UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA MAGISTRALE
IN
INFORMATICA

Identificazione di attacchi al CAN-bus mediante features temporali e tecniche di Machine-Learning in ambito Automotive

RELATORI:

Prof. Arcangelo Castiglione

Prof. Gianni D'Angelo

CANDIDATO:

Abagnale Giuseppe

Matr. 0522500547

ANNO ACCADEMICO 2018/2019

“Sono convinto che l'informatica abbia molto in comune con la fisica. Entrambe si occupano di come funziona il mondo a un livello abbastanza fondamentale. La differenza, naturalmente, è che mentre in fisica devi capire come è fatto il mondo, in informatica sei tu a crearlo. Dentro i confini del computer, sei tu il creatore. Controlli, almeno potenzialmente, tutto ciò che vi succede. Se sei abbastanza bravo, puoi essere un dio. Su piccola scala.”

Linus Torvalds

Sommario

Indice delle figure

Introduzione

Al giorno d'oggi le automobili sono i mezzi di trasporto più utilizzati dalla popolazione mondiale, nel corso degli anni è stata oggetto di profonde modifiche e miglioramenti. Dagli anni '70, c'è stato un crescente interesse per la sostituzione di componenti idrauliche o puramente meccaniche con sistemi elettronici alternativi, infatti le automobili moderne sono una rete sempre più complessa di sistemi informatici.
Partendo dai primi sistemi di iniezione del carburante e proseguendo verso i primi sistemi di antibloccaggio delle ruote in frenata (ABS), l'automobile si è dotata in maniera sempre più evidente di computer predisposti alla gestione di ogni singolo aspetto, con il risultato che le più importanti funzioni veicolari sono dotate di centraline elettroniche di controllo (ECU), connesse tra loro tramite una o più reti interne dove al centro di questa complessità c'è il Controller Area Network o CAN-bus.
Il CAN-bus è il sistema nervoso centrale di un veicolo moderno sul quale avviene la maggior parte della comunicazione intra-veicolare.

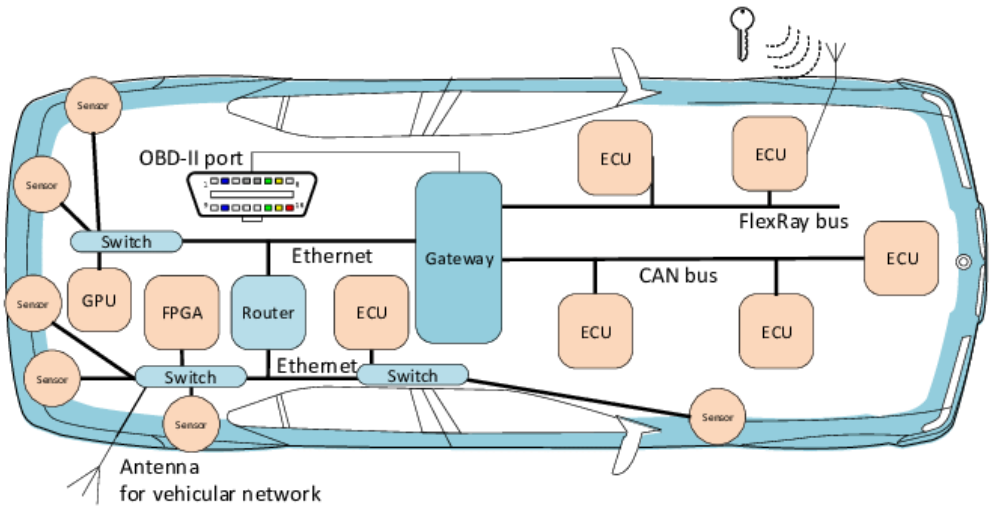

Figura 1 - Esempio di architettura hardware di un'automobile moderna

Seppur riconosciute e statisticamente rilevanti le migliorie apportate da tali sistemi, l'aggiunta sempre più ingente di computer per la gestione di aspetti anche critici del veicolo ha inevitabilmente portato con sé problemi di

natura di sicurezza informatica, problemi ignoti al mondo dell'ICT ma inediti nel mondo dell'automobilismo.

La frenetica evoluzione del settore automobilistico, la richiesta sempre più ingente di nuovi prodotti sul mercato, l'aggiunta sempre più preponderante sulle centraline della vettura di interfacce comunicanti con il mondo esterno e un mercato sempre più competitivo hanno portato negli ultimi anni alla comparsa di vulnerabilità informatiche sui veicoli sempre più concretamente fattibili da poter essere sfruttate da un eventuale aggressore.

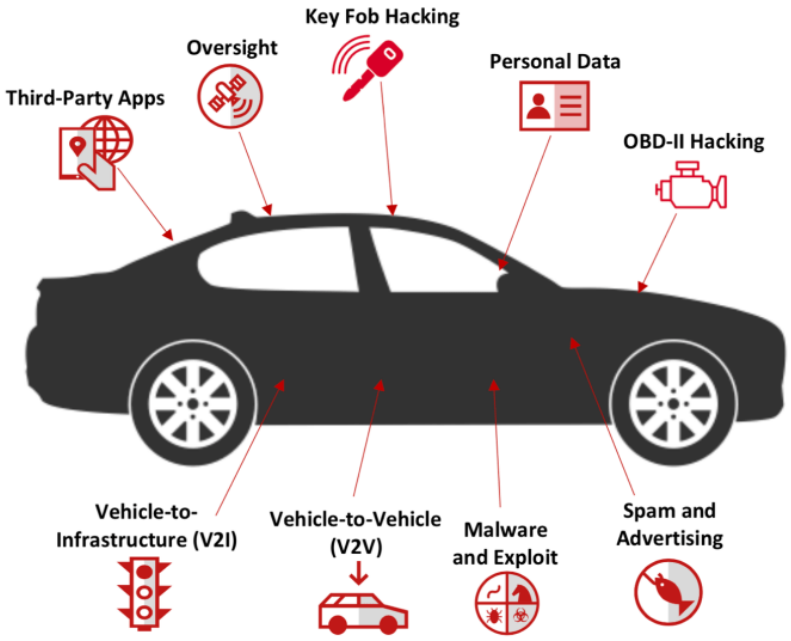

Figura 2 - L'auto moderna e le sue vulnerabilità

Vulnerabilità che possono portare alla capacità di controllare in maniera arbitraria e, in alcuni casi, persino totalmente da remoto, sistemi critici della vettura quali acceleratore, sterzo e freni, ponendo guidatore e passeggeri in potenziali pericoli.

La maggior parte degli attacchi informatici consiste, tramite connessione fisica alla vettura o tramite compromissione da remoto di una centralina già presente a bordo dell'auto, nella contraffazione ed invio indiscriminato di messaggi (frame) all'interno della rete CAN dell'automobile con lo scopo di compromettere in maniera vantaggiosa per l'avversario il funzionamento del veicolo.

A causa della natura di tali attacchi e della topologia bus della rete CAN, attualmente il modo più efficiente per rendere sicura a sufficienza un'auto da attacchi informatici consiste nell'installazione di un sistema di individuazione o di prevenzione delle intrusioni (IDS/IPS), che monitora ogni messaggio circolante sulla rete della macchina ed attua contromisure qualora stabilisca, con un livello di probabilità sufficientemente alto, l'esecuzione di un attacco in corso.

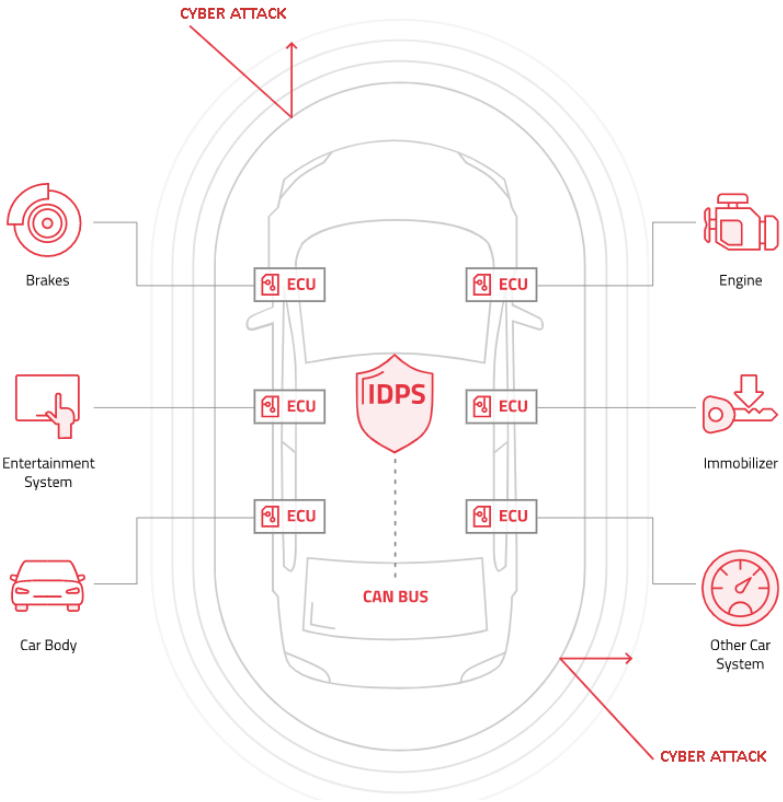

Figura 3 - Esempio di Scenario dove l'IDS/IPS protegge l'auto da attacchi indirizzati alle componenti del sistema

Nel seguente lavoro di tesi, si vogliono identificare i diversi tipi di attacchi che possono essere effettuati sul CAN-bus. In particolare, si andrà ad utilizzare un set di dati che includono attacchi Denial of Service (DoS), Fuzzy e Spoofing ottenuti registrando il traffico CAN attraverso la porta di on-board diagnostics (OBD-II) da un veicolo reale mentre si stava eseguendo un attacco di tipo injection message che inietta messaggi sviluppati per ingannare le ECU originali o causare malfunzionamenti.

L'obiettivo è quello di riuscire ad estrapolare delle informazioni in modo tale da poter riconoscere degli schemi temporali che delineano gli attacchi, facendo una netta distinzione tra malware e goodware.

Per raggiungere tale obiettivo si farà uso di alcune librerie di Python necessarie per la manipolazione dei dataset; una volta che i dati saranno in un formato comprensibile e adatto alla nostra necessità, verrà sviluppata una rete neurale ricorrente, ovvero una classe di rete neurale artificiale in cui i valori di uscita di uno strato di un livello superiore vengono utilizzati come ingresso ad uno strato di livello inferiore. Questa avrà il compito di effettuare un'analisi predittiva sulle sequenze temporali di valori, permettendo così di stabilire se gli attacchi inviati sul CAN-bus sono malware o goodware.

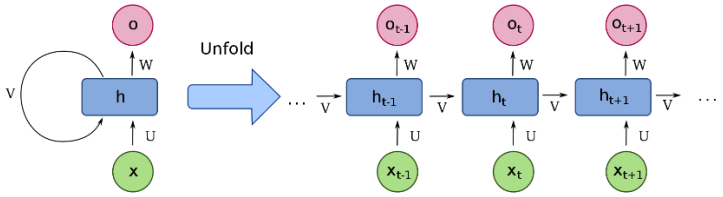

Figura 4 - Esempio di una rete neurale ricorrente

Di seguito è mostrata l'organizzazione del lavoro di tesi. Nel primo capitolo verrà mostrato il CAN nel dettaglio e verranno descritti i diversi tipi di attacchi che possono essere effettuati su questo. Il secondo capitolo tratterà il Machine Learning e le Reti Neurali, in particolare si andrà ad approfondire il funzionamento delle Reti e la loro classificazione. Il terzo capitolo descriverà le fasi di progettazione, l'implementazione, gli strumenti che sono stati utilizzati e la soluzione all'identificazione degli attacchi sul CAN-bus. Infine, le conclusioni e i lavori futuri concluderanno la tesi.

Controller Area Network

In questo capitolo vengono illustrati i concetti base del protocollo CAN, limitando però i dettagli alle caratteristiche che si ritengono di una certa importanza per comprendere meglio il funzionamento del sistema integrato complessivo.

Introduzione al CAN

Il Controller Area Network (CAN) è un protocollo di comunicazione seriale in grado di gestire con elevata efficienza sistemi di controllo distribuiti di tipo real-time, con un elevato livello di sicurezza e di integrità dei dati trasmessi.

Una rete CAN nella sua versione più tradizionale è composta da un bus lineare realizzato con una coppia intrecciata di fili (schermati o meno) e da un numero teoricamente infinito di interfacce (nodi) collegate fra loro attraverso questo mezzo trasmissivo (Figura 5).

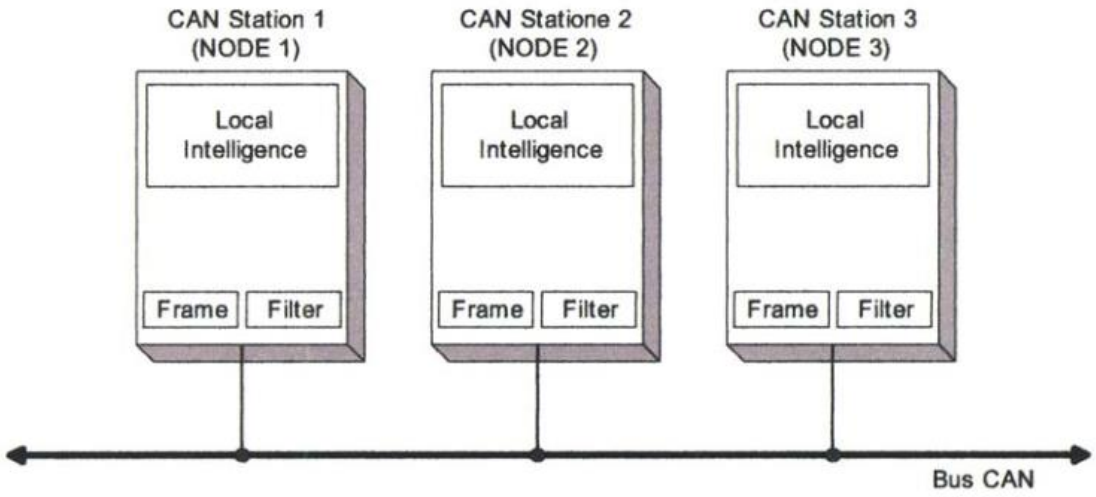

Figura 5 - Struttura schematica di una rete CAN

Il bus non viene occupato secondo uno schema predefinito. Tutti i componenti collegati sul bus hanno gli stessi diritti, ascoltano in "parallelo" tutti i messaggi, sono sempre pronti a ricevere e, quando necessario,

cominciano a trasmettere se nessun altro componente sta inviando messaggi sul bus.
E' la comunicazione *event-driven*, ossia la trasmissione viene inizializzata solo quando necessario.
Le specifiche del protocollo CAN non si occupano della descrizione del mezzo fisico quindi, per sopperire a questa esigenza sono nati diversi standard ognuno dei quali con proprie peculiarità ed in grado di soddisfare particolari esigenze applicative.
Uno degli elementi che distingue uno standard da un altro, è appunto il mezzo trasmissivo utilizzato per la realizzazione fisica della linea bus: coppie intrecciate di conduttori (twisted pair), bus a singolo conduttore, fibre ottiche, cavo coassiale, sistemi wireless a radio-frequenza, ed altro.
In ogni caso si tratta di un bus di tipo lineare.
Le specifiche CAN nella versione 1.2 e la successiva 2.0, forniscono una descrizione del protocollo limitatamente ad una parte dei due livelli più bassi del modello ISO/OSI, cioè il Physical Layer ed il Data Link.
Il CAN definisce le specifiche di basso livello del protocollo di comunicazione e lascia liberi di definire un HLP (Higher Layer Protocol, protocollo di livello superiore) adatto alle specifiche esigenze del progettista.

Cenni sull'accesso al bus

Il CAN[1] appartiene alla categoria dei protocolli *producer – consumer* in cui tutti i nodi della rete possono teoricamente essere in grado di inviare dati sul bus e tutti possono ricevere qualsiasi messaggio (si parla anche di *broadcasting* dei messaggi).
Non si può dire che i nodi abbiano un indirizzo specifico (per questo il CAN è definito un protocollo *Message-based* anziché *Address-based*) è invece più corretto affermare che l'informazione di indirizzo è contenuta nel

[1] Convenzionalmente nel resto del capitolo un bit con livello logico 1 verrà indicato come bit di tipo "recessivo" ed indicato con "r"; un bit con livello logico 0 è detto bit dominante e verrà indicato con il carattere "d"

campo *identifier* del frame trasmesso e che questo identificatore viene utilizzato da tutti i nodi della rete per controllare se i dati trasportati dal frame sono o meno di loro competenza (cioè se sono utili o meno al nodo stesso).
Il CAN non si basa su indirizzi "fisici" dati ad ogni nodo di rete, ma sul contenuto di una comunicazione che viene contrassegnato da un identificatore.
Non si può propriamente parlare di indirizzo perché più interfacce distinte possono riconoscere valido (utile) l'identificatore trasmesso e poi il campo *identifier* viene utilizzato per stabilire la priorità di trasmissione del messaggio sul bus.
Non essendoci, come detto, uno schema predefinito di accesso al bus, è necessario utilizzare un sistema di arbitraggio che "decida" quale messaggio può accedere al bus per primo, nel caso di contemporanea richiesta di accesso.

Struttura dei frame

Il campo *identifier* appena introdotto è una parte del così detto *Arbitration Field* di un frame di tipo DATA o REMOTE la cui struttura è riportata nel dettaglio nella figura 6 relativamente al protocollo standard (CAN 2.0A) ed in figura 7 dove i campi si riferiscono a quanto descritto nelle specifiche del protocollo esteso (CAN 2.0B).
Il CAN definisce cinque tipi di messaggi detti anche frame:

DATA frame: è il messaggio di tipo più comune ed è usato per trasmettere dati da un nodo ad un altro. Il frame inizia con un bit *Start of frame* usato per la sincronizzazione (hard synchronization) dei nodi e successivamente vengono trasmessi i campi di seguito elencati:

- L'*Arbitration Field* contiene le informazioni necessarie durante la fase di arbitraggio per assegnare il possesso del bus quando questo

viene conteso da più nodi. Questo campo a dimensioni differenti a seconda che il frame rispetti le specifiche 2.0A (standard frame) o 2.0B (extended frame) ma indipendentemente da wuesto, contiene l'identificatore (11 bit per frame standard, 29 bit per un frame extended) e l'RTR (*Remote Transmission Request*) bit con cui si distingue un DATA Frame da un REMOTE Frame.

- Il *Control Field* è composto da 6 bit, gli ultimi 4 in ordine di trasmissione costituiscono il campo DLC (*Data Length Code*) che codifica il numero di byte di dato contenuti nel messaggio (0-8 byte). Se il frame è di tipo standard il primo bit del campo di controllo è il bit di IDE (*Identifier Extention*), che specifica proprio il tipo di formato (IDE=d standard, IDE=r extended), questa informazione se il frame è di tipo esteso è contenuta nell'*Arbitration Field* descritto in precedenza.
- Il campo *Data Field* contiene i byte che codificano l'informazione trasmessa con il messaggio (il dato). Il campo può avere un'estensione che varia da 0 a 64 bit (solo multipli di 8) in base a quanto specificato nel campo DLC. I bit che compongono i singoli byte del campo vengono trasmessi partendo sempre dal più significativo.
- Il *Cyclic Redundancy Field* (*CRC Field*) viene impiegato come sistema (non unico) per la rilevazione degli errori di trasmissione ed è compost dalla *CRC sequence* (15 bit calcolati sulla base dei precedenti campi del frame) e da un bit recessive, il *CRC Delimiter* bit utilizzato per chiudere la trasmissione del CRC ed essere certi che il bus una volta trasmessi i 15 bit sia in stato recessivo e quindi pronto a gestire correttamente la fase di acknowledgment seguente.
- Il campo successivo è l'*Acknowledge Field*. Durante l'*ACK Slot* bit il nodo trasmittente invia un bit recessivo (r) e qualunque nodo della rete abbia ricevuto il messaggio formattato in modo corretto, risponde inviando un bit dominante (d) che "sovrascrive" il

precedente. Il nodo trasmittente riceve in questo modo l'ACK cioè la conferma dell'avvenuta corretta ricezione da parte di almeno un nodo e chiude la trasmissione con un bit recessivo che costituisce l'*ACK Delimiter*.

- Il DATA frame si chiude con il campo di *End Of Frame* (7 bit recessivi).

Remote frame: Una delle caratteristiche del protocollo CAN è che un nodo della rete non solo può trasmettere informazioni o rimanere in attesa di riceverne altre, ma può anche "chiedere" informazioni ad altri nodi della rete "presentando una domanda". Per mezzo del REMOTE frame il nodo trasmette una richiesta remota cioè una *Remote Transmit Request (RTR)*. Un REMOTE frame ha la stessa struttura di un DATA frame con le seguenti differenze:

- Il bit RTR è dominante per un DATA frame e recessivo per un REMOTE frame (per mezzo di esso il ricevitore riconosce il tipo di messaggio e si comporta di conseguenza).
- Non esiste il dato DATA (non ha senso prevederlo).
- Il campo DLC non codifica il numero di byte componenti il campo DATA ma può essere impiegato per specificare altre informazioni: il tipo di dato o il numero di informazioni che il nodo richiede.

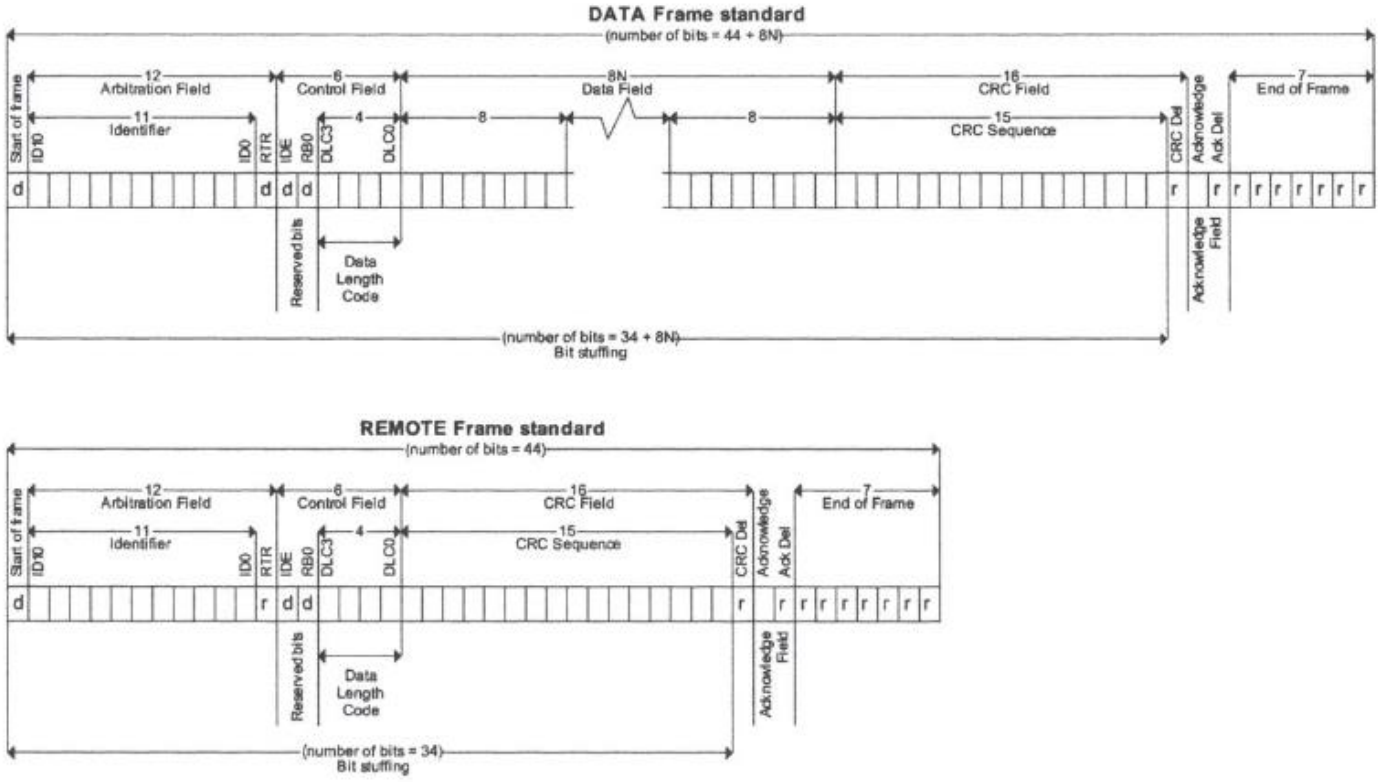

Figura 6 - DATA frame & REMOTE Frame standard

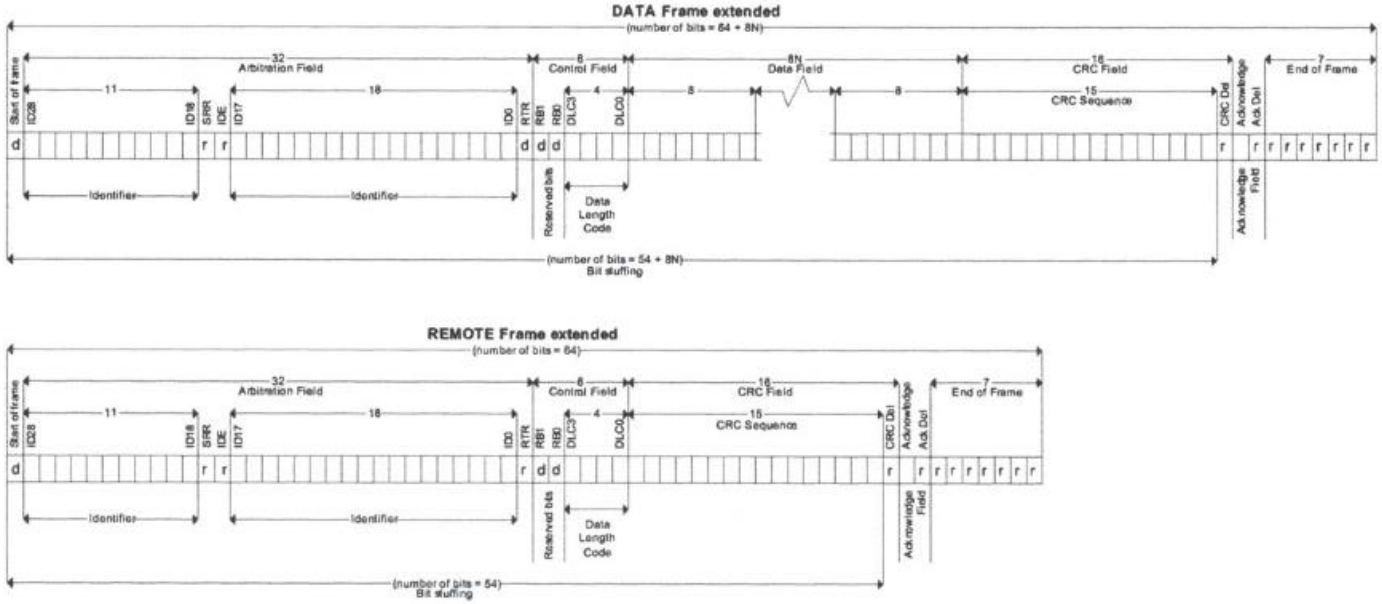

Figura 7 - DATA Frame & REMOTE Frame extended

ERROR frame: è generato da ogni nodo che rileva un errore. Gli errori a cui il protocollo si riferisce, sono i seguenti: *Bit Error*, *Stuff Error*, *Form Error* e *Ack Error*. Al rilevamento dei quali viene subito inviato un Error Frame. Se viene rilevato un *CRC Error*, il frame viene inviato anche in questo caso, ma solo successivamente all'*ACK Delimiter*.
Come mostrato in figura 8 il frame di errore è composto da due campi chiamati *Error Flag Field* e *Error Delimiter Field*, quest'ultimo è composto da 8 bit recessivi a seguito dei quali i nodi possono riprendere la comunicazione sul bus. La struttura del campo *Error Flag* dipende dallo stato in cui si trova il nodo al momento in cui il frame di errore deve essere generato.

- Se il nodo è *Error Active* l'Error Flag è composto da 6 bit dominanti. Questa sequenza a seconda del momento in cui viene trasmessa viola le regole di bit stuffing oppure la struttura del campo di Acknowledgement o ancora quella del campo di End Of Frame del messaggio che gli altri nodi della rete stanno decodificando. Indipendentemente da quando l'errore si verifica l'ERROR Frame viola qualche regola di formato e come conseguenza anche gli altri nodi rilevano a loro volta un errore e trasmettono un ERROR Frame, i vari Error Flag si sovrappongono sul bus producendo un "echo" che ha come effetto quello di allungare la permanenza della sequenza di bit dominanti; l'Error Flag che ne deriva può essere composto da un numero di bit dominanti che può variare tra 6 e 12.
- Se il nodo è *Error Passive* al rilevamento dell'errore il campo Error Flag prodotto è composto da 6 bit recessivi; il frame complessivamente è costituito da 14 bit recessivi. Come conseguenza l'ERROR Frame trasmesso da un nodo "passivo" non ha effetto sugli altri moduli della rete (l'informazione di errore trasmessa non è considerata attendibile perchè generata da una stazione CAN "passiva" quindi con qualche "problema"). Solo se il nodo Error Passive è in trasmissione ed esso stesso rileva l'errore, l'Error Flag di

tipo passivo viola le regole di bit stuffing e quindi l'errore viene rilevato anche dagli altri nodi, solo in questo caso un ERROR Frame di tipo passivo ha effetto sulle altre stazioni della rete.

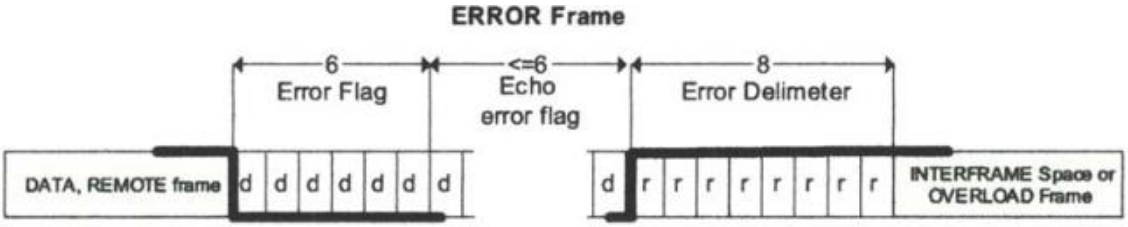

Figura 8 - Struttura dell'ERROR Frame

INTERFRAME Space: con questo termine non si indica un vero e proprio frame ma solo una sequenza di bit recessivi (minimo 3 che costituiscono l'*Intermission Field*) che separano un frame qualunque da un altro frame di tipo DATA o REMOTE. Successivamente alla trasmissione dell'*Intermission Field* la linea del bus rimane in stato recessivo (Bus Idle) fino alla trasmissione di un nuovo messaggio. Dopo la trasmissione dell'Intermission Fied il bus è considerato disponibile.Un'eccezione a quanto appena descritto è mostrata in figura 9-B : nel caso in cui l'ultimo nodo trasmittente sia di tipo Error Passive all'*Intermission Frame* segue una sequenza di otto bit recessivi detta *Suspend Transmission segment* che allunga i tempi minimi di trasmissione di un nuovo messaggio da parte dello stesso nodo, ciò consente ad altre interfacce anche con identificatori con priorità inferiore di iniziare una nuovo trasmissione prima del nodo passivo, così facendo il protocollo evita che un modulo ad altissima priorità ma difettoso possa rischiare di monopolizzare la rete.

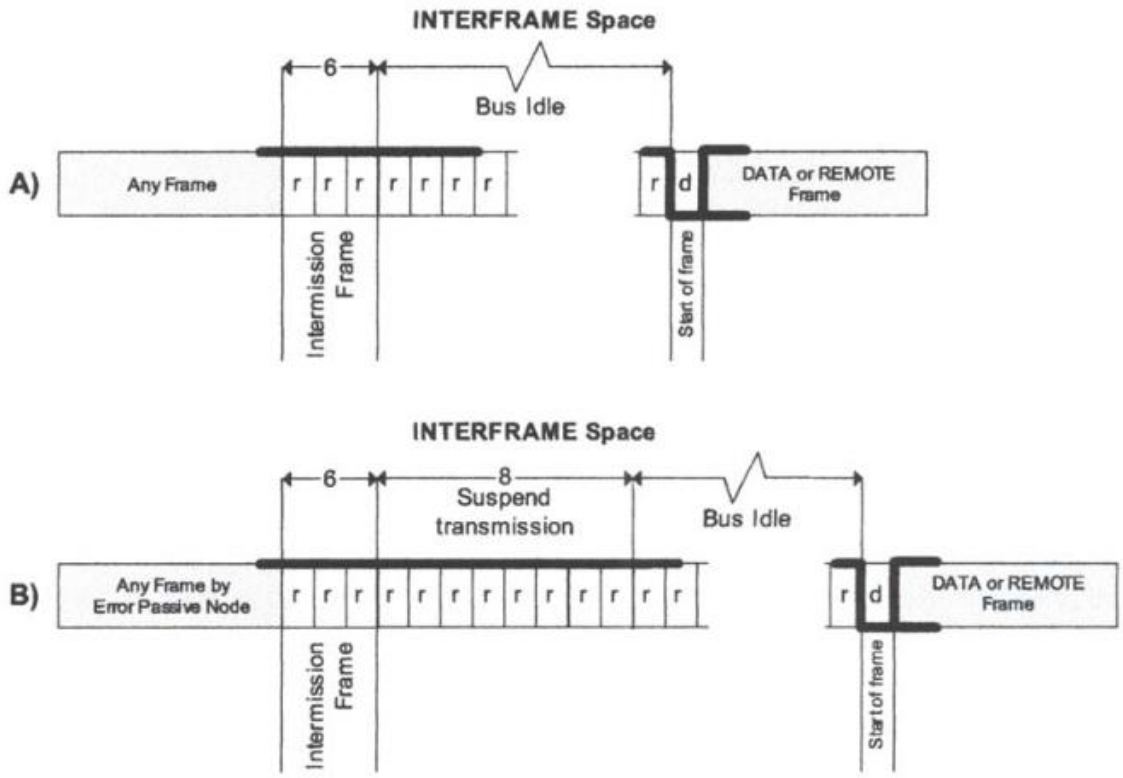

Figura 9 - Struttura dell'INTERFRAME Space

OVERLOAD Frame: questo frame ha la stessa struttura di un ERROR Active frame tuttavia può essere generato solo durante un *INTERFRAME Space* e questa è anche l'unica maniera attraverso la quale i due messaggi possono essere distinti. Rispetto ad un *ERROR Frame*, l'*OVERLOAD* non incrementa i contatori TEC e REC e non causa la ritrasmissione dei messaggi. Il messaggio ricevuto prima di un *OVERLOAD frame* è un messaggio valido a differenza di quello che precede un *ERROR Frame*.

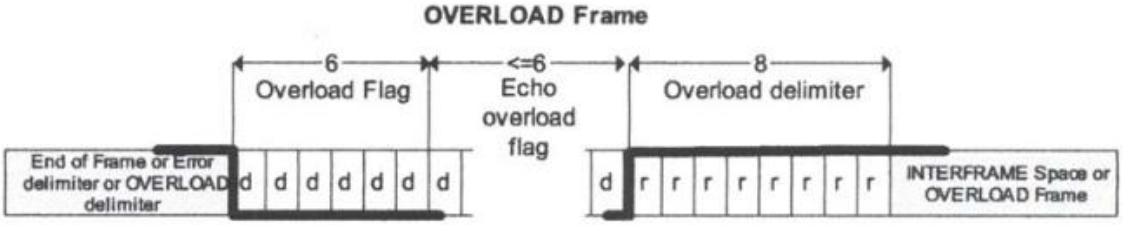

Figura 10 - Struttura dell'OVERLOAD Frame

L'OVERLOAD viene trasmesso al verificarsi di una delle seguenti condizioni:

1. Il nodo per qualche motivo non è in grado di riabilitarsi in tempo per una nuova ricezione dopo che la precedente si è conclusa.

2. Viene rilevato un bit dominante durante la trasmissione dell'Intermission Frame. Se il bit dominante è l'ultimo (il terzo) viene interpretato come uno Start of Frame. Dato che la regola del "bit stuffing" (descritta in seguito) viene applicata a partire dal bit di SOF, il sesto bit dell'Overload flag viola lo stuffing e come conseguenza i nodi attivi genereranno un Error Active flag.
3. Se un nodo campiona un bit dominante come ultimo bit di un Error Delimiter o di un Overload Delimiter.
4. Se un nodo in ricezione rileva un bit dominante come ultimo bit di un End or Frame riconosce il messaggio come valido. Il bit dominante ricevuto è interpretato appartenente all'Intermission Frame quindi il nodo si trova nella condizione descritta nel punto 1 (fig. 11).

La struttura dell'OVERLOAD è analoga a quella dell'ERROR Frame e come tale la condizione di overload viene notificata a tutti i nodi della rete che a loro volta generano l'echo mostrato in figura 10. L'obiettivo che il nodo generatore dell'OVERLOAD Frame deve raggiunge, è quello di ritardare l'istante in cui il bus tornerà in condizione di Idle e quindi sarà nuovamente disponibile a gestire altro traffico.

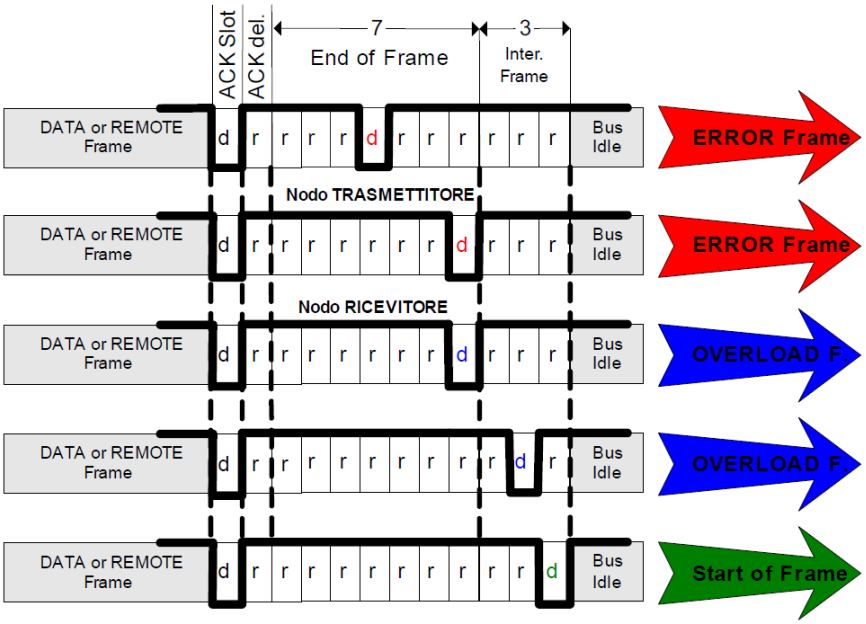

Figura 11 - Comportamento del nodo CAN al rilevamento di errori nei campi di EOF e Intermission Space

Perché eseguire l'hack sul CAN?

Prima di descrivere come funziona l'hacking del CAN bus, è importante considerare la logica che si nasconde dietro.

Per decenni, gli hacker etici hanno preso di mira sistemi informatici tradizionali. Questi hacker dal "cappello bianco" effettuano attacchi contro i sistemi al fine di esporre le loro vulnerabilità in modo tale da poter migliorare la loro sicurezza.

Uno dei modelli più utilizzati dall'industria è il modello "sicurezza tramite segretezza", si basa sull'assunto che tenere segreto il funzionamento interno di un sistema o di una componente lo renda più sicuro, dato che un eventuale attaccante avrebbe maggiore difficoltà nella scoperta di eventuali falle del sistema stesso. Purtroppo questo modello non funziona.

Invece di investire in soluzioni di sicurezza proattive, i produttori di automobili hanno la tendenza a tagliare fondi alla sicurezza a favore di un risparmio sui costi di produzione.

Prima ancora che l'hacking di veicoli difettosi diventasse comune, i produttori automobilistici credevano che non sarebbe stato necessario dover spendere una quantità significativa di budget o di attenzione. Tuttavia, ignorare le vulnerabilità o tentare di nasconderle non porta a sistemi più sicuri. L'unico modo per superare questo modello imperfetto è aumentare l'esposizione pubblica delle vulnerabilità di sicurezza.

L'hacking automobilistico può anche essere considerato come un tipo di audit di sicurezza. Riuscendo a controllare la sicurezza del proprio veicolo, è possibile acquisire una migliore comprensione dei modi in cui il veicolo potrebbe essere vulnerabile agli attacchi e alle possibili precauzioni da effettuare.

Legalità & Sicurezza

Nonostante i vari rapporti pubblicati sull'hacking delle auto da parte dei ricercatori, la manipolazione dei sistemi di sicurezza automobilistica non è priva di conseguenze legali.

Infatti, nel 2012, un gruppo di ricercatori dell'Università di Radboud e dell'Università di Birmingham hanno scoperto una significativa carenza di sicurezza nei sistemi di immobilizzazione del motore dei veicoli di diversi produttori. Quindi il team contattò le case automobilistiche per informarli di tali vulnerabilità ed era loro intenzione pubblicare i risultati nella prossima conferenza sulla sicurezza. Questo preoccupò molte case automobilistiche, in particolare la Volkswagen.

La Volkswagen preoccupata di questa divulgazione della vulnerabilità perché avrebbe potuto giovare a possibili aggressori, inoltre avrebbe comportato il richiamo di tutti i modelli interessati per rimediare al problema, rischiando così la perdita di molti soldi, ha intentato una causa per bloccare la pubblicazione del documento.

Così il gruppo di ricerca è stato costretto a negare le sue conclusioni e giungere a un'intesa con la Volkswagen.

L'hacking automobilistico è intrinsecamente pericoloso. Quando si interagisce con il CAN bus di un veicolo, è importante non perdere mai di vista il fatto che il sistema target è un metallo di diverse tonnellate, in grado di raggiungere velocità pericolose in breve tempo.

A differenza dell'hacking tradizionale del computer in cui un errore potrebbe portare alla corruzione del sistema operativo o ad *una schermata blu della morte,* un errore di hacking di un auto potrebbe portare a lesioni gravi o addirittura provocare la morte. Pertanto, è importante praticare l'hacking delle auto in modo sicuro e controllato.

Visto che nel CAN bus si trovano molte unità di controllo critiche di un veicolo, c'è la possibilità di provocare una risposta non intenzionale da parte del motore, freni, o altri componenti del veicolo durante la

sperimentazione con i messaggi inviati al CAN. Anche se il motore o le trasmissioni non sono l'obiettivo previsto, è importante pianificare il peggio.

L'approccio migliore per l'hacking sicuro delle auto è quello di sollevare il veicolo in modo che non sia più a contatto con il suolo, come mostrato in figura 12.

Figura 12 - Auto sollevata in sicurezza dal suolo

Tutto questo per eliminare qualsiasi preoccupazione da un'accelerazione involontaria. E' importante acquisire familiarità con il veicolo target per sapere se il veicolo invia segnali alle ruote anteriori, posteriori o a tutte e quattro. E' anche importante ricercare, in anticipo, come il veicolo target può essere spento in caso di emergenza.

Per poter effettuare l'hacking del CAN bus, bisogna conoscere a fondo il veicolo target, perché il CAN può essere implementato in modo diverso da ciascun produttore. Alcuni veicoli utilizzano un solo CAN bus, altri possono averne diversi e separati tra loro. E' quindi necessario eseguire alcune ricerche di base sul veicolo per sapere quali tipi di bus sono presenti.

OBD-II

I sistemi OBD forniscono al proprietario del veicolo o a un meccanico accesso alle informazioni sullo "stato di salute" dei vari sottosistemi del veicolo: la normativa standard (in Europa e Stati Uniti) è riferita però solo ai sottosistemi *emission revelant,* cioè quelli che, se rotti, possono portare a un aumento delle emissioni, come catalizzatore, sonda lambda, ecc., mentre gli altri sistemi (es. airbag, climatizzatore, ecc.) hanno un'autodiagnosi non standard, definita a piacimento da ogni costruttore automobilistico.

L'OBD-II permette di avere un controllo completo sui parametri del motore e monitorare altre parti di un veicolo come il telaio e gli accessori; inoltre permette di connettersi al sistema di diagnostica.

L'OBD-II è un'interfaccia per acquisire segnali di diagnostica. Lo standard OBD-II definisce inoltre alcuni comandi per il controllo dell'output, per le modalità di autocontrollo e per l'azzeramento della memoria KAM (Keep Alive Memory).

Questo sistema fornisce l'accesso diretto alle reti di bordo dei veicoli, mettendo così a rischio le unità di controllo da parte di possibili aggressori. Infatti, una volta trovati i giusti mezzi per accedervi, un aggressore sarebbe in grado di compromettere una componente o l'intera rete di bordo ed essere capace di disabilitare alcune funzioni del veicolo, come i freni, il motore e così via.

Nella specifica è incluso uno speciale connettore con una piedinatura standard per collegarsi alla rete. La piedinatura standard include un collegamento diretto al CAN bus. Per un ricercatore in sicurezza automobilistica, la porta OBD-II è essenzialmente una backdoor non protetta, infatti questa è aperta a chiunque abbia l'hardware appropriato.

Nella maggior parte dei veicoli, la porta OBD-II si trova sotto il cruscotto.

La figura 13 mostra dove la porta OBD-II di solito può essere trovata e come appare.

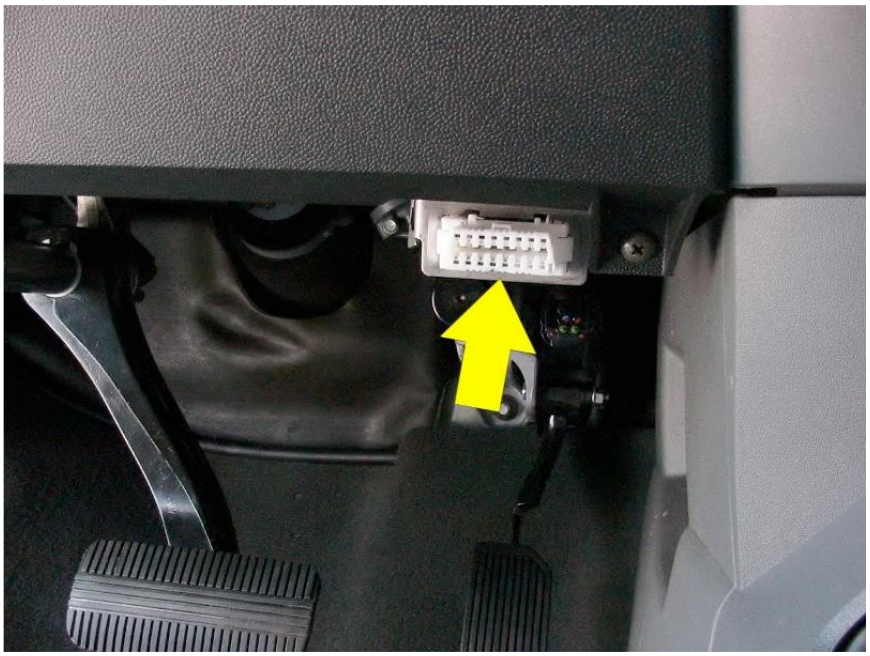

Figura 13 - Porta OBD-II

Car Hacking Hardware

Nonostante la porta OBD-II sia un'interfaccia non protetta, l'hacking delle auto tramite OBD-II non è così semplice collegare un computer direttamente alla porta. Ci sono scanner, comunemente usati dai meccanici, che si possono collegare direttamente alla porta OBD-II. Tuttavia, la capacità di questi dispositivi di solito non si estende oltre alla lettura e cancellazione di codici diagnostici.

Al fine di utilizzare un computer portatile per comunicare con il CAN bus, sono necessarie alcune periferiche hardware aggiuntive. La figura 14 mostra l'hardware che viene utilizzato per comunicare con possibile veicolo target.

Figura 14 - Hardware di base richiesto per l'hacking del CAN bus

Per poter effettuare l'hacking del CAN bus, oltre all'utilizzo del portatile viene utilizzato il dispositivo CANtact che si presenta come una scheda di rete, che consente al laptop di interagire con il CAN bus. Il dispositivo funziona su Linux, OS X e Windows, una volta stabilito il collegamento, l'interfaccia CANtact passerà i pacchetti che viaggiano sul CAN tra il veicolo e il computer.

Software

Oltre ad avere il giusto hardware, è necessario scaricare e installare il software appropriato per abilitare la comunicazione tra il veicolo e il computer.

Quando si seleziona un sistema operativo, è importante assicurarsi che sia pienamente compatibile con l'hardware scelto. Bisogna sapere che per effettuare l'hacking automobilistico non viene richiesto esplicitamente un sistema Linux, questo può essere eseguito anche su Windows, tuttavia la community preferisce Linux per questo molti strumenti e dispositivi open source disponibili sono progettati principalmente per l'uso con Linux.

SocketCAN

SocketCAN è un insieme di open source CAN driver e stack di rete originariamente noto come “Low Level CAN Framework” sviluppato dalla Volkswagen.
I CAN driver tradizionali per Linux si basano su modelli di dispositivi a caratteri. In genere consentono solo l’invio e la ricezione dal CAN controller. Le implementazioni convenzionali di questa classe di driver consentono l’accesso al dispositivo a un singolo processo, il che significa che gli altri processi vengono temporaneamente bloccati. Inoltre, questi driver in genere differiscono leggermente nell’interfaccia presentata dall’applicazione, riducendo la portabilità. Il concetto di SocketCAN estende l’*API* dei socket Berkeley in Linux introducendo una nuova famiglia di protocolli, *PF_CAN*, che coesiste con altre famiglie di protocolli come *PF_INET* per l’*Internet Protocol*. La comunicazione con il CAN bus viene quindi eseguita in modo analogo all’uso dell’*Internet Protocol* tramite socket. I componenti fondamentali del SocketCAN sono i driver del dispositivo di rete per i diversi CAN controller e l’implementazione della famiglia di protocolli CAN. Il *PF_CAN*, fornisce le strutture per abilitare diversi protocolli sul bus: socket per la comunicazione diretta con il CAN e protocolli di trasporto per connessioni *point-to-point*. Inoltre il gestore di trasmissione che fa parte della famiglia di protocolli CAN fornisce funzioni per inviare periodicamente messaggi CAN o funzioni per realizzare messaggi filtrati complessi.

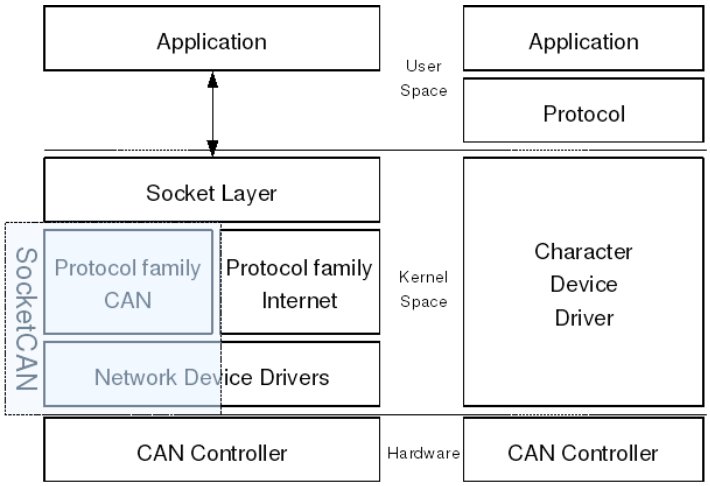

Figura 15 - Tipici livelli di comunicazione CAN. Con SocketCAN (a sinistra) e convenzionale (a destra)

Can-utils

Il pacchetto SocketCAN include una serie di tools che sono particolarmente utili quando si comunica con il CAN bus di un veicolo moderno:

- **candump:** come suggerisce il nome, stamperà tutti i dati ricevuti dal CAN direttamente sulla console. E' possibile aggiungere diversi filtri per migliorare l'utilità dell'output.
- **cansend:** consente l'invio di un singolo pacchetto da inviare al CAN bus. Quando si utilizza *cansend*, è necessario specificare l'interfaccia, il *CAN identifier* e il *CAN data*.
- **cansniffer:** probabilmente il più utile dei *can-utils* quando si cerca di decodificare messaggi CAN bus del veicolo, *cansniffer* funziona come *candump* ma esegue in tempo reale il filtraggio dell'output sulla console. Filtrando eventuali messaggi CAN ripetitivi contenenti dati che rimangono invariati, *cansniffer* visualizza solo i messaggi CAN per i quali i dati cambiano in tempo reale. Permettendo così di concentrarsi solo su pacchetti più rilevanti.

Wireshark con SocketCAN

Wireshark offre anche un supporto per i dispositivi SocketCAN. Può essere usato per visualizzare l'output del CAN bus in tempo reale, simile a *candump*, e può filtrare i messaggi CAN in base all'ID del messaggio.

Riconoscere messaggi CAN

Se sono stati eseguiti tutti i passaggi necessari, l'interfaccia dovrebbe essere attiva e pronta a comunicare le informazioni che attraversano il CAN bus del veicolo target. Per iniziare a vedere il traffico, potrebbe essere necessario accendere il veicolo, sebbene non sia necessario avviare il motore.
Un test semplice e veloce per vedere se i pacchetti CAN vengono ricevuti correttamente, bisogna eseguire il comando:

```
candump slcan0
```

A questo punto, una raffica di messaggi CAN dovrebbe iniziare a scorrere sullo schermo della console. Anche se l'output *candump* non sia molto utile, è comunque un modo efficace per confermare che l'interfaccia CAN funzioni correttamente.
Per avere un output migliore, bisogna eseguire il comando:

```
cansniffer –c slcan0
```

Come già è stato detto in precedenza, *cansniffer* esegue il filtraggio in tempo reale dei messaggi CAN che riceve. Tutti i messaggi che rimangono invariati verranno filtrati, rendendo così l'output più leggibile. Inoltre utilizzando l'opzione –c evidenzia i byte che cambiano.
Il vantaggio del *cansniffer* è che ogni CAN ID univoco ha una propria riga e rimane fisso in quella posizione, quindi quando si cerca di manipolare fisicamente le funzioni del veicolo, ad esempio, spostando la leva del cambio da P(parcheggio) a R(retromarcia) provocherà una modifica del CAN ID specifico.

Manipolazione del veicolo target

Realizzare manualmente messaggi CAN personalizzati è relativamente semplice. Ad esempio, il *cansend* consente di digitare manualmente i messaggi CAN nella console e inviarli al CAN bus. Tuttavia, l'invio di singoli messaggi in questo modo è improbabile che consenta un notevole controllo del veicolo. Questo perché quando un messaggio viene ricevuto ed elaborato, i suoi effetti di solito durano solo da 10 a 20 millisecondi prima che un altro messaggio venga ricevuto ed elaborato dal controller di ascolto. Pertanto, quando si tenta di controllare determinate funzioni veicolari sono richiesti un flusso continuo di messaggi.
Un modo per trasmettere continuamente i messaggi al veicolo è utilizzare un linguaggio di scripting come Python. In questo modo è possibile creare un ciclo per eseguire il comando *cansend* ad un intervallo specificato, ad esempio ogni 20ms.
Un modo alternativo è creare un file .log personalizzato da riprodurre su un veicolo che utilizza *canplayer*. Il file .log dovrà essere del seguente formato:

```
(1492287144.880000) slcan0 1DC # 023D1713
(1492287144.900000) slcan0 1DC # 023D1713
(1492287144.920000) slcan0 1DC # 023D1713
(1492287144.940000) slcan0 1DC # 023D1713
(1492287144.960000) slcan0 1DC # 023D1713
(1492287144.980000) slcan0 1DC # 023D1713
(1492287145.000000) slcan0 1DC # 023D1713
```

- La prima colonna rappresenta la data e l'ora corrente in cui viene lanciato il messaggio.
- La seconda colonna rappresenta l'interfaccia su cui il messaggio deve essere inviato.
- Le prime 3 cifre del messaggio rappresentano l'ID, il simbolo "#" viene utilizzato per separare l'ID dal corpo del messaggio, i caratteri

che seguono dopo rappresentano i dati del messaggio in esadecimale.

Una volta che è stato creato il file .log basterà eseguire il comando:

```
canplayer –v –I nomefile.log
```

Quando viene eseguito *canplayer*, l'opzione –v permette di visualizzare ogni riga del file sullo schermo in tempo reale, mentre viene riprodotto, invece l'opzione –I viene utilizzata per specificare il nome del file di input. L'esecuzione può essere effettuata sia con motore spento o in funzione, indipendentemente dal fatto che il veicolo sia parcheggiato o in movimento, sarà possibile prendere controllo del CAN bus e visionarne i comportamenti.

Attacchi al CAN bus

Come accennato in precedenza, a causa dei difetti del veicolo, gli aggressori sono fortemente motivati a sfruttare le vulnerabilità del CAN bus. Gli attacchi più conosciuti sono:

Attacco DoS: un aggressore può iniettare messaggi ad alta priorità in un breve ciclo sul bus. Poiché qualsiasi nodo non è posseduto o controllato da un gestore di rete, un nodo maligno può non rispettare le regole del protocollo per ottenere un accesso non autorizzato. I messaggi di attacco DoS mirano ad occupare il bus utilizzando l'identificatore di priorità teoricamente più elevato, ossia 0x000. La Figura 16 mostra i messaggi durante lo stato montato di un attacco DoS. Il primo record che ha il terzo indice come 100 in un frame remoto, e l'ultimo è il corrispondente messaggio di risposta. Ci sono tre messaggi iniettati 0x000 tra i messaggi di richiesta e i messaggi di risposta, che ritardano inevitabilmente la ricezione della risposta.

Poiché tutti i nodi condividono un unico bus, l'aumento dell'occupazione del bus può produrre latenze di altri messaggi e causare minacce per quanto riguardano la possibilità di ricezione ai comandi del conducente.

Attacco Fuzzy: un aggressore può iniettare messaggi di identificatori spoofed casuali con dati arbitrari. Di conseguenza, tutti i nodi riceveranno un sacco di messaggi funzionali, e ciò provoca comportamenti del veicolo non intenzionali. Per sfruttare l'attacco fuzzy, un aggressore ha osservato i messaggi a bordo del veicolo e ha selezionato degli identificatori bersaglio per produrre comportamenti inaspettati. Inserendo dati arbitrari negli identificatori falsificati, abbiamo scoperto che il volante viene scosso tremendamente, le luci degli indicatori di direzione si accendono in modo irregolare, il cruscotto lampeggia in innumerevoli modi e il cambio di marcia cambia automaticamente. A differenza dell'attacco DoS, paralizza le funzioni di un veicolo piuttosto che ritardare i normali messaggi tramite l'occupazione del bus.

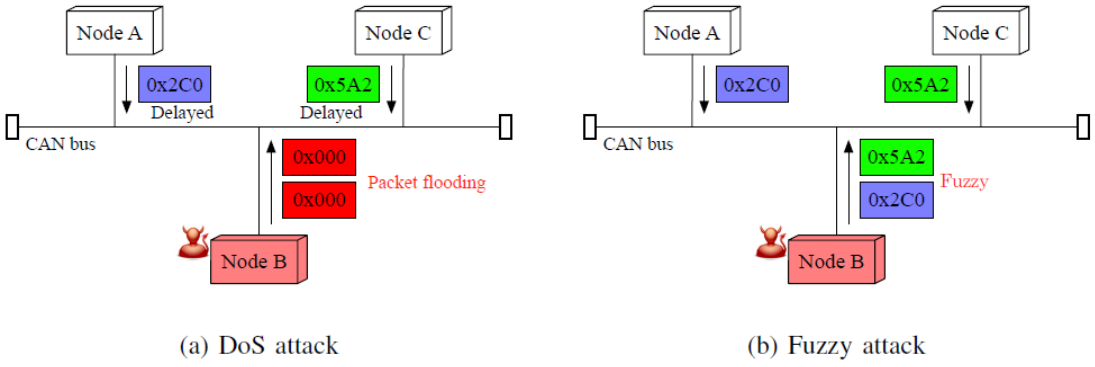

Figura 16 - Due Tipi di attacchi su CAN bus

Attacco Spoofing: comprendono tutti i casi in cui un aggressore, falsificando i dati, invia messaggi fingendo di essere un altro nodo della rete. Nella sicurezza automobilistica gli attacchi di spoofing si dividono solitamente in attacchi Masquerade e Replay. Gli attacchi Replay sono definiti come tutti quei casi in cui l'aggressore finge di essere la vittima e invia dati sniffati che la vittima ha inviato in un momento precedente. Masquerade sono invece tutti quegli attacchi di spoofing in cui il payload dei dati è stato creato dall'aggressore.

Machine Learning e Reti Neurali

Il Machine Learning e la Data Analysis

Il grande progresso nel campo della raccolta dei dati mette a disposizione una grande quantità di informazioni eterogenee fra loro, le quali una volta raccolte debbono essere opportunamente analizzate per trarne beneficio. Il processo di ispezione, pulizia, trasformazione e modellizzazione dei dati con lo scopo di estrarne informazioni utili è detto Data Analysis. La Data Analysis è un campo estremamente vasto che comprende numerosi approcci e tecniche in base alla natura dei dati a disposizione e del tipo di risultati che si persegue, dalla ricerca scientifica, economia e finanza, scienze sociali ed applicazioni tecniche. Le principali macro-categorie della Data Analysis sono:

- **Applicazioni statistiche**: gli studi statistici ricorrono a grandi quantità di informazioni utilizzabili in diverse applicazioni, la Data Analysis può essere applicata per studi descrittivi di un particolare fenomeno, per analisi esplorative (EDA-Explorative Data Analysis) o per la verifica delle ipotesi di un modello di studio (CDA-Confermatory Data Analysis);
- **Data Mining**: è una tecnica di analisi che si focalizza nella ricerca di correlazioni e pattern all'interno di grandi dataset. Il termine è spesso usato impropriamente poiché si riferisce all'estrazione di pattern dai dati a disposizione e non all'estrazione dei dati stessi. Il Data Mining rispetto ad altre tecniche di analisi è completamente rivolto alla creazione di modelli statistici utilizzati con scopi predittivi, in particolare grazie all'utilizzo del *machine learning*. Attualmente il compito di del Data Mining consiste nella analisi semi-automatizzata di grandi quantità di dati alla ricerca di

correlazioni (cluster analysis), anomalie e dipendenze al fine di fornire uno strumento di supporto decisionale in numerosi processi.

Le analisi predittive si concentrano sulla creazione di modelli sia per la previsione di fenomeni, sia per problemi di classificazione, come l'analisi di testi nel capo della linguistica e nell'estrazione di informazioni da fonti visive o audiovisive, le cui applicazioni vanno dalla sicurezza negli aeroporti al marketing, grazie ad una intelligenza artificiale.

Il processo di Data Analysis è composto da varie fasi che consentono di ottenere una efficace ed affidabile estrapolazione delle informazioni desiderate. La prima fase è la **raccolta** dei dati, durante la quale vengono selezionati tutti i dati, provenienti dalle più diverse fonti, che saranno utilizzati nel corso dell'analisi. Segue l'**organizzazione** dei dati, dove le informazioni raccolte vengono organizzate in maniera omogena in una forma adatta allo studio che si intende applicare, la struttura più utilizzata è un database organizzato in righe e colonne. In seguito occorre provvedere ad una **pulizia** dei dati, infatti molti elementi raccolti potrebbero incompleti, ripetuti o errati e il loro utilizzo comporterebbe errori e imprecisioni da parte del modello di studio. Solo dopo aver completato questi passaggi si può procedere con la **modellizzazione** vera e propria di un algoritmo in grado di identificare pattern ed eventuali correlazioni causa-effetto contenute nei dati. L'ultimo passaggio consiste nella produzione di un **risultato** dell'analisi in grado di fornire un supporto efficace al processo decisionale.

Con il termine *machine learning*, o apprendimento automatico in italiano, si intende un insieme di metodi e strumenti sviluppati nel campo dell'intelligenza artificiale col fine di rendere possibile ad un calcolatore l'apprendimento di pattern senza la necessità di una programmazione tradizionale esplicita, migliorando progressivamente la qualità dell'analisi effettuata. Il machine learning viene adoperato in tutte quelle situazioni in cui la scrittura di un algoritmo esplicito risulti impossibile o estremamente

sconveniente per vari motivi, come ad esempio un numero eccessivo di regole complesse di cui tenere conto, il riconoscimento di immagini o suoni o lo studio di una grande mole di dati di cui non si conosce precisamente la correlazione.

L'apprendimento automatico viene spesso utilizzato per elaborare delle predizioni sulla base di dati precedenti sfruttando la principale peculiarità che lo contraddistingue: la capacità di generalizzare in base alla propria esperienza e di produrre dei "ragionamenti induttivi" altrimenti impossibili con un programma tradizionale.

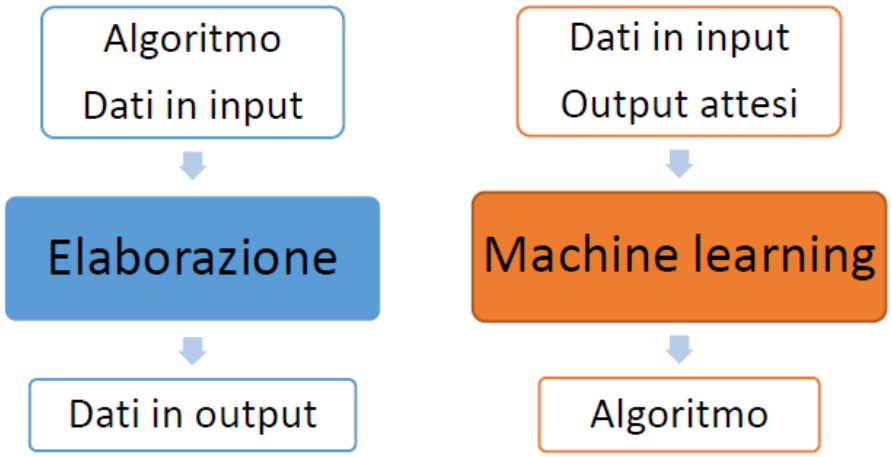

Figura 17 - programma tradizionale (sinistra) e programma di machine learning (destra)

La caratteristica principale del machine learning è la capacità di imparare e quindi il processo di apprendimento riveste un ruolo cruciale nella fase di creazione del modello. Esistono due approcci principali a questo procedimento.

L'inconveniente principale legato all'utilizzo di algoritmi di machine learning è che non è possibile conoscere la struttura analitica dell'algoritmo che sviluppato, rendendo, di fatto, quasi impossibile valutare la bontà dello stesso in termini assoluti. Per questo motivo è spesso difficile rilevare o meno la presenza di *overfitting* nel modello. L'overfitting, o adattamento eccessivo, è un fenomeno che si presenta nei modelli statistici ad elevato grado di complessità e consiste nell'eccessivo adattamento del modello ai dati causato dal numero eccessivo di parametri rispetto al numero di osservazioni. In pratica un modello non corretto si può comunque adattare

e fornire risultati apparentemente validi se risulta sufficientemente complesso, questo può avvenire se, ad esempio, i dati utilizzati per l'addestramento presentassero caratteristiche peculiari di quello specifico dataset, l'algoritmo riconoscerà quindi dei pattern inesistenti, perdendo la capacità di generalizzare nel momento in cui analizzerà dei dataset differenti. Sono stati sviluppati dei procedimenti per prevenire l'overfitting, tuttavia non esistono dei parametri oggettivi da misurare e il rischio di incappare in questo inconveniente è sempre presente.

Le reti neurali artificiali

Le reti neurali artificiali, o Artificial Neural Network, sono una delle principali branche nel campo del machine learning. Sono dei sistemi computazionali che tentano di replicare la struttura delle reti neurali biologiche.

L'unità principale è il nodo o percettrone (N), che come il suo corrispettivo biologico, tramite dei collegamenti (L), comunica gli altri nodi; il sistema è uno spazio dimensionale dove N>>L. I nodi sono organizzati in strati (*layer)*.

La struttura base prevede:

- Un layer di ingresso, il quale riceve gli input che devono essere elaborati dal sistema;
- Uno o più layer intermedi (detti anche *nascosti*);
- Un layer finale che fornisce gli output.

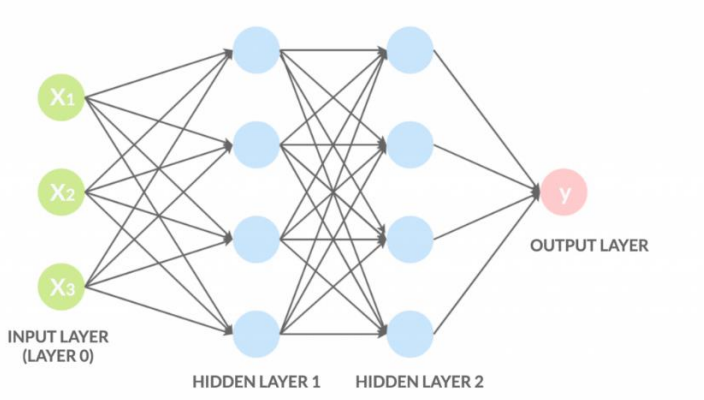

Figura 18 - struttura base di una rete neurale artificiale

L'elemento fondamentale di una rete neurale è il *percettrone*, il quale simula il neurone biologico. Il percettrone è formato da tre elementi fondamentali:

1. Un numero variabile di connessioni (replica delle sinapsi biologiche), ognuna della quali caratterizzata da un peso, che, differentemente dal corrispettivo biologico, può assumere anche un valore negativo;
2. Un sommatore, il quale somma gli input provenienti dalle varie connessioni in base al loro peso, producendo come output una combinazione lineare degli input ricevuti;
3. Una funzione di attivazione che serve a limitare l'ampiezza dell'output. Solitamente l'output è compreso fra i valori [0,1] o [-1,1].

Il percettrone comprende anche un valore soglia che permette di amplificare o ridurre l'input netto elaborato dalla funzione di approssimazione.

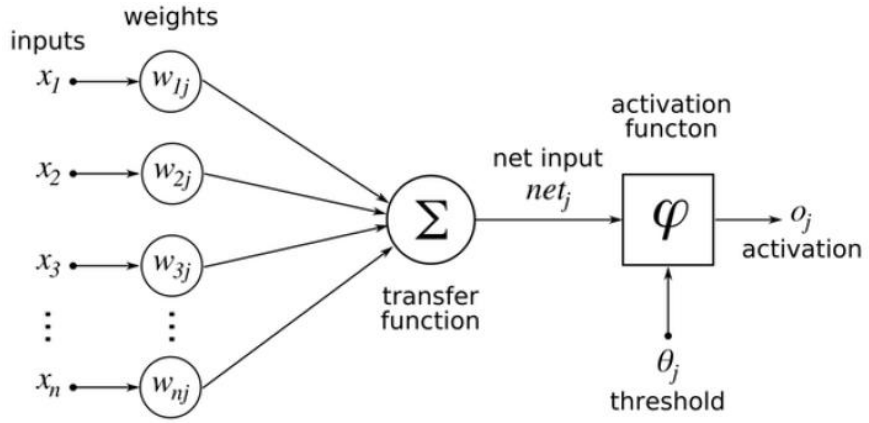

Figura 19 - Schematizzazione matematica di un percettrone

Da un punto di vista matematico un neurone k può essere rappresentato dalle seguenti equazioni:

$$u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j$$

$$y_k = \varphi(u_k + b_k)$$

Dove:

- x_i sono i pesi sinaptici del neurone k;

- u_k è la combinazione lineare degli input del neurone k;
- b_k è il valore di soglia del neurone k;
- $\varphi(x)$ è la funzione di attivazione;
- y_k è l'output elaborato dal neurone k.

Funzioni di attivazione

Ogni neurone che compone uno strato riceve degli input dai neuroni degli strati precedenti che vengono combinati in un unico valore che rappresenta l'input della funzione di attivazione.
Le funzioni di attivazioni devo rispettare dei criteri: ricevere degli input normalizzati e fornire un output quanto più possibile prossimo ad 1 se adeguatamente stimolata. Esistono tre tipi di funzione di attivazione generalmente utilizzati nelle reti neurali.
- La *funzione gradino* di Heaviside è la più semplice, è una funzione discontinua che restituisce 0 per tutti valori negativi e 1 per tutti quelli positivi. È definita come:

$$\varphi(v) = \begin{cases} 1 \; se \; v \geq 0 \\ 0 \; se \; v < 0 \end{cases}$$

L'andamento di questa funzione è il seguente:

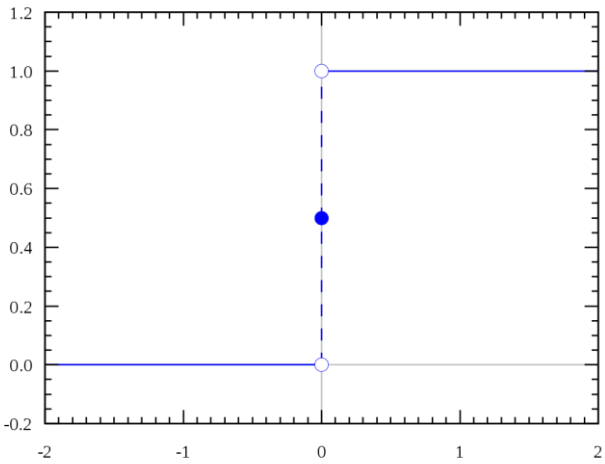

Figura 20 - funzione gradino di Heaviside

- *Funzione lineare a tratti*: spesso la funzione gradino risulta troppo semplice e viene utilizzata come base per una funzione definita ad hoc.

$$\varphi(v) = \begin{cases} 1 \; se \; v \geq \frac{1}{2} \\ v \; se - \frac{1}{2} < v < \frac{1}{2} \\ 0 \; se \; v \leq -\frac{1}{2} \end{cases}$$

L'andamento di questa funzione è il seguente:

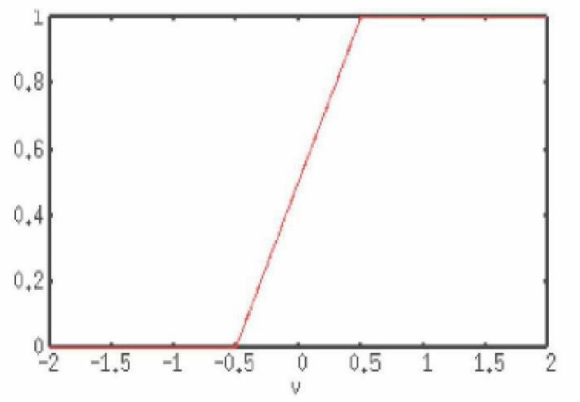

Figura 21 - funzione lineare a tratti

- *Funzioni sigmoidee:* sono le funzioni maggiormente utilizzate. Queste funzioni sono strettamente crescenti e comportano un bilanciamento fra un andamento lineare ed un andamento non lineare. Sono definite come:

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

Dove a è un parametro che permette di variare la pendenza della funzione si noti come per $a \to +\infty$ la funzione approssima l'andamento della funzione gradino. Questa funzione viene utilizzata perché spesso è utile una normalizzazione dei dati nell'intervallo [-1,1].

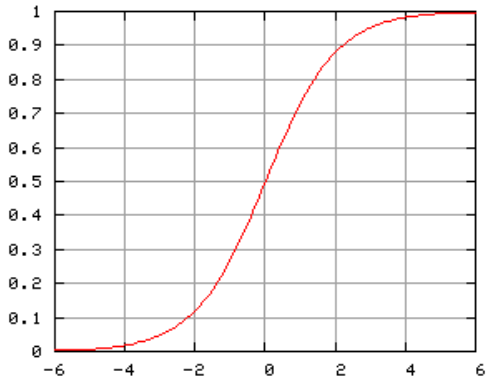

Figura 22 - Funzione sigmoidea

A questo gruppo di funzioni appartengono la tangente iperbolica $\varphi(v)=tanh(a \cdot v)$ e la funzione rettificatore $\varphi(v)=v_{+}=\max(0,v)$ indicata come ReLU (dall'inglese Rectified Linear Unit).
Lo strato finale di una rete necessita un tipo particolare di funzione di attivazione, la più utilizzata è la funzione softmax.

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{k} e^{T_l}}$$

Apprendimento

Apprendimento Supervisionato

L'apprendimento supervisionato è la soluzione più utilizzata per l'addestramento delle reti neurali. Nell'apprendimento supervisionato vengono sottoposti alla rete degli input ognuno dei quali è correlato a degli specifici output, detti etichette (labels). L'insieme di questi input e dei rispettivi labels costituisce il *training set*.
Nel processo di apprendimento i parametri liberi della rete, ossia il peso dei collegamenti, vengono modificati, attraverso una stimolazione, rispetto alle condizioni in cui essa è inserita.
Come visto in precedenza, esistono varie strategie per effettuare l'apprendimento della rete, il più utilizzato è l'apprendimento attraverso la riduzione dell'errore.

$$e_k(n) = d_k(n) - y_k(n)$$

Dove:

- $d_k(n)$ è la risposta attesa dal k-esimo neurone;
- $y_k(n)$ è la risposta generata dal sistema dal k-esimo neurone;
- $e_k(n)$ è il segnale di errore del k-esimo neurone.

Il segnale di errore attiva un procedimento di autocontrollo del sistema attraverso una serie di aggiustamenti del peso di ogni connessione del k-esimo neurone con l'obiettivo di ridurre la differenza fra la risposta generata e la risposta attesa.

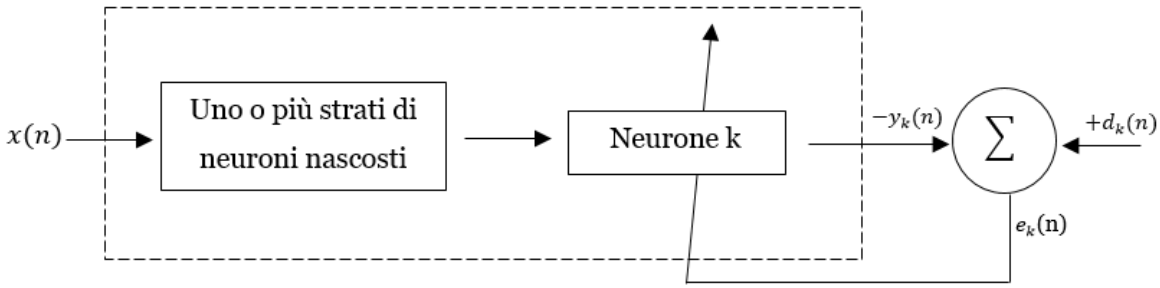

Figura 23 - apprendimento con correzione dell'errore

Questo procedimento avviene uno step alla volta andando a minimizzare una funzione costo:

$$E(n) = \frac{1}{2} \cdot e_k^2(n)$$

La minimizzazione della funzione costo avviene attraverso il metodo della discesa del gradiente. Chiamati $w_{kj}(n)$ i pesi delle connessioni del k-esimo neurone eccitati da un input $x_j(n)$ l'aggiustamento effettuato su $w_{kj}(n)$ è definito come:

$$\Delta w_{kj} = \eta \cdot e_k(n) \cdot x_j(n)$$

Dove η è una costante positiva chiamata tasso di apprendimento. I nuovi pesi così calcolati saranno:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

Il meccanismo di apprendimento attraverso la correzione dell'errore costituisce un sistema ricorrente in cui la stabilità dello stesso dipende dai parametri che lo costituiscono, in modo particolare da η poiché influenza direttamente la convergenza e la stabilità dell'intero processo.

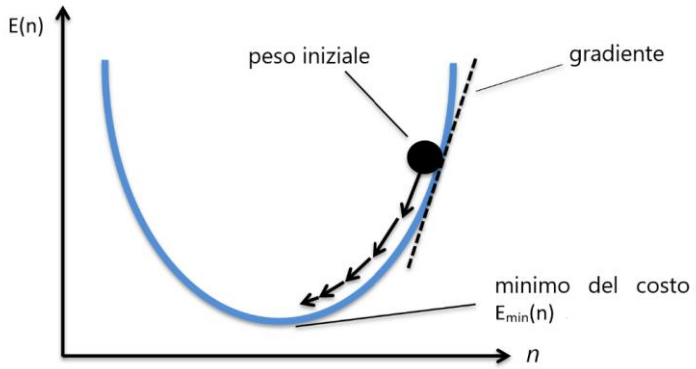

Figura 24 - discesa del gradiente

In seguito all'addestramento le prestazioni della rete sono valutate per mezzo di un insieme di input e labels differenti dal training set, detto *validation set,* attraverso il quale viene testata la capacità di generalizzazione da parte della rete (è fondamentale che questi dati non siano mai stati visualizzati in precedenza dal sistema).
I principali algoritmi per l'apprendimento supervisionato sono:

- K-Nearest neighbors (KNN);
- Logistic Regression (LR);
- Support Vector Machine (SVM);
- Decision Tree (DT);
- Random Forest (RF);
- Artificial Neural Network (ANNs).

Apprendimento non supervisionato

Nell'apprendimento non supervisionato vengono presentati alla rete un insieme di input a cui, differentemente rispetto all'apprendimento supervisionato, non sono associati a delle etichette. La classificazione degli input sarà svolta dal sistema andando ad analizzare i dati e trovando caratteristiche in comune fra di essi e provvedendo ad elaborare la classificazione.
Questo genere di apprendimento è utilizzato nei problemi di clustering, nei quali la conoscenza delle correlazioni fra i dati non è nota a priori e il compito della rete neurale consiste proprio nella loro individuazione; l'utilizzo di un apprendimento supervisionato risulterebbe impraticabile pe questa tipologia di problemi.

Attività di apprendimento

L'attività di apprendimento può essere eseguita in vari modi:

Associazione di Pattern

Viene sviluppata una memori associativa, ossia una memoria distribuita che apprende per associazioni. Le associazioni possono essere di due tipi:

- **Auto-associazioni**: vengono memorizzati i pattern presentando alla rete gli input varie volte. Quando alla rete viene presentato un nuovo input essa cerca di richiamare i pattern memorizzati anche nel caso in cui questi siano parziali o distorti. Solitamente il processo di apprendimento non è supervisionato;
- **Etero-associazioni**: durante l'apprendimento ogni pattern viene associato ad un altro pattern. Si ha dunque un pattern chiave x_k al quale viene associato un pattern y_k che viene memorizzato. Il pattern x_k funge da stimolo per richiamare il pattern y_k . Rispetto al caso precedente ($x_k = y_k$) l'abbinamento è $x_k \neq y_k$.

Il processo di associazione avviene in due fasi: nella prima, detta di memorizzazione, avviene l'associazione dei pattern, nella seconda, detta di richiamo, il pattern memorizzato viene richiamato in maniera integrale o parziale da uno stimolo.

Riconoscimento di Pattern

In questo procedimento il pattern viene associato ad una specifica classe o categoria. In seguito al processo di addestramento (nel quale il pattern viene presentato più volte) la rete è in grado di riconoscere i pattern che le vengono sottoposti in base alle categorie con le quali sono stati classificati. Nel caso in cui venisse sottoposto alla rete un pattern mai incontrato in precedenza, la rete provvederà a creare una nuova categoria sfruttando le conoscenze apprese dai pattern già noti.

Ogni pattern costituisce un singolo punto all'interno dello spazio di decisioni multidimensionale. Lo spazio decisionale è composto da varie

regioni ad ognuna delle quali è associata una categoria. Il processo di apprendimento definisce il confine fra le varie regioni.
Il riconoscimento di pattern può essere effettuato in due modi:

- La rete viene suddivisa in due parti. Una sottorete non supervisionata che estrapola le caratteristiche ed una sottorete supervisionata che effettua la classificazione;
- L'attività di estrapolazione e classificazione avviene in ogni singolo strato nascosto che compone la rete.

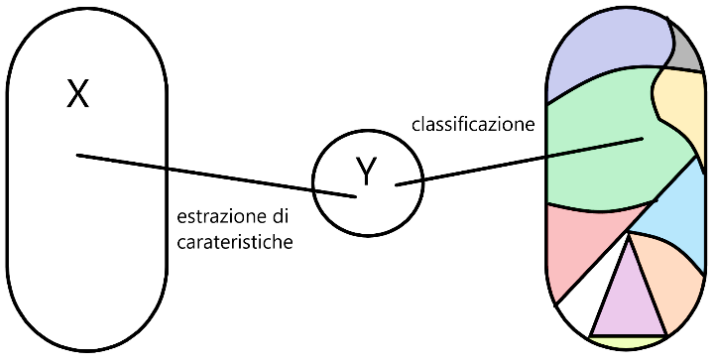

Figura 25 - classificazione dei pattern

Approssimazione di funzioni

Una mappatura input-output non lineare può essere descritta dalla relazione:

$$d = f(x)$$

Dove:

- x rappresenta l'input;
- d rappresenta l'output;
- f è la funzione incognita che relazione input e output.

Vengono fornite numerose coppie di elementi input-output descritte da dalla relazione f:

$$\{(x_i, d_i)\}_{i=1}^{n}$$

Queste coppie vengono usate dalla rete neurale per creare una funzione F che approssimi la funzione f tale per cui la mappa di input e output realizzata dalla rete sia vicina in senso euclideo ad ogni input:

$$||F(x) - f(x)|| < \varepsilon, \forall x$$

Dove ε è un numero positivo sufficientemente piccolo, tanto più piccolo quanto maggiore è il numero di parametri liberi che compongono la rete. L'approssimazione di funzioni è particolarmente indicata per l'apprendimento di supervisionato che utilizza vettori in input.

Generalizzazione

Lo scopo finale di una rete è quello di sviluppare la capacità di generalizzare efficacemente una tipologia di problemi. Una generalizzazione è corretta quando la mappa di input e output generata durante la fase di apprendimento è corretta per degli esempi di test ma visionati in precedenza dalla rete (il set di dati usati per il test appartiene alla medesima popolazione dei dati usati per il training).

La capacità di generalizzare è simile ad un problema di approssimazione di una curva: considerando la rete neurale come una mappa non lineare degli input e degli output, una buona generalizzazione equivale ad una buona interpolazione rispetto input dati.

Una generalizzazione efficacie consente di produrre una mappatura corretta anche se gli input utilizzati sono lievemente differenti rispetto a quelli utilizzati durante l'addestramento; una rete che ha visionato troppi esempi (coppie input output) nella fase di training corre il rischio di perdere la capacità di generalizzare correttamente andando a memorizzare informazioni non necessarie. Questo fenomeno è chiamato overfitting.

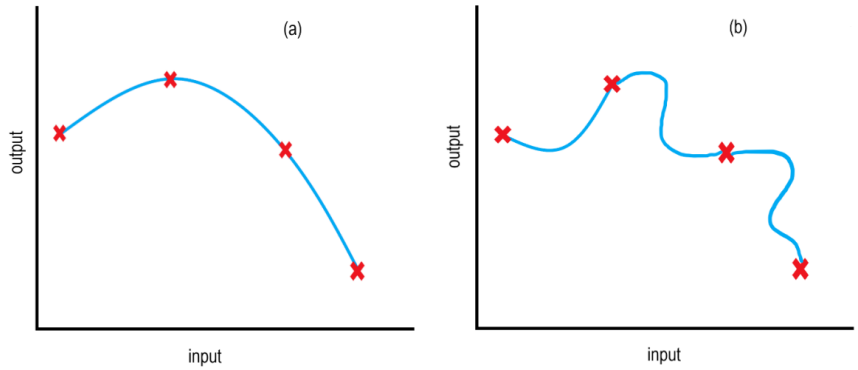

Figura 26 - confronto fra una buona generalizzazione (a) e una cattiva generalizzazione (b) dovuta ad overfitting

La generalizzazione è funzione di tre fattori:

- La dimensione del set di dati usato per addestrare il modello;
- La dimensione e il tipo di reti neurali;
- La complessità del problema analizzato.

La complessità dal problema che si vuole studiare con la rete neurale è un fattore sul quale non è possibile apportare modifiche, dunque per migliorare le prestazioni della rete occorre intervenire sugli altri due punti. Nel caso in cui la struttura della rete sia prefissata bisogna stabilire la dimensione ottimale del set di dati da usare nell'addestramento, nel caso in cui sia prefissata la quantità di dati del training set bisogna determinare la struttura ottimale della rete.
Una regola empirica che consente di ottenere buone generalizzazioni consiste nel rispettare la seguente condizione:

$$N = O\left(\frac{W}{\varepsilon}\right)$$

Dove:

- N è la dimensione del set di dati usato per l'addestramento;
- W è il numero di parametri liberi della rete;
- ε è la frazione di errori di classificazione sul set di tati usato per il test.

Di seguito è riportato un diagramma di flusso che schematizza il processo di addestramento di una rete neurale.

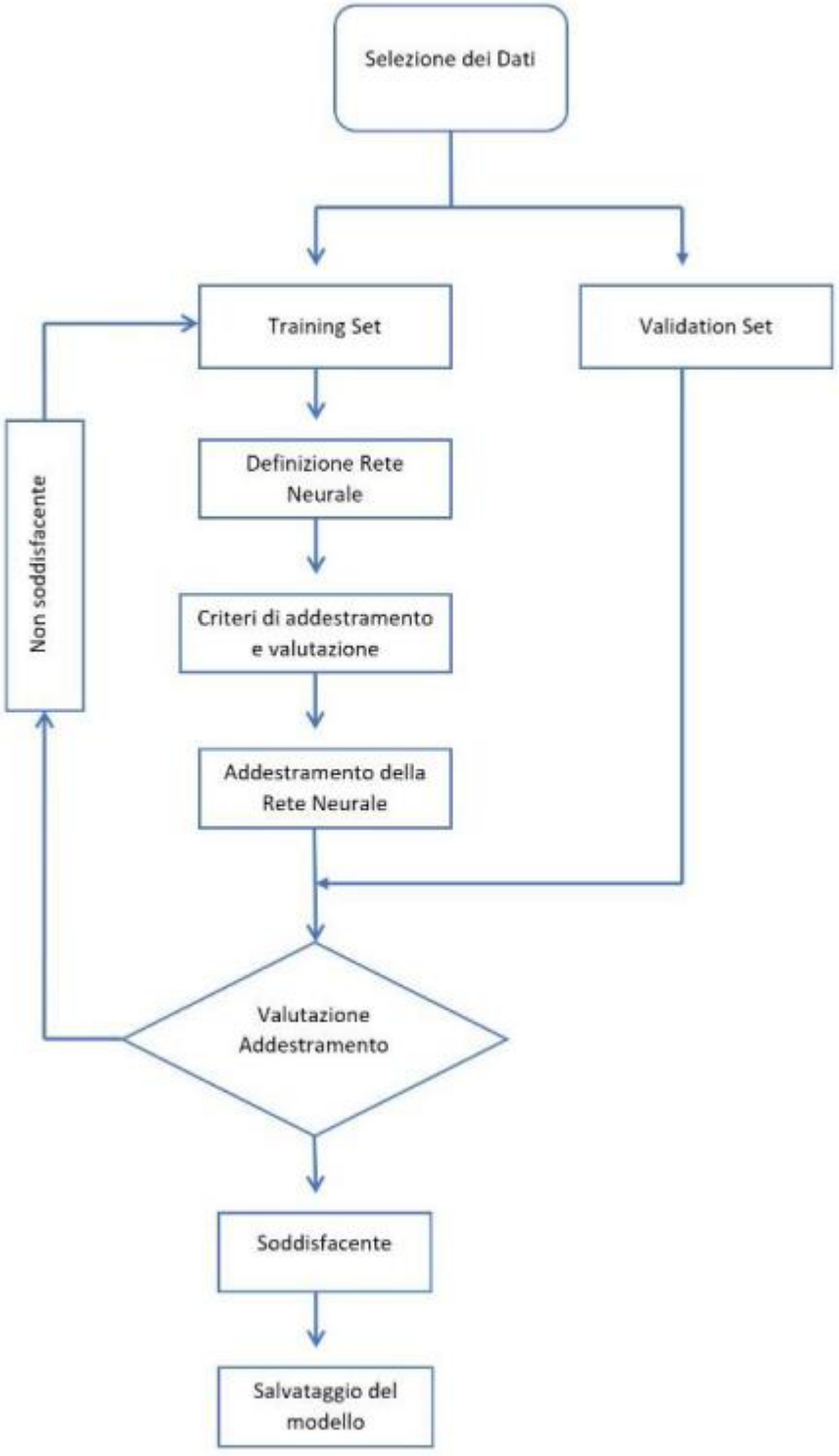

Figura 27 - processo di addestramento di una rete neurale

La funzione obiettivo e l'ottimizzatore

Il modello creato necessita di avere una funzione da minimizzare(Loss) ed un ottimizzatore. In base al fenomeno analizzato si adoperano diverse

funzioni, le due più utilizzate sono *binary_crossentropy* (nel caso in cui occorra calcolare delle probabilità) e *mean_squared_error*(usata nei modelli di predizione). Esistono vari ottimizzatori implementati nelle API Keras, ognuno con le sue peculiarità ed adatti ad uno specifico modello di calcolo. Il più utilizzato l'*AdamOptimizer*, particolarmente preformante nelle applicazioni stocastiche.

Prevenzione di Overfitting e Underfitting

Come visto in precedenza il modello viene allenato a riconoscere un determinato pattern attraverso la fase di apprendimento e ci si aspetta che sia in grado di *generalizzare* questo pattern anche con dati differenti, ossia di predire i risultati anche su dati che non ha mai visto in precedenza. Tuttavia, se l'apprendimento è stato eccessivo si corre il rischio che il modello perda la capacità di generalizzare poiché è troppo adattato ai dati usati per istruirlo (tipicamente si focalizza su alcuni aspetti tipici esclusivamente del training set, ma assenti nella maggior parte degli altri casi), questo fenomeno è detto ***overfitting***.
Esiste anche l'eventualità contraria, cioè che il modello non sia stato addestrato in maniera sufficiente. Questo fenomeno può verificarsi per vari motivi: i dati analizzati non sono organizzati in maniera adeguata, il modello richiede una complessità della rete (numero di strati e di neuroni) maggiore o i dati analizzati non sono abbastanza. Il tutto si traduce di nuovo nell'incapacità di generalizzare.
Spesso risulta difficile valutare la qualità di un modello in maniera univoca, ma esistono alcune regole empiriche che permettono di effettuare una valutazione qualitativa sull'addestramento della rete.
È buona norma implementare una cross-validation, ossia la divisione del training set in due insiemi distinti:

- Training sub-set: effettivamente usato per l'addestramento del modello;
- Validation sub-set: usato per la validazione del modello.

Osservando l'andamento reciproco dell'accuratezza dei due set di dati all'aumentare delle epoche dell'addestramento è possibile individuare eventuali fenomeni di overfitting e underfitting.

Un modello in underfitting ha una precisione migliore sul training sub-set rispetto al validation sub-set e questo può essere facilmente identificato se la curva del training sta al di sotto della curva del validation e quest'ultimo ha un andamento che suggerisce un miglioramento nel caso di un numero maggiore di epoche.

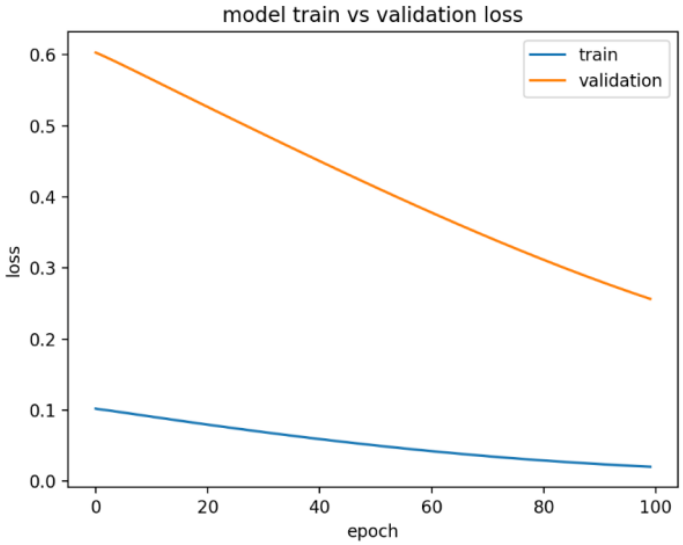

Figura 28 - modello in underfitting

Un modello con delle buone performance presenta un andamento convergente del training e validation set, in cui entrambi diminuiscono e si stabilizzano attorno allo stesso valore.

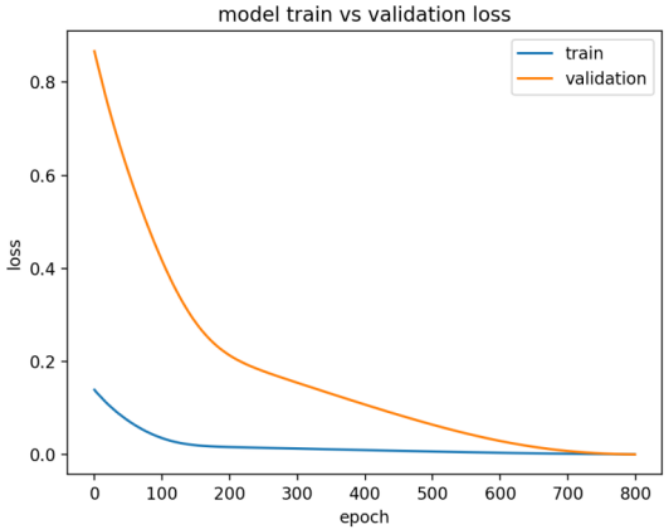

Figura 29 - modello con un addestramento efficace

In fine un modello affetto da overfitting è caratterizzato da prestazioni del training set che continuano ad aumentare mentre quelle del validation, ad un certo punto, tendono a peggiorare. Si nota solitamente che la curva di validation ha un minimo oltre il quale cambia pendenza.

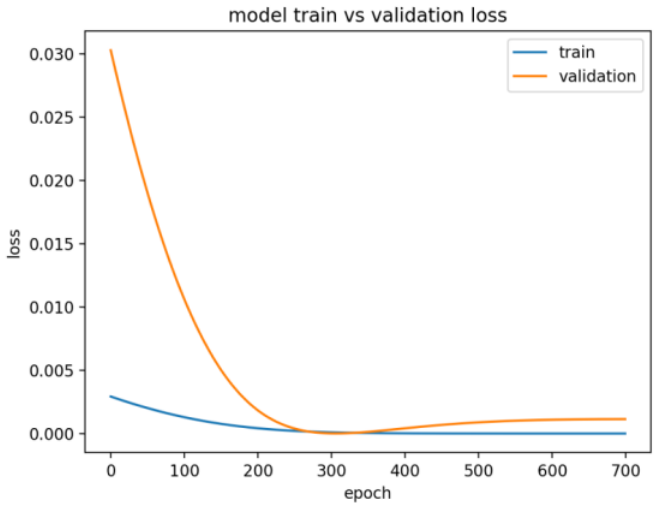

Figura 30 - modello in overfitting

Tensorflow, grazie alle API Keras, consente di implementare alcune funzioni utili a prevenire l'overfitting e l'underfitting, di seguito sono illustrate quelle utilizzate in questa analisi.

Early Stopping

In linea di principio il modo migliore per addestrare il modello consiste nel fargli visionare il maggior numero di dati differenti possibile, ma come abbiamo visto il rischio di incorrere in overfitting è sempre presente e, purtroppo, non esistono dei parametri analitici che possono indicare in maniera univoca la presenza di overfitting. Si può tuttavia ricorrere ad alcuni accorgimenti nella scrittura del codice che permettono di ridurre al minimo l'overfitting o riuscire ad intuire quando esso si verifica.

Una possibilità consiste nell'implementare una funzione che controlli il miglioramento della predizione rispetto ai dati attesi mano a mano che aumenta il numero di epoche ed interrompere l'addestramento nel momento in cui non si registrino più miglioramenti significativi. Questo metodo è detto "early stopping" ed è facilmente implementabile in Tensorflow attraverso una specifica funzione.

```
model = build_model()

# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)

history = model.fit(train_data, train_labels, epochs=10000,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history(history)
```

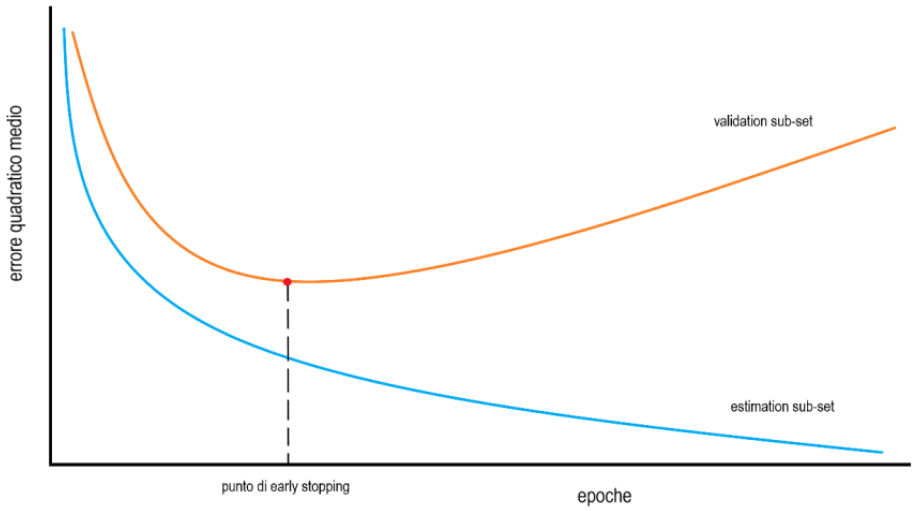

Figura 31 - metodo early stopping

Questa operazione viene fatta per validare la rete con dei dati differenti da quelli utilizzati nella fase di training, è così possibile valutare il set di addestramento calcolando l'errore alla fine di ogni epoca.

Osservando l'andamento dell'errore quadratico medio riferito all'estimation sub-set si nota come esso si riduca all'aumentare delle epoche portando a pensare che la precisione aumenti in maniera inversamente proporzionale ad esse. Tuttavia una volta superato il punto di early stopping la rete apprende solamente il rumore contenuto nei dati, quindi proseguire l'addestramento risulta dannoso.

Weight regularization

Un'altra possibilità è consiste nel "weight regularization": il modello che si viene a creare all'interno della rete è determinato dal peso specifico che ha ogni connessione fra i nodi della rete, più un modello è complesso e maggiore è il numero di possibili combinazioni che possono descriverlo

efficacemente. Per mitigare l'overfitting si può forzare il modello a sviluppare il set più semplice possibile aggiungendo un costo al alla funzione obiettivo associato ad avere dei pesi elevati.

Può avvenire in due modi:

- Regolarizzazione **L1**: il costo aggiunto è proporzionale al valore assoluto del peso dei coefficienti;
- Regolarizzazione **L2**: il costo aggiunto è proporzionale al quadrato del peso dei coefficienti. Questa tipologia di regolarizzazione è detta anche *decadimento del peso*.

Anche in questo caso è prevista una funzione specifica da implementare quando si crea il modello.

```
model = keras.Sequential([
    keras.layers.Dense(128, kernel_regularizer=keras.regularizers.l2(0.001),
                       activation=tf.nn.relu)
    keras.layers.Dense(10, activation=tf.nn.softmax) ])
```

Dropout

L'ultimo metodo, detto "dropout", consiste nell'eliminare alcuni output di un layer durante la fase di apprendimento. Questo sistema consiste nell'eliminazione casuale di un certo numero valori di un layer durante l'addestramento.

Immaginiamo di avere un layer che, dato un input, restituisca in output il vettore V= [0.2, 0.5, 1.3, 0.8 1.1]; applicando il drop out alcuni elementi del vettore vengono modificati in zero, ottenendo un novo vettore in output V^1= [0, 0.5, 1.3, 0 1.1]. Il rateo di dropout è la frazione dei valori che vengono tramutati in zero, solitamente compreso fra il 20% e il 50%.

```
dpt_model = keras.models.Sequential([
    keras.layers.Dense(16, activation=tf.nn.relu, input_shape=(NUM_WORDS,)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(16, activation=tf.nn.relu),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation=tf.nn.sigmoid) ])
```

Il dropout viene applicato attraverso lo specifico comando implementato.

Classificazione delle reti neurali

Esistono molti tipi di reti neurali, possono essere suddivisi in base alla struttura, agli operatori matematici che implementano e alle applicazioni in cui vengono utilizzate, ognuna con i suoi pro e contro. Le principali categorie sono riassunte nella seguente tabella:

Tipologia di rete	**Ottime prestazioni**	**Buone prestazioni**
Feed Foreward	-Dataset tabulari -Prevedere una classificazione -Prevedere una regressione	-Classificazione di immagini -Classificazione di testi -Serie temporali
Convoluzionali	-Classificazione di immagini -Prevedere una classificazione -Prevedere una regressione	-Classificazione di immagini -Serie temporali -Sentiment analysis
Ricorrenti	-Classificazione di testi -Comprensione del linguaggio -Prevedere una regressione -Prevedere una classificazione	-Serie temporali

Feedforeward

Le reti feedforeward costituiscono il modello più semplice di struttura neurale. In queste reti le informazioni si muovono sempre in un'unica direzione e i layer intermedi sono densi, ossia ogni neurone è connesso con tutti i neuroni dello strato precedente e successivo (*fully connected)*. La struttura molto semplice consente a questo tipo di rete di essere molto performante nei problemi di classificazione e regressione. Hanno lo svantaggio di essere tendenti all'overfitting quando la rete diventa molto complessa (molti layer e molti neuroni).

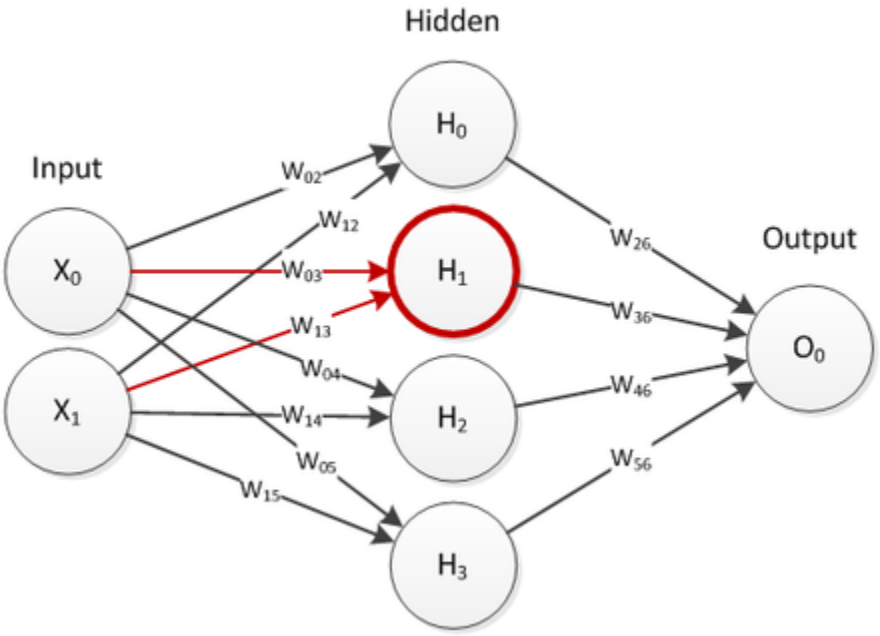

Figura 32 - rete feedforeward con uno strato nascosto

Le reti dette Radial Basis network sono un particolare tipo di reti feedforeward, si differenziano dal caso generale per l'utilizzo di un particolare tipo di funzione invece della classica funzione logistica (Sigmoid). Le funzioni logistiche generano una scala di valori da 0 ad 1 e possono rispondere a domande con risposte binarie (sì o no), tuttavia lavorano male quando devono maneggiare valori continui, le funzioni radial basis, invece, valutano la distanza del risultato ottenuto dall'obiettivo che si vuole raggiungere. Risultano quindi ottime per problemi in cui si verifica necessariamente un'approssimazione, come, ad esempio, il controllo di macchine operatrici.

Questo genere di reti è ottimo per i problemi di predizione dove gli input forniti sono classificati in label e nei problemi di regressione dove occorre predire dei valori basandosi su di uno storico. Data la notevole flessibilità possono tuttavia essere utilizzate anche per il riconoscimento di immagini e testi.

Convoluzionali

Le reti convoluzionali elaborano le informazioni sotto forma di array multipli e sono le più utilizzate per il riconoscimento di immagini.

Estraggono le caratteristiche di un'immagine attraverso la convoluzione, partendo dai bordi, e man mano che si procede attraverso la rete migliora il livello di precisione con la quale vengono riconosciuti i dettagli che formano l'immagine. Un'immagine a colori, ad esempio, viene rappresentata come tre array bidimensionali contenenti l'intensità dei tre colori primari per ogni singolo pixel.

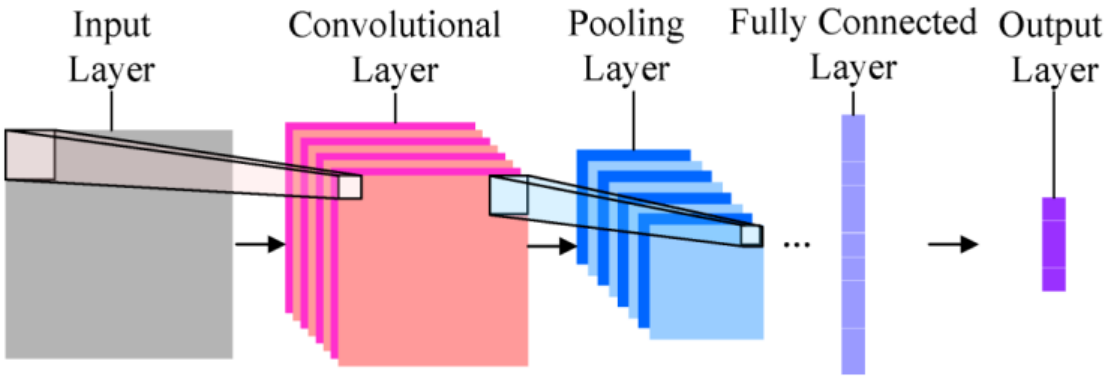

Figura 33 - rete convoluzionale

La principale differenza rispetto alle altre tipologie di reti neurali è costituita dalla loro capacità di operare direttamente sulle immagini e non sulle caratteristiche estratte dalle stesse. Le reti convoluzionali sono costituite da blocchi strati: i primi blocchi sono *convolutional layer* e i secondi *pooling layer*.

Le reti convoluzionali utilizzano il concetto di connettività locale, cioè la capacità di sfruttare le correlazioni spaziali presenti all'interno dei dati in input e che si propagano lungo gli strati della rete. Gli input dei neuroni dello strato *n* sono gli output di un sottoinsieme dei neuroni dello strato *n-1* e hanno campi ricettivi con una contiguità spaziale.

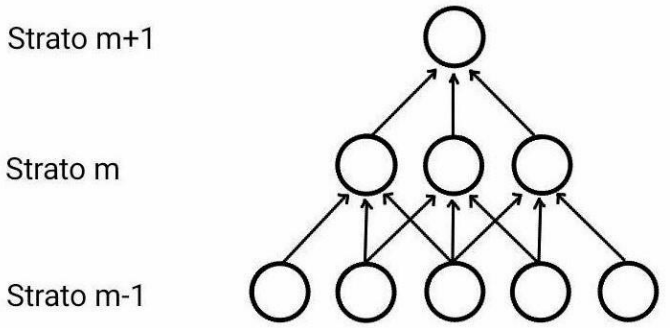

Figura 34 - struttura interna

Supponendo che lo strato m-1 sia quello di input, i nodi dello strato m risultano avere un campo di ricezione con ampiezza 3 rispetto all'input, poiché ognuno di essi è connesso con solo tre nodi del layer precedente, mentre in nodi dello strato n+1 hanno un'ampiezza di 3 rispetto allo strato n e un'ampiezza di 5 rispetto allo strato n-1. Ogni unità di un layer elabora le informazioni che appartengono al proprio campo ricettivo indipendentemente da eventuali variazioni esterne ad esso. Andando a sovrapporre più strati connessi fra di loro le informazioni elaborate da ogni noto hanno un carattere sempre più globale man mano che si sale di livello, aumentando il livello di astrazione. La connettività locale, dunque, consiste nel fatto che i dati organizzati sotto forma di array, in particolare nel caso di immagini, i sottoinsiemi locali e adiacenti di dati sono fortemente correlati fra loro e possono essere agevolmente riconosciuti dalla rete.
I nodi degli strati convoluzionali che compongono la rete sono connessi ai sottoinsiemi dei nodi degli strati precedenti per mezzo di un insieme di pesi, chiamato *filter bank,* con cui viene svolto il processo di convoluzione vero e proprio. Il valore dei pesi è inizialmente casuale e vengono mano a mano aggiornati durante l'addestramento della rete per mezzo dell'algoritmo di back propagation.
Un'altra peculiarità delle reti convoluzionali è la condivisione dei pesi, i nodi degli strati convoluzionali sono organizzati in gruppi detti feature map condividendo fra di essi il valore di pesi. Le feature map comprendono tutti gli input del layer precedente e, per mezzo del filter bank condiviso, elaborano l'immagine e possono andare individuarne le caratteristiche peculiari indipendentemente dalla loro posizione.
Il vantaggio nell'utilizzo di queste reti è la loro abilità di sviluppare una rappresentazione interna di immagini 2D, permettendo al modello di imparare la posizione e la scala di molteplici strutture (i.e. riconoscere un gilè da una camicia dalla presenza o meno di maniche).

Queste reti sono particolarmente performanti la classificazione di immagini e, in generale, con tutti i dati legati relazioni spaziali (come la disposizione di parole in un testo, *sentiment analysis*).

Recurrent-Long Short Term Memory

Le reti ricorrenti, a differenza delle altre tipologie di reti, nelle quali gli input vengono trattati come ognuno indipendente dagli altri, analizzano le informazioni in modo sequenziale tenendo conto dell'ordine con cui essi sono presentati al sistema.

La rete elabora gli elementi sequenziali uno per volta mantenendo la memoria di quelli precedenti e hanno la capacità di propagare i dati non solo in avanti ma anche in senso inverso. Questa caratteristica è particolarmente utile quando si analizzano dati di lunghezza variabile, come frasi o tracce audio. Sono utilizzate per estrarre un elemento (ad esempio una singola parola) e capire il suo significato all'interno del contesto.

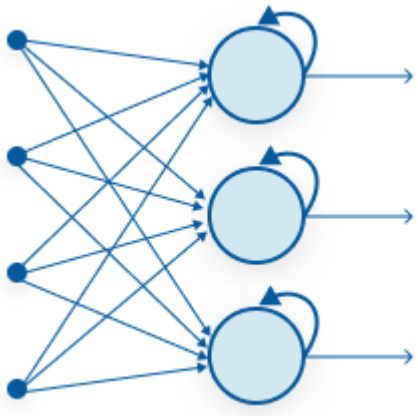

Figura 35 - rete ricorrente

Durante la fase di addestramento queste reti possono essere affette dal fenomeno di *vanishing* o *exploding gradient,* poiché i gradienti che vengono propagati all'indietro hanno la tendenza ad aumentare o diminuire per ogni istante temporale e far quindi divergere verso infinito o convergere verso zero.

Per ovviare a questo problema le reti LSTM sono dotate di particolari unità nascoste, dette celle di memoria che hanno la capacità di ricordare

memorizzare gli input precedenti per lunghi periodi. Queste celle comparano l'input dell'istante corrente con quello dell'istante precedente e decidono quale mantenere in memoria, garantendo la capacità di memorizzare le informazioni importanti durante tutto il procedimento di apprendimento.

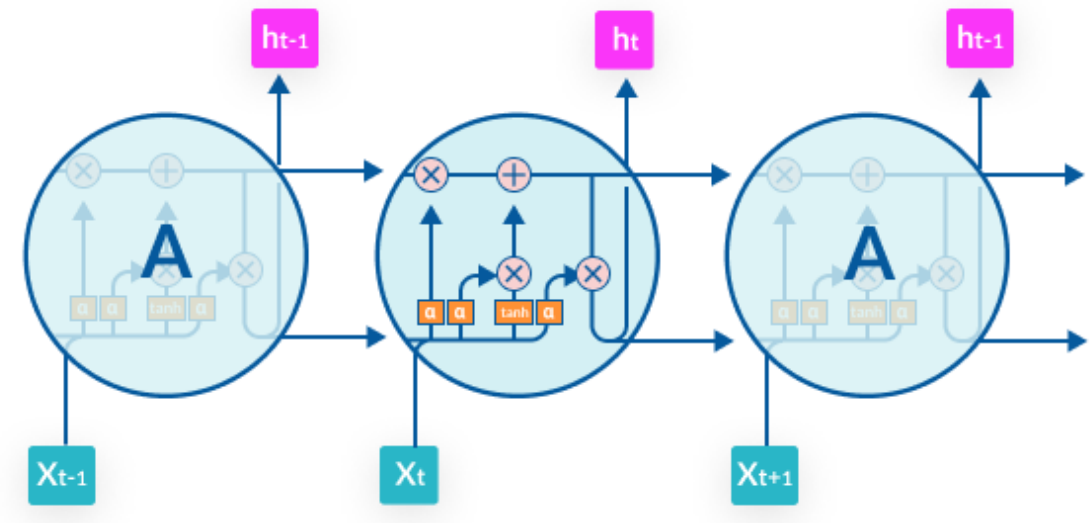

Figura 36 - celle LSTM

In LSTM lo stato h_t è suddiviso in due vettori h_t e x_t :

- h_t è uno stato (o memoria) a breve termine;
- x_t è uno stato (o memoria) a lungo termine;
- La cella apprende durante il training cosa è importante dimenticare (forget gate) dello stato passato x_{t-1} e cosa estrarre e aggiungere (input gate) dall'input corrente x_t ;
- Per il calcolo dell'output si combina l'input corrente (output gate) con informazioni estratte dalla memoria a lungo termine.

Sono state progettate per lavorare con problemi di predizione sequenziale, ossia problemi in cui è fondamentale da dipendenza temporale. Risultano quindi molto performanti nella comprensione di un discorso (sia scritto sia in forma audio) e per generare modelli che forniscono output in sequenza. Non sono adatti per la classificazione di immagini.

Progettazione e Implementazione

In questo studio si andrà a sviluppare un modello di rete neurale in grado di predire delle tipologie di attacco effettuate sul CAN-bus, il modello estrapolerà dei pattern temporali che ci permetteranno di distinguere se gli attacchi forniti siano dei malware o dei goodware.

Impostazione dello studio

Prima di procedere con lo sviluppo di un modello di predizione basato sulle reti neurali, è necessario compiere una fase preliminare in cui si impostano i seguenti step di lavoro:

- **Definire l'obiettivo di studio:** la rete neurale sarà sviluppata con lo scopo di fornire una previsione relativa al tipo di attacco eseguito sul CAN-bus, distinguendo se esso sia benevolo o malevolo;
- **Acquisire e preparare i dati necessari per l'analisi:** i set di dati utilizzati sono divisi per tipologia di attacchi, per cui ogni intrusione di una certa tipologia di attacco viene distinta da una variabile "Flag" che indica se l'intrusione è buona o cattiva;
- **Scegliere il modello da utilizzare:** come visto nel capitolo precedente, esistono numerose tipologie differenti di reti neurali, ognuna con le sue caratteristiche e peculiarità per cui è fondamentale utilizzare un modello performante per la tipologia di studio che si intende effettuare;
- **Sviluppare il modello:** una volta scelto il modello più appropriato si può sviluppare ed addestrare la rete neurale;
- **Valutare il modello:** l'ultimo passaggio consiste nella valutazione del modello, analizzando le previsioni fornite.

I dati in nostro possesso costituiscono una serie di valori legati fra loro da correlazioni temporali da utilizzare per compiere una regressione e

prevedere l'andamento futuro. Per questo motivo, si è scelto di utilizzare la rete ricorrente Long Short Term Memory (LSTM).
Le reti ricorrenti prevedono collegamenti all'indietro o verso lo stesso livello.
A ogni step della sequenza (ad esempio a ogni istante temporale t) il livello riceve, oltre all'input $x_{(t)}$, anche il suo output dello step precedente $x_{(t-1)}$. Questo consentirà alla rete di basare le sue decisioni sulla storia passata (effetto memoria) ovvero su tutti gli elementi della sequenza che andremo a stabilire e sulla loro posizione reciproca.

Set di dati

Per poter sviluppare il modello, è stato utilizzato un set di dati di car-hacking per il rilevamento delle intrusioni, che includono attacchi DoS, Fuzzy e Spoofing. I dati sono stati creati registrando il traffico CAN attraverso la porta OBD-II, da un veicolo reale, mentre si stavano eseguendo attacchi di injection message. Ogni intrusione è stata eseguita per circa 3-5 secondi e ogni set di dati ha circa 30-40 minuti di traffico effettuato sul CAN.
Quello che si può notare attraverso i vari dataset comprende:

- **DoS Attack:** il CAN ID '0000' dura ogni 0.3 millisecondi ed è il più dominante;
- **Fuzzy Attack:** i valori dei CAN ID e DATA sono totalmente casuali ogni 0,5 millisecondi;
- **Spoofing Attack:** la maggior parte dei CAN ID ha la durata di 1 millisecondi.

Inoltre, i dati sono rappresentati in questo modo:

- **Timestamp:** rappresenta la data e l'ora registrata.
- **CAN ID:** identificatore del messaggio CAN in esadecimale (es. 043f).
- **DLC:** numero di byte dei dati, da 0 a 8.

- **DATA [0 ~ 7]:** valore dei dati (byte).
- **Flag:** T o R, dove T rappresenta il messaggio iniettato mentre R rappresenta il messaggio normale.

	Timestamp	CAN ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7	Flag
0	1.478198e+09	0316	8	05	21	68	09	21	21	00	6f	R
1	1.478198e+09	018f	8	fe	5b	00	00	00	3c	00	00	R
2	1.478198e+09	0260	8	19	21	22	30	08	8e	6d	3a	R
3	1.478198e+09	02a0	8	64	00	9a	1d	97	02	bd	00	R
4	1.478198e+09	0329	8	40	bb	7f	14	11	20	00	14	R

Figura 37 - Esempio di rappresentazione del set di dati

Strumenti

Gli strumenti utilizzati nello sviluppo del modello sono:

- **Anaconda:** una distribuzione gratuita e open source dei linguaggi di programmazione *Python* e *R* per il calcolo scientifico (Data Scienze, tecniche di machine learning, elaborazione di dati su larga scala, analisi predittiva ecc.), che mira a semplificare la gestione dei pacchetti. Le versioni dei pacchetti sono gestite dal sistema di gestione dei pacchetti *conda* che installa, esegue e aggiorna i pacchetti e le loro dipendenze.

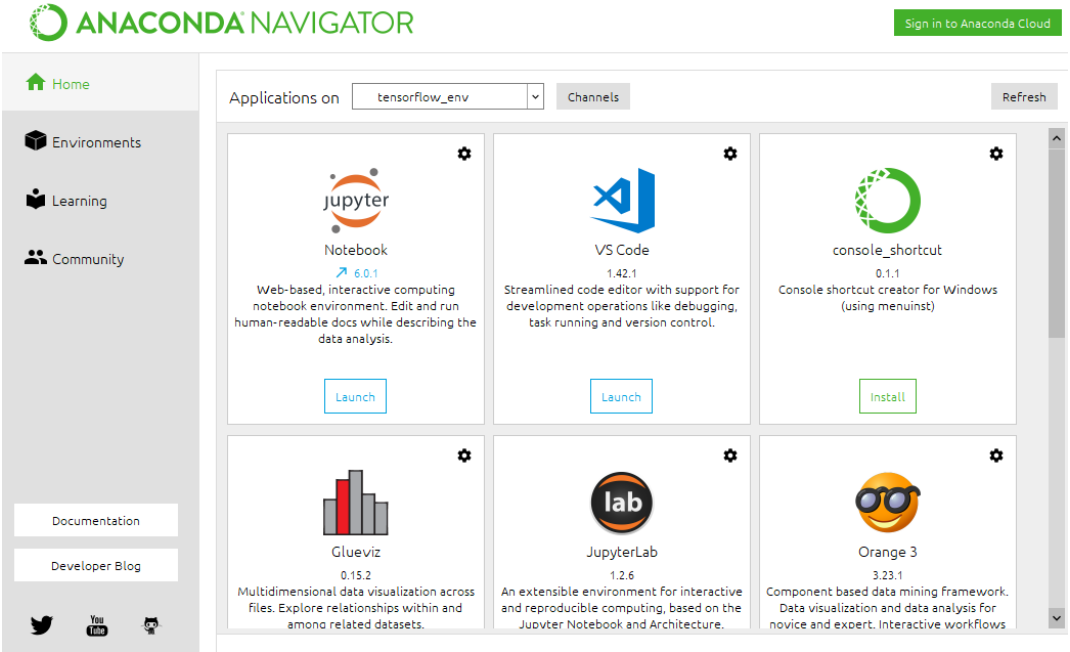

Figura 38 - Interfaccia grafica di Anaconda Navigator

- **Notebook Jupyter:** un'applicazione basata sul modello client-server che permette la creazione e la condivisione di documenti web nel formato JSON, che seguono uno schema e una lista ordinata di celle input/output. I due componenti centrali sono un set di diversi *kernel* (interpreti) e la *dashboard*. Un kernel standard è IPython, un interprete della riga di comando che permette di lavorare con Python. La dashboard serve da una parte come interfaccia di gestione per i singoli kernel e dall'altra come centro per la creazione di nuovi documenti Notebook o per aprire progetti già esistenti. Notebook Jupiter è molto utilizzato per la pulizia dei dati, modellazione statistica, creazione di modelli di machine learning e rappresentazione dei dati.

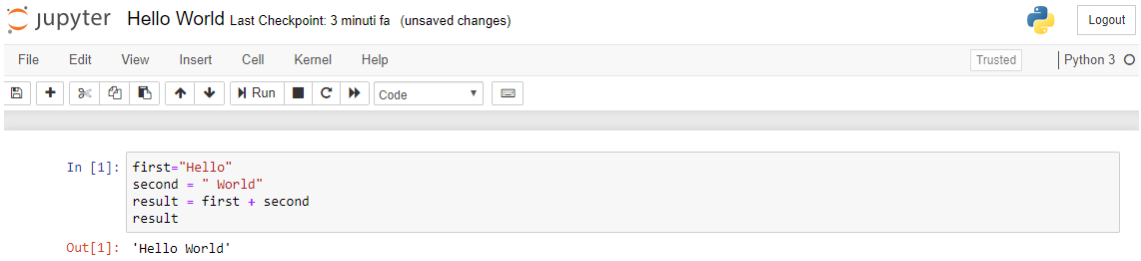

Figura 39 - Dashboard Notebook Jupyter

- **Python:** è un linguaggio di programmazione ad alto livello adatto alla computazione numerica, gratuito e molto flessibile. Sono disponibili moltissime librerie, anch'esse gratuite, che ampliano enormemente le possibilità di sviluppo. Rispetto a molti altri software che permettono di sviluppare algoritmi di machine learning, Python può essere utilizzato anche senza avere una conoscenza approfondita del linguaggio ed è supportato da una vastissima community on-line.
- **Tensorflow**: è una libreria per il machine learning sviluppata da Google che fornisce moduli testati ed ottimizzati molto facili da usare per merito della sintassi estremamente sintetica ed intuitiva. Include al suo interno le API Keras, una delle librerie più utilizzate nell'ambito dell'apprendimento automatico. Tensorflow è

pienamente compatibile con i linguaggi Python, C++, Java, R e Scala. Se ne consiglia l'utilizzo su macchine dotate di una scheda per l'accelerazione grafica dedicata, in quanto i calcoli sui tensori risultano molto più performanti se eseguiti da una GPU rispetto ad una CPU (non è possibile utilizzare l'accelerazione grafica nei sistemi MacOs di Apple).

- **Scikit-Learn**: è una libreria Python per il machine learning. Al suo interno sono presenti numerosi comandi e funzioni per la classificazione, il clustering, la regressione e l'analisi statistica. Questa libreria è ideata per funzionare assieme a Numpy.
- **Numpy**: acronimo di Numeric Python, è una delle più diffuse librerie per applicazioni scientifiche che include moltissime funzioni precompilate per calcoli matematici e scientifici. È molto performante nella computazione di array multidimensionali e matrici.
- **Pandas**: è una libreria utilizzata per manipolare database in modo semplice e intuitivo, progettata per funzionare con archivi relazionali e labeled. Risulta quindi molto utile nella creazione di dataframe da utilizzare in un addestramento supervisionato (il sistema adoperato in questo studio). Viene utilizzato in coppia con Numpy.
- **Matplotlib**: è una libreria per la creazione di grafici per il linguaggio di programmazione Python e la libreria matematica di Numpy. Fornisce API orientate agli oggetti che permettono di inserire grafici all'interno di applicativi usando toolkit GUI generici.

Indici utilizzati

Per valutare la bontà delle previsioni esposte di seguito, sono stati utilizzati alcuni indici caratteristici. In particolare, si fa riferimento a:

Errore quadratico medio: è una misura utilizzata per quantificare la qualità della stima effettuata.

L’errore quadratico medio misura quanto i valori predetti si discostano da quelli reali.

$$MSE = \frac{\sum_{i=1}^{N}(real_i - pred_i)^2}{N}$$

In particolare, essa valuta per ogni dato la distanza esistente tra il punto predetto e il corrispondente reale. Il quadrato è inserito per evitare che i valori negativi cancellino i positivi.

Matrice di Confusione: Rappresentazione tabellare del risultato di una classificazione. Ogni riga rappresenta un'istanza di una classe predetta, mentre ogni colonna rappresenta un’istanza di una classe reale.

Consente di capire le reali prestazioni di un classificatore anche nel caso di dataset sbilanciati. Un vantaggio di questa rappresentazione è la possibilità di individuare subito se il sistema confonde le classi.

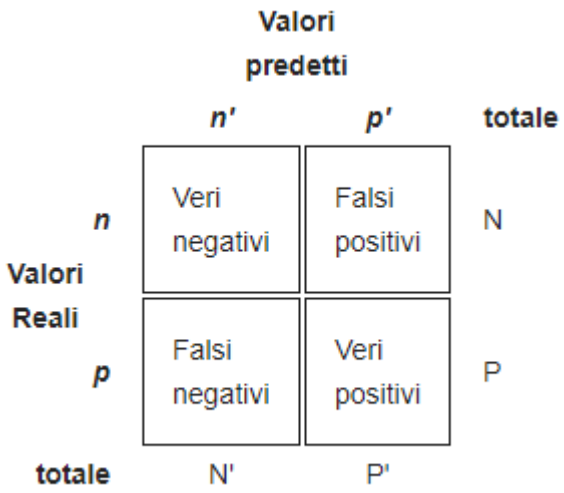

Figura 40 - Matrice di confusione

Accuracy: è una misura statistica, utilizzata per quantificare quanto bene funzioni un classificatore nell’identificare le classi. Essa è definita come rapporto tra tutti i risultati true rispetto all’intera popolazione.

$$Acc = \frac{VP + VN}{VP + VN + FP + FN}$$

Preparazione e formattazione dei dati

I dati a nostra disposizione sono disponibili in file .csv. Questo formato viene utilizzato perché è molto leggero e consente una lettura più veloce da parte del programma rispetto ad un file .xlsx (file in formato Excel).
Come è stato descritto precedentemente, ogni riga rappresenta un'intrusione effettuata sul CAN-bus. I dati a nostra disposizione sono:

- Il tempo di esecuzione dell'intrusione;
- Il codice identificativo del comando;
- Il numero di byte utilizzati;
- Il valore dei byte;
- Il valore di flag che ci indica se l'intrusione è buona o cattiva.

Iniziamo il primo passo nella scrittura del programma organizzando i dati a nostra disposizione. Il primo dataset che andiamo ad analizzare è quello degli attacchi Denial of Service, comunemente chiamati DoS.

```
dos = pd.read_csv("C:/Users/pio-f/Desktop/DoS_dataset.csv")
dos
```

	Timestamp	CAN ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7	Flag
0	1.478198e+09	0316	8	05	21	68	09	21	21	00	6f	R
1	1.478198e+09	018f	8	fe	5b	00	00	00	3c	00	00	T
2	1.478198e+09	0260	8	19	21	22	30	08	8e	6d	3a	R
3	1.478198e+09	02a0	8	64	00	9a	1d	97	02	bd	00	T
4	1.478198e+09	0329	8	40	bb	7f	14	11	20	00	14	R
...	...	...	...	...	...	...	...	...	...	...	...	...

Figura 41 - Anteprima del DoS dataset

Dalla figura 41, possiamo notare che i dati sono rappresentati in formato esadecimale, quindi il primo passo da compiere per la fase di pre-processing dei dati consiste nel convertire i dati in un formato facilmente leggibile per il nostro modello.

```
dos = pd.read_csv("C:/Users/pio-f/Desktop/DoS_dataset.csv", converters={"DATA0":lambda x: int(x, 16),"DATA1":lambda x: int(x, 16),"DATA2":lambda x: int(x, 16),"DATA3":lambda x: int(x, 16),"DATA4":lambda x: int(x, 16),"DATA5":lambda x: int(x, 16),"DATA6":lambda x: int(x, 16),"DATA7":lambda x: int(x, 16)})
dos.head()
```

	Timestamp	CAN ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7	Flag
0	1.478198e+09	0316	8	5	33	104	9	33	33	0	111	R
1	1.478198e+09	018f	8	254	91	0	0	0	60	0	0	R
2	1.478198e+09	0260	8	25	33	34	48	8	142	109	58	R
3	1.478198e+09	02a0	8	100	0	154	29	151	2	189	0	R
4	1.478198e+09	0329	8	64	187	127	20	17	32	0	20	R

Figura 42 - Conversione dei dati

Tramite il comando *converters* i dati di input vengono analizzati per poi essere convertiti nel tipo desiderato, utilizzando una funzione di conversione in determinate colonne. Nel nostro caso, i dati sono convertiti in interi.

Una volta che i valori sono stati convertiti, bisogna riorganizzare il dataset, ordinando le intrusioni per CAN ID e per Timestamp, in modo tale da avere i dati in sequenza temporale. Così facendo, il modello sarà in grado di estrapolare delle feature temporali che si ripetono nel tempo, che andranno poi a identificare se l'intrusione sarà benevola o malevola. Questi schemi potrebbero ripetersi in maniera diversa per il malware e per il goodware. Non ci interessa il tempo che passa tra un comando e un altro, ma semplicemente chi viene prima e chi dopo.

```
dos['Timestamp'] = pd.to_datetime(dos['Timestamp'], origin=pd.Timestamp('2020-01-01'))
dos.sort_values(by = ["Timestamp","CAN ID"], inplace = True)
dos.head()
```

	Timestamp	CAN ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7	Flag
1460	2020-01-01 00:00:01.478198272	0000	8	0	0	0	0	0	0	0	0	T
1462	2020-01-01 00:00:01.478198272	0000	8	0	0	0	0	0	0	0	0	T
1464	2020-01-01 00:00:01.478198272	0000	8	0	0	0	0	0	0	0	0	T
1466	2020-01-01 00:00:01.478198272	0000	8	0	0	0	0	0	0	0	0	T
1468	2020-01-01 00:00:01.478198272	0000	8	0	0	0	0	0	0	0	0	T

Figura 43 - Dati ordinati in base al Timestamp e al CAN ID

A questo punto il dataset viene diviso in due dataset distinti:

- **DoST,** che contiene tutti i CAN ID malevoli;
- **DoSR,** che contiene tutti i CAN ID benevoli.

Questa suddivisione viene effettuata per due motivi principali:

- Bisogna evitare che ci sia una sovrapposizione tra i CAN ID buoni e cattivi;
- Bisogna evitare che durante l'addestramento le feature temporali estratte possano essere confuse per una diversa tipologia di attacco.

Una volta che i dati sono stati divisi e ordinati come volevamo, possiamo concatenare i due dataset. Così facendo, avremo la prima parte del dataset che andremo ad analizzare con tutti i CAN ID di tipo T, e l'altra metà di tipo R.

Adesso estrapoliamo dal dataset due array numpy aventi diverse proprietà:

```
X_input = dataset[['DATA0','DATA1','DATA2','DATA3','DATA4','DATA5','DATA6','DATA7']].values
y_target = dataset['Flag'].values
```

Figura 44 - Suddivisione dei dati in ingresso e in uscita del modello

- X_input, contenente la sequenza di 8 byte dei vari frame contenuti nel dataset;
- y_target, contenente un'unica proprietà, che vogliamo predire.

Utilizziamo adesso una tecnica di pre-elaborazione dei dati chiamata Label Encoder che permette di gestire le variabili categoriali ordinali.

Essa viene utilizzata allo scopo di preparare i dati da analizzare così da poterli rappresentare ogni dato in modo che il modello possa comprenderli facilmente.

I dati categorici che andremo a convertire sono T ed R. Questo significa che andremo ad assegnare ai due valori un numero per rappresentare quella variabile (0 per T e 1 per R).

```
le = LabelEncoder()
y_target = le.fit_transform(y_target)
```

Figura 45 - Utilizzo della tecnica LabelEncoder

Funzione temporalize

Affinché il modello riesca ad estrapolare degli schemi temporali, abbiamo bisogno di trasformare i dati in serie temporali con lunghezza fissa. Per questo motivo utilizzeremo la funzione temporalize.

```
def temporalize(X,y, lookback):
    output_X = []
    output_y = []
    for i in range(len(X)-lookback):
        t = []
        for j in range(0, lookback):
            t.append(X[[i+j], :])
        output_X.append(t)
        output_y.append(y[i+lookback])
    return output_X,output_y
```

Figura 46 - Funzione temporalize

La funzione temporalize prende in ingresso due vettori (che sono quelli che abbiamo creato in precedenza) e un valore di lookback, che rappresenta la dimensione di timestep che vogliamo far assumere ai vettori in uscita.
Costruiamo quindi i due vettori di output, dove il vettore *output_X* conterrà al suo interno altri vettori, ognuno dei quali contenente serie di 10 elementi, mentre il vettore *output_y* avrà associato, per ogni timestep, una variabile target T o R.
Una volta fatta la temporalizzazione costruiamo il vettore *output_X* come un array numpy ed eseguiamo il reshape in modo tale da ottenere il vettore descritto in precedenza.

```
X_in, y_t = temporalize(X = X_input, y=y_target, lookback=10)
```

```
X_in = np.array(X_in)
```

```
X_in = X_in.reshape(X_in.shape[0],10,8)
```

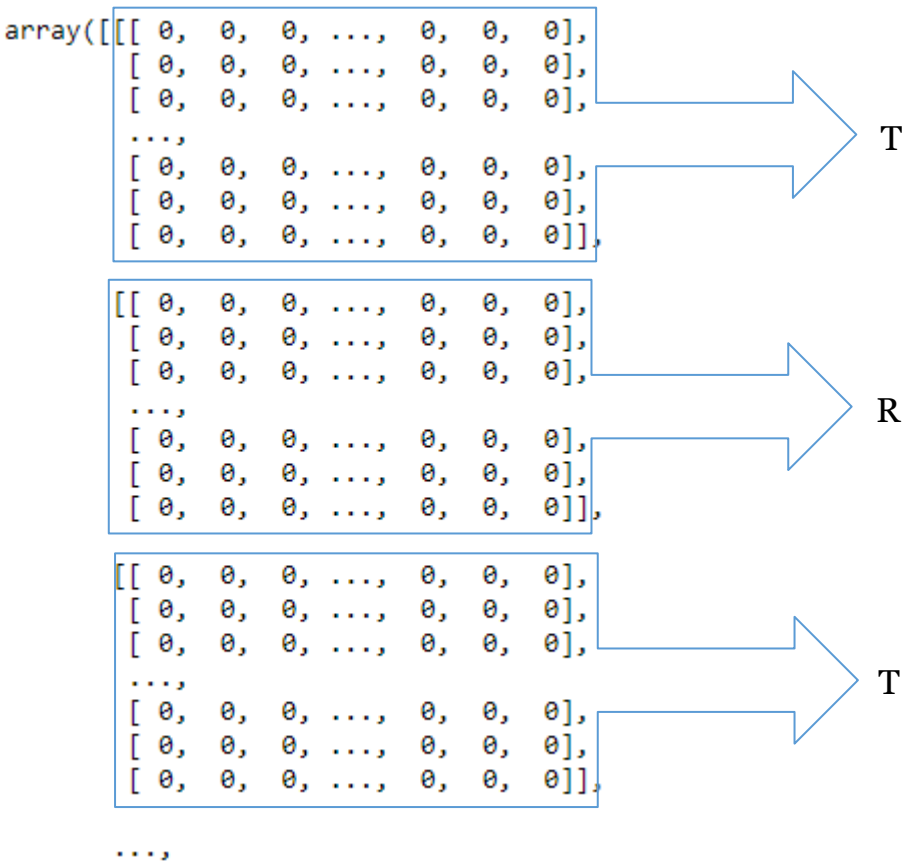

Figura 47 – Vettori di X raggruppati per timestep dove ogni uscita corrisponde a T o R

Realizzazione del modello LSTM

Suddivisione dei dati

Si è scelto di suddividere il dataset in due sottoinsiemi:

- Il **Training Set:** comprende il 70% dei dati e viene utilizzato per addestrare il modello, che a sua volta sarà suddiviso automaticamente dal programma, durante la fase di addestramento, nel Training Subset e nel Validation Subset, che rappresentano rispettivamente 10% e 90% del totale;
- Il **Test Set:** è composto dal 30% dei dati e viene utilizzato per verificare la bontà del modello testandolo su dei dati che non ha mai visto in precedenza.

```
X_train, X_test, y_train, y_test = train_test_split(X_in, y_t, test_size=0.3)
```

Figura 48 - Suddivisione del dataset in Training Set e Test Set

Creazione della rete

Una volta che i dati sono preparati adeguatamente, si può procedere alla realizzazione della struttura della rete. Il modello ha l'obiettivo di prevedere in maniera temporale se la tipologia di attacco sia *malware* o *goodware*.
La rete può essere schematizzata come due blocchi separati composti dagli strati nascosti e dallo strato finale di output, i quali ricevono l'input e lo elaborano in base ai pesi delle connessioni e alla rispettiva funzione di attivazione dello strato.
La sintassi per la creazione della struttura della rete è molto compatta. Il numero di nodi che compongono gli strati nascosti della rete può essere agevolmente modificato per testare le varie configurazioni e trovare quella più performante in base allo studio effettuato, mentre il numero di neuroni

dello strato di output deve necessariamente essere pari al numero delle tipologie di attacco che si vuole predire (in questo caso, 2 valori).

```
model = Sequential()
model.add(LSTM(20, return_sequences=True, input_shape=(10,8)))
model.add(Dropout(0.2))
model.add(LSTM(40, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(60, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(40, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(20, activation='relu'))
model.add(Dense(2, activation="softmax" ))
print(model.summary())
```

Figura 49 - Modello LSTM

Sono stati utilizzati quattro strati nascosti LSTM, dove alla fine di ogni strato è stato applicato un layer con funzione dropout per prevenire fenomeni di overfitting, uno strato nascosto Dense con funzione di attivazione *relu* e uno strato di output Dense con funzione di attivazione *softmax*.

Il layer LSTM non accetta un unico elemento d'ingresso, ma accetta un insieme di elementi. Per questo, nel parametro *input_shape*, che imposta la dimensione e la lunghezza delle sequenze, inseriamo come primo valore la dimensione di batch (o timestep) pari a 10, mentre come secondo valore la dimensione del messaggio, che è di 8 byte.

Quello che si può notare nel modello è che nei primi tre layer LSTM il parametro *return_sequences* non è impostato su *false*, dato che non vogliamo che gli strati restituiscano solo l'ultimo output generato ma vogliamo ottenere l'intera sequenza; nel caso specifico, quindi, vogliamo che ad ogni dieci elementi restituisca un output. Di conseguenza, eseguendo le *return_sequences* impostate a *true*, riusciamo ad ottenere più informazioni, perché riusciamo ad ottenere non solo gli output finali ma anche quelli intermedi. Inoltre, lo stride utilizzato è pari ad 1 perché così riusciamo a capire se ci sono delle ripetizioni temporali.

Il motivo per cui vengono utilizzati diversi strati nascosti è perché riteniamo che all'inizio vengano estratte feature temporali semplici.

Utilizzando più strati, riusciamo sicuramente ad estrapolarne di più complesse, utilizzando più strati nascosti, la rete viene definita dall'inglese Deep Neural Network.

L'ultimo strato nascosto è di tipo denso, con funzione di attivazione *relu*, quindi ogni nodo di questo strato sarà collegato a ogni nodo dello strato successivo. Infine, nello strato di output, viene utilizzato uno strato denso con la funzione di attivazione, ossia la *softmax*. Questo perché, come output finale, si vuole stabilire la probabilità che una sequenza sia di una certa tipologia di attacco, cioè *malware* o *goodware*.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 10, 20)            2320
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 20)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 10, 40)            9760
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 40)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 10, 60)            24240
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 60)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 40)                16160
_________________________________________________________________
dropout_4 (Dropout)          (None, 40)                0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                820
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 42
=================================================================
Total params: 53,342
Trainable params: 53,342
Non-trainable params: 0
```

Figura 50 - Struttura del modello LSTM

Successivamente, si è definita la funzione per compilare la rete definendo i criteri di addestramento e valutazione. Occorre definire l'ottimizzatore e il criterio da seguire per minimizzare la funzione Loss, che rappresenta la differenza fra l'output del sistema e l'output atteso. Esistono vari criteri, a

seconda del tipo di problema affrontato. Gli argomenti necessari alla funzione sono:

- **optimizer:** definisce la funzione di ottimizzazione;
- **loss:** definisce la funzione obiettivo che deve essere minimizzata dall'ottimizzatore;
- **metrics:** definisce la funzione metrica e serve a valutare le performance del modello.

La funzione obiettivo utilizzata è l'errore quadratico medio (Mean Squared Error – MSE), mentre la funzione di ottimizzazione è la RMSprop, che definisce un ottimizzatore performante sulle reti neurali ricorsive.

Come metrica utilizzata per determinare la qualità del modello, viene utilizzata l'accuracy, che in sostanza indica la percentuale di predizioni che il nostro modello ha azzeccato.

```
model.compile(optimizer='rmsprop' , loss= 'mean_squared_error', metrics=['accuracy'])
```

Figura 51 - Funzione di compilazione

Apprendimento

Ora il modello deve essere "addestrato", dandogli da analizzare il set di dati appositamente preparato. Come accennato in precedenza, vengono forniti al modello una serie di dati reali e i risultati attesi, anch'essi reali, così che il sistema possa procedere con l'elaborazione dell'algoritmo che definisce le relazioni esistenti fra i due. Questa operazione richiede una sola riga di codice.

```
model.fit(X_train, y_train, epochs=5, batch_size = 1000, validation_split=0.1, callbacks=[history])
```

Figura 52 - Funzione di addestramento

Il parametro *epochs* (epoche) definisce il numero di volte che i dati vengono processati dall'intero sistema. Questo è un aspetto molto delicato, in quanto più i dati vengono riprocessati dalla rete, maggiore è l'accuratezza del modello. Tuttavia, questo processo non scala in modo lineare, per cui si individua quindi un punto di minimo oltre il quale aumentare le epochs sarebbe poco redditizio e si rischierebbe di incorrere in overfitting.

Il parametro *batch_size* definisce un set di campioni che vengono processati dal modello. Più grande è la dimensione, migliore è l'approssimazione, fermo restando che richiederà più tempo per essere elaborato.
Il parametro *validation_split* permette di suddividere ulteriormente il set di dati per includere un set di convalida dove il modello non si addestra su di esso, ma lo utilizza per valutare il valore di loss e quello della metrica alla fine di ogni epoca.
Il parametro *callbacks* permette di inserire un elenco di funzioni da applicare durante il training e la validazione. In questo caso, abbiamo inserito la funzione *history* che permette di registrare i valori di loss e delle metriche nelle varie epoche.

```
Train on 2289780 samples, validate on 254421 samples
Epoch 1/5
2289780/2289780 [==============================] - 235s 103us/step - loss: 0.02
17 - acc: 0.9748 - val_loss: 0.0210 - val_acc: 0.9759
Epoch 2/5
2289780/2289780 [==============================] - 262s 114us/step - loss: 0.02
08 - acc: 0.9762 - val_loss: 0.0210 - val_acc: 0.9759
Epoch 3/5
2289780/2289780 [==============================] - 264s 115us/step - loss: 0.02
08 - acc: 0.9761 - val_loss: 0.0210 - val_acc: 0.9759
Epoch 4/5
2289780/2289780 [==============================] - 263s 115us/step - loss: 0.02
08 - acc: 0.9762 - val_loss: 0.0210 - val_acc: 0.9759
Epoch 5/5
2289780/2289780 [==============================] - 267s 117us/step - loss: 0.02
07 - acc: 0.9762 - val_loss: 0.0210 - val_acc: 0.9759
```

Figura 53 - Addestramento della rete

Valutazione

Dopo la fase di apprendimento, il modello deve essere testato facendogli elaborare un set di dati che non ha mai visto in precedenza. I risultati del modello vengono comparati con i risultati attesi (reali).

```
model.evaluate(X_test, y_test)
```

```
1090372/1090372 [==============================] - 123s 113us/step

[0.020685571027994, 0.9762759865440418]
```

Figura 54 - Funzione di valutazione

La figura 54 mostra i risultati ottenuti dalla valutazione. Tra questi, il valore di loss calcolato che corrisponde all'errore quadratico medio è molto vicino allo zero, mentre il valore dell'accuratezza raggiunge un valore del 97%, il che comporta che siamo riusciti a realizzare un modello ottimale.

Risultati e prestazioni

Sono state provate varie configurazioni della rete, variando il numero di nodi che compongono gli strati nascosti per valutare la configurazione più performante. In alcuni casi, lo script è stato impostato per fermare automaticamente l'apprendimento una volta che la funzione Loss selezionata (in questo caso si è adoperato l'errore quadratico medio) non migliori dopo un certo numero di epoche.

Struttura nodi	Training		Validation		Epoche
	MSE	ACC	MSE	ACC	
5 – 10	0.0314	0.89	0.0289	0.90	1
	0.0308	0.90	0.0306	0.91	3
	0.0308	0.90	0.0297	0.91	5
10 - 30	0.0276	0.93	0.0255	0.93	1
	0.0274	0.93	0.0254	0.93	3
	0.0273	0.93	0.0249	0.94	5
15-45	0.0235	0.95	0.0231	0.96	1
	0.0224	0.96	0.0217	0.96	3
	0.0224	0.96	0.0216	0.96	5
20-60	0.0217	0.97	0.0210	0.97	1
	0.0208	0.97	0.0210	0.97	3
	0.0207	0.97	0.0210	0.97	5
25 – 70	0.0233	0.97	0.0194	0.97	1
	0.0205	0.97	0.0195	0.97	3
	0.0204	0.97	0.0194	0.97	5

Quello che si può notare osservando la tabella è che i valori dell'errore quadratico medio e dell'accuratezza, con l'aumentare del numero dei nodi, continuano a migliorare ma, essendo questi miglioramenti essendo molto vicini, non possiamo definire un miglioramento significativo, visto che già con pochi nodi la rete ci forniva una buona statistica. Con le diverse configurazioni effettuate, possiamo dire che la rete con la configurazione 20-60 sia il compromesso migliore, sia per i risultati ottenuti e sia per il tempo di addestramento. Infatti andando ad aumentare ulteriormente i numeri di nodi, la rete ci impiega più tempo per addestrare i dati e i risultati risultano molto vicini alla configurazione precedente.

La figura 55 mostra l'andamento della precisione del modello in funzione delle epoche durante la fase di addestramento e di test. Quello che possiamo rilevare è che la rete LSTM impiega poco tempo per stabilizzarsi durante l'addestramento; al contrario, per quanto riguarda il test, la rete sembra stabilizzarsi fin da subito.

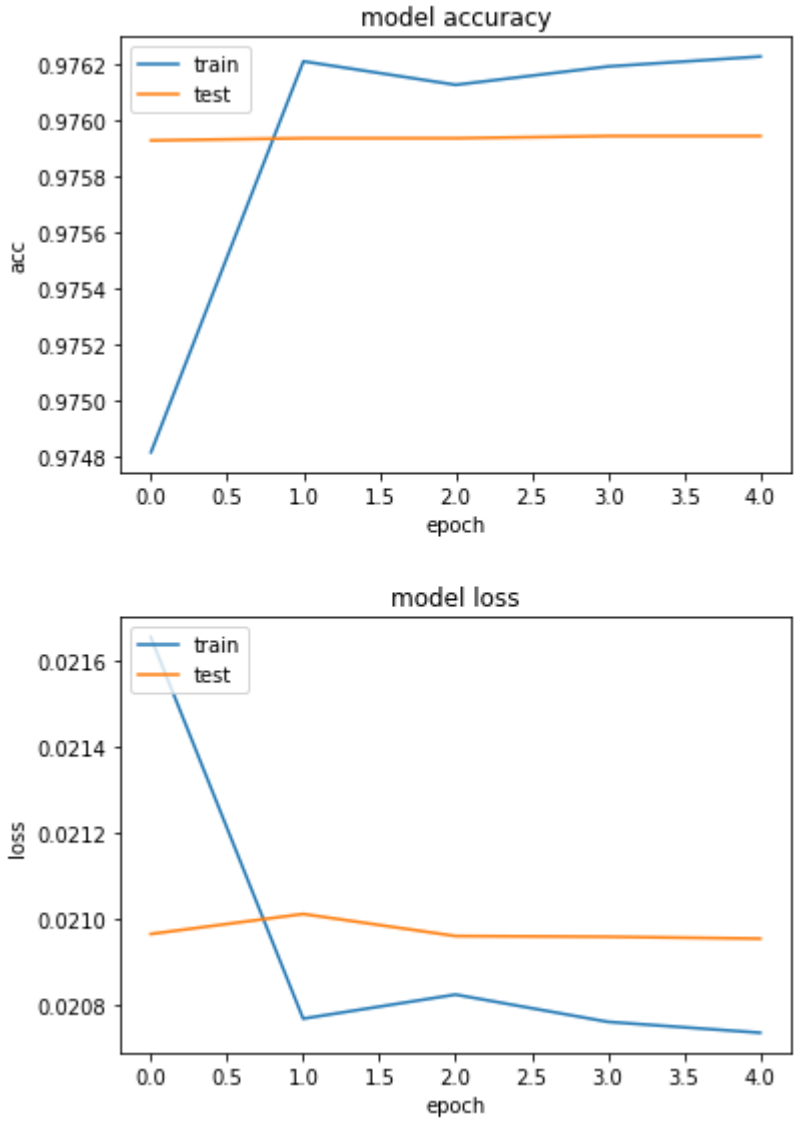

Figura 55 – Andamento di Accuracy e Loss della rete LSTM con configurazione 20-60

Predizione

Ora il modello è pronto per essere utilizzato. Fornendo dei dati coerenti rispetto a quelli utilizzati in fase di apprendimento è in grado di predirne i risultati efficientemente.

```
y_pred = model.predict_classes(X_test, batch_size=1000, verbose = 0)
```

```
y_labels = np.argmax(y_test, axis=1)
```

```
cm = confusion_matrix(y_labels, y_pred)
```

```
plot_confusion_matrix(cm, ['T','R'])
```

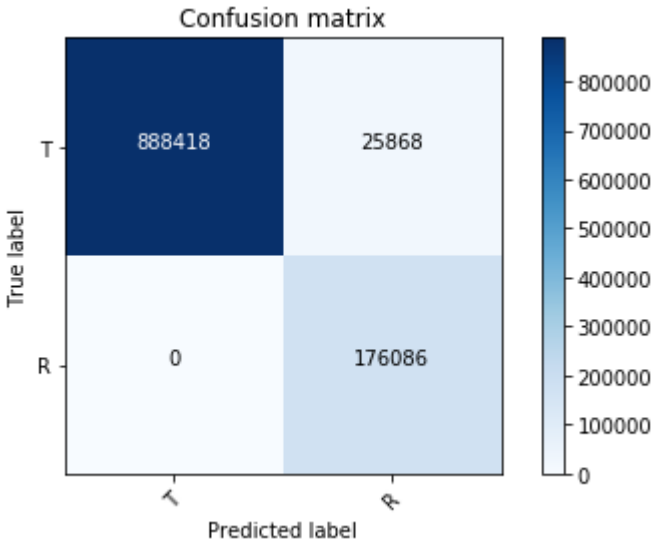

Figura 56 - matrice di confusione

La matrice di confusione ci permette di vedere dove il nostro modello ha commesso più errori. Il nostro test-set contiene circa 888.000 attacchi maligni che sono stati classificati correttamente e circa 200.000 attacchi benigni dove 176.000 sono stati classificati correttamente e 25.000 sono stati classificati come malware invece di goodware.

L'errore commesso sta a significare che, molto probabilmente, nei goodware ci sono degli schemi molto simili a quelli dei malware e per questo il modello non è stato in grado di riconoscerli.

Risultati del modello su attacchi Fuzzy e Spoofing

Adesso andremo ad analizzare i risultati ottenuti dagli altri due set di dati, la struttura dei dati, per entrambi i dataset di attacchi Fuzzy e Spoofing, si presenta allo stesso modo di quella degli attacchi DoS mostrata in precedenza.

Per tale motivo, in questo paragrafo, poiché la pre-analisi dei dati e il modello LSTM utilizzato sono gli stessi di quelli usati in precedenza, verrà mostrato graficamente l'andamento dell'accuratezza e dell'errore quadratico medio di entrambi gli attacchi, insieme alle rispettive matrici di confusione.

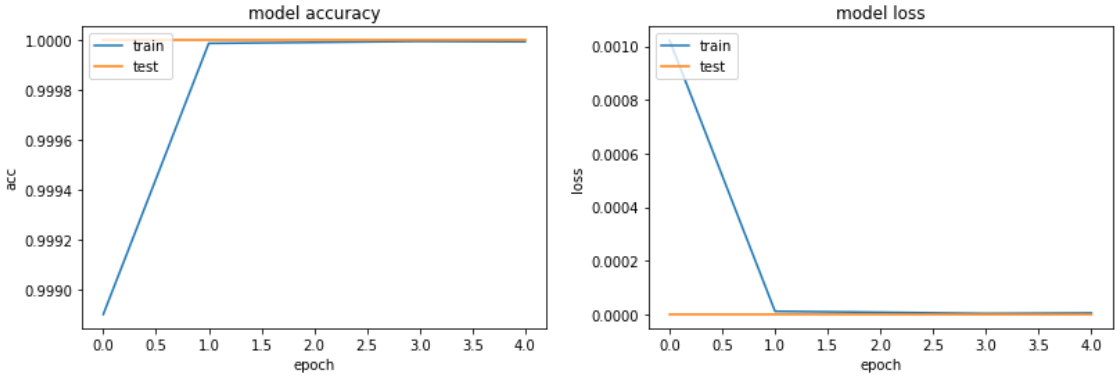

Figura 57 - Andamento Accuracy e Loss - Fuzzy

La figura 57 mostra l'andamento della precisione del modello in funzione delle epoche durante la fase di addestramento e di test. Quello che possiamo rilevare è che la rete LSTM in questo caso impiega un tempo minore rispetto a quello degli attacchi DoS a stabilizzarsi durante l'addestramento raggiungendo quasi il 100% di accuratezza, mentre per quanto riguarda l'errore quadratico medio assume un valore pari a 0.

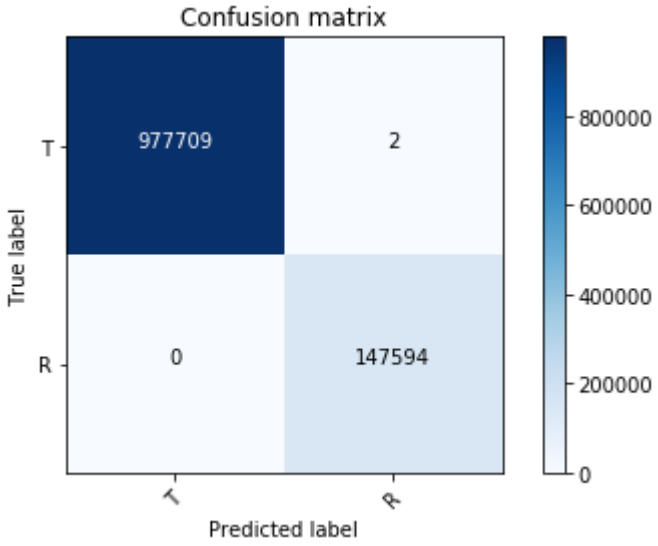

Figura 58 - Matrice di confusione – Fuzzy

La matrice di confusione ci conferma che la predizione è stata ottimale, infatti i falsi positivi, infatti, risultano essere soltanto due.

L'andamento della precisione del modello in funzione delle epoche durante la fase di addestramento e di test nel caso degli attacchi Spoofing, sembra

essere anch'esso ottimale. Durante l'addestramento del modello, solo nella seconda epoca sembra esserci stata un calo di accuratezza e loss.

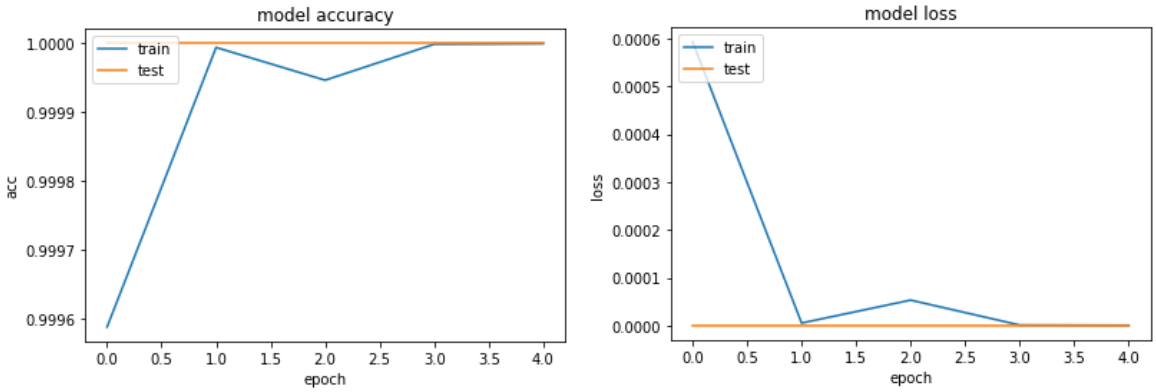

Figura 59 - Andamento di Accuracy e Loss - Spoofing

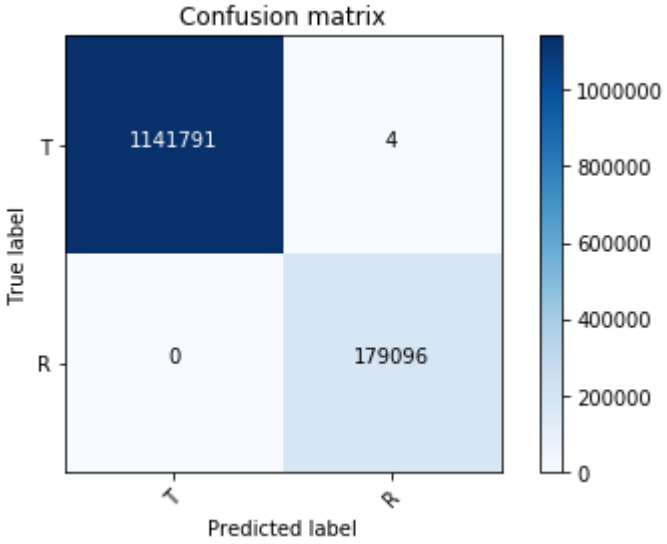

Figura 60 - Matrice di confusione – Spoofing

Come possiamo vedere dalla figura 60, anche questa matrice di confusione ci mostra una predizione ottimale, infatti, si sono solo quattro falsi positivi.

Conclusioni

In questa tesi si è esposto l'utilizzo del Machine Learning applicato alla classificazione di attacchi in ambito automotive, insieme ad uno studio teorico sul CAN-bus, gli attacchi che possono essere effettuati su di esso. Infine, è stato realizzato un approfondimento sulle varie reti che possono essere utilizzate, per poi nella pratica realizzare un modello adatto a questo scopo.

L'utilizzo di algoritmi di Machine Learning ha consentito di sviluppare un modello che, basandosi sullo studio dei dati, permette di classificare efficacemente i malware e goodware.

Nella ricerca svolta è emerso che, nella classificazione degli attacchi, i dati più "difficili" da distinguere tra malware o goodware sono risultati gli attacchi Denial of Services mentre, per gli altri tipi di attacchi, l'errore è stata di poca rilevanza data la grossa mole dei campioni forniti.

Un'ultima considerazione si evince nel caso in cui volessimo raggruppare tutti i tipi di attacchi DoS, Fuzzy e Spoofing, ordinarli e separarli tra malevoli e benevoli, per poi valutare il comportamento del modello.

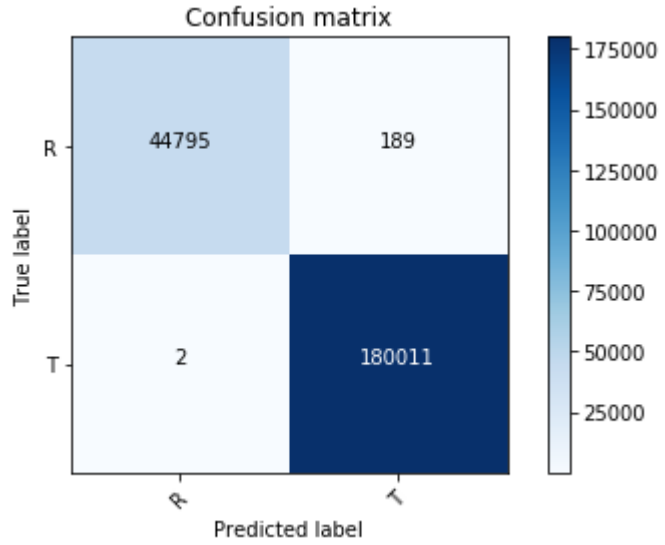

Figura 61 - Matrice di confusione con T (DoS, Fuzzy, Spoofing) e R(traffico normale)

Per quanto riguarda la predizione da parte del modello LSTM, la matrice di confusione mostra che su 225.000 intrusioni, circa 45.000 sono stati

classificati come benevoli, mentre 180.000 come malware, tra cui 189 sono stati classificati come falsi positivi e 2 come falsi negativi.
Per quanto riguarda l'accuratezza, il modello ha raggiunto un valore del 98% e un errore quadratico medio dello 0.0073, per cui possiamo dire che il modello è ottimale anche nel caso in cui i malware mostrano più di una differenziazione negli schemi temporali estratti.
Dalle analisi effettuate nella tesi, sono emersi alcuni elementi che necessitano di particolare attenzione nell'utilizzo di algoritmi di Machine Learning, i quali devono essere sempre tenuti in considerazione nello sviluppo di questi modelli:

- L'importanza di selezionare con cura le diverse grandezze in input, utilizzando solo quelle che sono strettamente correlate fra loro da un rapporto di causalità rispetto all'output desiderato;
- La necessità di avere un campione di dati quanto più grande possibile, in modo da poter addestrare efficacemente il modello per sviluppare la capacità di generalizzare e quindi di fornire una previsione affidabile e, contemporaneamente, la possibilità di garantire una continua raccolta dei dati al fine di effettuare migliori predizioni.

La presente tesi ha dimostrato la solidità delle predizioni ottenibili grazie all'utilizzo del Machine Learning, che si è rivelato in grado di fornire un efficace strumento di supporto nel campo della classificazione di attacchi in ambito automotive.

Codice completo del modello LSTM

```
"""
Modello LSTM
"""
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import pandas as pd
import os
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import itertools
from keras.callbacks import History, EarlyStopping,TensorBoard,
from keras.models import Sequential
from keras.layers import Dense, Input, GRU, Embedding, LSTM, Dropout

#Caricamento del dataset con la conversione dei dati in forma decimale
dos = pd.read_csv("C:/Users/pio-f/Desktop/DoS_dataset.csv",
        converters={"DATA0":lambda x: int(x, 16),"DATA1":lambda x: int(x, 16),
                    "DATA2":lambda x: int(x, 16),"DATA3":lambda x: int(x, 16),
                    "DATA4":lambda x: int(x, 16),"DATA5":lambda x: int(x, 16),
                    "DATA6":lambda x: int(x, 16),"DATA7":lambda x: int(x, 16)})
#Conversione del Timestamp
dos['Timestamp'] = pd.to_datetime(dos['Timestamp'],
                    origin=pd.Timestamp('2020-01-01'))
#Ordinamento dei dati per Timestamp e CAN ID
dos.sort_values(by = ["Timestamp","CAN ID"], inplace = True)
#Suddivisione del dataset per tipologia di attacco
dosT = dos[dos["Flag"] == "T"]
dosR = dos[dos["Flag"] == "R"]
# Concatenazione dei due dataset
frames = [dosT,dosR]
dataset = pd.concat(frames)
print(dataset)
# Creazione dei dati d'ingresso e target
X_input = dataset[['DATA0','DATA1','DATA2','DATA3','DATA4','DATA5'
                   ,'DATA6','DATA7']].values
y_target = dataset['Flag'].values
# utilizzo della tecnica LabelEncoder
le = LabelEncoder()
y_target = le.fit_transform(y_target)

# Definizione della funzione temporalize
def temporalize(X,y, lookback):
    output_X = []
    output_y = []
    for i in range(len(X)-lookback):
        t = []
        for j in range(0, lookback):
            t.append(X[[i+j], :])
        output_X.append(t)
        output_y.append(y[i+lookback])
    return output_X,output_y

# Dati temporalizzati
X_in, y_t = temporalize(X = X_input, y=y_target, lookback=10)
# Conversione del vettore in numpy array
X_in = np.array(X_in)
# reshape del vettore con timestep=10 e feature=8
```

```
X_in = X_in.reshape(X_in.shape[0],10,8)
# Categorizzazione dei target
y_t = to_categorical(y_t, 2)
# Creazione del set di train e di test
X_train, X_test, y_train, y_test = train_test_split(X_in, y_t, test_size=0.3)
# Creazione del modello
model = Sequential()
model.add(LSTM(20, return_sequences=True, input_shape=(10,8)))
model.add(Dropout(0.2))
model.add(LSTM(40, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(60, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(40, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(20, activation='relu'))
model.add(Dense(2, activation="softmax" ))
print(model.summary())

model.compile(optimizer='rmsprop',loss='mean_squared_error',metrics=['accuracy'])

# Apprendimento da parte del modello
history = History()
early_stop = EarlyStopping(monitor='val_loss', patience=3)
model.fit(X_train, y_train, epochs=5, batch_size = 1000, validation_split=0.1,
          callbacks=[early_stop,history])

# Valutazione dell'apprendimento
model.evaluate(X_test, y_test)

# Definizione della funzione per la realizzazione della matrice di confusione
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

# Predizione del modello
y_pred = model.predict_classes(X_test, batch_size=1000, verbose = 0)
y_labels = np.argmax(y_test, axis=1)
cm = confusion_matrix(y_labels, y_pred)
```

```
plot_confusion_matrix(cm, ['T','R'])

# Lista di tutti i dati in history
print(history.history.keys())
# riepilogo history per accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# riepilogo history per loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Bibliografia

1. [Online] “Controller Area Network – (CAN)”, *Ing. Stefano Maggi*

2. “Hacking the CAN Bus: Basic Manipulation of a Modern Automobile Through CAN Bus Reverse Engineering”, *Roderick Currie*, 2017

3. [Paper] “A Novel Intrusion Detection System for In-vehicle Network by using Remote Frame”, *Hyunsung Lee, Seong Hoon Jeong, Huy Kang Kim*

4. [Online] “Appunti di reti neurali”, *S.Rota Bulò*, 2015

5. “Machine learning with Tensorflow”, *N.Shukla*, 2017

6. “Deep Learning with Keras”, A. Gulli, 2017

7. “Long Short-Term Memory Networks whit Python”, *Jason Brownlee*, 2017

8. [Paper] “GIDS: GAN based Intrusion Detection System for In-Vehicle Network”, *Eunbi Seo, Hyun Min Song, Huy Kang Kim*, 2019

9. [Online] “Deep Learning e Reti Neurali con Python”, *Giuseppe Gullo*

Ringraziamenti

A conclusione di questo elaborato desidero menzionare tutte le persone che mi sono state vicine in questo percorso di crescita personale e professionale.
Ringrazio i miei relatori Castiglione Arcangelo e D'Angelo Gianni, che in questi mesi di lavoro hanno saputo guidarmi, con la loro infinita disponibilità, con suggerimenti pratici, nelle ricerche e nella stesura dell'elaborato.
Ringrazio i miei genitori e a mia sorella che sono il pilastro della mia vita, le fondamenta dei miei giorni. Questa tesi è la dedico a voi, che mi avete sempre amato e sostenuto ogni giorno, per avermi permesso di studiare, di scegliere la mia strada, di seguire le mie inclinazioni.
Ringrazio con tutto il mio cuore il mio amore, Maria Camilla, la persona che più di ogni altra sa cosa rappresenta per me questo giorno e che mi ha sempre sostenuto, supportato, sopportato, calmato, spronato, incoraggiato. Sei sempre stata con me, non mi hai mai lasciato solo e mi hai sempre fatto sentire quanto tu credessi in me, ogni giorno, ogni minuto, dopo ogni caduta, prima di ogni vittoria. Oggi la mia laurea la condivido con te, grazie amore mio, ti sarò per sempre grato.
Un ringraziamento speciale va alla professoressa Giordano Maria Rosaria, che durante questi ultimi anni di Università mi ha aiutato e sostenuto con lo studio, attraverso la traduzione di diverse pubblicazioni.
Grazie amici miei, per tutto. Per esserci stati sempre, in ogni istante. Per aver condiviso le cose belle e quelle brutte, in questi anni universitari così pieni di stimoli e avventure. Grazie per avermi sempre incoraggiato a non mollare mai e a guardare avanti a testa alta per arrivare fino a questo momento. Questa laurea è anche un po' vostra e a parole non riesco ad esprimere il bene che vi voglio e quanto vi sono grato.

In particolare vorrei ringraziare Roberto, con la quale ho condiviso con lui centinaia di ore in facoltà, a lezione, le ansie pre-esame, lo studio su Skype, le fantastiche avventure su Dark Souls, la pausa caffè e merendina.
Ringrazio i miei cari amici Giovanni, Orlando, Antonio e Gerardo, per il sostegno che ci siamo dati fino ad ora, per tutti i momenti di gioia, festa e "cazzeggio", per tutti i consigli, le critiche e gli spoiler scambiati. Vi ringrazio per essere miei amici, mi sento fortunato ad avervi conosciuto. Grazie per essere stati sempre al mio fianco!
Un sentito grazie lo rivolgo ad Assunta, un'amica speciale con cui ho condiviso diversi momenti, il part-time insieme al Centro Linguistico, la colazione al bar, le lunghe passeggiate tra la biblioteca e i diversi uffici della facoltà di Lettere e il sostegno reciproco durante questo periodo universitario.
Ringrazio Giulio per essermi stato mio collega e amico durante l'ultimo esame della magistrale, non dimenticherò mai le tue bestemmie di quando eri bloccato in tangenziale per venire all'Università a studiare per il progetto.
Un ringraziamento va anche ai miei cari amici di lunga data, Nello, Mauro e Cabu, amici così diversi ma così importanti, ognuno per ragioni uniche e speciali, voglio esprimere la mia più assoluta gratitudine.
Un grosso grazie a tutti gli amici incontrati in questi anni, che in modi diversi, attraverso parole, gesti, messaggi, risate, camminate e chiacchierate mi hanno incoraggiato!
Infine, un ultimo ringraziamento va a me stesso, perché se sono arrivato a questo punto, infondo è perché non mi sono mai arreso e perché voglio riuscire ad ottenere il meglio da questa vita, in modo tale da rendere felice non solo me stesso, ma anche tutte le persone che mi sono accanto.

"May your heart be your guiding key"

Giuseppe Abagnale