 2018. http://dx.doi.org/10.1109/TII.2018.2856884.

[22] R. Karmakar, S. Chattopadhyay, and S. Chakraborty, "Impact of IEEE 802.11n/ac PHY/MAC high throughput enhancements on transport and application protocols – a survey," *IEEE Commun. Surv. Tutor. 19 (4)*, p. 2050–2091, 2017. http://dx.doi.org/10.1109/COMST.2017.2745052.

[23] M. Lucas-Estan, J. Maestre, B. Coll-Perales, J. Gozalvez, and I. Lluvia, "An experimental evaluation of redundancy in industrial wireless communications," *IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, vol. vol 1, p. 1075–1078, 2018. http://dx.doi.org/10.1109/ETFA.2018.8502497.

[24] F. Tramarin, S. Vitturi, and M. Luvisotto, "A dynamic rate selection algorithm for IEEE 802.11 industrial wireless LAN," *IEEE Trans. Ind. Inform.*, vol. 13 (2), p. 846–855, 2017. http://dx.doi.org/10.1109/TII.2016.2616327.

[25] M. Rentschler and P. Laukemann, "Towards a reliable parallel redundant WLAN black channel," *IEEE Int. Workshop on Factory Communication Systems (WFCS)*, p. 255–264, 2012. http://dx.doi.org/10.1109/WFCS.2012.6242573.

[26] G. Cena, S. Scanzio, and A. Valenzano, "Seamless link-level redundancy to improve reliability of industrial Wi-Fi networks," *IEEE Trans. Ind. Inform.*, vol. 12 (2), p. 608–620, 2016. http://dx.doi.org/10.1109/TII.2016.2522768.

[27] G. Cena, S. Scanzio, and A. Valenzano, "Improving effectiveness of seamless redundancy in real industrial Wi-Fi networks," *IEEE Trans. Ind. Inform.*, vol. 14 (5), p. 2095–2107, 2018a. http://dx.doi.org/10.1109/TII.2017.2759788.

[28] G. Cena, S. Scanzio, and A. Valenzano, "A Prototype Implementation of Wi-Fi Seamless Redundancy with Reactive Duplication Avoidance," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 179–186, 2018.

[29] G. Cena, L. Seno, A. Valenzano, and C. Zunino, "On the performance of IEEE 802.11e wireless infrastructures for soft-real-time industrial applications," *IEEE Trans. Ind. Inform.*, vol. 6 (3), p. 425–437, 2010. http://dx.doi.org/10.1109/TII.2010.2052058.

[30] V. Sevani, B. Raman, and P. Joshi, "Implementation-based evaluation of a full-fledged multihop TDMA-MAC for WiFi mesh networks," *IEEE Trans. Mobile Comput.*, vol. 13 (2), p. 392–406, 2014. http://dx.doi.org/10.1109/TMC.2012.251.

[31] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Implementation and evaluation of the reference broadcast infrastructure synchronization protocol," *IEEE Trans. Ind. Inform.*, vol. 11 (3), p. 801–811, 2015. http://dx.doi.org/10.1109/TII.2015.2396003.

[32] A. Mahmood, R. Exel, H. Trsek, and T. Sauter, "Clock synchronization over IEEE 802.11 – a survey of methodologies and protocols," *IEEE Trans. Ind. Inform.*, vol. 13 (2), p. 907–922, 2017. http://dx.doi.org/10.1109/TII.2016.2629669.

[33] S. Pollin, I. Tan, B. Hodge, C. Chun, and A. Bahai, "Harmful coexistence between 802.15.4 and 802.11: a measurement-based study," *Int. Conf. on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom)*, p. 1–6, 2008. http://dx.doi.org/10.1109/CROWNCOM.2008.4562460.

[34] L. Seno, G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Enhancing communication determinism in Wi-Fi networks for soft real-time industrial applications," *IEEE Trans. Ind. Inform.*, vol. 13 (2), p. 866–876, 2017. http://dx.doi.org/10.1109/TII.2016.2641468.

[35] A. Ghosh, A. Maeder, M. Baker, and D. Chandramouli, "5G evolution: a view on 5G cellular technology beyond 3GPP release 15," *IEEE Access*, vol. 7, p. 127639–127651, 2019. http://dx.doi.org/10.1109/ACCESS.2019.2939938.

[36] J. Walia, H. Hämmäinen, K. Kilkki, and S. Yrjölä, "5G network slicing strategies for a smart factory," *Comput. Ind.*, vol. 111, p. 108–120, 2019. http://dx.doi.org/10.1016/j.compind.2019.07.006.

[37] A. Neumann, L. Wisniewski, R. Ganesan, P. Rost, and J. Jasperneite, "Towards integration of industrial ethernet with 5G mobile networks," *IEEE Int. Workshop on Factory Communication Systems (WFCS)*, p. 1–4, 2018. http://dx.doi.org/10.1109/WFCS.2018.84023736.

[38] A. Neumann, L. Wisniewski, and P. Rost, "About integrating 5G into profinet as a switch function," *In: KommA 2019 – Jahreskolloquium Kommunikation in der Automation)*, 2019. .

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," vol. 60, no. 6, 2017.

[40] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," 2018.

[41] S. Scanzio, S. Cumani, R. Gemello, F. Mana, and P. Laface, "Parallel Implementation of Artificial Neural Network Training for Speech Recognition," *Pattern Recogn. Lett.*, vol. 31, p. 1302–1309, Aug. 2010.

[42] M. Mongelli and S. Scanzio, "A neural approach to synchronization in wireless networks with heterogeneous sources of noise," *Ad Hoc Networks*, vol. 49, pp. 1–16, 2016. https://www.sciencedirect.com/science/article/pii/S1570870516301366.

[43] S. Scanzio, P. Laface, L. Fissore, R. Gemello, and F. Mana, "On the Use of a Multilingual Neural Network Front-End," *Conference of the International Speech Communication Association (INTERSPEECH 2008)*, pp. 2711–2714, September 2008.

[44] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011. https://proceedings.mlr.press/v15/glorot11a.html.

[45] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series*, vol. 1168, p. 022022, feb 2019. https://doi.org/10.1088/1742-6596/1168/2/022022.

[46] G. Cena, S. Scanzio, and A. Valenzano, "A software-defined MAC architecture for Wi-Fi operating in user space on conventional PCs," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, pp. 1–10, 2017.

[47] G. Cena, S. Scanzio, and A. Valenzano, "SDMAC: A Software-Defined MAC for Wi-Fi to Ease Implementation of Soft Real-Time Applications," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3143–3154, 2019.

[48] F. Tramarin, S. Vitturi, and M. Luvisotto, "A Dynamic Rate Selection Algorithm for IEEE 802.11 Industrial Wireless LAN," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 846–855, 2017.

[49] D. Xia, J. Hart, and Q. Fu, "Evaluation of the Minstrel rate adaptation algorithm in IEEE 802.11g WLANs," in *2013 IEEE International Conference on Communications (ICC)*, pp. 2223–2228, 2013.

[50] A. K. Gizzini, M. Chafii, A. Nimr, and G. Fettweis, "Deep Learning Based Channel Estimation Schemes for IEEE 802.11p Standard," *IEEE Access*, vol. 8, pp. 113751–113765, 2020.

[51] W. Jiang and H. D. Schotten, "Neural Network-Based Fading Channel Prediction: A Comprehensive Overview," *IEEE Access*, vol. 7, pp. 118112–118124, 2019.