UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN Ingegneria Informatica

"Utilizzo di un Autoencoder per la realizzazione di un Sistema di Rilevamento delle Intrusioni nella rete locale"

Relatore: Prof./Dott. Loris Nanni

Laureando: Nicolò Tesser

Correlatore: Dott. Pietro Boccaletto

ANNO ACCADEMICO 2021-2022

Data di laurea 21/07/2022

Indice

Indice delle figure

Glossario

Application Program Interface: insieme di procedure che permettono la compatibilità tra diversi computer o tra diversi software o tra diversi componenti di software
Batch size: grandezza del lotto/porzione di un dataset, è un numero che rappresenta quanti elementi devono presenti nei sottoinsiemi in cui il training set viene diviso
Comunità virtuale: insieme di persone interessate ad un determinato argomento che corrispondono tra loro attraverso internet costituendo una rete sociale con caratteristiche particolari
Epoca: unità temporale che rappresenta una completa elaborazione del training set da parte della rete
Features: in italiano caratteristica, è una proprietà di un dato espressa numericamente o categoricamente.
Open source: letteralmente sorgente libero, software non soggetto a licenza
Payload: contenuto informativo di un pacchetto, il suo "carico utile"
Seed: valore numerico che si usa per inizializzare le funzioni randomiche di un software
Signature: tradotto letteralmente in firma, è una sequenza di byte comune ad alcuni tipi di malware, essi possono così venir riconosciuti da software basati sul confronto delle firme
Sniffer: software in grado di catturare i pacchetti in transito in una LAN
User-friendly: letteralmente amichevole all'utente, sono una serie di tecniche volte a migliorare l'uso di sistemi complessi agli utenti finali, ad esempio la resa grafica.

Elenco delle abbreviazioni

ANN Artificial Neural Network
API Application Program Interface
DARPA Defense Advanced Research Projects Agency
DL Deep Learning
FNN FeedForward Neural Network
HIDS Host-based IDS
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
IA Intelligenza Artificiale
IDS Intrusion Detection System
IDPS Intrusion Detection and Prevention System
IP Internet Protocol
ISO International Standard Organization
KDD Knowledge Discovery in Databases
LAN Local Area Network
ML Machine Learning
NIDS Network IDS
NIST National Institute of Standard Technology
OSI Open System Interconnection
WAN Wide Area Network
WWW World Wide Web

Introduzione

Alla base di questo lavoro di tesi vi è lo studio dei sistemi atti a proteggere i dispositivi degli utenti finali analizzando il traffico che transita nella rete LAN, che sia essa una rete privata o aziendale.
Negli ultimi due decenni la rete internet è diventata il cuore di molte attività quotidiane, dalla messaggistica istantanea al versamento/prelievo di fondi dai nostri conti correnti. In futuro è lecito immaginare che il coinvolgimento della rete sarà sempre maggiore ed è quindi necessario avere un grado elevato di sicurezza nei dispositivi che sfruttano questa tecnologia.
Il lavoro svolto in questa tesi mira alla realizzazione di una prova di concetto volta a dimostrare come l'Intelligenza Artificiale possa aiutarci a proteggere i nostri dispositivi connessi alla rete. Gli Intrusion Detection System sono sistemi di protezione nati a tal scopo, in questo lavoro si presenta un tipo di IDS basato sull'Intelligenza Artificiale realizzato attraverso l'utilizzo di un particolare tipo di rete neurale che prende il nome di Autoencoder.
La Tesi è organizzata in tre capitoli.
Nel primo capitolo viene presentata una panoramica sulla rete Internet, sulle tipologie e di attacchi informatici e sui sistemi di protezione esistenti con particolare attenzione per gli Intrusion Detection System.
Nel secondo capitolo viene presentata una panoramica sulle tecniche di Intelligenza Artificiale e in particolare sulla topologia di rete neurale utilizzata nel progetto, con una breve introduzione a Python e alle librerie utilizzate nel progetto.
Nel terzo ed ultimo capitolo viene presentata una analisi dettagliata sulla metodologia seguita per la realizzazione dell'IDS basato su Autoencoder completata dai risultati ottenuti e dai possibili sviluppi futuri, a seguire le conclusioni e i ringraziamenti.

Capitolo 1

In questo capitolo verrà presentata una panoramica sulla rete Internet, sugli attacchi informatici portati attraverso la stessa e sui sistemi di protezione dei dispositivi attualmente utilizzati.

Rete Internet

Internet è una rete di telecomunicazioni che connette vari dispositivi in tutto il mondo.
L'avvento e la diffusione di Internet e dei suoi servizi hanno rappresentato una vera e propria rivoluzione tecnologica e socioculturale dagli inizi degli anni Novanta nonché uno dei motori dello sviluppo economico mondiale.
L'interconnessione globale tra reti informatiche di natura e di estensione diversa è resa possibile da un insieme di protocolli di rete chiamato TCP/IP. I due protocolli, TCP e IP, costituiscono il fondamento comune con cui i computer connessi a Internet possono comunicare tra loro indipendentemente dalla loro sottostante architettura hardware e software, garantendo così l'interoperabilità tra sistemi e sottoreti fisiche diverse.
Negli anni '90 nacque il World Wide Web (WWW), esso rappresenta uno dei principali e più utilizzati servizi di Internet; abbreviato in web permette di navigare e usufruire di un insieme molto vasto di contenuti messi in relazione tramite link (collegamenti ipertestuali), e di ulteriori servizi accessibili a tutti o ad una parte selezionata di utenti; questa facile reperibilità di informazioni è resa possibile, oltre che dai protocolli di rete, anche dalla presenza, diffusione, facilità d'uso ed efficienza dei motori di ricerca e dei web browser in un modello di architettura di rete definito Client-Server. Il modello Client-Server (Figura 1) è un'architettura di rete in cui si distinguono due tipi di dispositivi, i client e i server. Il client esegue una richiesta e il server restituisce la risposta. In generale il termine client indica una componente che accede ai servizi o alle risorse di un'altra componente, detta server.
Nell'uso comune il client corrisponde al computer dell'utilizzatore che tramite browser web può eseguire le richieste al server, il quale invia le risposte al client. Tali risposte possono essere visualizzate sempre grazie al browser web.

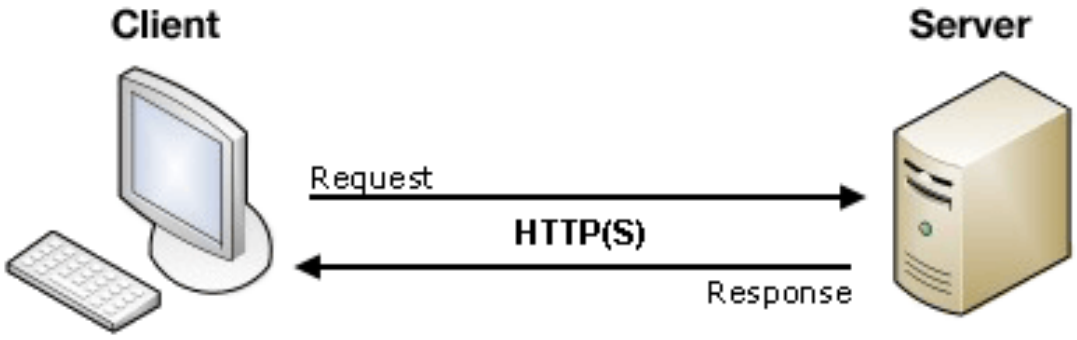

Figura 1. Rappresentazione di una connessione Client-Server.

Sempre negli anni ‘90 Tim Berners-Lee definì il protocollo HTTP (HyperText Transfer Protocol) basandosi sul modello client-server, grazie a questo protocollo è possibile l’interscambio di contenuti ipertestuali in cui è quindi possibile la lettura non sequenziale di documenti per mezzo di collegamenti link. Il protocollo HTTP trasmette le informazioni totalmente in chiaro, un malintenzionato potrebbe quindi carpire dati personali semplicemente ponendosi nel mezzo della connessione tra client e server in una tipologia di attacco che prende il nome di Man in the Middle; per questo motivo ad oggi è in uso il protocollo HTTPS (HyperText Transfer Protocol over Secure Socket Layer) in grado di criptare una connessione HTTP per mezzo di chiavi di cifratura asimmetrica.
Nel World Wide Web, le risorse sono organizzate secondo un sistema di pagine scritte in linguaggio HTML (HyperText Markup Language), a cui si può accedere, lato client, utilizzando programmi detti web browser che ne interpretano i contenuti rendendo accessibili visualmente file, testi, ipertesti, suoni, immagini e altro. La creazione di questo protocollo ha reso possibile la massiccia diffusione della rete Internet e in particolare del servizio web.
Internet rappresenta sicuramente il più diffuso ed utilizzato esempio di telecomunicazione a cui siamo abituati, la trasmissione delle informazioni è però permesso da un più ampio insieme di tecnologie che prendono il nome di Ethernet. Ethernet è una famiglia di tecnologie di rete comunemente usate nelle reti locali, metropolitane e geografiche; è stato introdotto nel 1980 e standardizzato come IEEE 802.3 (Institute of Electrics and Electronics Engineers). L’Ethernet standard utilizza cavi coassiali come mezzo di trasmissione fisico dei dati, recentemente la tecnologia è stata sostituita dai collegamenti in fibra ottica in grado di migliorare notevolmente la quantità di informazione trasferita nell’unità di tempo. I sistemi che comunicano tramite Ethernet dividono un flusso di dati in parti più brevi chiamate frame. Ogni frame contiene indirizzi di origine e destinazione e metodi di controllo degli errori in modo che i frame danneggiati possano essere rilevati ed eliminati; molto spesso, i protocolli di livello superiore attivano la ritrasmissione dei frame persi.
Il primo modello presentato per la telecomunicazione basato su Ethernet è stata la pila OSI (Open System Interconnection) definita da ISO (International Standard Organization); OSI è un modello concettuale che concentra la sua attenzione sulla definizione dei livelli di stratificazione della procedura che permette l’intercomunicabilità tra dispositivi diversi e delle interfacce presenti tra essi.
La pila OSI è formata da 7 livelli, a partire dal basso troviamo i livelli: fisico, accesso alla rete/collegamento, rete, trasporto, sessione, presentazione e applicazione. L’intento principale del modello di riferimento OSI è sancire delle linee guida per la progettazione e lo sviluppo dell’hardware, dei dispositivi e del software di comunicazione digitale in modo che possano interagire in modo efficiente.
La pila TCP/IP rappresenta un’implementazione del modello concettuale OSI sopra descritto, differentemente dalla pila OSI esso è composto da 4 livelli in cui sono racchiusi i 7 della pila OSI, in particolare troviamo i livelli: accesso alla rete, Internet, trasporto e applicazione. TCP/IP è considerato il modello standard per il networking. Il protocollo TCP gestisce la trasmissione dei dati e il protocollo IP gestisce gli indirizzi. Il modello TCP/IP racchiude una serie di protocolli tra i quali TCP, UDP, ARP, DNS, HTTP, ICMP e altri. Il modello TCP/IP viene utilizzato principalmente per l’interconnessione di computer su Internet ed è un modello robusto e flessibile.

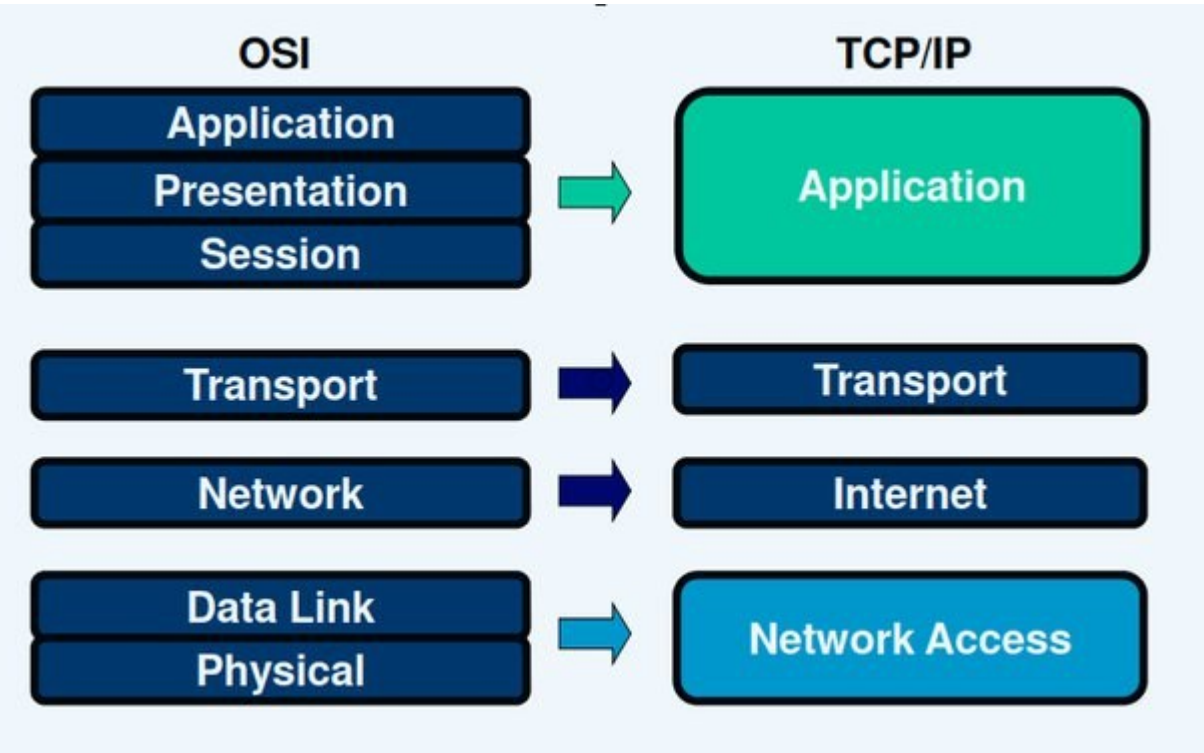

Figura 2. Rappresentazione Pila OSI e dell*a* Pila TCP/IP.

Ogni pacchetto inviato in rete, generalmente creato a livello applicativo da un utente, subisce un processo di incapsulamento in cui il contenuto del pacchetto (payload) è completato da un header di livello; nella discesa attraverso la pila TCP/IP il pacchetto del livello precedente, completato dal suo header, diventa payload del livello successivo; esso a sua volta viene completato dall'header di livello e la discesa prosegue al livello successivo; una volta raggiunto il livello di accesso alla rete, quello fisico, il pacchetto viene trasmesso al destinatario; all'arrivo il pacchetto subisce un processo complementare a quello precedente, nella risalita attraverso la pila TCP/IP esso viene man mano privato dei suoi header fino ad arrivare al livello applicativo in cui il payload originale può essere visualizzato.
Molti dei nostri dati personali fluiscono in rete grazie a questa tecnologia, perciò, la sicurezza di questi sistemi di interscambio rappresenta un pilastro nella tutela della nostra privacy. Non solo la nostra privacy, ma anche infrastrutture critiche come centrali energetiche, ospedali, banche e simili utilizzano la rete come mezzo di interscambio di informazioni, negli ultimi decenni a causa di questa grande quantità di informazioni circolanti in rete gli attacchi informatici si sono intensificati in frequenza e portata dei danni, anche in Italia; per questa ragione la sicurezza informatica assume oggi un ruolo quanto mai importante nella nostra vita.

Sicurezza Informatica

La Sicurezza Informatica, secondo l'enciclopedia Treccani, è un ramo dell'informatica che si occupa di tutelare i sistemi di elaborazione, siano essi reti complesse o singoli computer, dalla possibile violazione, sottrazione o modifica non autorizzata di dati riservati in essi contenuti. Un sistema informatico si può definire sicuro se è in grado di preservare la confidenzialità, l'integrità e la disponibilità dei dati che ha in gestione.
Con confidenzialità si intende la protezione del dato nel momento in cui viene scambiato tra un mittente e un destinatario, eventuali terze parti non devono essere in grado di carpire informazioni dalla comunicazione.
Con integrità di un dato si intende la protezione del dato contro le modifiche non autorizzate a cui potrebbe essere soggetto, un sistema che garantisce l'integrità può predisporre dei meccanismi in grado di rilevare un tentativo di manomissione delle informazioni.
Con disponibilità si intende la prevenzione della non accessibilità ai legittimi fruitori di dati o risorse che il sistema mette a disposizione.

Con il termine hacking si intende la capacità di intaccare un sistema, scoprirne le vulnerabilità e sfruttarle per ottenere un accesso allo stesso col fine di estorcere informazioni confidenziali, danneggiare l'integrità dei dati o renderli inaccessibili.
La prima fase di un'intrusione consiste in un tentativo di attacco, generalmente di tipo esplorativo, una volta scoperta una vulnerabilità nel sistema essa può essere sfruttata tramite un exploit, ovvero un pezzo di codice in grado di generare comportamenti inaspettati all'interno del sistema come, ad esempio, ottenere un accesso non autorizzato.
Esistono vari tipi di attacchi informatici, essi sfruttano diverse mal configurazioni o errori nei sistemi per scopi malevoli, tra i più noti e diffusi troviamo:

- Port Scanning: è un attacco che consiste nell'enumerazione delle porte e dei servizi attivi su un certo IP o intervallo di IP, un software tipicamente usato in questo tipo di azioni è Nmap;
- Man-in-the-middle (MitM): è un attacco in cui l'attaccante riesce a inserirsi nel mezzo di una comunicazione tra due utenti, questo permette di compromettere la confidenzialità delle informazioni trasmesse;
- User to Root (U2R): è un attacco che consente ad un attaccante, che ha già avuto accesso ad un account utente all'interno del sistema, di scalare i suoi privilegi diventando così amministratore, in questo modo acquisisce il pieno controllo del sistema e può, ad esempio, compromettere l'integrità dei dati contenuti;
- Denial of Service (DoS): è un attacco in cui l'attaccante sovraccarica il sistema rendendolo inutilizzabile per un certo periodo di tempo, questo tipo di azione compromette la disponibilità dei dati o dei servizi del sistema;

Tali tentativi di violazione e molti altri possono essere contrastati mediante strumenti sia software che hardware.
In questo lavoro di tesi ci siamo occupati di soluzioni software volte a prevenire attacchi informatici portati tramite la tecnologia Ethernet, esistono già diversi mezzi per contrastare questo tipo di azioni malevole: antivirus, firewall e Intrusion Detection System.

Antivirus

Un antivirus è un software capace di rilevare e potenzialmente rendere inoffensivi codici dannosi o malware penetrati nel nostro sistema. Gli antivirus odierni si basano su una serie di signature (firme) predefinite basate sullo studio e la modellizzazione di vettori di attacco noti, queste regole vengono aggiornate quasi quotidianamente dai fornitori del servizio in modo che l'azione di filtraggio sia, quanto più possibile, in linea con l'evoluzione delle tecniche di attacco. Il loro funzionamento si basa proprio sul confronto tra il codice potenzialmente dannoso in esame e un insieme di signature estratte da malware noti o codice malevolo conosciuto, per questo motivo nuovi virus (denominati Zero-day) o versioni modificate di virus noti potrebbero non venir riconosciute da questi strumenti.

Firewall

Nell'ambito della sicurezza web il firewall (Figura 3) è il primo sistema di protezione della nostra rete privata (LAN), esso permette di consentire o bloccare connessioni da o verso il nostro sistema utilizzando una serie predefinita di regole, ad esempio i pacchetti vengono filtrati in base all'indirizzo IP sorgente o destinazione e alle rispettive porte, queste regole vengono definite dal proprietario o gestore della rete informatica. Il firewall, per il ruolo che ricopre, deve essere altamente sicuro, esso infatti è l'unico punto di contatto tra la rete privata che vogliamo proteggere (LAN) e la rete esterna (WAN), la sua posizione fisica nella rete gli permette di far transitare solo le connessioni autorizzate sulla base delle regole definite sopra citate.

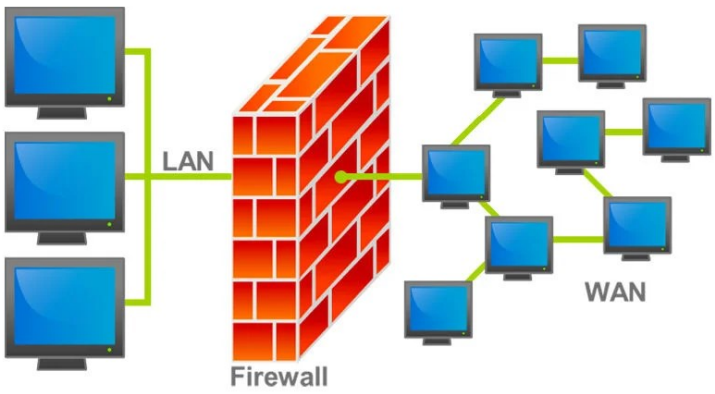

Figura 3. Rappresentazione del posizionamento tipico di un firewall in una LAN.

Sempre a causa della posizione che occupa nell'infrastruttura di rete esso è in grado di bloccare le sole minacce provenienti dall'ambiente esterno, non è quindi in grado di monitorare eventuali minacce sviluppatesi all'interno della rete privata, il traffico locale generato e indirizzato ad altri host locali non transita infatti per il firewall.

Intrusion Detection System

Gli Intrusion Detection System (IDS), secondo il National Institute of Standards and Technology (NIST) (*1*), sono sistemi atti a monitorare gli eventi che avvengono in un sistema o il traffico di rete che lo coinvolge. Dall'analisi di questi eventi è possibile rilevare segni di intrusioni, definiti come tentativi di compromissione della confidenzialità, integrità, disponibilità dei dati, o per aggirare i meccanismi di sicurezza di un sistema o di una rete.
Un IDS è quindi una componente hardware/software in grado di monitorare sistemi e di lanciare eccezioni nel caso in cui avvenga un'intrusione o sia riscontrata un'anomalia di qualche tipo.
Gli IDS si possono differenziare sulla base del metodo di monitoraggio:

- Host-based IDS: un IDS basato su host (HIDS) è un componente software specializzato nell'analisi e monitoraggio di un singolo host o computer. Il principio di funzionamento degli HIDS è basato sulla considerazione che se l'attaccante riesce a forzare il sistema egli lascia traccia del suo passaggio, probabilmente egli installerà nel computer componenti deputate alla raccolta di informazioni o alla gestione remota della macchina. L'installazione di questi programmi modifica lo stato interno della macchina e l'HIDS dovrebbe essere così in grado di rilevare un'intrusione. Per raggiungere l'obiettivo sopra citato un HIDS sorveglia il funzionamento del sistema operativo verificando che le politiche di sicurezza non siano aggirate o scavalcate, per farlo confronta istantanee consecutive dello stato generale del sistema alla ricerca di modifiche. Gli HIDS monitorano lo stato delle risorse di sistema e dei processi associati all'esecuzione dei programmi nell'host, verificano anche che i file critici di sistema non vengono modificati e permette quindi di segnalare comportamenti potenzialmente dannosi all'utente.
- Network-based IDS: i Network Intrusion Detection System (NIDS) sono degli strumenti informatici software e/o hardware dedicati all'analisi del traffico di rete di una LAN al fine di individuare anomalie nei flussi o probabili intrusioni informatiche.
  I più comuni NIDS sono organizzati secondo un'architettura master-slave in cui diversi collettori di pacchetti di rete (slave) vengono collegati agli switch dislocati all'interno della rete LAN. I vari slave comunicano con un hardware centralizzato (master), che in genere si appoggia ad un Database. Le attività anomale che possono essere rilevate da un NIDS sono quelle tipicamente connesse all'attività di rete come accessi non autorizzati, propagazione di software malevolo, acquisizione abusiva di privilegi, intercettazione del traffico (sniffing), negazioni di servizio (DoS) e svariate altre.
  Il posizionamento dell'IDS all'interno della rete LAN permette al sistema di intrusione di notificare attività sospetta generata dall'esterno e diretta all'interno così come quella generata all'interno e diretta sempre all'interno, questo succede se il traffico attraversa lo switch in cui vi è collegato il NIDS.

I due metodi di monitoraggio possono essere combinati per offrire un doppio livello di sicurezza, il primo è volto a monitorare i cambiamenti interni del sistema, il secondo monitora la rete a cui è collegato il dispositivo e notifica attività sospette provenienti da essa.

Gli IDS basati sulla rete possono essere usati in due diverse modalità di monitoraggio:

- modalità mirroring: in questa modalità l'IDS viene collegato a una porta del router/switch che prende il nome di porta di mirroring; il traffico in transito nel router/switch viene copiato integralmente e reso disponibile attraverso la porta di mirroring a cui è collegato l'IDS. In questa modalità il sistema, operando solamente su una copia del traffico, non è quindi in grado di segnalare in tempo reale un'attività dannosa in quanto il traffico analizzato dall'IDS è, in realtà, già stato consegnato al destinatario; maggiore è la velocità con la quale il sistema è in grado di generare avvisi di intrusione maggiore sarà la vicinanza ad un sistema di allarme real-time. Questo tipo di modalità è stato diffusamente utilizzato in quanto all'epoca della prima introduzione di questo tipo di sistemi la profondità di calcolo necessaria per il rilevamento di intrusioni combinata ad una minor capacità computazionale avrebbero portato ad un peggioramento della qualità della connessione;
- modalità in-line: in questa modalità l'IDS viene posto nel mezzo della linea dati, il traffico viene quindi fatto fluire attraverso l'IDS che in questo modo può reagire istantaneamente bloccando la connessione anomala, questo tipo di sistemi prende il nome di Intrusion Detection and Prevention System (IDPS).
  A causa dell'azione di filtraggio dell'IDPS il sistema deve avere prestazioni ottime per non inficiare la qualità della connessione di rete.

Gli IDS possono inoltre essere divisi in due categorie basate sul criterio di rilevamento dell'attività sospetta:

- signature-based IDS: gli IDS basati su signature hanno un funzionamento simile a firewall e antivirus, essi fondano la loro conoscenza su un database continuamente aggiornato con i nuovi pattern malevoli, confrontando il traffico in entrata con questo insieme di signature emettono avvisi di attività sospetta ogni qual volta ci si sia una corrispondenza.
- anomaly-based IDS: gli IDS basati sul riconoscimento di anomalie operano in modo simile ma le regole non sono basate su pattern malevoli noti bensì sono scritte in funzione del normale comportamento del sistema, tutto ciò che devia da questo standard è segnalato come anomalia, in questa specifica applicazione risulta utile introdurre tecniche di Intelligenza Artificiale (IA) per la segnalazione di anomalie.
  L'idea è quella di far apprendere all'IDS basato su IA il normale comportamento del sistema (inteso come normale comportamento degli utenti e dei dispositivi ad essi assegnati), una volta che il modello è stato addestrato esso è in grado di distinguere le successive attività tra normali e anomale basandosi sulla propria abilità di discernimento, in questo modo anche pattern malevoli non noti in precedenza possono essere intercettati dal sistema di allarme e segnalati all'utente che può così essere messo al corrente di una anomalia nel sistema, queste non sempre sono conseguenza di un attacco in corso ma potrebbero essere semplicemente attività non consuete per il sistema.

Questi differenti approcci possono essere combinati:

- in parallelo: in questo modo il traffico viene analizzato da due diversi sistemi, uno basato su signature e uno basato sul riconoscimento di anomalie, così si ha una doppia valutazione per determinare se il dato traffico è malevolo o meno;
- in serie: in questo modo si posiziona un IDS basato sul rilevamento di anomalie, che spesso ha un alto tasso di falsi positivi, in cascata ad uno basato su signature che può così notificare con precisione che tipo di attacco è avvenuto; questo metodo perde però la capacità di notificare attività sospetta generata da nuovi tipi di attacchi informatici non noti, un importante pregio che un sistema puro basato sul riconoscimento di anomalie preserva.

Nessuno tra gli strumenti di protezione dei sistemi informatici sopra citati può, da solo, garantire la sicurezza del sistema: per questo motivo le varie tecniche vengono stratificate in un modello a cipolla, questo permette di avere un alto livello di sicurezza per le nostre infrastrutture e i dati contenuti al loro interno.
La protezione offerta non è comunque mai totale, si tenta di raggiungere un buon compromesso tra rendere difficile l'ingresso dell'attaccante e mantenere i costi di gestione commisurati al livello di rischio che l'azienda può accettare.

Capitolo 2

In questo capitolo verrà presentato un quadro generale sull'Intelligenza Artificiale, sul machine learning e sul deep learning, con un approfondimento particolare per la struttura direttamente connessa con il lavoro svolto in questa tesi, l'Autoencoder.

Intelligenza Artificiale

L'intelligenza artificiale consiste nell'emulazione di alcune facoltà cognitive umane tra le quali troviamo: comprensione di testi scritti, riconoscimento di immagini e comprensione del linguaggio, solo per citarne alcune. L'intelligenza artificiale permette anche di aiutarci nella risoluzione di compiti che sarebbero impossibili da processare per il nostro cervello in quanto coinvolgono grandi quantità di dati simultaneamente.
Ad oggi la capacità di computazione, che continua a crescere nel rispetto della legge di Moore, è tale da permettere l'elaborazione di grandi insiemi di dati chiamati big data, questo permette l'applicazione di tecniche di intelligenza artificiale a quasi tutti i settori della conoscenza in cui è possibile collezionare un'ingente mole di dati, così ingente da non permettere un'analisi umana dettagliata.
L'intelligenza artificiale ha iniziato a suscitare interesse quando, per la prima volta, è stata in grado di portare a termine con successo attività che, per concezione comune, sembravano non adatte ad una risoluzione programmata e prestabilita tipica dei calcolatori ma che necessitano di un ragionamento complesso, di previsioni e strategie. Celebre è l'esempio di Deep Blue, il primo software che nel 1998 sconfisse il campione mondiale in carica di scacchi. Un altro esempio pertinente è quello di Alpha-Go, il primo software in grado di sconfiggere il campione mondiale in carica del gioco orientale di strategia Go.
Nell'ambito del riconoscimento di immagini abbiamo due esempi famosi, AlexNet e ResNet, rispettivamente i primi classificati nella competizione mondiale di riconoscimento immagini nel 2012 e nel 2015.
Da questi esempi, anche se in numero ristretto, si può però cogliere il concetto fondamentale che sta alla base dell'intelligenza artificiale ovvero permettere che un calcolatore acquisisca abilità "umane" simulando il comportamento delle stesse tramite algoritmi. Questi algoritmi si differenziano da quelli classici in quanto permettono al sistema di imparare dai dati di input che ricevono modificando la struttura interna del modello al crescere degli esempi che gli vengono presentati.

Machine Learning

Il Machine Learning (ML), in italiano Apprendimento Automatico, è un sottoinsieme di tecniche facenti parte dell'Intelligenza Artificiale. Esso è una variante della programmazione tradizionale nella quale in una macchina si predispone l'abilità di apprendere qualcosa dai dati in maniera autonoma, senza istruzioni esplicite. Gli algoritmi di apprendimento automatico sono utilizzati in un'ampia varietà di applicazioni per le quali risulta difficile o non fattibile sviluppare algoritmi convenzionali per eseguire i compiti richiesti.
Questa tipologia di algoritmi è denominata data-driven ovvero guidata dai dati, compiti di diversa natura possono essere risolti dagli stessi algoritmi in quanto il risultato finale non dipende dal codice stesso ma dai dati che vengono elaborati al proprio interno. I dati devono essere rappresentati in un formato che ne permetta l'elaborazione, in generale essi vanno rappresentati numericamente come un vettore n-dimensionale, ognuna delle n dimensioni rappresenta una caratteristica (features) del dato stesso, per fare un esempio potremmo rappresentare diverse persone con un vettore contenente altezza e peso espresse numericamente.
Il processo di apprendimento si articola in tre fasi:

- training: la prima fase consiste nell'addestrare il modello sui dati contenuti nel dataset di addestramento denominato training-set. L'addestramento si svolge presentando ripetutamente al modello il training-set, ogni presentazione completa al modello dell'insieme di dati di train prende il nome di epoca, al crescere delle epoche il modello si specializza sui dati che gli sono presentati modificando la propria struttura interna. Parallelamente il modello viene validato su una porzione separata dei dati di train chiamata validation-set, questa operazione è necessaria per non incorrere nell'overfitting o nell'underfitting. L'overfitting è un fenomeno che consiste nell'eccessiva specializzazione del modello sui dati di train, questa eccessiva specializzazione porta le performance ad essere eccellenti nell'addestramento ma pessime nell'ambiente reale per il quale il modello è stato creato, questo avviene poiché il modello ha "imparato a memoria" la struttura dei dati del training-set e non è grado di generalizzare sui nuovi dati che gli vengono presentati in ambiente reale o in fase di test. L'underfitting è il fenomeno opposto in quanto il modello non è in grado di comprendere la struttura interna dei dati, le prestazioni sia sui dati di train che in ambiente reale sono scarse poiché il modello era troppo semplice o non è stato addestrato per un numero di epoche sufficienti affinché potesse comprendere la struttura interna dei dati;
- testing: la fase di test consiste nel presentare al modello addestrato nuovi dati, mai visti in precedenza, in questo modo si possono calcolare le prestazioni dello stesso e in caso non fossero sufficienti si può addestrare nuovamente il modello modificando alcuni parametri per tentare di ottenere performance migliori.

Per addestrare un modello possiamo servirci di diverse tecniche che dipendono dall'applicazione per cui il modello è stato creato:

- Apprendimento supervisionato: questi algoritmi costruiscono un modello matematico a partire da un dataset etichettato, è quindi presente una ulteriore feature ovvero la classe di appartenenza di ogni singolo esempio dell'insieme di dati di train, di validazione e di test.

  I principali algoritmi di apprendimento supervisionato sono la classificazione e la regressione. Gli algoritmi di classificazione, una volta addestrati, permettono di

assegnare ad ogni dato presentato in fase di test la sua classe di appartenenza, vengono utilizzati solitamente quando i possibili valori di output sono limitati ad un certo insieme. Gli algoritmi di regressione permettono di fare interpolazione, perciò, vengono utilizzati quando i valori di output possono assumere un qualunque valore reale all'interno di un certo intervallo.

- Apprendimento non supervisionato: questi algoritmi utilizzano set di dati non etichettati, non è perciò presente l'etichetta classe per gli esempi. La loro particolarità è quindi quella di imparare schemi da dati che non sono stati categorizzati precedentemente.

  L'applicazione più comune di queste tecniche è quella del clustering, in italiano raggruppamento; questa tecnica consente di creare raggruppamenti di dati, apparentemente non correlati, sulla base della vicinanza degli stessi nello spazio multi-dimensionale che li contiene; ciò è molto utile quando si vogliono creare dei modelli che mettano in relazione caratteristiche dei dati non noti a priori, e, in particolare, casi in cui è impossibile effettuare manualmente questo raggruppamento, ad esempio a causa dell'elevato numero di features dell'insieme di dati, esso infatti è uguale al numero di dimensioni dello spazio che li contiene.
- Apprendimento semi-supervisionato: si tratta di modelli ibridi in cui il set di dati è parzialmente etichettato, sono quindi presenti solo alcuni esempi nel training-set recanti l'etichetta classe mentre molti altri sono non etichettati; diversi studi hanno mostrato come l'utilizzo di dati non etichettati, in congiunzione con una piccola quantità di dati etichettati, possono produrre un aumento non indifferente nell'accuratezza del modello.
- Apprendimento per rinforzo: questo approccio si fonda sul concetto di ricompensa, il sistema è spinto a prendere delle decisioni, in base alla correttezza della scelta viene premiato o punito con delle ricompense, lo scopo è massimizzare questi premi.

Ognuno di questi approcci fonda la sua bontà e qualità sul corretto utilizzo dei dati.
Il trattamento dei dati di input prende il nome di pre-processamento, esso ricopre un ruolo molto importante soprattutto quando i dati presentano anomalie al loro interno, valori fuori scala, etichette errate, grande variabilità dei valori delle features e simili. Tecniche di pre-processamento comuni comprendono, ad esempio, la rimozione degli outlier (i valori fuori scala) e la normalizzazione: tali tecniche permettono di lavorare su un training set omogeneo e pulito dai valori anomali portando così ad un miglioramento nelle prestazioni del modello.

Deep Learning

Il Deep Learning (DL), in italiano Apprendimento Approfondito, è un sottoinsieme di tecniche facenti parte del Machine Learning. Si tratta di un insieme di metodi basati su reti neurali artificiali profonde quindi formate da almeno un livello tra lo strato di input e lo strato di output (definito livello hidden o nascosto).

Le reti neurali artificiali (ANN) sono modelli computazionali composti di neuroni artificiali capaci di imparare a risolvere dei compiti senza essere stati precedentemente codificati per risolverli ma imparando dall'esperienza accumulata durante la fase di addestramento.

Una rete neurale con almeno un livello tra lo strato di input e lo strato di output prende il nome di Deep Neural Network (Rete Neurale Profonda). Il Teorema di Approssimazione universale sancisce che una funzione reale può essere approssimata con un arbitrario grado di precisione da una rete neurale con almeno uno strato hidden (strato interno nascosto). Questo potrebbe far pensare che non serva spingersi oltre i 3 livelli in una rete neurale, l'affermazione non è però del tutto vera: anche se una rete neurale con 3 livelli totali può approssimare con arbitraria precisione una qualsiasi funzione reale aggiungere ulteriori strati hidden ha portato empiricamente a risultati migliori per diverse applicazioni (*2*), il teorema sopra citato non fornisce infatti un metodo di costruzione della data rete ma si limita ad affermare che tale costruzione è possibile.

Per quanto le tecniche di Deep Learning siano di recente applicazione esse affondano le proprie radici nel passato. Le reti neurali multistrato venivano già studiate negli anni Ottanta, ma soltanto negli ultimi dieci anni si è riusciti a dimostrare la loro utilità in un'ampia gamma di applicazioni grazie alla capacità computazionale raggiunta ad oggi e alla grande mole di dati così analizzabile.

Le reti neurali hanno infatti trovato ampia applicazione recentemente perché in grado di risolvere diversi compiti, eccellono particolarmente nel riconoscimento e processamento di immagini e caratteri poiché in grado di accettare input di elevate dimensioni senza bisogno di pre-processamento, sono in grado di processare il linguaggio naturale, leggere, creare dipinti e molto altro.

Il motore delle reti neurali (profonde e non) è il Perceptron, in italiano percettrone; esso è stato concepito verso la fine degli anni '50 dallo psicologo statunitense Frank Rosenblatt.

Il percettrone (Figura 4) è nato come imitazione artificiale del neurone biologico ed è infatti strutturalmente molto simile allo stesso.

Il percettrone è il componente principale e fondamentale per una rete neurale, è il neurone artificiale: si tratta di un particolare elemento che ha la capacità di prendere in input dei dati, elaborarli e restituire un risultato in output:

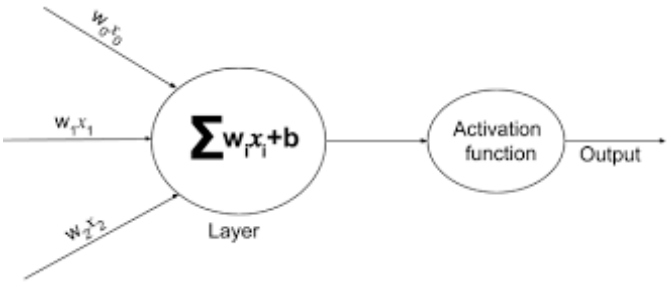

Figura 4. Schematizzazione di un percettrone.

Il percettrone è diviso in due strati, abbiamo uno strato di input composto da collegamenti multipli, e uno strato di output composto da un singolo collegamento.
Il percettrone riceve, come detto, più input, essi sono il risultato della moltiplicazione di uno scalare (rappresentati in figura da x_i) per un ramo pesato, alla somma di questi prodotti è sommato un ulteriore scalare, il bias (rappresentato in figura da b), il risultato finale costituisce l'input del neurone, questo input è passato ad una funzione di attivazione che restituisce l'output in funzione dell'input ricevuto, esso sarà moltiplicato per i vari rami pesati (rappresentati in figura come w_i al variare di i) e diventerà quindi parte dell'input per i neuroni del livello successivo.
La funzione di attivazione di base è detta funzione a gradino, la sua espressione analitica è:

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

mentre la sua rappresentazione grafica è:

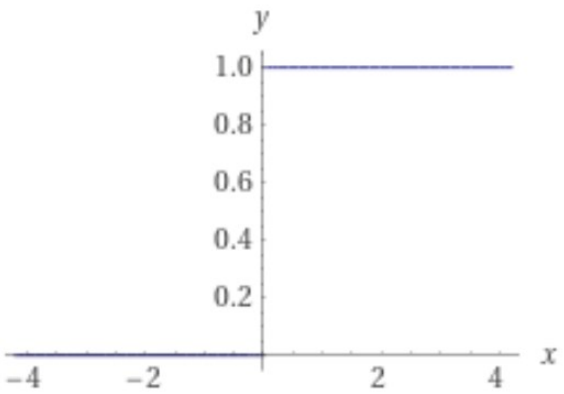

Figura 5. Rappresentazione grafica della funzione a gradino.

questa funzione viene utilizzata, ad esempio, quando si ha un problema di classificazione binaria, dove l'output deve essere un'etichetta binaria.
Esistono molti problemi in cui avere solo due possibili valori di output non è sufficiente: si utilizzano perciò diverse funzioni di attivazioni, tra le quali:

- Sigmoide: è una funzione derivabile, mantiene l'output limitato all'intervallo [0,1], spesso usata per introdurre non linearità alla rete in modo da poter modellare superfici di discriminazione non lineari, analiticamente:

$$f(x) = \frac{e^t}{1 + e^t}$$

graficamente:

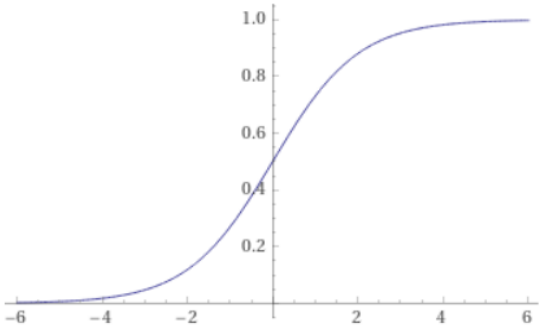

Figura 6. Rappresentazione grafica della funzione sigmoide.

- Rectified Linear Unit (ReLu): funzione con output limitato nell'intervallo [0,1], spesso usata per la velocità computazionale che porta, anche la derivata è costante ed è la funzione a gradino sopra citata, analiticamente:

$$f(x) = max\ (0, x)$$

graficamente:

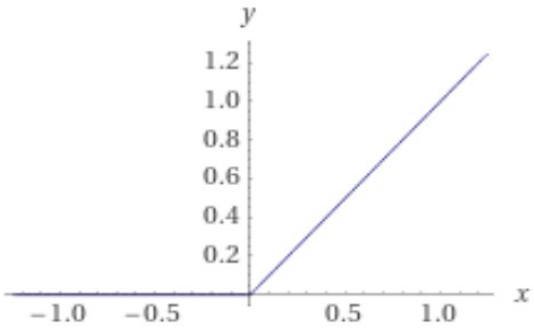

Figura 7. Rappresentazione grafica della funzione ReLu.

la ReLu ha anche delle varianti molto simili in cui però l'output per input corrispondenti a valori minori di 0 non è 0 ma un valore molto vicino e negativo, ad esempio esistono ELu, LeakyReLu e GeLu;

- Tangente Iperbolica: molto simile alla sigmoide, è una funzione derivabile, l'output è limitato nell'intervallo [-1, 1], analiticamente:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

graficamente:

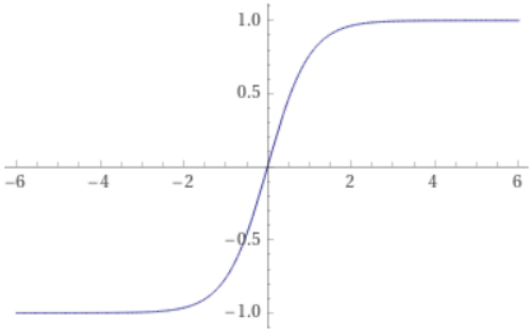

Figura 8. Rappresentazione grafica della tangente iperbolica.

Una rete neurale è costituita da interconnessioni di neuroni organizzati in livelli (layer), ognuno dei quali contiene un determinato numero di neuroni; connettendo diversi livelli possiamo creare infinite topologie di rete.
In una rete neurale possiamo denominare i livelli in base alla posizione che occupano nella rete stessa, abbiamo così:

- input layer: livello di input con un numero di neuroni che corrisponde al numero di features degli esempi del training-set;
- output layer: livello di output che solitamente conta un singolo neurone per le applicazioni comuni, in caso di problemi di multi-classificazione l'etichetta è assegnata in funzione di soglie prestabilite sui valori di output possibili per la funzione di attivazione dell'ultimo neurone;
- hidden layers : livelli nascosti, in reti neurali non profonda è un singolo livello che ha spesso un numero di neuroni compreso tra il numero di neuroni del livello di input e quelli del livello di output, per reti deep sono più livelli di neuroni che rispettano spesso la caratteristica sopra descritta.

Una rete neurale in cui non sono presenti cicli prende il nome di rete FeedForward Neural Network (FNN), in questa topologia di rete le sole connessioni valide sono tra neuroni di livelli contigui, i neuroni di un livello sono quindi connessi ai soli neuroni dei livelli precedente e successivo.

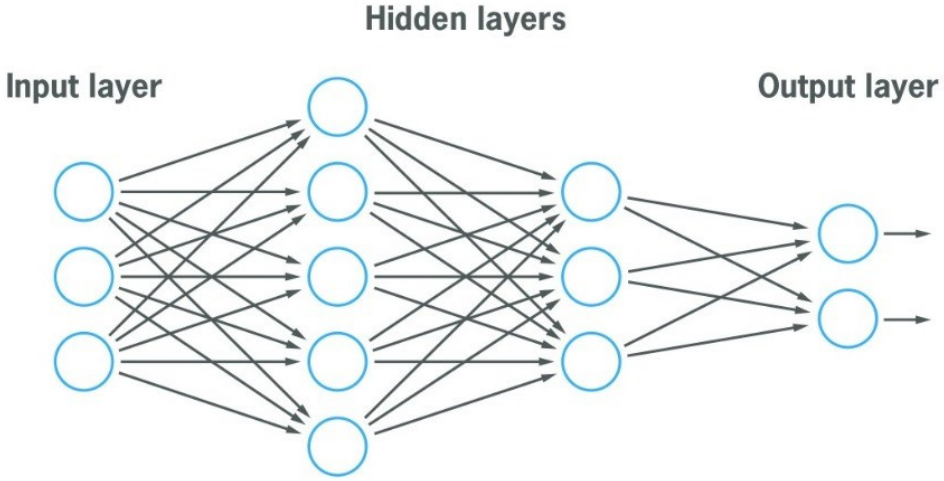

Figura 9. Rappresentazione di una rete Feed-Forward.

Le reti che prendono il nome di Recurrent Neural Network permettono invece connessioni tra neuroni dello stesso livello così come tra neuroni di livelli non contigui.
Nelle reti FeedForward le connessioni tra diversi livelli sono di diverso tipo, ad esempio:

- Fully-connected: ogni neurone in un livello è connesso a tutti i neuroni del livello successivo;
- Pooling: gruppi di neuroni condividono gli stessi pesi, così facendo i pesi totali della rete diminuiscono.

Una volta che la rete è stata topologicamente definita inizia il training; il processo si compone di due fasi:

- Forward propagation: fase di propagazione in avanti, gli esempi del training set vengono presentati alla rete, ogni feature dell'input viene processata da un diverso neurone

dell'input layer, questi procedono col calcolo dell'output che diventa input per i neuroni del livello successivo, fino ad arrivare all'output layer;

- backpropagation: fase di propagazione all'indietro o di retropropagazione dell'errore, la rete calcola il gradiente (ovvero una derivata parziale) della funzione di perdita in funzione al peso dei rami attuali, l'aggiornamento dei pesi avviene per mezzo di un ottimizzatore, ne esistono di diversi tipi ad esempio: Stochastic Gradient Descent, Adaptive Gradient Algorithm, Adam, Root Mean Square Propagation e altri.

L'aggiornamento dei pesi della rete può avvenire per ogni singolo esempio del training set oppure per sottoinsiemi del training set o alla presentazione completa dell'intero training set.
Addestrare una rete neurale significa quindi stabilire il valore ottimale dei pesi tale da minimizzare il valore della loss function (funzione di perdita).
Esistono varie loss function basate su diverse formule d'errore, esse vanno scelte coerentemente problema che si vuole risolvere, ad esempio:

- Mean Squared Error (MSE): errore quadratico medio, questa funzione di perdita calcola l'errore quadratico medio tra il valore predetto e il valore reale dell'esempio in esame tramite la formula:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(X_i - \hat{X}_i)^2$$

- Mean Absolute Error (MAE): errore medio assoluto, questa funzione calcola l'errore medio assoluto tra il valore predetto e reale dell'esempio in esame tramite la formula

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|X_i - \hat{X}_i|$$

- Binary Cross Entropy (BCE): entropia binaria incrociata, questa funzione calcola l'entropia binaria incrociata tra valore predetto e valore reale, utile per problemi di classificazione binaria tramite la formula

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}(X_i log\,\hat{X}_i + (1 - X_i) log\,(1 - \hat{X}_i))$$

In questo insieme di formule (MSE, MAE, BCE) X_i rappresenta il valore della feature i-esima del dato di input e $\hat{X}_i$ il valore della feature i-esima del dato di input ricostruito.
Le topologie di rete, come detto sopra, sono infinite, alcuni modelli hanno riscosso particolare interesse per i risultati ottenuti e sono quindi divenute topologie note, la topologia usata in questo lavoro di tesi prende il nome di Autoencoder.

Gli Autoencoder sono tipi particolari di reti neurali nate con lo scopo di ricavare dei modelli di codifica efficiente dei dati in un approccio non supervisionato.
Il nome di questa particolare topologia di rete, Autoencoder, tradotto letteralmente in Auto codificatore, è dato dal fatto che, intuitivamente, questa impara a generare una codifica degli esempi da cui è in grado di ricostruire gli esempi originali con un errore minimo.
Questa rete è costituita da due blocchi con compiti complementari:

- encoder: prima parte della rete che mappa gli input in una rappresentazione codificata o latente
- decoder: seconda parte della rete che mappa la rappresentazione codificata o latente in una ricostruzione dell'input

Più precisamente, l'obiettivo di un Autoencoder è quello di codificare dati in un formato compresso e a partire da quel dato compresso cercare di decomprimerlo alla grandezza iniziale minimizzando le differenze tra dato iniziale e dato ricostruito (*3*). Spesso questa tecnica viene utilizzata con lo scopo di effettuare una riduzione del numero di feature, così da ridurre la dimensione dello spazio n-dimensionale che contiene gli esempi del dataset; si addestra un Autoencoder formato da encoder e decoder per assicurare che le features conservate nel dato compresso siano le più rilevanti del dato stesso, è da queste, infatti, che il decoder riesce a ricostruire una versione molto simile del dato iniziale.
L'architettura standard di un Autoencoder è quella di una rete feedforward dove input e output layer hanno lo stesso numero di neuroni, mentre gli hidden layer ne hanno un numero minore, la struttura tipica è mostrata in figura

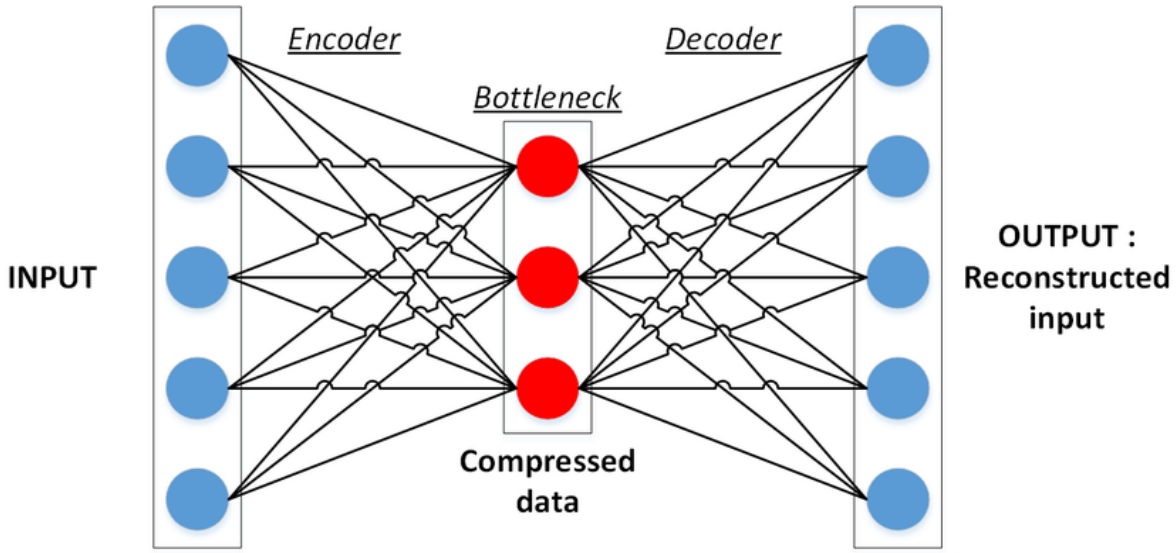

Figura 10. Rappresentazione della topologia di un Autoencoder con bottleneck centrale.

Il livello centrale prende il nome di bottleneck, tradotto collo di bottiglia, poiché nella topologia della rete esso è il livello che conta il minor numero di neuroni, esso rappresenta una versione codificata (rappresentazione latente), quindi a ridotta dimensionalità, dell'input.

Esistono diversi tipi di Autoencoder specializzati nella risoluzione di diversi compiti, ad esempio:

- Deep Autoencoder: gli Autoencoder profondi sono semplicemente degli Autoencoder in cui sono presenti più livelli hidden oltre al sempre presente livello bottleneck, bisogna prestare attenzione nell'aggiungere livelli ulteriori in quanto un numero elevato di neuroni potrebbe creare una superficie troppo aderente ai dati di training rendendo così difficile la generalizzazione su nuovi esempi;
- Denoising Autoencoder: gli Autoencoder per la rimozione del rumore accettano un input parzialmente danneggiato e tentano di recuperare l'input originale non distorto. Con questo approccio il modello non è in grado di sviluppare semplicemente una mappatura che memorizzi i dati di allenamento perché input e output target non sono più gli stessi. Piuttosto il modello apprende un campo vettoriale per mappare i dati di input verso un campo di dimensioni inferiori; se questo collettore descrive accuratamente i dati naturali, abbiamo effettivamente rimosso il rumore aggiunto ai dati di train;
- Sparse Autoencoder: gli Autoencoder sparsi ci offrono un metodo alternativo per introdurre un collo di bottiglia delle informazioni senza richiedere una riduzione del numero di nodi nei nostri livelli nascosti. Per farlo la funzione di perdita sarà fatta in modo tale da penalizzare le attivazioni all'interno di un livello. Per ogni data osservazione la rete è spinta ad apprendere una codifica e una decodifica che si basano solo sull'attivazione di un piccolo numero di neuroni dei livelli hidden;
- Contractive Autoencoder: L'obiettivo di un Autoencoder contrattivo è apprendere una solida rappresentazione che sia poco sensibile a piccole variazioni nei dati. La robustezza della rappresentazione dei dati viene ottenuta applicando un termine di penalità alla funzione di perdita. L'Autoencoder contrattivo è un'altra tecnica di regolarizzazione, proprio come gli Autoencoder sparsi e quelli per la rimozione del rumore;
- Convolutional Autoencoder: gli Autoencoder nella loro formulazione tradizionale non tengono conto del fatto che un segnale può essere visto come una somma di altri segnali. Gli Autoencoder convoluzionali utilizzano l'operatore di convoluzione per sfruttare questa osservazione. Imparano a codificare l'input in un insieme di segnali semplici e poi provano a ricostruire l'input da essi. Sono gli strumenti all'avanguardia per l'apprendimento non supervisionato dei filtri convoluzionali. Una volta appresi questi filtri, possono essere applicati a qualsiasi input per estrarre le caratteristiche principali. Queste caratteristiche compatte, quindi, possono essere utilizzate per eseguire qualsiasi attività che richieda una rappresentazione compatta dell'input, come la classificazione;
- Variational Autoencoder: I modelli variazionali dell'Autoencoder fanno forti ipotesi sulla distribuzione delle variabili latenti. Usano un approccio variazionale per l'apprendimento della rappresentazione latente, che si traduce in una componente di perdita aggiuntiva e uno stimatore specifico per l'algoritmo di addestramento chiamato stimatore Stochastic Gradient Variational Bayes;

In questo lavoro di tesi è stato usato un Deep Autoencoder composto da 7 livelli totali.
Le proprietà dell'Autoencoder sono state utilizzate per il rilevamento di anomalie, nel capitolo dedicato alla presentazione della metodologia seguita per la costruzione dell'IDS questo concetto sarà descritto dettagliatamente.
Per la realizzazione di reti neurali il linguaggio di programmazione più utilizzato, a causa della grande disponibilità di librerie ottimizzate a tal scopo, è Python.

Python

Python è un linguaggio di programmazione ad alto livello Ideato da Guido van Rossum negli anni Novanta. Tra le caratteristiche principali troviamo la semplicità di utilizzo e la flessibilità nell'applicazione. Le caratteristiche che rendono python distinguibile dagli altri linguaggi di programmazione di uso comune (come Java e C++) sono le variabili non tipizzate, l'uso dell'indentazione per la sintassi delle istruzioni al posto delle parentesi e della punteggiatura e la presenza di una vasta gamma di librerie e funzioni che permettono la scrittura agevole di software complesso anche ai neofiti della programmazione.
Utilizzare Python come linguaggio di programmazione è stato cruciale per poter accedere ad un grande numero di librerie open source dedicate alla manipolazione di dati e all'Intelligenza Artificiale, di seguito un breve elenco:

- NumPy (*4*): libreria per la manipolazione di grandi array multidimensionali e matrici tramite l'ausilio di una collezione di funzioni matematiche di alto livello;
- Pandas (*5*): libreria per la manipolazione efficiente ad alto livello di dati strutturati, come ad esempio dataset di grandi dimensioni; offre algoritmi per tutti i classici processi di manipolazione oltre a strumenti per preprocessamento e la pulizia dei dati;
- Scikit-Learn (*6*): libreria di machine learning che offre algoritmi per i classici problemi di clustering e classificazione; è ben integrata con altre librerie e offre inoltre loss function e metriche utili per lo studio dei risultati ottenuti;
- Matplotlib (*7*): libreria che offre strumenti di visualizzazione dei dati, istogrammi, punti sparsi e altri.

Le librerie sopra citate sono state fondamentali nella fase di esplorazione e visualizzazione dei dati contenuti nel dataset NSL-KDD, esse mettono infatti a disposizione tecniche atte alla manipolazione di dati di grandi dimensioni.
Per la creazione del modello di apprendimento è stato necessario utilizzare due ulteriori moduli di Python: TensorFlow e Keras.

TensorFlow

TensorFlow (*8*) è una librearia open source per l'Intelligenza Artificiale sviluppato inizialmente dal team di Google Brain. TensorFlow comprende strumenti e risorse che permettono un processo di sviluppo facilitato di applicazioni di machine learning; l'obiettivo è ottenuto fornendo dei moduli compatibili con Python e C++, i quali sono implementati a basso livello con quest'ultimo linguaggio.
Tensorflow si configura come una libreria per la definizione e il calcolo di operazioni che coinvolgono tensori, rappresentazioni generiche di vettori e matrici a dimensioni maggiori. Un tensore non è altro che un array n-dimensionale: per capire meglio il concetto l'array di uso comune, seguendo questa data nomenclatura, è un array 1-dimensionale, una matrice è un array

2-dimensionale e per generalizzazione possiamo estendere la definizione a più di 2 dimensioni arrivando a definire array n-dimensionali.
TensorFlow fornisce quindi un'implementazione efficiente delle operazioni necessarie per eseguire i calcoli coinvolti nell'uso di tecniche di IA, per l'implementazione degli algoritmi si usa un'API dedicata, Keras.

Keras

Keras (*9*) è una API open source che fornisce un'interfaccia Python per reti neurali artificiali.
Keras funge da interfaccia per la libreria TensorFlow in particolare.
Keras è stato progettato per consentire una rapida sperimentazione con reti neurali profonde, si concentra sull'essere intuitivo, modulare ed estensibile. Pensato per essere facile ed intuibile anche al pubblico meno esperto a causa dell'esplosione della popolarità del concetto di deep learning, molti sviluppatori lo utilizzano.
All'epoca dell'introduzione di Keras non vi erano molte librerie per il deep learning; esse, inoltre, richiedevano una conoscenza approfondita e l'utilizzo di metodi di programmazione di basso livello ed erano per questo complesse da utilizzare.
Keras, al contrario risulta estremamente semplice, permettendo in particolare a ricercatori e sviluppatori di realizzare esperimenti complessi molto più velocemente e con poche righe di codice.
Essendo solo un'API di alto livello, Keras richiede una librearia di basso livello che svolga effettivamente le operazioni necessarie per rendere il modello funzionante, ovvero un componente di fondo che si occupi di eseguire i calcoli di interesse a basso livello.
TensorFlow è stato scelto per ricoprire questa funzionalità, essendosi affermato nell'ambiente come la libreria di riferimento per il machine learning e il deep learning in particolare; la conseguenza di ciò è che, man mano, lo sviluppo dei due software è andato avanti di pari passo, fino ad arrivare all'integrazione completa delle API di Keras all'interno dell'installazione principale di TensorFlow.
Keras, è stato fondamentale nella realizzazione della rete neurale, a partire dalla costruzione della topologia per arrivare all'addestramento e alla fase finale di test, esso ha messo a disposizione le componenti fondamentali per la costruzione delle reti neurali sotto forma di semplici funzioni che implementano complesse operazioni di basso livello; in particolare vengono messi a disposizione dei metodi che vanno a far parte di un oggetto principale, ovvero la classe Model, astrazione di un modello di rete neurale, il quale una volta costruito deve essere compilato e poi addestrato.
Alcuni dei moduli disponibili sono:

- Layer: modulo che consente la costruzione dei vari livelli della rete, essi possono essere di varie tipologie e possono essere semplicemente collegati tra loro in diverse configurazioni; tra i tipi possibili abbiamo Input, Dense, Activation, Dropout e Flatten, Conv1D, Conv2D, MaxPooling1D e MaxPooling2D citando i principali
- Activation: modulo che contiene diversi tipi funzioni di attivazione, in particolare quelle citate precedentemente, per i neuroni artificiali; per essere usate vanno richiamate all'interno della definizione di un livello della rete all'interno della definizione del modello tramite il metodo layer;
- Losses: modulo che contiene funzioni di perdita implementate vengono richiamate al momento della costruzione del modello tramite la funzione compile;

- Metrics: modulo che contiene metriche per la valutazione delle performance del modello in fase di addestramento; come per le funzioni di perdita vengono richiamate nel momento della costruzione del modello tramite la funzione compile;
- Optimizers: modulo che contiene vari ottimizzatori implementati e pronti all'uso, essi consistono in diversi metodi di aggiornamento dei pesi in seguito alla fase di backpropagation; come per funzioni di perdita e metriche di valutazione delle performance va specificato al momento della creazione del modello attraverso la funzione compile;
- Callbacks: insieme di funzioni che possono essere applicate a determinati stadi della procedura di allenamento; solitamente vengono utilizzate per ottenere una visione degli stati interni e delle statistiche del modello; esempi di funzioni di callback sono la Early Stopping, questa funzione blocca il training non appena la quantità monitorata dalla funzione cessa di migliorare;

Non appena il modello è stato definito e costruito è possibile iniziare la fase di addestramento sul dataset di training.
Per addestrare un modello Keras mette a disposizioni la funzione fit, essa si compone di vari iperparametri che vanno specificati per l'addestramento della rete, troviamo ad esempio:

- numero di epoche: questo numero rappresenta quante volte la rete deve processare l'intero training set prima che l'addestramento possa ritenersi concluso, questo è un iperparametro molto importante che gioca un ruolo fondamentale in termini di prestazioni finali del modello;
- batch size: è la dimensione di sottoinsiemi (letteralmente lotti) del dataset che vengono presentati alla rete, i pesi della rete vengono aggiornati solo dopo che un intero batch è stato processato dalla rete, anche questo iperparametro risulta essere fondamentale in quanto le prestazioni finali della rete dipendono dall'aggiornamento dei pesi e dalla loro inizializzazione randomica; se il batch size è grande come il training set l'algoritmo di apprendimento prende il nome di Batch Gradient Descent, se il batch size è unitario si parla di Stochastic Gradient Descent e se è compreso tra 1 e l'intera dimensione del dataset Mini-Batch Gradient Descent;
- learning rate: il tasso di apprendimento è un iperparametro di ottimizzazione che determina la dimensione del passo compiuta dall'ottimizzatore, ad ogni iterazione, mentre si sposta verso un minimo di una funzione di perdita;
- decay: decadimento dell'apprendimento, è un iperparametro che regola la velocità di discesa del gradiente della funzione di loss.

Capitolo 3

In questo capitolo verrà presentata un'analisi sulla metodologia seguita per la costruzione del prototipo di un IDS basato sulla rete ed implementato grazie a tecniche di Intelligenza Artificiale, in particolare servendosi di una topologia di rete neurale artificiale che prende il nome di Autoencoder.

Motivazioni

Il fine del progetto è stato quello di eseguire una prova di concetto volta a dimostrare come l'Intelligenza Artificiale sia un importante strumento in grado di migliorare la protezione dei dispositivi nei confronti di attacchi portati per mezzo della tecnologia Ethernet. A tal scopo è stato scelto di implementare un Sistema di Rilevamento delle Intrusioni (IDS) nella rete locale servendosi di tecniche di Intelligenza Artificiale.
In letteratura esistono diversi esempi di come queste tecniche di Machine Learning e Deep Learning vengano usate al fine rilevare anomalie nel traffico di una LAN come, ad esempio, approcci basati su tecniche di clustering (*10*), su alberi di decisione (*11*), one-class SVM (*12*), reti generative avversarie (GAN)(*13*), reti convoluzionali (*14*) e molti altri.
Nel lavoro proposto è stato usato un differente approccio che si basa sull'utilizzo di un Autoencoder come discriminatore del traffico. L'Autoencoder permette di lavorare in un approccio non supervisionato poiché la rete neurale tenta di copiare l'input nell'output e non ha quindi bisogno di lavorare su dataset etichettati. Nella nostra applicazione questa caratteristica è risultata di fondamentale importanza in quanto l'insieme di dati di training è stato costruito localmente e perciò risultava privo di etichette. Inoltre, il traffico usato per la fase di addestramento, nella particolare applicazione d'uso dell'Autoencoder, deve essere composto da sola attività normale, costruire un dataset di training localmente ha permesso di avere maggior accuratezza nella definizione di "traffico normale per la rete".
In aggiunta a ciò, il nostro lavoro segue questa strategia in quanto utilizzare l'Autoencoder come rilevatore di anomalie rappresenta un approccio poco esplorato in letteratura, si è quindi voluto presentare una tecnica relativamente innovativa per la risoluzione del problema del rilevamento di anomalie nel traffico di rete in modo da permettere un ulteriore sviluppo futuro della metodologia.

Lavori correlati

La letteratura scientifica conta diversi articoli in merito all'uso di un Autoencoder per il rilevamento delle anomalie nel traffico LAN. In molti di questi lavori il ruolo dell'Autoencoder nel Sistema di Rilevamento delle Anomalie è quello di estrattore di features; queste vengono poi usate in differenti approcci di Machine Learning o Deep Learning per il rilevamento di anomalie. Altri lavori, invece, riservano un ruolo centrale all'Autoencoder ovvero quello di essere direttamente il rilevatore di anomalie. Un lavoro svolto nel 2018 in questa ottica è quello di Gonzalo Marin, Pedro Casas e Germán Capdehourat (*15*), in questo progetto i ricercatori hanno usato come input per l'Autoencoder il traffico LAN grezzo convertendo ogni byte in numeri naturali compresi tra 0 e 255, questo è un interessante approccio in quanto non è necessario nessun pre-processamento dei dati di input. Un approccio simile, ma con pre-processamento dei dati è stato proposto da Jinghui Chen, Saket Sathe, Charu Aggarwal, e Deepak Turaga nel 2017 (*16*).

Approccio proposto

Per realizzare l'IDS basato su Autoencoder è stata fatta leva sulla principale abilità dell'Autoencoder: ridurre la dimensionalità di un insieme di dati attraverso l'apprendimento di una funzione in grado di copiare l'input nell'output passando per una rappresentazione compresa dell'input.
Per utilizzare un Autoencoder come rilevatore di anomalie bisogna fare una forte ipotesi sui dati in esame: esiste una grande differenza tra dati normali ed anormali. Tale ipotesi è necessaria in quanto l'errore di ricostruzione sui dati viene utilizzato per stabilire una soglia di discriminazione tra dati normali ed anormali.
Al fine di rilevare anomalie nel traffico LAN l'Autoencoder va addestrato sul solo traffico benigno, così facendo esso è forzato ad apprendere una rappresentazione latente dell'input (composto da solo traffico benigno) a partire dalla quale è possibile ricostruire l'input stesso con un minimo errore di ricostruzione: in questo modo è possibile determinare una soglia (che rappresenta l'errore di ricostruzione massimo ottenuto in fase di addestramento) che permette di discriminare il traffico benigno (errore di ricostruzione minore della soglia) dal traffico anomalo (errore di ricostruzione maggiore della soglia). Si avrà così che più l'errore di ricostruzione sul traffico benigno è basso e quello sul traffico anomalo è alto migliori saranno le prestazioni della rete neurale.
Costruire un dataset bilanciato ed etichettato, composto sia da traffico normale che da tentativi di attacco per la fase di test, avrebbe richiesto un notevole sforzo in termini temporali: si è perciò deciso di utilizzare NSL-KDD come dataset di riferimento per eseguire la valutazione delle prestazioni del modello ottenuto.
Come detto precedentemente, il training set deve essere invece composto da solo traffico benigno, nel progetto si è perciò scelto di collezionare il traffico localmente ed etichettarlo interamente come normale per poi processarlo al fine di estrarre features omogenee a quelle presenti nel dataset di test NSL-KDD.

Analisi del dataset NSL-KDD

In accordo con la descrizione del dataset presente nel sito ufficiale (*17*), NSL-KDD è una collezione di flussi di traffico di rete creato per risolvere alcuni dei problemi inerenti alla precedente versione del dataset, il KDD'99 (Knowledge Discovery in Databases).
KDD'99 è un'elaborazione della collezione di dati del DARPA (Defense Advanced Research Projects Agency) realizzata nel 1998, questo dataset è composto dal traffico collezionato in 9 settimane simulando il traffico nella rete LAN dell'U.S. Air Force. KDD'99 è stato realizzato mediante l'estrazione di 41 features sia numeriche che categoriche le quali caratterizzano il traffico di rete raccolto da DARPA nel 1998.
Sebbene NSL-KDD soffra ancora di alcuni dei problemi discussi da Al Tobi (*18*) e Revathi (*19*) e potrebbe non essere un perfetto rappresentante delle reti reali esistenti, a causa della mancanza di set di dati pubblici per IDS basati su rete, NSL-KDD resta ancora un valido insieme di dati di riferimento per aiutare i ricercatori a confrontare diversi metodologie di rilevamento delle intrusioni.
Il dataset è reperibile in formato nativo pcap oppure già elaborato e diviso in training set e test set, nel nostro progetto solo il test set è stato utilizzato in fase di valutazione.
Il test set si compone di 22543 flussi di cui 9711 normali e i restanti 12832 anormali, questi ultimi raggruppano flussi di quattro diverse tipologie di attacco, come mostrato nella sottostante tabella:

Data set type	No of data samples					
	Records	Normal	DoS	Probe	U2R	R2L
NSL-KDD Train	125973	67343	45927	11656	52	995
	%	53.46	36.45	9.25	0.04	0.79
NSL-KDD Test	22543	9711	7458	2421	200	2754
	%	43.08	33.08	10.74	0.89	12.22

Figura 11. Numero di flussi divisi per tipologia nel NSL-KDD dataset.

Ogni flusso viene descritto da un insieme di 41 features sia numeriche che categoriche più un'ulteriore etichetta classe (normale o anormale), l'elenco è visionabile nella sottostante figura:

F#	Feature name	F#	Feature name	F#	Feature name
F1	Duration	F15	Su attempted	F29	Same srv rate
F2	Protocol type	F16	Num root	F30	Diff srv rate
F3	Service	F17	Num file creations	F31	Srv diff host rate
F4	Flag	F18	Num shells	F32	Dst host count
F5	Source bytes	F19	Num access files	F33	Dst host srv count
F6	Destination bytes	F20	Num outbound cmds	F34	Dst host same srv rate
F7	Land	F21	Is host login	F35	Dst host diff srv rate
F8	Wrong fragment	F22	Is guest login	F36	Dst host same src port rate
F9	Urgent	F23	Count	F37	Dst host srv diff host rate
F10	Hot	F24	Srv count	F38	Dst host serror rate
F11	Number failed logins	F25	Serror rate	F39	Dst host srv serror rate
F12	Logged in	F26	Srv serror rate	F40	Dst host rerror rate
F13	Num compromised	F27	Rerror rate	F41	Dst host srv rerror rate
F14	Root shell	F28	Srv rerror rate	F42	Class label

Figura 12. Lista delle features del dataset NSL-KDD

Il numero di flussi di NSL-KDD è in quantità ragionevole: questo vantaggio rende conveniente eseguire gli esperimenti sul set completo senza la necessità di selezionare casualmente una piccola porzione, di conseguenza i risultati della valutazione di diversi lavori di ricerca saranno coerenti e comparabili.
Nonostante NSL-KDD contenga traffico datato (ormai di oltre 20 anni fa) esso rappresenta, per le sue dimensioni, per la scarsità di dataset presenti per il rilevamento di anomalie in campo di rete internet e per il diffuso uso in diversi lavori di ricerca, il miglior insieme di dati di riferimento per questo tipo di analisi.

Raccolta dati

La prima fase del progetto è stata quella della raccolta dati.
Nella particolare applicazione i dati corrispondono al traffico di rete in una LAN, per raccogliere questo tipo di dati è stato utilizzato uno strumento software che prende il nome di analizzatore di traffico di rete, successivamente il traffico è stato processato grazie ad un altro software, il kdd'99 extractor reperibile su github (*20*). Questo software di estrazione ha consentito di ricostruire i flussi a partire dal traffico collezionato localmente nelle tre settimane precedenti ed ha estratto un sottoinsieme delle features del dataset NSL-KDD; in particolare le stesse del dataset NSL-KDD, escluse quelle comprese tra la posizione 10 e la posizione 22 della Figura 12. Anche NSL-KDD è stato processato nello stesso modo, così facendo training set e test set sono diventati omogenei nel numero di features.

Scelta dell'analizzatore di traffico di rete

Un analizzatore di traffico di rete (sniffer) è un software in grado di intercettare i pacchetti in transito nella LAN, esso inoltre mette a disposizione una serie di strumenti utili alla loro analisi.
Il primo problema affrontato nella progettazione dell'IDS è stato quello di capire quale tra i vari strumenti software fosse il più adatto per catturare, analizzare ed estrarre statistiche dal traffico di rete, a tal scopo sono stati presi in considerazione i seguenti sniffer:

- Tcpdump
- Tstat
- Wireshark / Tshark

Questi tool si basano tutti sulla libreria C/C++ libpcap. Libpcap è una libreria open source, essa fornisce un'interfaccia ad alto livello per interagire con i moduli del kernel che si occupano di catturare e filtrare i pacchetti, ovvero il BPF (Berkeley Packet Filter). Questa libreria è alla base della maggior parte degli strumenti di analisi di rete odierni.
Di seguito una lista dei software presi in considerazione con una breve descrizione delle funzionalità:

- Tcpdump: Tcpdump (*21*) è uno strumento da riga di comando per l'analisi dei pacchetti di rete che transitano sull'interfaccia di rete scelta. Nello specifico tcpdump permette di catturare e visualizzare il traffico di rete in tempo reale a schermo o salvando l'output su file. Questo software offre anche la possibilità di applicare filtri al flusso di rete per visualizzare solo alcuni di questi pacchetti, ciò è reso possibile dal supporto ai filtri BPF. BPF è un'architettura generica per realizzare filtri di pacchetti direttamente nel kernel del sistema operativo. BPF non interagisce direttamente con lo stack di rete del sistema operativo (TCP/IP), ma su copie dei pacchetti grezzi in transito.

- Tstat: Tstat (*22*) è un analizzatore di traffico che si presenta sia come strumento da riga di comando che come libreria scritta in C (libtstat). Tstat, infatti, è in grado di riconoscere diversi protocolli ed estrarne caratteristiche specifiche.
  Tstat può analizzare sia il traffico in tempo reale che il traffico precedentemente catturato e salvato in formato pcap (Packet CAPture). Esso è in grado di ricostruire flussi TCP e ricavare statistiche complessive sull'intera connessione.
- Tshar/Wireshark: Tshark (*23*) è un analizzatore di protocolli di rete da riga di comando e Wireshark (*24*) la sua versione GUI, entrambi sono open source. Come gli altri software anche tshark si basa sulla libreria libpcap, esso può così catturare traffico in tempo reale e interagire con pacchetti in formato pcap. Wireshark permette di collezionare il traffico di rete in formato pcap e tramite l'interfaccia grafica di visualizzare il contenuto di ogni singolo pacchetto di rete, Wireshark consente inoltre di visualizzare gli header dei pacchetti livello per livello in conformità con l'incapsulamento subito per essere inviati tramite Ethernet come descritto nel capitolo 1. In assenza di particolari settaggi si comporta come tcpdump e può eseguire catture filtrate grazie alla possibilità di interagire direttamente con il BPF. I filtri esprimibili in Wireshark sono molto complessi poiché possono essere composti da diversi livelli di filtraggio, questo fa di Wireshark uno strumento professionale usato anche in ambiente reale per l'analisi del traffico di rete. Wireshark permette inoltre la ricostruzione di particolari flussi di rete come la ricostruzione dei flussi HTTP (ovvero l'insieme di connessioni avvenute tra un server HTTP e un client particolare all'interno della LAN) o dei flussi TCP.

Nonostante tutti gli strumenti possono contare su ottime caratteristiche, Wireshark (in particolare la sua versione a riga di comando Tshark) è stato scelto come analizzatore di traffico di rete per il progetto: le sue funzionalità apprezzate da diversi analisti professionisti, la facilità di utilizzo e la massiccia presenza di citazioni riguardanti Wireshark all'interno dei forum della comunità virtuale ha consentito di trovare notevoli spunti e di poter accedere ad una vasta documentazione che ha permesso di comprendere dettagliatamente il funzionamento dello strumento stesso. Nel nostro lavoro Wireshark ci ha permesso di collezionare quotidianamente i pacchetti associati ad un normale utilizzo della rete, l'insieme di tutti i pacchetti di traffico benigno così raccolti sono stati riuniti in un unico file pcap. Una volta ottenuto il file pcap totale esso è stato processato attraverso il software kdd'99 extractor come descritto precedentemente a formare il training set.

Costruzione di training, validation e test set

Il training set, come riportato di sopra, è composto da solo traffico normale catturato localmente e processato al fine di estrarre features compatibili con quelle del dataset di test NSL-KDD. Dal dataset così creato è stata estrapolata una porzione del 20%, questa è andata a costituire il validation set.
Per quanto riguarda il test set esso è equivalente al test set del NSL-KDD privato delle features recanti gli indici compresi tra 10 e 22 in Figura 12, così facendo è stato possibile rendere omogenee le features di training, validation e test set.
Dal test set è stata estrapolata una porzione del 15%, questa porzione è stata privata dei flussi anomali ed è andata a costituire il dataset che ha poi permesso di calcolare la soglia di discriminazione tra gli errori di ricostruzione sul traffico normale e anormale, questo principio sarà descritto più dettagliatamente di seguito.

Pre-processamento dei dati

Un Autoencoder, essendo una FNN, accetta in input un vettore numerico di dimensione (intesa come numero di features) pari al numero di neuroni dell'input layer della rete, così facendo ogni feature viene inizialmente processata da un diverso neurone.
Il pre-processamento dei dati risulta essere un'operazione fondamentale quando si lavora su grandi quantità di dati, infatti, spesso le features non sono tutte numeriche come desiderato, esistono correlazioni tra le diverse features, esse presentano unità di misura diverse e possono avere valori fuori scala (outlier).
L'etichetta classe del test set è stata primariamente convertita in valori binari: 1 per i flussi normali, 0 per i flussi anomali. Nel training set l'etichetta classe è invece stata rimossa.
La successiva operazione eseguita è stata la conversione delle features categoriche, esistono due metodi principali:

- associare ad ogni categoria un valore numerico;
- creare una nuova feature per ogni categoria e assegnare il valore 1 alla sola feature che rappresenta la categoria dell'esempio in esame e 0 a tutte le altre. Questa seconda tecnica prende il nome di one-hot encoding ed è la strategia scelta per la conversione delle features categoriche nel nostro lavoro.

Applicando il one-hot encoding ai dataset, il numero di features è passato dalle 28 iniziali alle 41 finali.
Successivamente i dati sono stati normalizzati nel range [0,1], così facendo la distribuzione dei valori delle features viene mantenuta ma si può disporre di una scala univoca per tutte le features risolvendo così il problema delle diverse unità di misura. Per questa operazione ci si è avvalsi della formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

in cui:
x_{scaled} è il valore della feature i-esima normalizzata, x è il valore iniziale della feature i-esima, x_{max} è il massimo valore della feature i-esima, x_{min} il minimo valore della feature i-esima.

Costruzione della rete

La realizzazione di una rete neurale non può contare su regole fisse che portano a risultati certi, al contrario, la costruzione di una rete neurale può, per certi versi, essere definita un'arte: la scelta del numero di livelli della rete, il numero di neuroni per ogni livello, le funzioni di attivazione dei neuroni, l'ottimizzatore, l'inizializzatore dei pesi della rete e molti altri fattori di regolazione, detti iperparametri, devono essere scelti sulla base dei risultati che portano.
Nel progetto si è scelto di implementare un Autoencoder composto da 7 livelli con numero di neuroni speculari nella parte encoder e decoder, tutti fully-connected:

- input layer: 41 neuroni per corrispondenza con la dimensione dell'input, funzione di attivazione ReLu e inizializzatore dei pesi HeNormal
- hidden layers: 4 livelli, 2 per la parte encoder e 2 per quella decoder, essi contano rispettivamente 24 e 12 neuroni per la parte encoder, 12 e 24 per la parte decoder, funzione di attivazione ReLu e inizializzatore dei pesi HeNormal
- bottleneck: 6 neuroni con funzione di attivazione Relu e inizializzatore dei pesi HeNormal
- output layer: 41 neuroni, come per l'input layer, funzione di attivazione Sigmoide e inizializzatore dei pesi GlorotNormal

Nel progetto si è scelto di utilizzare due funzioni di attivazione:

- ReLu per tutti i livelli tranne l'ultimo, l'output è limitato nel range 0-1 così come erano stati normalizzati i dati di input;
- Sigmoid per l'ultimo livello, per la particolare applicazione, essendo l'input scalato nell'intervallo 0-1 e dovendo calcolare un errore tra dato in input e dato in output è conveniente usare una funzione che ha un output limitato tra 0 e 1.

L'inizializzazione dei pesi della rete è randomica per default: a causa di questa inizializzazione randomica le performance corrispondenti a valori iniziali diversi dei pesi possono essere molto differenti tra loro. Nel progetto si è perciò deciso di usare un'inizializzazione che si basa su una distribuzione normale dipendente dal numero di neuroni, essa può ricevere un seed in input in modo da rendere i risultati replicabili in iterazioni successive dell'algoritmo, questo tipo di inizializzazione prende il nome di HeNormal. Anche l'inizializzatore GlorotNormal, che si basa su principi simili, è stato utilizzato.

Addestramento

Una volta che il modello è stato costruito esso va compilato. L'operazione di compilazione è possibile grazie al metodo compile messo a disposizione da Keras. Il metodo compile accetta diversi parametri tra cui: ottimizzatore, funzione di perdita, metriche di valutazione e altri; tutti questi iperparametri rendono completo il modello che prima di questa fase era composto dalla solo topologia della rete.
Nel progetto sono stati utilizzati:

- optimizer: NAdam, è simile ad Adam (L'ottimizzazione di Adam è un metodo di discesa del gradiente stocastico basato sulla stima adattiva dei momenti del primo e del secondo ordine) ma con l'aggiunta del momento di Nesterov, esso è quindi un'estensione dell'algoritmo di discesa del gradiente
- loss: Mean Absolute Error (media dell'errore assoluto)

L'addestramento vero e proprio della rete, grazie all'utilizzo di Keras, è un processo semplice in quanto è sufficiente chiamare il metodo fit. Fit ha diversi parametri di input: vettore/matrice di input e di output, la grandezza del batch, il numero di epoche e altri che permettono una maggior personalizzazione della fase di addestramento.
Nel nostro lavoro al metodo solo un sottoinsieme di tutti i parametri disponibili è stato utilizzato:

- input data: il training set
- target data: il training set in quando lo scopo dell'Autoencoder è minimizzare l'errore di ricostruzione tra l'input e l'output (ovvero l'input ricostruito)
- batch_size: 128 flussi, ogni 128 flussi processati nella fase feedforward i pesi vengono aggiornati
- epochs: 100 epoche, il training set è stato presentato un totale di 100 volte alla rete
- validation_data: il validation set
- shuffle: False, questo parametro, settato True, permette di rimescolare i dati di training prima di ogni epoca, settato False permette di rendere i risultati riproducibili
- callbacks: EarlyStopping, funzione che permette di interrompere prematuramente l'addestramento quando non si ha una crescita/decrescita superiore al delta imposto per un numero di epoche consecutive fissato

Gli output del metodo fit possono essere salvati in un oggetto di tipo history messo a disposizione da Keras. Dall'oggetto history è poi possibile estrarre i dati utili a graficare la funzione di perdita in funzione del numero di epoche trascorse sia per il training che per il validation set. Il grafico citato permette di monitorare l'andamento dell'apprendimento del modello per fare in modo di evitare overfitting e underfitting.

L'addestramento dell'Autoencoder ha prodotto il seguente grafico:

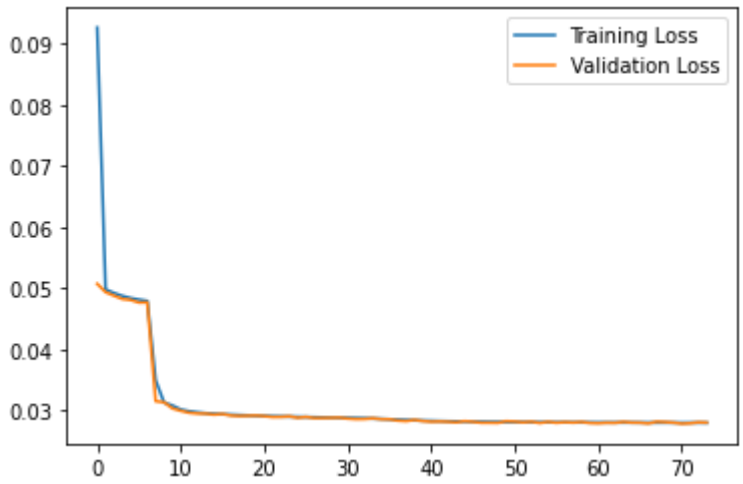

Figura 13. Funzione di perdita di training e validation set in funzione delle epoche.

si può evincere facilmente che il modello non ha riscontrato problemi di overfitting o underfitting nell'addestramento in quanto si ha la convergenza delle due funzioni di perdita sul training e sul validation set a valori prossimi a 0. Questo significa che l'errore di ricostruzione sui flussi normali è vicino allo 0 e che il modello è inoltre in grado di generalizzare su nuovo traffico.
In questo tipo di applicazione l'addestramento della rete non completa la fase di addestramento: è necessario calcolare la soglia di discriminazione tra l'errore di ricostruzione che contraddistingue il traffico normale, tendente a 0, e quello che contraddistingue il traffico anomalo, sperabilmente tendente a valori molto maggiori di 0.
Per calcolare la soglia si usa una piccola porzione di flussi normali estrapolati dal dataset NSL-KDD, che non è mai stato presentato alla rete, e si applica la formula:

$$threshold = np.mean(train\_loss) + np.std(train\_loss)$$

in cui: *train_loss* è l'insieme degli errori di ricostruzione tra i dati estrapolati da NSL-KDD e la loro versione ricostruita, la soglia (threshold) è calcolata come la media di questi valori a cui si aggiunge la deviazione standard sugli stessi.
I flussi processati dalla rete neurale vengono associati ad un errore di ricostruzione: il confronto di questo valore con la soglia trovata permette di determinare se il dato flusso è normale o anomalo, si ha normalità quando l'errore di ricostruzione è minore della soglia, anormalità quando è maggiore. Il risultato della fase di addestramento è quindi classificatore binario.

Risultati ottenuti sul dataset NSL-KDD

Valutando le performance ottenute sul dataset di test NSL-KDD è stato possibile andare a modificare gli iperparametri in modo da ottenere le prestazioni migliori.
Esiste anche un'altra tecnica per eseguire il tuning automatico degli iperparametri, un modulo di Keras chiamato keras tuner. Questo modulo permette di provare varie combinazioni di iperparametri secondo diversi metodi di combinazione e ritorna quindi la combinazione ideale, ovvero quella che ha portato a risultati migliori. Keras tuner presenta però anche degli svantaggi: per svolgere una ricerca completa sull'intero spazio di combinazioni di iperparametri esso richiede una grande quantità di tempo. Inoltre, spesso, i risultati ottenuti non corrispondono alle reali migliori performance ottenibili in quanto, al fine di ottenere tempi di ricerca accettabili, ogni configurazione di iperparametri viene testata un ridotto numero di volte; come spiegato in precedenza, però, le performance del modello dipendono molto dall'inizializzazione randomica dei pesi della rete, perciò alcune ottime combinazioni potrebbero venir scartate a causa degli scarsi risultati ottenuti che potrebbero però non essere rappresentativi per la data configurazione della rete.
Nel progetto si è perciò deciso di applicare un tuning manuale, andando, man mano, a modificare alcuni parametri del modello al fine di ottenere le migliori prestazioni.
Per valutare le performance di un classificatore binario esistono diverse metriche, una delle più importanti è la matrice di confusione: essa rappresenta una matrice quadrata composta da n righe e n colonne in cui n rappresenta il numero di classi presenti nel problema (nel nostro caso n = 2), le celle che si vengono a formare contengono il numero di esempi di classe predetta i-esima e classe reale j-esima, nel caso di un classificatore binario, essendoci solo 2 valori possibili e perciò 4 celle risultanti, i valori contenuti all'interno delle stesse assumono nomi particolari:

- True Positive (TP): esempi positivi correttamente classificati come appartenenti alla classe positiva;
- True Negative (TN): esempi negativi correttamente classificati come appartenenti alla classe negativa;
- False Positive (FP): esempi negativi erroneamente classificati come appartenenti alla classe positiva;
- False Negative (FN): esempi positivi erroneamente classificati come appartenenti alla classe negativa.

Altre metriche di valutazione degne di nota comprendono:

- accuracy: letteralmente accuratezza, rappresenta quante volte un'etichetta predetta dal modello e una etichetta reale combaciano, è calcolata come:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- precision: letteralmente precisione, rappresenta quanti esempi positivi sono stati correttamente etichettati rispetto a tutti gli esempi positivi presenti, è calcolata come:

$$Precision = \frac{TP}{TP + FP}$$

- recall: letteralmente richiamo, rappresenta quanti esempi positivi sono stati correttamente etichettati rispetto a tutti i potenziali positivi che potevano essere predetti, è calcolata come:

$$Recall = \frac{TP}{TP + FN}$$

- F1-score: è una metrica che coinvolge altre due metriche sopra citate secondo la formula:

$$F1 = 2 \bullet \frac{precision \bullet recall}{precision + recall}$$

Il lavoro svolto si basa sul confronto tra l'errore di ricostruzione ottenuto per un dato flusso e la soglia prestabilita in fase di training. Di seguito vengono presentati tre istogrammi:

- il primo istogramma rappresenta l'insieme degli errori di ricostruzione ottenuti sugli esempi normali del test set, graficamente:

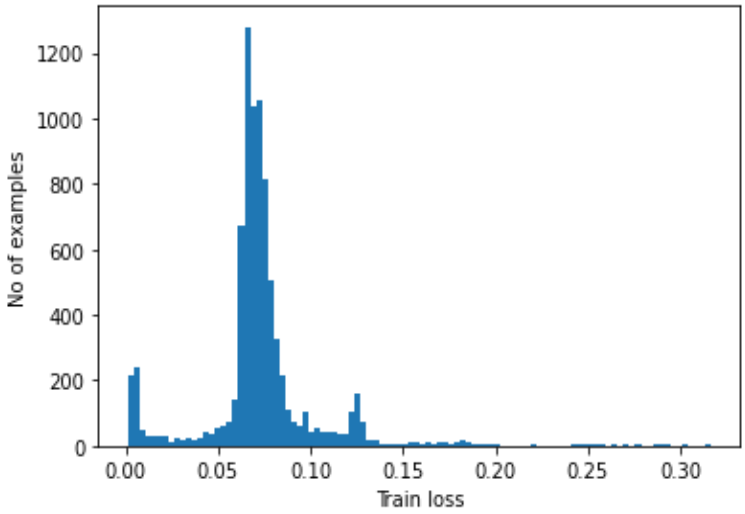

Figura 14. Istogramma, errore di ricostruzione sugli esempi normali del test set.

- il secondo istogramma rappresenta l'insieme degli errori di ricostruzione ottenuti sugli esempi anomali del test set, graficamente:

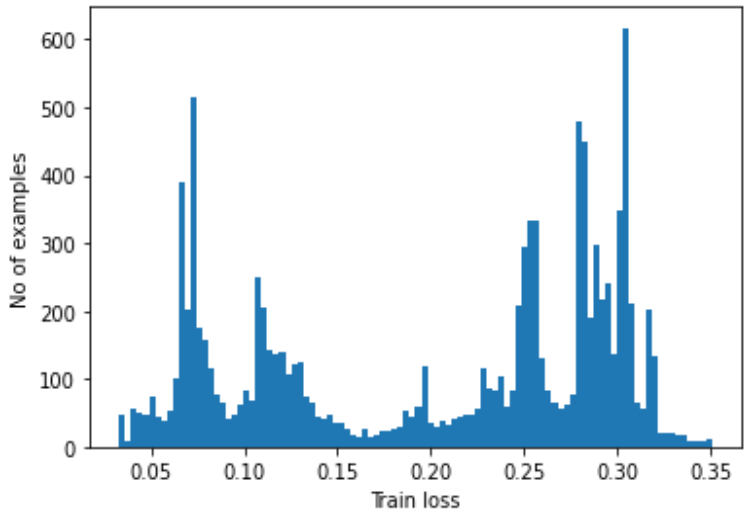

Figura 15. Istogramma, errore di ricostruzione sugli esempi anomali del test set.

- il terzo istogramma rappresenta la sovrapposizione dei due precedenti istogrammi in cui è inoltre evidenziata la soglia di discriminazione (linea rossa tratteggiata) ottenuta precedentemente, graficamente:

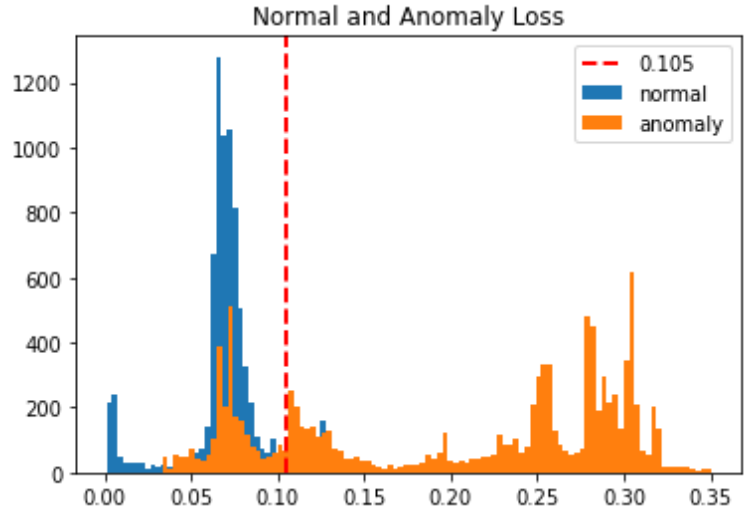

Figura 16. Sovrapposizione degli istogrammi con soglia di discriminazione in rosso

Da questo grafico finale è possibile notare come ci siano alcuni flussi anomali che si trovano a sinistra della soglia e alcuni flussi normali che invece si trovano alla destra: questi rappresentano le errate classificazioni del modello.
Si può altresì notare come la maggior parte dei flussi normali si trovino a sinistra della soglia così come per i flussi anomali che si trovano alla destra: questi rappresentano le corrette classificazioni del modello.

Le considerazioni qualitative fatte sono meglio riassunte nella confusion matrix ottenuta:

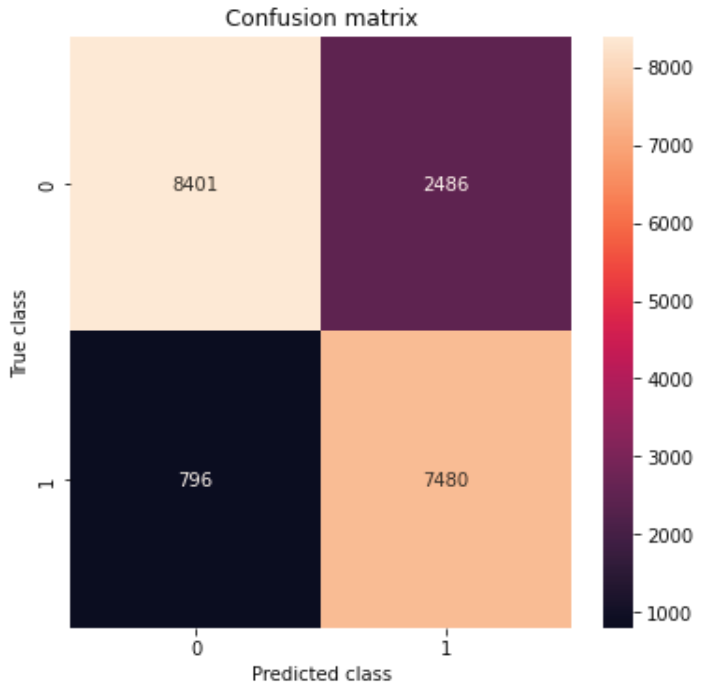

Figura 17. Matrice di confusione ottenuta sul test set.

in cui i flussi normali hanno etichetta pari a 1 mentre quelli anomali pari a 0, interpretando la matrice risulta che il sistema ottenuto classifica:

- 8401 correttamente come flussi anomali (TP);
- 7480 correttamente come flussi anomali (TN);
- 2486 erroneamente come flussi normali, in realtà essi sono anomali (FP);
- 796 erroneamente come flussi anomali, in realtà essi sono normali (FN);

Elaborando questi risultati secondo le metriche sopra citate si ottiene:

- Accuracy = 82,87%
- Precision = 75,05%
- Recall = 90,38%
- F1-score = 82,00%

Il problema maggiore è rappresentato dai 2486 flussi classificati erroneamente come normali, questi sono in realtà associati a tentativi di attacco e non vengono rilevati dal classificatore.
Il motivo dietro a questo alto tasso di falsi positivi (per positivo si intende un flusso normale etichettato come 1) si può ritrovare nel fatto che l'Autoencoder è stato addestrato su traffico catturato localmente, la fase di test della rete ha invece coinvolto il dataset NSL-KDD che per sua natura contiene traffico ormai datato: è possibile che il modello abbia appreso delle caratteristiche che non sono state ben generalizzate alla presentazione del traffico datato contenuto nel NSL-KDD. Inoltre, la natura degli attacchi è varia, l'Autoencoder impara a ricostruire i flussi normali con minor errore di ricostruzione possibile, non è però assicurato che anche qualche tentativo di attacco venga ricostruito molto bene dall'Autoencoder e che perciò venga mal interpretato come traffico normale.

Le prestazioni finali risultano essere comunque accettabili, combinando questo con altri approcci di rilevamento delle intrusioni si può facilmente ridurre il numero di queste errate classificazioni sotto una certa soglia di usabilità del sistema.

Sviluppi futuri

Il lavoro proposto rappresenta una prova di concetto: i risultati ottenuti, pur avendo usato un dataset di test contenente traffico datato, fanno sperare che l'approccio sia valido in ambiente reale. Per testare questa ulteriore ipotesi è possibile creare un dataset composto da traffico normale e da tentativi di attacco, svolti in ambiente predisposto, al fine di testare le prestazioni del sistema su traffico conforme all'uso odierno della rete.
Per la realizzazione di un prodotto commerciale si può pensare di introdurre diversi metodi di rilevazione, in un approccio in parallelo, in cui il singolo flusso viene processato da diversi classificatori: la classe che riceve il maggior numero di voti è la classe finale predetta.
Uno sviluppo ulteriore del sistema potrebbe essere quello di aver maggior informazioni sui flussi che vengono classificati come anomali creando un rilevatore ibrido composto da un approccio basato su IA in serie ad uno basato su firma in modo tale da poter specificare l'effettiva causa dell'anomalia inizialmente rilevata per mezzo del confronto con firme di malware noti. Inoltre, un sistema di protezione in ambiente reale dovrebbe permettere di bloccare in tempo utile il sistema sul quale viene rilevata un'anomalia così da permettere una prematura interruzione dell'attività malevola. Ogni minuto, ad esempio, si potrebbe processare i dati raccolti dall'analizzatore di traffico in modo da poterli far processare dalla rete neurale già addestrata e confrontare l'errore di ricostruzione con la soglia precedentemente fissata.

Conclusioni

Questo lavoro di tesi si è svolto nell'ottica di portare una prova di concetto volta a dimostrare come l'Intelligenza Artificiale possa essere la base per la costruzione di strumenti utili nel contrasto di attacchi informatici. Questi ultimi sono in forte crescita negli ultimi due decenni ed è perciò importante aggiungere ulteriori livelli di sicurezza ai nostri dispositivi, i quali, sempre più spesso, contengono informazioni personali e sensibili. A tal scopo si è scelto di realizzare un Sistema di Rilevamento delle Intrusioni (IDS) nella rete locale basato su una topologia di rete neurale che prende il nome di Autoencoder.

Implementare algoritmi di Intelligenza Artificiale è spesso un processo lungo e difficoltoso in quanto i dati utilizzati presentano un elevato numero di dimensioni. Trovare la giusta combinazione di parametri che portano il modello ad avere performance ottimali consiste in gran parte nel trovare la giusta modalità di estrapolare informazioni rilevanti dai dati.

La soluzione adottata nel progetto non ha surclassato lo stato dell'arte in campo di rilevamento di anomalie nel traffico di rete ma i risultati sono stati incoraggianti: su un totale di 19163 flussi, 15881 vengono correttamente classificati, portando così l'accuratezza del modello a 82,87%. La metrica di precisione ha invece raggiunto il 75,05%. Questa metrica rappresenta quanti flussi di traffico normale sono stati correttamente classificati rispetto a tutti i positivi predetti; risulta evidente che un 24,95% dei flussi classificati come normali appartenevano però alla classe anomala. Questo problema può essere risolto, come descritto nel paragrafo precedente, creando un sistema formato da diversi classificatori in parallelo e fondendo i loro output a livello di decisione. Così facendo ogni classificatore può processare diverse features, essere addestrato su porzioni diverse del dataset, usare algoritmi di classificazione diversi e perciò valutare lo stesso problema secondo diverse prospettive. In questo modo si può estrapolare un più ampio insieme di informazioni dai dati e portare quindi ad un miglioramento delle prestazioni totali del sistema.

Al di là dei possibili miglioramenti, l'obiettivo del lavoro è stato raggiunto in quanto si è dimostrato come l'Intelligenza Artificiale, in particolare il Deep Learning, possa essere, in combinazione con gli altri strati di protezione già esistenti, un valido approccio per creare ulteriori strati protettivi in grado di rendere i dispositivi connessi più sicuri rispetto al passato.

Ringraziamenti

Ci tengo a ringraziare la mia famiglia, la mia fidanzata, gli amici, i colleghi universitari e tutte le persone che in diversi modi hanno contribuito a rendere possibile questo momento.
Un ringraziamento speciale a Pietro e Andrea che mi hanno ospitato nella loro azienda in questo periodo di tirocinio e hanno reso così possibile la realizzazione del progetto. Infine, ci tengo a ringraziare il Professor Nanni che mi ha seguito in tutto il percorso rispondendo prontamente ad ogni richiesta con preziosi spunti.
Senza di voi tutto questo non sarebbe mai stato possibile.
Grazie a tutti di cuore.

Bibliografia

1. National Institute of Standards and Technology. Available: https://www.nist.gov/
2. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016).
3. H. Schulz, S. Behnke, Deep learning. *KI - Künstl. Intell.* **26**, 357–363 (2012).
4. NumPy, (available at https://numpy.org/).
5. pandas, (available at https://pandas.pydata.org/).
6. G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, A. Mueller, Scikit-learn. *GetMob. Mob. Comput. Commun.* **19**, 29–33 (2015).
7. Matplotlib — visualization with python, (available at https://matplotlib.org/).
8. G. Blokdyk, *Tensorflow: A Complete Guide* (5starcooks, 2018).
9. Keras: the Python deep learning API, (available at https://keras.io/).
10. G. Pu, L. Wang, J. Shen, F. Dong, A hybrid unsupervised clustering-based anomaly detection method. *Tsinghua Sci. Technol.* **26**, 146–153 (2021).
11. P. Sangkatsanee, N. Wattanapongsakorn, C. Charnsripinyo, Practical real-time intrusion detection using machine learning approaches. *Comput. Commun.* **34**, 2227–2235 (2011).
12. A. Vikram, Mohana, Anomaly detection in Network Traffic Using Unsupervised Machine learning Approach. *2020 5th International Conference on Communication and Electronics Systems (ICCES)* (2020), , doi:10.1109/icces48766.2020.9137987.
13. T. Truong-Huu, N. Dheenadhayalan, P. P. Kundu, S. P. Kadiyala, in *1st Security and Privacy on Artificial Intelligent Workshop (SPAI'20)* (unknown, 2020; https://www.researchgate.net/publication/343039092_An_Empirical_Study_on_Unsupervised_Network_Anomaly_Detection_using_Generative_Adversarial_Networks).
14. D. Kwon, K. Natarajan, S. C. Suh, H. Kim, J. Kim, An Empirical Study on Network Anomaly Detection Using Convolutional Neural Networks. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (2018), , doi:10.1109/icdcs.2018.00178.
15. G. Marín, P. Casas, G. Capdehourat, in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos* (Association for Computing Machinery, New York, NY, USA, 2018), *SIGCOMM '18*, pp. 75–77.
16. J. Chen, S. Sathe, C. Aggarwal, D. Turaga, in *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)* (Society for Industrial and Applied Mathematics, 2017), *Proceedings*, pp. 90–98.
17. Nsl-kdd, (available at https://www.unb.ca/cic/datasets/nsl.html).
18. A. M. Al Tobi, I. Duncan, KDD 1999 generation faults: a review and analysis. *Journal of Cyber Security Technology*. **2**, 164–200 (2018).
19. A Detailed Analysis on NSL-KDD Dataset Using ... - CiteSeerX.

https://citeseerx.ist.psu.edu › viewdoc › downloadhttps://citeseerx.ist.psu.edu › viewdoc › download (available at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.6760&rep=rep1&type=pdf).

20. AI-IDS/kdd99_feature_extractor: Utility for extraction ... - GitHub. *https://github.com › AI-IDS › kd...https://github.com › AI-IDS › kd...* (available at https://github.com/AI-IDS/kdd99_feature_extractor).

21. Home, (available at https://www.tcpdump.org/).

22. D. Rossi, Tstat - TCP STatistic and Analysis Tool, (available at http://tstat.polito.it/docs.php).

23. Tshark(1), (available at https://www.wireshark.org/docs/man-pages/tshark.html).

24. Wireshark · Documentation, (available at https://www.wireshark.org/docs/).