UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea in Informatica

Reti Antagoniste Generative in ambito Cybersecurity:

Un approccio pratico alla classificazione del traffico di rete

Relatore
Prof. Francesco Palmieri

Candidato
Lucio Grimaldi

Mat:0512103243

Anno Accademico 2017/2018

INDICE

INTRODUZIONE

Nella battaglia quotidiana fra aziende e cybercriminalità, le prime possono avvalersi oggi di nuovi strumenti di protezione dalle minacce basati sull'utilizzo dell'intelligenza artificiale e, in particolare, di una sua sottocategoria: il **Machine Learning**, il cui utilizzo non è una novità nell'ambito della Cybersecurity. Infatti, vengono già utilizzati diversi algoritmi e framework per analizzare i comportamenti anomali tramite l'osservazione e la catalogazione di possibili virus, malware, attacchi di rete e intrusioni. Ciò consente di studiare in che modo queste minacce funzionano per costruire sistemi che aiutano le imprese che cercano di evitare i possibili incidenti che potrebbero avvenire quando collegano le proprie macchine alla rete.

Con il passare degli anni la necessità di essere connessi in rete è aumentata repentinamente. Basti pensare al processo evolutivo dei servizi della Pubblica Amministrazione come la posta elettronica certificata o sportelli virtuali tramite i quali, ora, i cittadini possono avere accesso ad ulteriori informazioni e servizi in modo vantaggioso. In questo modo la comunicazione pubblica sta migrando verso una comunicazione on-line, il che permette a malintenzionati di mettere in atto attacchi agli utenti, aumentando il rischio di furti di dati sia dei privati che delle pubbliche amministrazioni. Un esempio di attacco molto diffuso negli ultimi tempi è quello del "phishing", col quale un malintenzionato impersona un ente affidabile con lo scopo di ottenere dalla vittima dati riservati, utilizzati per l'accesso ad un determinato servizio.

Gli enti esperti nell'ambito della sicurezza informatica periodicamente condividono nuove statistiche riguardo il traffico dati dalle quali emerge che con il passare degli anni crescono le falle di sicurezza trovate e di conseguenza anche il numero di attacchi che le sfruttano. Nel primo trimestre del 2017, secondo i report del Kaspersky Security Network, il software anti-malware ha bloccato 479.528.279 attacchi condotti attraverso siti Internet compromessi e questi numeri sono in costante crescita. Per dare dunque un aiuto alla difesa dei nostri dati sensibili e dei nostri sistemi, entra in gioco il Machine Learning ed in particolare una sua specializzazione, il Deep Learning. Quest'ultimo offre la possibilità di emulare il comportamento del cervello umano tramite schemi di riconoscimento più "profondi", il che porta ad un incremento della possibilità di identificare le minacce sconosciute. Quindi, sfruttando il Deep Learning, è possibile costruire dei frameworks affiancabili alle già esistenti soluzioni per apportare un contributo importante alle stesse. Uno dei punti chiave per la costruzione del modello è

quello di disporre di un ampio insieme di campioni che il modello può analizzare e approssimare con lo scopo di riuscire a identificare eventuali schemi anomali. La collezione dei campioni è chiamato **dataset**. Nella classificazione del traffico di rete il problema principale è quello di distinguere il traffico "normale" dal traffico anomalo. Nello specifico, bisogna costruire un framework che usa il Deep Learning per distinguere il traffico in due categorie o **classi**. In rete è possibile trovare diversi dataset gratuiti messi a disposizione da organizzazioni no-profit, Università e laboratori di ricerca che è possibile utilizzare in diversi ambiti. Certamente non mancano dataset contenenti informazioni legate al traffico di rete, divise in opportune classi. Solitamente i dataset disponibili contengono alcune imperfezioni che devono essere corrette. Tale problematica si presenta soprattutto nei dataset per l'analisi del traffico di rete, i quali, essendo costruiti in laboratorio, risultano essere una approssimazione dei dati che realmente viaggiano sulla rete per diverse questioni. Per raggiungere il nostro scopo, ovvero quello della classificazione del traffico di rete, intesa come anomaly detection, è stata utilizzata una particolare classe di algoritmi di Deep Learning, le Reti Generative Antagoniste o **GAN**, dall'inglese Generative Adversarial Networks. Tali reti ci permettono sia di correggere alcune imperfezioni del dataset, sia di raggiungere una precisione maggiore nella classificazione del traffico di rete.

CAPITOLO 1

LE RETI NEURALI

1.1 Introduzione

Una rete neurale artificiale è un modello matematico, composto di "neuroni" artificiali, basato sulle reti neurali biologiche, utilizzato per tentare di risolvere problemi ingegneristici di intelligenza artificiale. [1] Essa può essere realizzata sia da programmi software che da hardware dedicato. Nella maggior parte dei casi una rete neurale artificiale è un sistema adattivo che cambia la sua struttura basata su informazioni esterne o interne che scorrono attraverso la rete durante la fase di apprendimento. In termini pratici le reti neurali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione. Esse possono essere utilizzate per simulare relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riescono a rappresentare. Una rete neurale artificiale riceve segnali esterni su uno strato di nodi (unità di elaborazione) d'ingresso, ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi. In linea di massima, le reti neurali sono formate da tre strati (che però possono coinvolgere migliaia di neuroni e decine di migliaia di connessioni):

- lo **strato degli ingressi** (I – Input): è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete;
- lo **strato nascosto** (H – hidden): è quello che ha in carica il processo di elaborazione vero e proprio (e può anche essere strutturato con più colonne-livelli di neuroni);
- lo **strato di uscita** (O – Output): qui vengono raccolti i risultati dell'elaborazione dello strato H e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

1.2 Le Reti Neurali Biologiche

Il prototipo delle reti neurali artificiali sono quelle biologiche. Le reti neurali biologiche sono la sede della nostra capacità di comprendere l'ambiente e i suoi mutamenti, e di fornire quindi risposte adattive calibrate sulle esigenze che si presentano. [2] Sono costituite da insiemi di cellule nervose fittamente interconnesse fra loro. Al cui interno troviamo:

- i **somi neuronali**, ossia i corpi dei neuroni. Ricevono e processano le informazioni;

- i **neurotrasmettitori**, responsabili della modulazione degli impulsi nervosi;
- gli **assoni o neuriti**: la via di comunicazione in uscita da un neurone. Ogni cellula nervosa ne possiede di norma soltanto uno;
- i **dendriti**: la principale via di comunicazione in ingresso; sono multipli per ogni neurone, formando il cosiddetto albero dendritico;
- le **sinapsi**, o giunzioni sinaptiche: i siti funzionali ad alta specializzazione nei quali avviene il passaggio delle informazioni fra neuroni. Ognuno di questi ne possiede migliaia. A seconda dell'azione esercitata dai neurotrasmettitori, le sinapsi hanno una funzione eccitatoria, facilitando **la trasmissione dell'impulso nervoso**, oppure inibitoria, tendente a smorzarlo.

Un singolo neurone può ricevere simultaneamente segnali da diverse sinapsi. Una sua capacità intrinseca è quella di misurare il potenziale elettrico di tali segnali in modo globale, stabilendo quindi se è stata raggiunta la soglia di attivazione per generare a sua volta un impulso nervoso. Tale proprietà è implementata anche nelle reti artificiali. La configurazione sinaptica all'interno di ogni rete neurale biologica è dinamica. Il numero di sinapsi può incrementare o diminuire a seconda degli stimoli che riceve la rete. Più sono numerosi, maggiori connessioni sinaptiche vengono create, e viceversa. In questo modo, la risposta adattiva fornita dai circuiti neurali è più calibrata, e anche questa è una peculiarità implementata nelle reti neurali artificiali.

1.3 Storia ed evoluzione delle Reti Neurali

Il primo modello teorico di un rudimentale neurone artificiale vede la luce nel 1943. A proporlo è una coppia di scienziati, **McCulloch** e **Pitts**. I due descrivono un apparato in grado di ricevere n dati binari in ingresso in ognuno dei suoi elementi, a cui segue un singolo dato in uscita per ciascuno.

Tale macchina è in grado di lavorare su funzioni booleane elementari, e solo su quelle.

Figura 1.1 Walter Pitts

Nel 1949 D. O. **Hebb** ipotizza la possibilità di istruire le macchine con un apprendimento che emuli quello alla base dell'intelligenza umana.

Nel 1958 viene proposta da **Rosenblatt** la prima rete neurale: **Perceptron**. Perceptron di Rosenblatt possiede uno strato di nodi (neuroni artificiali) di input e un nodo di output. I pesi sinaptici (un peso indica la forza di una connessione fra due nodi) sono dinamici, permettendo alla macchina di apprendere, in un modo sommariamente simile, anche se molto più elementare, a quello delle reti neurali biologiche. Il modello è **feedforward**: gli impulsi si propagano in un'unica direzione, in avanti. Il suo campo di applicazione è molto limitato. Consiste nel riconoscere forme, classificandole in due gruppi separati, e nel calcolare semplici funzioni.

Il passo successivo è il **Perceptron multistrato** (MLP).

Al suo interno, fra i nodi di input e quello di output si trova uno strato hidden, dove avviene l'elaborazione delle informazioni provenienti dallo strato di input, che poi vengono inviate al nodo di output. Questa è una rete feedforward non lineare: le connessioni in ingresso e in uscita da ogni singolo nodo sono multiple. A merito di tale architettura, il MLP può computare qualsiasi funzione.

Werbos, nel 1974, descrive nella sua tesi di dottorato come impostare l'apprendimento di un **MLP**.

Il suo lavoro viene poi ripreso e perfezionato da Rumelhart, Hinton e Williams, che nel 1986 elaborano il celebre **Error Back-Propagation**, con il quale entriamo nel presente, essendo tuttora utilizzato. L'EBP permette di perfezionare in stadi successivi l'apprendimento automatico di una rete neurale. Si implementa modificando i pesi delle connessioni fra nodi che non producono l'output ottimale, finché non si ottiene quest'ultimo.

Non meno importante, in tal senso, risulta il precedente lavoro di Hebb, relativo alle connessioni reciproche fra neuroni. Hebb postula che il loro peso deve incrementare unicamente in caso di convergenza fra i due valori pre e post-sinaptico, da cui deriva la

"Legge di Hebb":

"Se un neurone A è abbastanza vicino ad un neurone B da contribuire ripetutamente o in maniera duratura alla sua eccitazione, allora ha luogo in entrambi i neuroni un processo di crescita o di cambiamento metabolico tale per cui l'efficacia di A nell'eccitare B viene accresciuta" [3]

Con i MLP e l'EPB il machine learning trova ora alcuni campi di applicazione pratica entrando così negli anni '90.

Intanto vengono implementate anche **reti neurali con architetture feedback** (reti **Hopfield**, dal nome del fisico che nel 1982 ne propone il modello). In tali architetture, le informazioni fra nodi viaggiano in qualunque direzione: in avanti, all'indietro e fra nodi di uno stesso strato. Il campo delle applicazioni si amplia ulteriormente.

Sempre in questi anni viene progettata la rete neurale proposta da **Elman** nel 1990. Anche questo è un modello di rete ricorrente (bidirezionale), ma con la variante che alla classica struttura MLP viene aggiunto un gruppo di nodi aventi lo scopo di conservare le informazioni della precedente configurazione di valori della rete. Grazie a tale modifica, la rete di Elman si rivela vantaggiosa nel calcolo delle sequenze temporali.

Nel 1982, invece, **Kohonen** progetta un tipo di rete neurale dall'architettura sia feedforward che feedback. Sua caratteristica peculiare è la capacità di modificare la configurazione (mappa) dei propri nodi in base al peso che assumono man mano che vengono forniti gli input. I nodi con pesi simili si avvicinano, quelli con pesi molto

diversi si allontanano. La rete di Kohonen è anche conosciuta come rete **Self-Organizing Maps** (SOM).

L'evoluzione delle reti neurali prosegue con la tecnologia adottata da **IBM**, tramite la quale è stata sviluppata una rete neurale basata su materiali a cambiamento di fase. L'hardware di tale architettura utilizza leghe come il germanio, tellururo di ammonio, che presentano l'interessante proprietà di assumere due diversi stati: cristallino (configurazione spaziale omogenea) e amorfo (configurazione spaziale poco definita). Nei neuroni a cambiamento di fase, gli impulsi elettrici sono in grado di provocare una cristallizzazione del materiale, innescandone infine il **Firing** (attivazione). Ebbene, questo è analogo a quello che avviene nelle cellule nervose. Per ora il neurone artificiale di IBM permette di scrivervi informazioni ma non le memorizza stabilmente, però è certo che il suo funzionamento è quanto di più simile esista **all'emulazione di un cervello umano**.

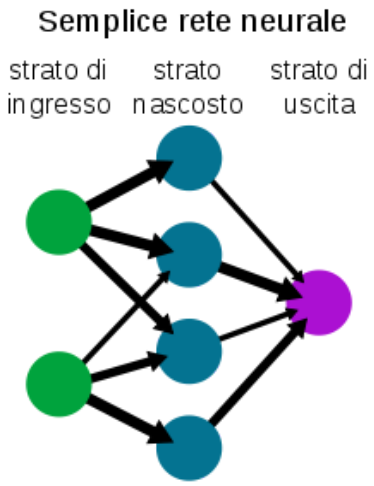

Figura 1.2 Rete Neurale

1.4 Algoritmi di Apprendimento

Nelle reti artificiali ovviamente il processo di apprendimento automatico è semplificato rispetto a quello delle reti biologiche. Non esistono analoghi dei neurotrasmettitori, ma lo schema di funzionamento è simile. I **nodi** ricevono dati in input, li processano e sono in grado di inviare le informazioni ad altri neuroni. Attraverso cicli più o meno numerosi di input-elaborazione-output, in cui gli input presentano variabili differenti, diventano in grado di generalizzare e fornire output corretti associati ad input non

facenti parte del training set. Affinché questo processo risulti performante è necessario "**addestrare**" le reti neurali, ossia fare in modo che apprendano come comportarsi nel momento in cui andrà risolto un problema ingegneristico, come per esempio il riconoscimento di un essere umano dall'analisi delle immagini (attraverso per esempio la tecnologia del riconoscimento facciale). Gli **algoritmi di apprendimento** utilizzati per istruire le reti neurali sono divisi in 3 categorie e sono:

- **Apprendimento supervisionato** (supervised learning)

  Nell'apprendimento supervisionato si fornisce alla rete un insieme di input ai quali corrispondono output noti (training set). Analizzandoli, la rete apprende il nesso che li unisce. In tal modo impara a generalizzare, ossia a calcolare nuove associazioni corrette input-output processando input esterni al training set. Man mano che la macchina elabora output, si procede a correggerla per migliorarne le risposte variando i pesi. Ovviamente, aumentano i pesi che determinano gli output corretti e diminuiscono quelli che generano valori non validi.

  Il meccanismo di apprendimento supervisionato impiega quindi l'Error Back-Propagation, ma è molto importante l'esperienza dell'operatore che istruisce la rete. Il motivo risiede nel non facile compito di trovare un rapporto adeguato fra le dimensioni del training set, quelle della rete e l'abilità a generalizzare che si tenta di ottenere. Un numero eccessivo di parametri in ingresso e una troppo potente capacità di elaborazione, paradossalmente, rendono difficile alla rete neurale imparare a generalizzare, perché gli input esterni al training set vengono valutati dalla rete come troppo dissimili ai sofisticati e dettagliati modelli che conosce. D'altro canto, un training set con variabili scarse porta per la via opposta alla stessa conclusione: la rete, in questo caso, non ha sufficienti parametri per apprendere a generalizzare. Il giusto compromesso, insomma, è un compito che necessita di molta preparazione ed esperienza.

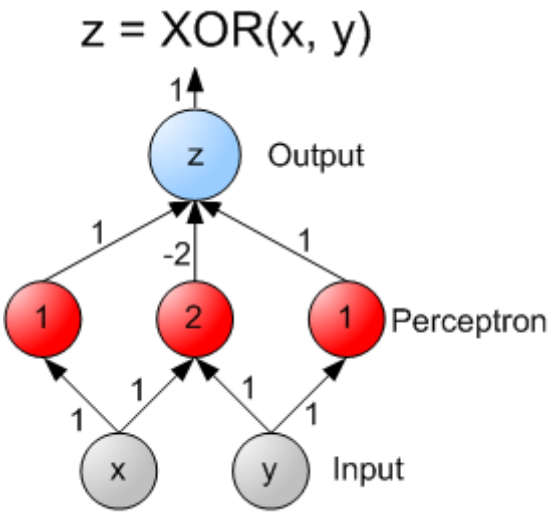

Figura 1.3 Apprendimento

- **Apprendimento non supervisionato** (unsupervised learning)

  In una rete neurale ad apprendimento non supervisionato, la medesima riceve solo un insieme di variabili di input. Analizzandole, la rete deve creare dei **cluster** rappresentativi per categorizzarle. Anche in questo caso i valori dei pesi sono dinamici, ma sono i nodi stessi a modificarli.

- **Apprendimento per rinforzo** (reinforcement learning)

  Nelle reti neurali che apprendono mediante **l'algoritmo di rinforzo**, non esistono né associazioni input-output di esempi, né un aggiustamento esplicito degli output da ottimizzare. I circuiti neurali imparano esclusivamente dall'interazione con l'ambiente. Su di esso, eseguono una serie di azioni. Dato un risultato da ottenere, è considerato rinforzo l'azione che avvicina al risultato; viceversa, la rete apprende a eliminare le azioni negative, ossia foriere di errore. Detto in altri termini, un algoritmo di apprendimento per rinforzo mira a indirizzare la rete neurale verso il risultato sperato con una politica di **incentivi** (azioni positive) e disincentivi (azioni negative). Usando tale algoritmo, una macchina impara a trovare soluzioni che non è esagerato definire creative. Una rete neurale così implementata, per esempio, è stata utilizzata per giocare ad Arcade Breakout. Risultato: dopo sole 4 ore di continuo miglioramento, i circuiti hanno individuato una strategia di gioco mai ideata da un essere umano in questo videogame.

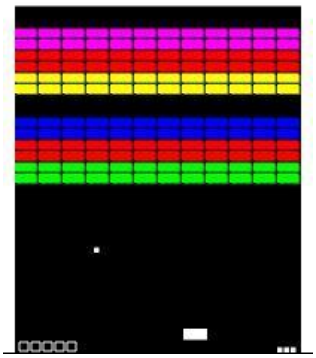

Figura 1.4 Arcade Breakout

1.5 Vantaggi delle Reti Neurali

L'utilizzo delle varie tipologie di reti neurali nasce dagli importanti vantaggi che presentano:

- elevato **parallelismo**, grazie al quale possono processare in tempi relativamente rapidi grandi moli di dati;
- **tolleranza ai guasti**, anche questo grazie all'architettura parallela;
- **tolleranza al rumore**, ossia la capacità di operare, in molti casi, in modo corretto nonostante input imprecisi o incompleti;
- **evoluzione adattiva**: una rete neurale ben implementata è in grado di auto aggiornarsi in presenza di modifiche ambientali.

1.6 Applicazioni

Le reti neurali vengono solitamente usate in contesti dove i dati possono essere parzialmente errati oppure dove non esistano modelli analitici in grado di affrontare il problema. Un loro tipico utilizzo è nei software di **OCR**, nei sistemi di riconoscimento facciale e più in generale nei sistemi che si occupano di trattare dati soggetti a errori o rumore. Esse sono anche uno degli strumenti maggiormente utilizzati nelle analisi di **Data mining**. Le reti neurali vengono anche utilizzate come mezzo per previsioni nell'analisi finanziaria o meteorologica. Negli ultimi anni è aumentata notevolmente la loro importanza anche nel campo della **bioinformatica** nel quale vengono utilizzate per la ricerca di pattern funzionali e/o strutturali in proteine e acidi nucleici. Mostrando opportunamente una lunga serie di input (fase di training o apprendimento), la rete è in grado di fornire l'output più probabile. Negli ultimi anni inoltre sono in corso studi per il loro utilizzo nella previsione degli **attacchi Epilettici** (Analisi dei Dati provenienti dall'

EEG). Recenti studi hanno dimostrato buone potenzialità delle reti neurali in sismologia per la localizzazione di epicentri di terremoti e predizione della loro intensità.

1.7 Limiti

Le **reti neurali artificiali** hanno comunque dei limiti, ed è difficile prevedere se col tempo potranno essere eliminati o attenuati. I più importanti sono:

- **funzionamento a black box**: Un handicap rimarchevole delle reti neurali artificiali è il fatto che la loro computazione non è analizzabile in modo completo. Con questo si intende dire che sono in grado di fornire output corretti, o sufficientemente corretti, ma non permettono di esaminare i singoli stadi di elaborazione che li determinano;
- non è possibile avere la certezza a priori che un problema sarà risolto;
- **gli output forniti spesso non rappresentano la soluzione perfetta**, anche se in molti casi questo non è necessario;
- **il periodo di learning è più o meno lungo**: le iterazioni necessarie dipendono da fattori quali numero e complessità delle variabili di input, algoritmo utilizzato, etc. In realtà, in tale ambito sono stati fatti importanti progressi, ed è ragionevole ipotizzare che in futuro il periodo di learning potrà ulteriormente ridursi;
- **le reti neurali non sono idonee a risolvere determinate categorie di problemi**. Un esempio è un tipo di input costituito da un numero elevato di variabili categoriche.

CAPITOLO 2
GAN

2.1 Introduzione

Le reti antagoniste generative (o GANs, dall'inglese Generative Adversarial Networks) sono una classe di algoritmi di intelligenza artificiale usati nell'apprendimento automatico delle reti neurali. Le GANs, proposte per la prima volta nel 2014 [4] da Ian Goodfellow, implementano una coppia di reti neurali che si sfidano l'una con l'altra in un gioco a somma zero [5]. Una analogia comune per descrivere le due reti è quella di pensare ad una rete come un falsario di banconote, che sfida il poliziotto esperto di banconote, ovvero la rete antagonista. Il falsario, che per convenzione chiamiamo generatore (G), produce delle banconote false con lo scopo di renderle indistinguibili da quelle originali mentre il poliziotto, che per convenzione chiamiamo discriminatore (D), invece ha il compito di distinguere quali banconote sono false e quali sono originali. Entrambe le reti, G e D, sono addestrate simultaneamente facendole sfidare tra loro, in una sorta di competizione, che permette al generatore di costruire dei campioni sempre più accurati, ovvero che si avvicinano sempre più agli esempi originali, e al discriminatore di distinguere di volta in volta i campioni generati dai campioni originali, con una accuratezza sempre maggiore.

Il generatore, per costruire dei campioni, non usa in alcun modo i dati presenti nel dataset, ma utilizza come input un vettore di numeri casuali. L'unico modo per apprendere come generare dei campioni molto simili a quelli originali è quello di interagire con il discriminatore, che a sua volta apprende utilizzando direttamente i campioni originali oltre che quelli provenienti dal generatore. Per distinguere un campione generato da uno originale contenuto nel dataset, il discriminatore utilizza una strategia molto semplice: ricorre ad una ground-truth che indica se un dato campione proviene dal generatore o dal dataset, la cui costruzione è immediata. La ground-truth è quindi responsabile di segnalare eventuali errori al discriminatore che provvede a correggere la sua strategia di riconoscimento. La ground-truth utilizzata per segnalare gli errori al discriminatore, viene utilizzata anche per segnalare gli errori al generatore attraverso il discriminatore stesso.

Il processo di addestramento delle due reti termina quando sia il generatore che il discriminatore raggiungono il livello ottimale di accuratezza ma facendo attenzione ad evitare situazioni in cui una delle due reti prevale sull'altra. In altre parole, si cerca di far raggiungere alle reti un equilibrio di Nash, secondo il quale nessuna delle due reti riesce a migliorare in maniera unilaterale la propria strategia per avere la meglio sull'altra. Durante l'addestramento congiunto, quando il discriminatore raggiunge il livello ottimale di accuratezza i suoi parametri vengono congelati e si continua ad addestrare il generatore fino a raggiungere un livello di accuratezza tale da produrre campioni che riescono ad ingannare il discriminatore e di conseguenza abbassare il suo livello di accuratezza.

2.2 Architettura

Come già accennato, le GANs sono costituite essenzialmente da una rete neurale generatrice, G, ed una rete neurale discriminatoria, D, come mostrato in figura

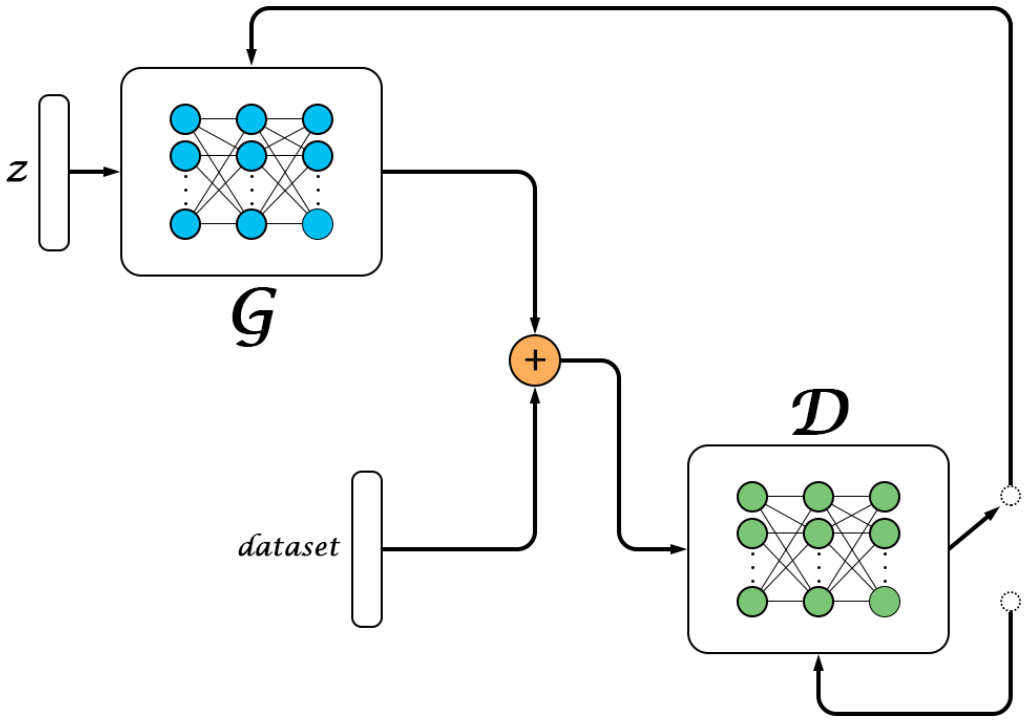

Figura 2.1 Architettura GAN

Nella maggior parte dei casi, le due reti sono o di tipo feedforward o di tipo feedback.

La più semplice rete feedforward è costituita da un primo strato, quello di input, composto da nodi che si collegano direttamente ad un secondo strato, quello di output. Il segnale nella rete si propaga in avanti in modo aciclico, partendo dallo strato di input e terminando in quello di output. Quindi non ci sono né connessioni trasversali all'interno degli strati né connessioni all'indietro. Reti feedforward più complesse presentano anche uno o più strati intermedi, chiamati strati nascosti, i cui nodi sono connessi sempre in modo tale da non creare cicli all'interno della rete.

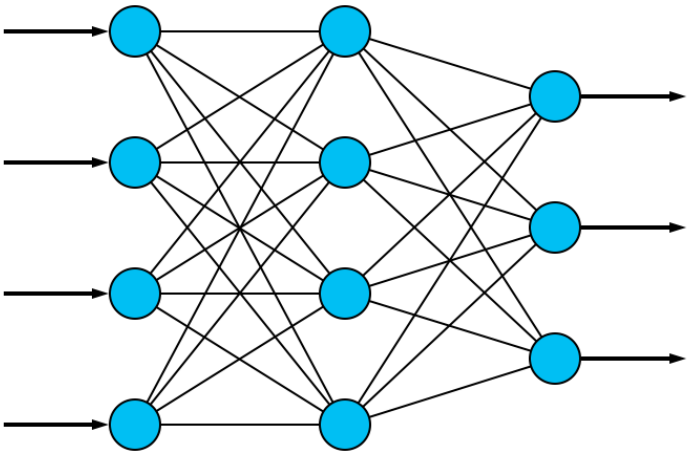

Figura 2.2 Reti FeedForward

Una rete feedback, chiamata anche rete ricorrente, presenta uno strato di input connesso a uno o più strati intermedi ed uno strato di output ma in questo caso i nodi di ogni strato possono essere connessi in modo trasversale e all'indietro.

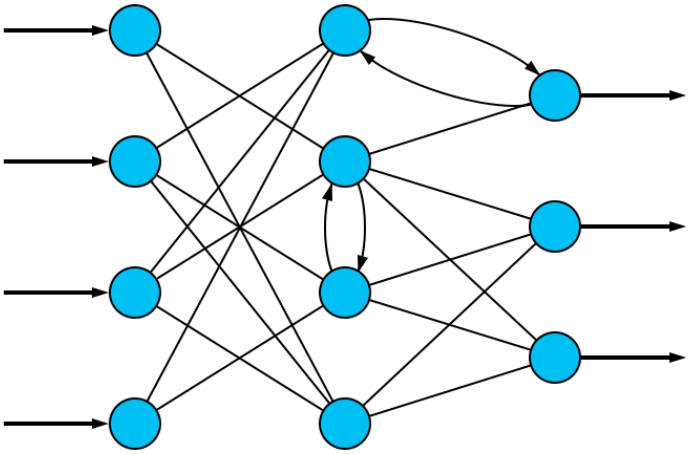

Figura 2.2 Reti FeedBack

La scelta di uno o dell'altro tipo dipende dal dominio di applicazione. Ad esempio, nell'ambito della sintesi delle immagini, uno dei modelli di GAN più utilizzato è la variante DC-GAN (Deep Convolutional GAN) costituita da due reti di tipo feedforward, in particolare da reti convoluzionali che presentano una organizzazione interna dei nodi simile a quella della corteccia visiva animale [6, 7]. Grazie a questo modello, è stato possibile generare immagini fotorealistiche di facce [6], caratteri scritti a mano [4], lesioni della pelle ad alta definizione [8], paesaggi [6] ed altro ancora.

Un altro aspetto importante, per entrambe le reti, è quello di definire il numero di strati, il numero di nodi per ogni strato e il modo in cui sono connessi i nodi di uno strato con

quelli del successivo. Per quanto riguarda l'input e l'output delle reti non c'è alcun dubbio sulla scelta del numero di nodi. Nell'esempio del riconoscimento dei caratteri numerici scritti a mano, la dimensione dell'input del discriminatore è pari al numero di pixel della singola immagine presa in esame ad ogni iterazione del processo di apprendimento (ovvero 28x28 = 784), mentre l'output ha una dimensione pari al numero delle classi di dati presenti nel dataset, ovvero 10 in quanto si prendono in considerazione le immagini delle cifre da 0 a 9. Per quanto riguarda il numero di nodi negli strati intermedi ed il numero stesso di strati intermedi non esiste una regola fissa. La scelta di questi parametri può essere fatta sulla base di alcune supposizioni e/o linee guida, sempre in dipendenza del problema che si sta studiando. Certo è che variando il numero di strati intermedi e il numero di nodi per strato si possono raggiungere vari livelli di astrazione [9]. Per quanto riguarda la scelta della modalità di connessione dei nodi di uno strato ai nodi dello strato successivo, alcuni autori suggeriscono di utilizzare una rete completamente connessa, mentre altri consigliano di minimizzare il numero di strati completamente connessi per aumentare la fattibilità delle "deep neural networks".

Un ultimo aspetto, anch'esso di notevole importanza, è la scelta della funzione di attivazione dei nodi, per ogni strato, e la scelta delle metriche di valutazione della rete. La funzione di attivazione, chiamata anche funzione di trasferimento, definisce l'output di un neurone a partire da un input. Per quanto riguarda la scelta di tale funzione, alcuni autori adottano un approccio euristico. Per esempio, Radford [6], nello studio delle DC-GAN, ha mostrato che utilizzando la funzione di attivazione "Leaky ReLU" negli stati intermedi del discriminatore si ottengono prestazioni migliori, rispetto all'utilizzo di altre funzioni ReLU generiche.

In questa trattazione utilizzeremo due semplici reti neurali di tipo feedforward completamente connesse, in particolare utilizzeremo due reti MLP costituite da uno strato di ingresso, due strati intermedi ed uno stato di output.

2.3 L’addestramento

Lo scopo del generatore è riuscire ad ingannare il discriminatore, riuscendo a far passare un campione generato per uno reale. Mentre il discriminatore ha lo scopo di minimizzare l'errore di predizione, il generatore tenta di massimizzare tale errore. Ciò

vuol dire che le due reti sono in competizione in un gioco minimax a somma zero che possiamo formalizzare nel seguente modo:

$$\left(E_{x\sim p_D}[\log log\ D(x)\ ] + E_{z\sim p_Z}[\log log\ 1\ -\ G(D(z))\ ]\right)$$

Ma cosa vuol dire addestrare una rete neurale? Come fa la rete ad apprendere?

Uno dei metodi maggiormente noti per addestrare una rete neurale è l'algoritmo di backpropagation, che sistematicamente modifica i pesi delle connessioni sinaptiche in modo tale che la risposta complessiva della rete si avvicini sempre più alla risposta desiderata. Tale tecnica è utilizzata nell'apprendimento supervisionato tramite esempi. Invece l'algoritmo di apprendimento di una rete si basa su un metodo chiamato discesa del gradiente, che permette di trovare un minimo locale di una funzione, la funzione di loss. Inizialmente, in una rete neurale i pesi sinaptici sono impostati a valori casuali molto piccoli che, durante il processo di addestramento, tenderanno a cambiare e ad assumere valori non prevedibili. Il processo di addestramento consiste nel presentare in input alla rete diversi schemi o pattern di esempio ed applicare l'algoritmo di backpropagation. Nello specifico, l'addestramento avviene in due fasi:

- nella prima fase si presenta in input alla rete un esempio o un gruppo di esempi, (batch) che la rete propaga in avanti attraverso tutti i suoi strati. In questa fase i pesi sinaptici sono tutti fissati;

- nella seconda fase si confronta la risposta complessiva della rete con la risposta desiderata e si misura l'errore che viene propagato all'indietro della rete (backpropagation).

Dopo queste due fasi, i pesi vengono modificati tramite il metodo della discesa del gradiente in modo tale da minimizzare la differenza tra l'output della rete e l'output desiderato.

Addestrare una GAN può rivelarsi un compito non molto semplice: nel caso della singola rete discriminatoria bisogna preoccuparsi soltanto della struttura di quest'ultima

mentre per una GAN bisogna preoccuparsi anche di diverse problematiche relative all'accoppiamento e all'addestramento.

Il modello di GAN discusso in precedenza, anche se motivato da numerosi successi in diversi ambiti, presenta diversi problemi.

Un primo problema è legato alla limitata capacità del generatore di modellare i dati reali. Non sempre si riesce a modellare perfettamente i campioni presenti in un dataset soprattutto nel caso in cui i campioni presentano caratteristiche miste, sia qualitative che quantitative.

Un altro problema è legato alla possibilità che il discriminatore vada in overfitting ovvero che il discriminatore si adatti a caratteristiche che sono specifiche solo per il training set utilizzato ma che non hanno riscontro negli altri casi non ancora analizzati. Per questo motivo, in presenza di overfitting, le prestazioni della rete sui dati di addestramento saranno ottime mentre le prestazioni sui dati non ancora visionati saranno scadenti. Questo problema si presenta solitamente quando si hanno pochi dati rappresentanti per una determinata classe.

Un ulteriore problema è quindi costituito dallo sbilanciamento delle classi. Per descrivere in termini pratici cosa sia lo sbilanciamento delle classi, consideriamo un problema di classificazione binaria per il quale disponiamo di un dataset costituito da dati divisi in due classi, A e B. Quando una delle due classi, ad esempio A, presenta un numero di campioni molto maggiore o molto minore rispetto al numero di rappresentanti della classe B si ha uno sbilanciamento. Ciò potrebbe portare il discriminatore a raggiungere immediatamente un alto livello di accuratezza, il che fa pensare di aver risolto immediatamente il problema quando in realtà, sottoponendo al discriminatore nuovi esempi, ci si accorge che su tali esempi il discriminatore nella maggior parte dei casi fallisce.

Durante l'addestramento della GAN il generatore non attinge in alcun modo dal dataset dei campioni reali. Quindi sorge il problema di cosa dare in input alla rete. Tipicamente il generatore prende in input un vettore di numeri casuali, z, che chiamiamo rumore, e che attraverso una funzione, viene trasformato e propagato all'interno della rete fino a raggiungere l'output. Qui accade che i campioni generati vengono mescolati con i

campioni reali del dataset costruendo così un training set aumentato ed una ground-truth. Il training set e la ground-truth vengono utilizzati come input del discriminatore che, attraverso una funzione, trasforma e propaga i dati attraverso i suoi strati fino all'ultimo stato di output. Quando il discriminatore produce una risposta, il risultato viene confrontato con la ground-truth. Il risultato del confronto è cruciale: se il confronto ha successo, ovvero la predizione del discriminatore risulta veritiera in accordo con la ground-truth, allora bisogna segnalare al generatore che la strategia che sta usando per la generazione di campioni è ancora poco precisa e di conseguenza si deve procedere alla correzione dei pesi interni del generatore, affinché produca dei campioni più veritieri. Se invece il confronto non ha successo vuol dire che il discriminatore è stato ingannato dal generatore, di conseguenza si deve adottare una strategia di correzione dei parametri del discriminatore, affinché si possa aumentare l'accuratezza del riconoscimento dei campioni generati da quelli reali. La correzione avviene tramite l'algoritmo di backpropagation. La fase di addestramento richiede alcuni step preparativi. Prima di tutto bisogna disporre di un dataset costruito ad hoc. Quasi sempre i dataset disponibili, per le diverse applicazioni, hanno i problemi descritti in precedenza. Per questo motivo bisogna operare su di esso per eliminare le cause che potrebbero portare ad un fallimento complessivo della rete. Una prima operazione possibile è quella di eliminare i dati che presentano informazioni superflue, prettamente descrittive. Ad esempio, nel problema della classificazione del traffico di rete, una informazione superflua può essere l'indirizzo di rete della sorgente del flusso, in quanto un flusso di traffico maligno o benigno non è legato strettamente agli indirizzi della sorgente o della destinazione. Il successivo passo è quello di normalizzare o standardizzare i valori presenti nel dataset. Nel caso in cui le distribuzioni delle variabili sono direttamente confrontabili allora si ricorre alla normalizzazione, ovvero si scalano i valori all'interno del range [0, 1]. Quando invece abbiamo più variabili le cui distribuzioni non sono confrontabili direttamente, si ricorre alla standardizzazione in modo da renderle confrontabili.

Un esempio pratico:

nel problema della generazione di cifre scritte a mano, il dataset utilizzato è costituito da immagini in bianco e nero in cui ogni pixel è codificato con 8 bit. Quindi il colore che può assumere ogni pixel è compreso tra 0 e 255, dove 0 rappresenta il nero, 255 il

bianco e i numeri intermedi rappresentano le diverse varianti di grigio. In questo caso il dataset viene sottoposto ad un processo di normalizzazione, in quanto tutti i pixel hanno valori che si trovano nello stesso range [0, 255] e quindi sono confrontabili direttamente. Invece, in altri casi, potremmo avere alcune variabili che assumono valori compresi in un range, ad esempio [0, 100], ed altre variabili che assumono valori compresi in un range diverso, ad esempio [-1000, +1000]. In tali casi si ricorre alla standardizzazione che trasforma tali valori in modo da rendere le distribuzioni confrontabili.

Dopo la standardizzazione/normalizzazione del dataset di solito si procede con uno shuffle dei campioni in modo tale da escludere la possibilità che durante la fase di addestramento si selezionino dei gruppi di campioni non rappresentativi dell'intero dataset.

2.4 La Validazione

Dopo aver portato a termine la fase di addestramento del modello, il successivo passo è quello di validare il modello. Il processo di validazione consiste nel sottoporre alla rete discriminatoria dei campioni, che non ha mai processato, e verificare se la risposta data coincide con la risposta attesa. Tale validazione può essere fatta utilizzando un dataset diverso dal dataset di addestramento. In questo modo possiamo facilmente verificare se il discriminatore è andato in overfitting e correggere gli iperparametri della rete. Solitamente, dopo aver costruito un dataset equilibrato si procede col porzionarlo in due differenti dataset: uno costituisce il training set e presenta circa 70/80% dei dati presenti nel dataset originale mentre l'altro costituisce il validation set, formato dalla restante parte dei dati.

2.5 Applicazioni

Il dominio di applicazione in cui le GANs hanno riscosso maggior successo è quello della sintesi di immagini. Come accennato in precedenza con l'utilizzo delle GANs e alcune sue varianti è stato possibile generare esempi di immagini fotorealistiche che rappresentano volti umani [6], lesioni della pelle [8], animali, paesaggi, caratteri numerici scritti a mano [4] e così via. In altri ambiti come quello della Sicurezza, tali

reti sono state utilizzate per migliorare l'efficacia dei sistemi di rilevamento di transazioni fraudolente [11], per attaccare sistemi di riconoscimento delle impronte digitali [10] e testarne la loro sicurezza, per costruire sistemi di riconoscimento delle intrusioni ed in generale delle anomalie [12].

CAPITOLO 3

CLASSIFICAZIONE DEL TRAFFICO DI RETE

3.1 Introduzione

Il problema della classificazione del traffico di rete può essere enunciato come un problema di classificazione binaria in cui si distingue il traffico di rete benigno dal traffico di rete maligno. La maggior parte dei dataset utilizzati per questo problema, essendo costruiti per lo più in laboratorio, approssimano il traffico di rete reale. Inoltre, tali dataset contengono diverse imperfezioni, ed in particolare sono fortemente sbilanciati in quanto i campioni del dataset che identificano il traffico maligno risultano essere la minoranza. Nello sviluppo del problema sono state create due classi di traffico di rete, quella relativa al traffico benigno correlata alle normali comunicazioni e quella relativa al traffico maligno correlata al traffico generato da Botnet. Una Botnet è una rete di dispositivi infettati da un malware che vengono controllati da un botmaster che potrà fruttarli per eseguire, nella maggior parte dei casi, attacchi di tipo Distributed Denial of Service (DDoS) all'oscuro dei reali proprietari dei device.

3.2 Dataset

Il dataset utilizzato è stato fornito su richiesta dalla University of New Brunswick [19], costruito utilizzando il software CICFlowMeter [20] che analizza il traffico di rete estrapolando diverse features. Questo dataset presentava delle imperfezioni che sono state risolte in più fasi:

- Pulizia del dataset
- Standardizzazione/Normalizzazione
- Isolamento dei campioni delle classi di interesse
- Downsampling e costruzione dei dataset
- Shuffling

Alla fine di queste fasi di preparazione otteniamo:

- un training set bilanciato composto da 3800 campioni da un totale di 191.000
- un test set bilanciato composto da 132 campioni

3.3 Conclusioni

Per quanto riguarda il dataset aumentato bisogna prima di tutto addestrare la GAN per poi sfruttare la capacità del generatore di sintetizzare dei dati di traffico maligno. Dopo l'addestramento della GAN quindi si chiede al generatore di costruire dei campioni che saranno utilizzati insieme ai campioni di traffico benigno e maligno esistenti. In questo caso il dataset aumentato è stato costruito inserendo i 1900 campioni di traffico benigno utilizzati nel training set iniziale, i 1900 campioni di traffico benigno del dataset aggiuntivo, i 1900 campioni di traffico maligno selezionati dal training set aumentato e i 1900 campioni prodotti dal generatore. Una volta uniti tali campioni, ognuno avente la propria etichetta, è stato applicato anche in questo caso lo shuffling dei campioni.

CAPITOLO 4
FRAMEWORK

4.1 Introduzione

Il framework utilizzato per affrontare il problema della classificazione binaria del traffico di rete nelle classi di "Traffico Benigno" e "Traffico Maligno" è composto da una rete GAN e da un discriminatore che chiameremo “Discriminatore BM”, responsabile di distinguere a quale delle due classi appartiene il campione che gli viene sottoposto.

4.2 La Rete GAN

Tale rete, responsabile della generazione dei campioni di traffico maligno, è costituita da una architettura a più strati.

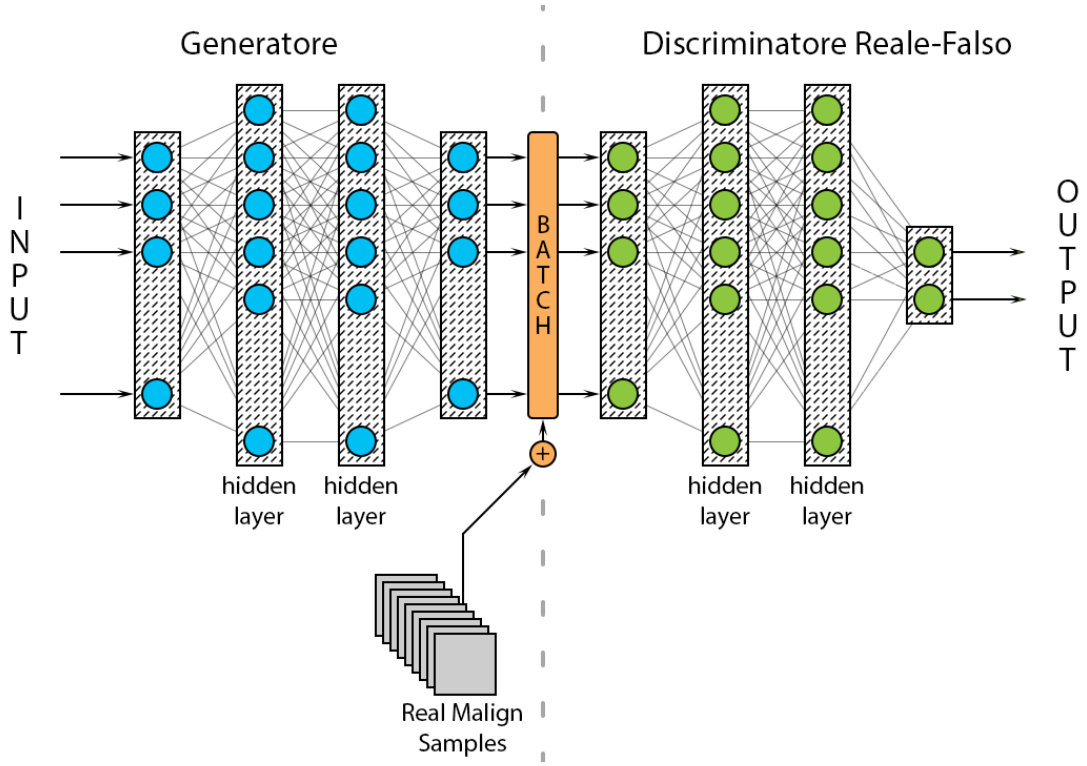

Figura 4.1 Architettura GAN

La **prima sezione** della rete è quella dedicata al generatore, composto da uno strato di input la cui taglia è pari a 100. In questo strato vengono immessi 100 valori casuali, equamente distribuiti nell'intervallo [0, 1], tramite un vettore che ad ogni ciclo di addestramento viene ricalcolato. La taglia di input del generatore si può scegliere in modo arbitrario ma per mantenere una sorta di convenzione è stato scelto un valore pari a 100. Lo strato di input ha un output di taglia 280, scelta arbitrariamente. Il secondo strato, chiamato layer di normalizzazione (o Batch Normalization Layer) è stato inserito per aiutare la rete a raggiungere un determinato livello di accuratezza utilizzando meno passi di apprendimento [13]. Tale strato, nella pratica, applica una trasformazione all'input proveniente dallo strato precedente che mira a mantenere la media dei valori dell'input vicina allo 0 e la deviazione standard vicina a 1. Le taglie di input e output sono pari a 280. Il terzo strato, chiamato layer di dropout, è stato inserito per aiutare la rete a prevenire o, nel peggiore dei casi, ridurre l'overfitting [14]. Tale strato, ad ogni ciclo di addestramento, seleziona in modo casuale una percentuale di nodi e relative connessioni da escludere temporaneamente durante l'addestramento. Alla fine di un ciclo di addestramento tali nodi vengono inclusi nuovamente. Anche per questo strato le taglie dell'input e dell'output sono pari a 280. Il quarto strato, ovvero il primo dei due strati nascosti, simile a quello di input, è costituito da una taglia di input di 280 ed una taglia di output di 140. I successivi due strati che seguono sono uguali agli strati 2 e 3, ovvero quelli di normalizzazione e dropout. L'unica differenza è nella taglia di input e output, pari a 140 per entrambi gli strati. Il successivo strato, ovvero il secondo strato nascosto, anche questo simile al primo, ha una taglia di input pari a 140 ed una taglia di output pari a 70. L'ultimo strato, quello di output, fornisce un vettore di 70 elementi che corrisponde al campione di traffico generato.

Tutti gli strati sono completamente connessi tra loro, ovvero per ogni strato, ogni nodo dello strato stesso è connesso a tutti i nodi dello strato successivo.

Per quanto riguarda le funzioni di attivazione sono state scelte la “Rectified Linear activation Units” (ReLU) e la Softmax. Tra lo strato di input ed il primo strato nascosto, così come tra i due strati nascosti, è stata utilizzata la funzione ReLU, che in termini pratici propaga il dato ricevuto in input se esso è maggiore di 0 e propaga 0 se il dato in input è minore di 0, mentre tra l'ultimo strato nascosto e lo strato di output è stata utilizzata la funzione Softmax che ha lo scopo pratico di mettere in evidenza i valori più

grandi e attenuare quelli che sono significativamente più piccoli del valore massimo. La scelta di utilizzare la funzione ReLU è dettata dal fatto che, generalmente, tale funzione offre le migliori prestazioni nei modelli multistrato di Deep Learning [15].

La **seconda sezione** della rete GAN è costituita da un discriminatore che chiameremo Discriminatore Reale-Falso in breve RF, che viene addestrato a riconoscere i campioni generati dai campioni reali. La struttura di tale discriminatore ha un primo strato di input con taglia di 70 in quanto ogni campione è composto da 70 features. La taglia dell'output di questo strato è di 256. Anche stavolta la scelta della taglia dell'output è arbitraria ed è scelta dopo diversi tentativi riscontrando appunto prestazioni migliori. Il secondo strato, un layer di dropout, è simile a quello utilizzato nel generatore e ha la medesima funzione di ridurre l'overfitting. Le taglie di input e output per tale strato sono di 256. Il successivo strato, ovvero il primo strato nascosto, ha una taglia di input di 256 ed una di output pari a 128. Segue uno strato di dropout che ha taglie di input e output pari a 128. Il successivo strato, il secondo dei due strati nascosti è simile al primo ma ha taglia di input pari a 128 mentre la taglia di output è uguale a 2. L'ultimo strato, quello di output ha una taglia pari a 2 in quanto il discriminatore ha il compito di individuare se il campione appartiene alla classe del traffico generato oppure a quella del traffico reale.

Anche per il Discriminatore RF sono state utilizzate le medesime funzioni di attivazione, la ReLU e la Softmax.

4.3 Il Discriminatore Benigno-Maligno

Il discriminatore Benigno-Malignio, in breve BM, distingue a quale classe appartiene il campione che gli viene sottoposto ed ha la medesima struttura del discriminatore RF utilizzato all'interno della GAN, non ci sono differenze sostanziali. L'unica differenza sta nel fatto che il discriminatore viene posto in coda alla GAN e, invece di distinguere i

campioni di traffico tra i campioni reali e quelli generati, ha il compito di distinguere se il campione appartiene alla classe dei campioni di traffico benigno o alla classe dei campioni di traffico maligno.

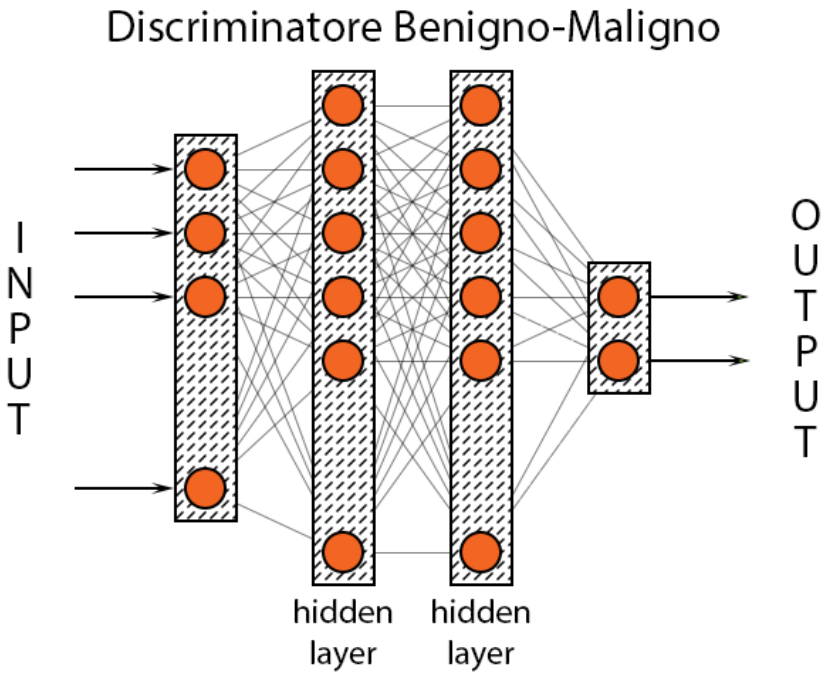

Figura 4.1 Architettura Discriminatore BM

4.4 Framework Completo

Nella figura seguente è illustrato il framework completo.

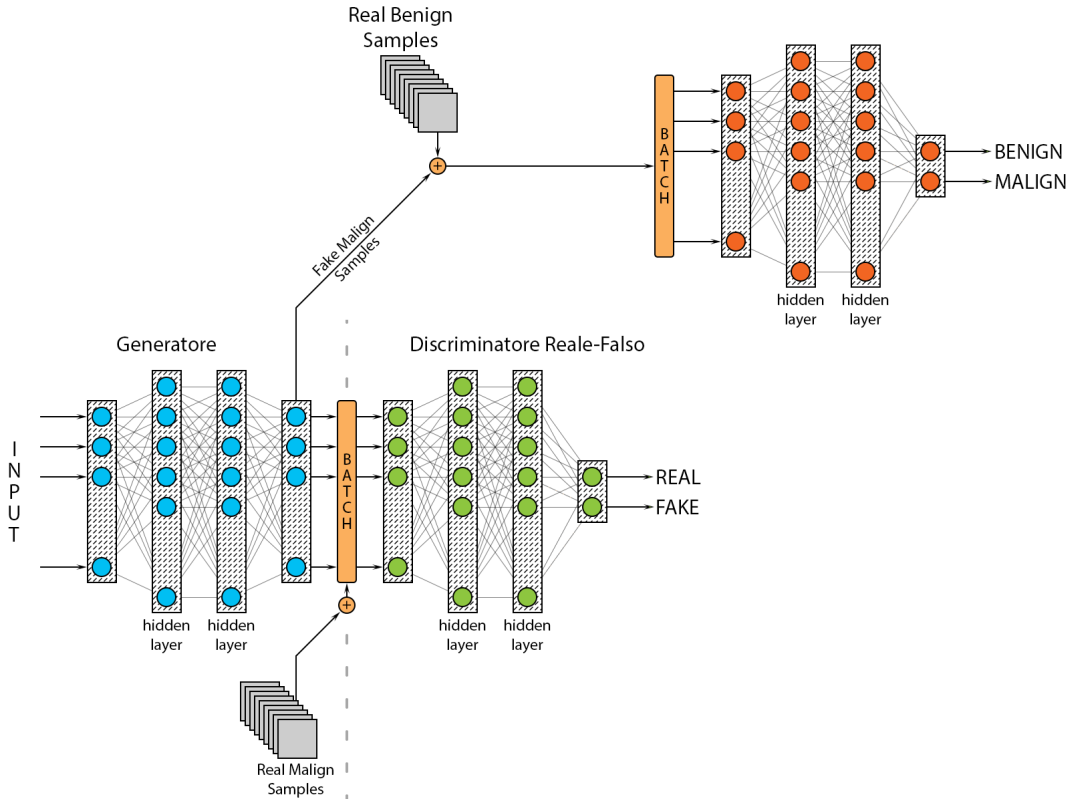

Figura 4.1 Architettura Completa

CAPITOLO 5

RISULTATI

5.1 Introduzione

Una volta definiti e compilati i modelli (Discriminatore Benigno-Maligno e Generatore) è stato possibile implementare il processo di apprendimento degli stessi. L'apprendimento che abbiamo ritenuto opportuno utilizzare è quello supervisionato, nel quale si fornisce alla rete un gruppo di input ai quali corrispondono output noti. La rete analizza i dati e apprende come generalizzarli, ossia riesce a computare coppie (input-output) corrette. Per far sì che ciò sia possibile si utilizzano algoritmi di apprendimento che partono dal presupposto che, fornendo un numero ampio di dati al modello, questo accumulerà un'esperienza tale da permettere alla stessa di creare una nuova funzione, capace di approssimare quella di partenza. Quando verranno proposti dei nuovi esempi alla rete, non presenti nella sua esperienza, essa dovrà essere in grado di approssimare i dati e fornire una risposta corretta.

5.2 Addestramento Discriminatore BM

Il Discriminatore BM è stato addestrato utilizzando il dataset iniziale bilanciato, contenente soltanto dati reali Benigni e dati reali Maligni. Per l'apprendimento abbiamo previsto 80 cicli, chiamati "epoche". In pratica, durante l'addestramento, la rete non impara ad associare ogni input ad un output ma impara a riconoscere la relazione che esiste tra input e output. Possiamo vederla come una "scatola nera", che non ci svelerà la formula matematica che unisce i dati, ma ci permetterà di ottenere risposte significative a input di cui non sappiamo ancora la loro classificazione. Per monitorare l'apprendimento della rete abbiamo utilizzato la funzione di loss e una metrica semplice, l'accuracy (accuratezza), con le quali ad ogni epoca abbiamo potuto quantificare l'apprendimento. In pratica la funzione di loss, misura la distanza tra la risposta y del discriminatore per un dato vettore di input x e la risposta f (x, w) restituita dall'algoritmo di apprendimento.

$$L(y, f(x, w))$$

L’accuracy, o accuratezza è una misura di concordanza data dalla semplice equazione:

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

con TP (Veri Positivi), TN (Veri Negativi), FP (Falsi Positivi) e FN (Falsi Negativi) rappresentanti le quantità delle quattro categorie riscontrate nella matrice di confusione illustrata più avanti in questo paragrafo. Questa misura potrebbe portare ad una interpretazione errata della qualità delle classificazioni, quando le osservazioni appartengono a classi fortemente sbilanciate verso i veri positivi o i veri negativi. Tuttavia, risulta facile da calcolare e può dare una visione rapida della correttezza di un classificatore. Al termine dell’addestramento i valori di accuracy e loss ottenuti sono mostrati di seguito:

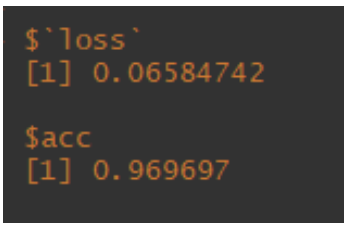

Figura 5.1 Output addestramento

Per produrre tali risultati è stato utilizzato un dataset di validazione, che non è mai stato sottoposto alla rete durante l’addestramento. Tali valori sono ottimi ma non abbastanza, in quanto gli errori rilevati sono ancora molti, come è possibile notare dalla matrice di confusione presente nelle prossime pagine.

Per rappresentare l’andamento dei parametri loss e accuracy sono state utilizzate apposite funzioni con le quali è stato ottenuto il grafico che segue. Come si evince dalla figura, il parametro di loss decresce ad ogni epoca raggiungendo valori sempre più piccoli fino ad un valore dello 0.06, mentre il valore di accuracy cresce ad ogni epoca, arrivando ad un valore di accuratezza del 96.96%.

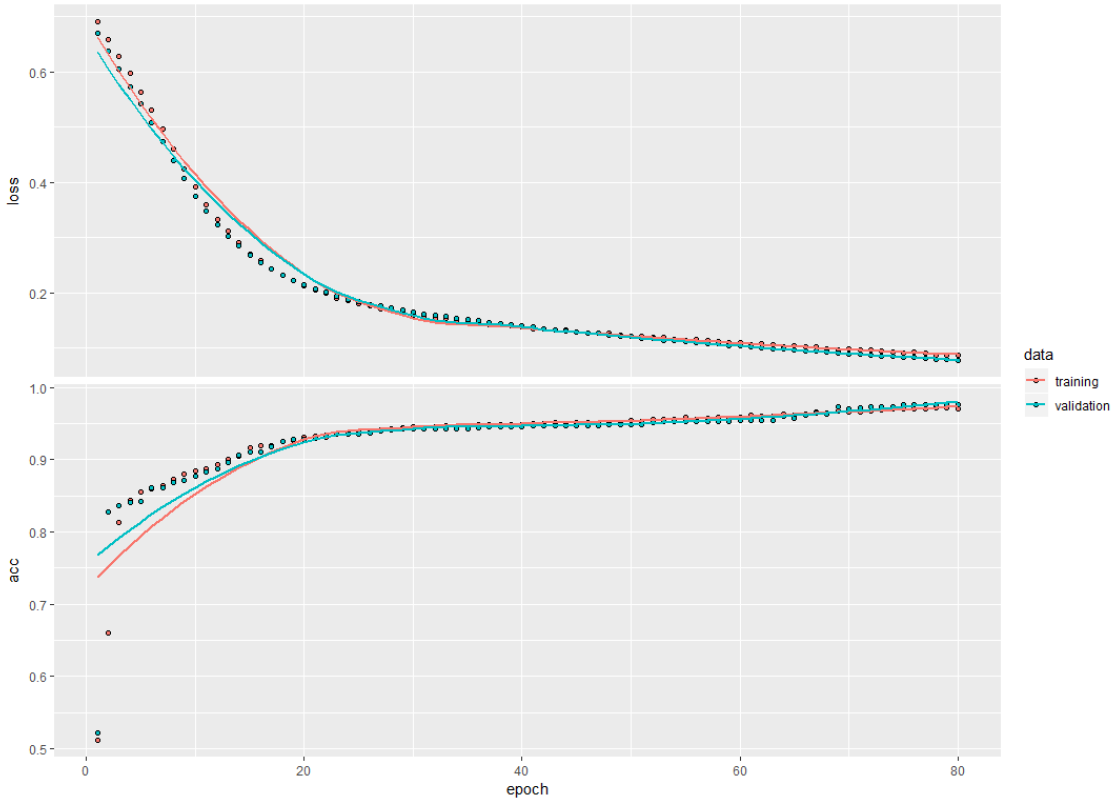

Figura 5.2 Grafico andamento addestramento

Abbiamo, inoltre, costruito una "matrice di confusione", chiamata anche matrice di errore [16], la quale è una tabella con un particolare layout, che permette la visualizzazione delle prestazioni di un algoritmo di apprendimento supervisionato. Ogni riga della matrice rappresenta le istanze in una classe prevista mentre ciascuna colonna rappresenta le istanze di una classe effettiva (o viceversa) [17]. Il nome deriva dal fatto che rende facile vedere se il sistema confonde due classi.

In una matrice di confusione, c'è una tabella con due righe e due colonne che riporta il numero di falsi positivi, falsi negativi, veri positivi e veri negativi. Ciò consente un'analisi più dettagliata rispetto alla semplice proporzione tra classificazioni corrette e classificazioni errate. L'accuratezza non è una metrica attendibile per le prestazioni reali di un classificatore, poiché produrrà risultati fuorvianti se il set di dati non è bilanciato.

Questi sono i risultati riportati dalla nostra confusion matrix:

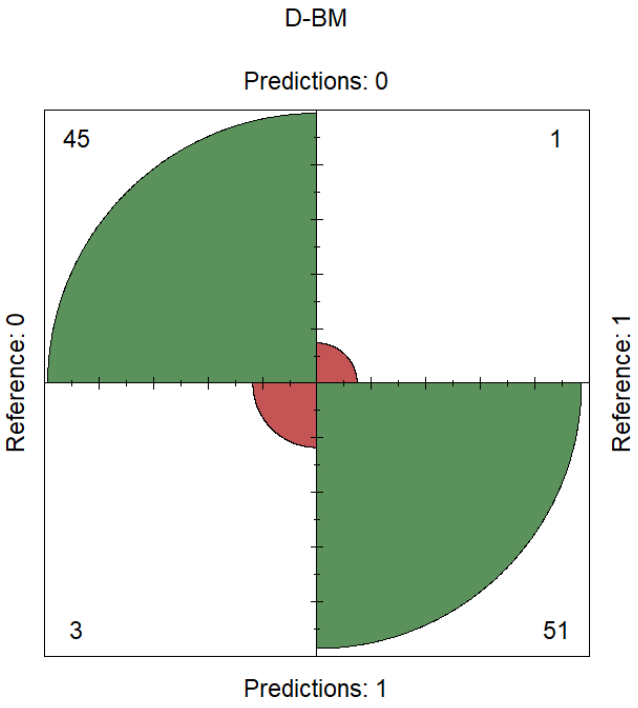

Figura 5.3 Matrice di Confusione

Come si evince dalla figura si hanno:

- **45 Veri negativi**, campioni “benigni”, correttamente classificati come tali.
- **51 Veri positivi**, campioni “maligni”, correttamente classificati come tali.
- **1 Falso positivo**, campioni “benigni”, erroneamente classificati come “maligno”
- **3 Falsi negativi**, campioni “maligni”, erroneamente classificati come “benigni”

I “falsi positivi” ed i “falsi negativi” sono noti come, errori di Tipo I e Tipo II.

Un errore di tipo I (falso positivo) porta a concludere che un presunto campione di traffico di rete è maligno quando in realtà è benigno. Quest’errore, in relazione all’errore di tipo II, è meno grave. Volendo fare un’analogia in ambito medico, affermare che un paziente sano sia portatore di una grave malattia è meno grave di affermare che un paziente malato in realtà non lo sia.

5.3 Addestramento Modello Avversariale e Costruzione Dataset BM Aumentato

Per iniziare è stato creato un batch di 512 campioni. La prima metà è costituita da campioni di traffico maligno presi dal dataset originale, mentre la seconda metà è costituita da vettori di numeri casuali generati a partire dalla distribuzione normale, o di Gauss (o gaussiana). Di questo batch, la label correlata al vettore di numeri casuali è stata impostata a 1, mentre per i dati presi dal dataset originale la label viene impostata a 0. Lo 0 rappresenta la classe dei dati reali, mentre l'1 rappresenta la classe dei dati generati. Il batch viene dato in input al discriminatore e con esso viene effettuato l'addestramento, con lo scopo di riuscire a generare dati sempre più vicini a quelli reali. Si procede con l'addestramento del modello avversariale, utilizzando un batch di dati generati ma etichettati come reali. Durante questa fase, il discriminatore deve etichettare i campioni come reali o falsi. In caso di errore da parte del discriminatore, viene generato un segnale d'errore per il discriminatore stesso, così che possa minimizzare il suo errore. Viceversa, in caso di errore da parte del generatore, quindi il campione generato viene etichettato come "falso", il generatore impara dall'errore cosi da minimizzare la possibilità di essere "scoperto" dal discriminatore, ovvero massimizzare l'errore del discriminatore.

5.4 Addestramento Discriminatore con dataset Aumentato

Infine, abbiamo sottoposto il Discriminatore Benigno-Maligno ad un ultimo addestramento, questa volta però utilizzando un dataset aumentato. Questa scelta è stata fatta poiché nell'analisi del traffico di rete, sorgono molti problemi quando ci si imbatte nei dataset. Infatti, quest'ultimi presentano delle problematiche, come sbilanciamento dei campioni, campioni generati artificialmente (non da reti neurali) e quindi lontani da quelli reali. Per ovviare a questi problemi abbiamo sfruttato la capacità del generatore di produrre traffico maligno. Il dataset aumentato è costituito da 1900 campioni di traffico benigno utilizzati nel training set iniziale, 1900 campioni di traffico benigno del dataset aggiuntivo, 1900 campioni di traffico maligno selezionati dal training set aumentato e 1900 campioni prodotti dal generatore. Una volta uniti tali campioni, ognuno avente la propria etichetta, è stato applicato anche in questo caso lo shuffling dei campioni.

Al termine delle 80 epoche di addestramento abbiamo ottenuto risultati migliori, come si evince dalla figura sottostante.

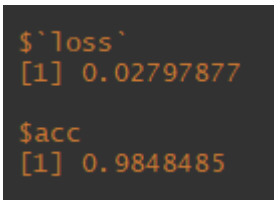

Figura 5.4 Output addestramento

Come fatto inizialmente, abbiamo costruito un grafico attraverso apposite funzioni, che rappresenta l'andamento dei parametri loss e accuracy.

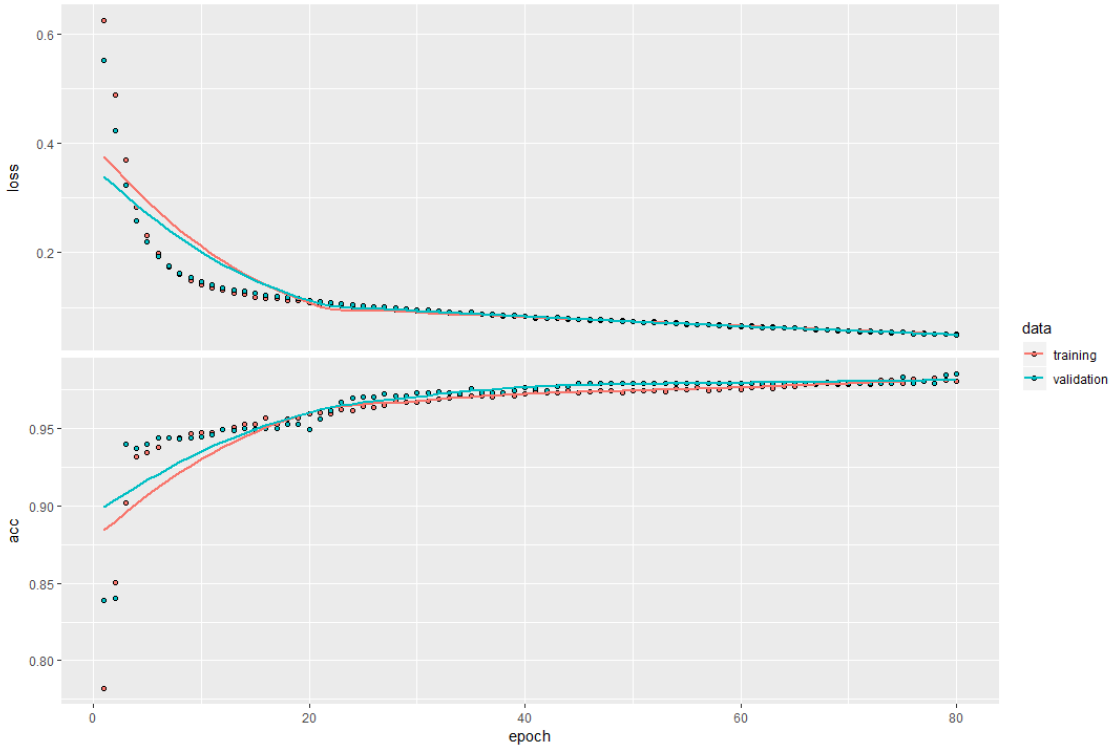

Figura 5.5 Grafico andamento addestramento

Come si evince dalla figura, il parametro di loss decresce ad ogni epoca raggiungendo valori sempre più piccoli fino ad un valore dello 0.02, mentre il valore di accuracy cresce ad ogni epoca, arrivando ad un valore di accuratezza del 98.48%.

Abbiamo poi infine costruito la matrice di confusione (confusion matrix), ottenendo altrettanti ottimi risultati.

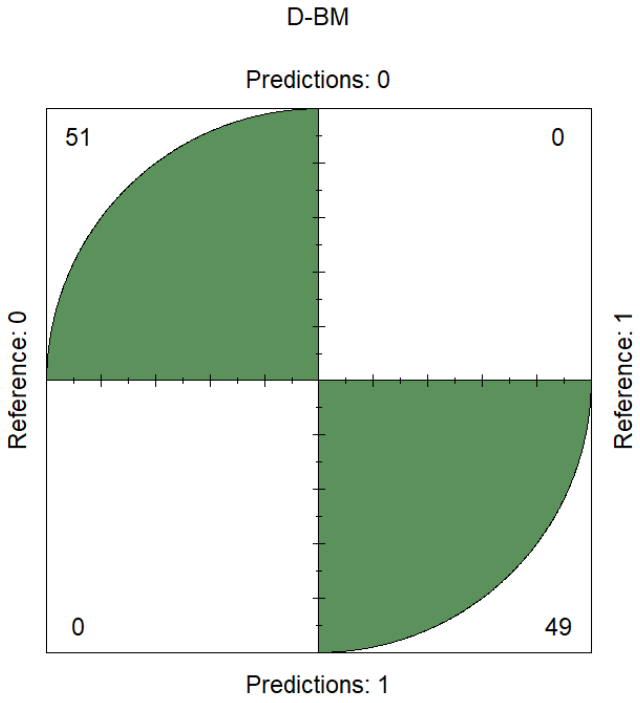

Figura 5.6 Matrice di Confusione

Come si può ben notare nella figura precedente, gli errori di Tipo I e errori di Tipo II sono pari a 0. Ciò vuol dire che dopo l'addestramento del discriminatore BM con il dataset aumentato, le prestazioni di tale rete hanno subito un forte incremento.

CAPITOLO 6

STRUMENTI UTILIZZATI

6.1 Introduzione

Negli ultimi anni gli strumenti software per il Machine Learning stanno diventando sempre più numerosi, ad oggi se ne contano alcune decine [18]. La maggior parte di essi sono librerie che possono essere usate in diversi linguaggi come C++, Java, Python, R e così via.

Gli strumenti utilizzati per sviluppare il sistema di riconoscimento del traffico di rete descritto nei precedenti capitoli sono le librerie Keras, TensorFlow, il linguaggio R e la libreria NVIDIA CUDA Deep Neural Network (cuDNN). Di seguito sono illustrati i dettagli per ogni strumento.

6.2 Il linguaggio R

R è un linguaggio utilizzato prevalentemente nell'ambito della statistica e analisi dei dati, distribuito con la licenza GNU/GPL e quindi di libero utilizzo. Inoltre, tale ambiente è disponibile per diversi sistemi operativi tra cui Linux, Microsoft Windows e Apple MacOS. Le principali motivazioni, che ci portano alla scelta di tale linguaggio, sono date dal fatto di essere uno strumento molto performante nel processare e gestire grandi quantità di dati e di avere una sintassi semplice e flessibile. In più è uno strumento gratuito che tutti possono installare. Nonostante possa sembrare uno strumento di nicchia, la comunità che ne supporta lo sviluppo è molto vasta.

R, come molti altri linguaggi, può essere utilizzato in un ambiente di sviluppo integrato (IDE) che aiuta il programmatore a scrivere il codice sorgente, compilarlo e testarlo sfruttando diversi strumenti messi a disposizione dalla IDE stessa. Nel nostro caso la IDE scelta è RStudio, costruita appositamente per il linguaggio R che offre diversi vantaggi, come l'integrazione di strumenti di versioning, l'installazione di plugin direttamente dai repository in rete, la possibilità di visualizzare i dati tramite grafici, che possono essere esportati in diversi formati e la possibilità di sfruttare una console interattiva.

Figura 6.1 Rstudio logo

6.3 TensorFlow

Questa libreria open source, sviluppata da ricercatori del team di Google Brain, è progettata per il calcolo numerico ad alte prestazioni. L'architettura flessibile di TensorFlow permette di computare dati su differenti piattaforme ed architetture, nello specifico è possibile eseguire calcoli sfruttando l'utilizzo di CPU e anche GPU, in ambienti Mobile, Desktop e Cloud. La computazione su GPU è possibile grazie al supporto di Cuda e OpenCL. Oltre al classico funzionamento su CPU o GPU, TensorFlow può funzionare su processori ASIC, progettati da Google espressamente per questo linguaggio, chiamati Tensor Processing Unit o, più brevemente, TPU. TensorFlow ad oggi è uno degli strumenti più utilizzati per il Machine Learning che fornisce dei moduli ottimizzati per diversi scopi. Attualmente la libreria è utilizzata sia in ambito di ricerca che di produzione.

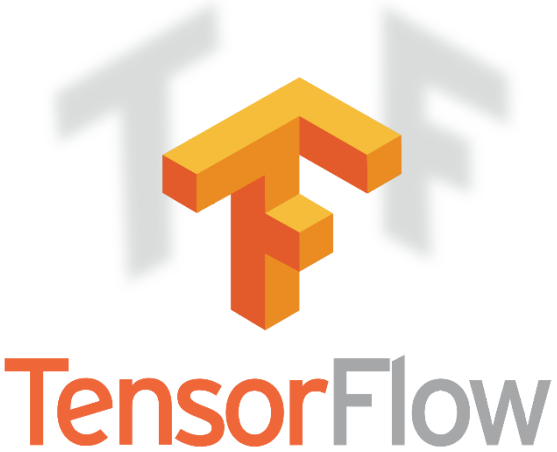

Figura 6.2 TensorFlow logo

6.4 cuDNN

La libreria NVIDIA Cuda Deep Neural Network, in breve cuDNN, è uno strumento che fornisce un supporto per le routine comuni nello sviluppo di reti neurali utilizzando la potenza computazionale delle GPU NVIDIA. Grazie alla piattaforma CUDA, una piattaforma di computazione parallela inventata da NVIDIA, cuDNN riesce a sfruttare i più performati core della GPU che nel complesso possono eseguire migliaia di threads. cuDNN quindi aiuta i programmatori a concentrarsi sulla costruzione e sull'addestramento delle reti neurali senza preoccuparsi della messa a punto dei parametri di basso livello della GPU che permettono alla rete di raggiungere le migliori prestazioni. Questa libreria è ampiamente utilizzata per accelerare il calcolo numerico nei framework di Deep Learning come Keras, TensorFlow, Theano ed altri.

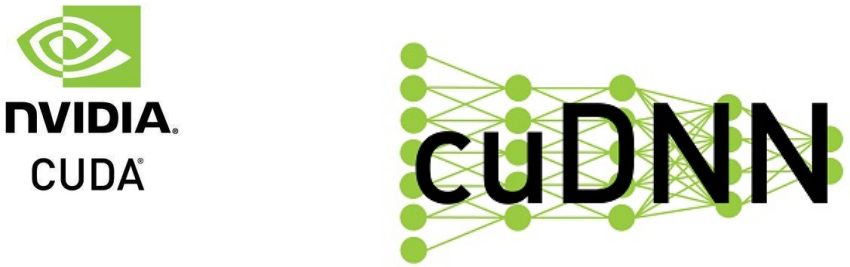

Figura 6.3 Nvidia Cuda Toolkit

6.5 Keras

Keras è una API ad alto livello per lo sviluppo e la valutazione di modelli di Deep Learning capace di sfruttare diverse librerie di calcolo numerico ad alte prestazioni come TensorFlow e Theano. La potenza di questa libreria è quella di aiutare lo sviluppatore a costruire, addestrare e valutare un modello di rete neurale con poche righe di codice. Keras quindi risulta essere uno strumento potente per la prototipazione veloce di reti neurali.

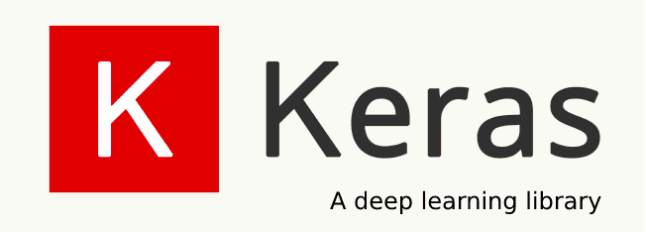

Figura 6.4 Keras Logo

6.5 Commenti finali

Tutti gli strumenti scelti hanno la caratteristica di essere di libero utilizzo ed in quanto tali possono essere scaricati liberamente. Inoltre, essendo software di ampia diffusione, dispongono di una documentazione dettagliata, ricca di esempi pratici che facilitano in modo notevole il primo approccio con tali strumenti. Oltre alla documentazione ufficiale, in rete ci sono diverse community di supporto attive e diversi blog che facilitano ulteriormente l'approccio iniziale.

CAPITOLO 7

CONCLUSIONI

Al termine di questo progetto di tesi, posso affermare che i risultati ottenuti hanno superato di molto le aspettative iniziali.

In primis, per quanto riguarda il problema della classificazione del traffico di rete, intesa come anomaly detection, in *traffico benigno* e *traffico maligno,* il quale poteva essere risolto con diverse strategie, è stato risolto utilizzando una classe di algoritmi di intelligenza artificiale, **il modello "GAN".** Questo modello permetterà in futuro ai computer di avere immaginazione e creatività, avendo un'idea di base molto semplice riassumibile in pochi passaggi:

- scelta di una collezione di dati (dataset) correlata al problema affrontato, nel nostro caso fornita dalla *"University of New Brunswick";*
- pulizia del dataset utilizzato;
- analisi ed eliminazione dei campioni poco significativi;
- analisi del numero di campioni per categoria, così da determinare il livello di sbilanciamento del dataset;
- costruzione di un dataset bilanciato, partendo da quello iniziale sbilanciato, utilizzato come *training set.*

Una volta completati questi passaggi è stato implementato il processo di addestramento della rete, con l'obiettivo di perfezionare l'approssimazione dei dati da parte della rete e quindi fornire una risposta corretta. L'addestramento vede il Discriminatore Benigno-Maligno apprendere a partire dal dataset iniziale bilanciato. Per produrre tali risultati è stato utilizzato un dataset di validazione, che non è mai stato sottoposto alla rete durante l'addestramento. Tali valori sono ottimi ma non abbastanza, infatti, il nostro **obiettivo** è stato quello di migliorare questi risultati con l'ausilio della **rete GAN.**

Attraverso un procedimento molto complesso è stato addestrato il modello avversariale, con lo scopo di riuscire a generare dati sempre più vicini a quelli reali, grazie al quale è stato costruito un dataset aumentato bilanciato. Sfruttato questa capacità del generatore di produrre *"traffico maligno"*, abbiamo potuto ovviare al problema iniziale della scarsità di campioni di *"traffico maligno"*. I risultati ottenuti sono stati sbalorditivi, infatti, avendo utilizzato, come dataset iniziale, uno di tipo bilanciato (situazione irreale), i risultati attesi, con l'ausilio di una rete GAN, dovevano essere peggiori di quelli ottenuti inizialmente. Al Termine del processo abbiamo avuto valori di loss e

accuracy, rispettivamente di 0.027 e 98.84%. Tali valori rispetto a quelli iniziali sono altamente migliorati. Il valore di loss è addirittura diminuito del 65% , mentre l'accuracy è aumentata del 2%.

Infine, con l'ausilio della rete GAN, è stato possibile così migliorare l'apprendimento del classificatore, Benigno-Maligno, laddove non sarebbe stato possibile migliorarne le prestazioni tramite sistemi convenzionali o in ogni caso senza l'uso di classi di algoritmi di Machine Learning di base.

CAPITOLO 8

SVILUPPI FUTURI

Le reti GAN sono algoritmi fondamentali per il progresso tecnologico e sociale, dato che le potenzialità espresse attualmente sono solo una minima parte e legate esclusivamente alla capacità degli studiosi della computer science di poterle sfruttare al meglio. A seguito di questo elaborato si possono facilmente dedurre le possibilità di impiego pratico di queste reti, infatti così come è stato possibile generare nuovi esempi di trasmissioni sia benigne che maligne tra due dispositivi, l'impego delle reti GAN è praticabile in qualunque ambito in cui è possibile tradurre un problema in un data set da dare in pasto all'algoritmo ovvero che ha il giusto numero di features e bilanciato, basti pensare all'ambito medico dove ogni tipo di patologia è quantificabile tramite dei dati numerici e quindi modellabile ai fini di un utilizzo per l'addestramento di reti neurali artificiali. Inoltre la key feature di questi algoritmi è proprio la loro capacità di essere usate laddove la mancanza di dati non permette un uso ideale dei classici modelli di machine learning, e dunque ritornando all'ambito medico, potenzialmente potrebbero essere usate per generare nuovi sample di dati, che rappresentano ad esempio un determinato tipo di malattia aiutando così la comprensione e lo studio più approfondito di determinati fenomeni.

APPENDICE A

10.1 Codice Sorgente

Di seguito è riportato il **codice sorgente** completo.

```
setwd("C:/Users/izzol/Desktop/GAN-Progetto/Tesi")

library(keras)
library(ROCR)
library(caret)
library(stringr)

use_implementation("tensorflow")
seed <- sample(1000, 1) + 5000
use_session_with_seed(seed, disable_gpu = FALSE, disable_parallel_cpu = FALSE)

# -------------------- SEZIONE COSTANTI
N_FEATURES <- 70
LABELS_COL <- 71
REAL <- 0
BENIGN <- 0
FAKE <- 1
MALIGN <- 1

# -------------------- SEZIONE VARIABILI
ds_bm_training <- NULL                          # Dataset BM Training
ds_bm_testing <- NULL                           # Dataset BM Test
lbl_bm_training <- NULL                         # Etichette Dataset BM Training
lbl_bm_testing <- NULL                          # Etichette Dataset BM Test

ds_bm_aug <- NULL                               # Dataset BM Aumentato
lbl_bm_aug <- NULL                              # Etichette Dataset BM Aumentato

ds_b_add <- NULL                                # Dataset B Addizionale

# -------------------- SEZIONE IPERPARAMETRI
g_in = 100                                      # Dimensione Input Generatore
g_out = N_FEATURES                              # Dimensione Output Generatore
g_lr = 0.0001                                   # Learning Rate Generatore
g_opt = optimizer_adam(lr = g_lr)               # Funzione Ottimizzazione Generatore
g_dropout = 0.15                                # Percentuale Dropout Generatore
g_noise = NULL                                  # Vettore Input Generatore
g_momentum = 0.9                                # Momentum Generatore

d_in = N_FEATURES                               # Dimensione Input Discriminatore
d_out = 2                                       # Dimensione Output Discriminatore
d_lr = 0.0001                                   # Learning Rate Discriminatore
d_opt = optimizer_adam(lr = d_lr)               # Funzione Ottimizzazione Discriminatore
d_dropout = 0.15                                # Percentuale Dropout Discriminatore
d_maxepoch = 80                                 # Epoche Massime Discriminatore
d_batch_size = 256                              # Dimensione Batch Discriminatore
d_validation_split = 0.15                       # Percentuale Validation Split Discriminatore

h_layer_out = 256                               # Dimensione Output Strati Nascosti

a_lr = 0.00015                                  # Learning Rate Modello Avversariale
a_opt = optimizer_adam(lr = a_lr)               # Funzione Ottimizzazione Modello Avversariale
a_maxsteps = 2500                               # Numero Massimo Step Apprendimento Modello Avversariale
a_batch_size = 512                              # Dimensione Batch Modello Avversariale

i_layer_activation = "relu"                     # Funzione Attivazione Input -> Hidden
h_layer_activation = "relu"                     # Funzione Attivazione Hidden -> Hidden
o_layer_activation = "softmax"                  # Funzione Attivazione Hidden -> Output
```

```
# -------------------- COSTRUZIONE DATASETs
ds_full <- read.csv(file = "Datasets/ds_full_71.csv", header = TRUE, sep = ";", fileEncoding = "UTF-8-BOM")

#Standardizzazione
temp <- NULL
#temp <- scale(ds_full, center = TRUE, scale = FALSE)
temp <- apply(ds_full, MARGIN = 2, FUN = function(x) (x-min(x))/(max(x)-min(x)))
temp[is.nan(temp)] <- 0
ds_full_std <- as.matrix(temp)

ds_m_train <- ds_full_std[1:1900, ]
ds_m_test <- ds_full_std[1901:1966, ]
ds_b_full <- ds_full_std[1967:nrow(ds_full_std), ]
ds_b_train <- temp[sample(nrow(temp), 1900), ]
ds_b_test <- temp[sample(nrow(temp), 66), ]
ds_b_add <- temp[sample(nrow(temp), 1900), ]

ds_bm_training <- rbind(ds_bm_training, ds_b_train)
ds_bm_training <- rbind(ds_bm_training, ds_m_train)
ds_bm_training <- ds_bm_training[sample(nrow(ds_bm_training)), ]

ds_bm_testing <- rbind(ds_bm_testing, ds_b_test)
ds_bm_testing <- rbind(ds_bm_testing, ds_m_test)
ds_bm_testing <- ds_bm_testing[sample(nrow(ds_bm_testing)), ]

lbl_bm_training <- ds_bm_training[, LABELS_COL]
ds_bm_training <- ds_bm_training[, 1:N_FEATURES]

lbl_bm_testing <- ds_bm_testing[, LABELS_COL]
ds_bm_testing <- ds_bm_testing[, 1:N_FEATURES]



# -------------------- GENERATORE
gen <- keras_model_sequential()
gen %>%
  layer_dense(units = g_out*4, activation = i_layer_activation, input_shape = g_in) %>%
  layer_batch_normalization(momentum = g_momentum) %>%
  layer_dropout(rate = g_dropout) %>%
  layer_dense(units = g_out*2, activation = h_layer_activation) %>%
  layer_batch_normalization(momentum = g_momentum) %>%
  layer_dropout(rate = g_dropout/2.0) %>%
  layer_dense(units = g_out, activation = o_layer_activation)
#summary(gen)

# -------------------- DISCRIMINATORE BM
dbm <- keras_model_sequential()
dbm %>%
  layer_dense(units = h_layer_out, activation = i_layer_activation, input_shape = d_in) %>%
  layer_dropout(rate = d_dropout) %>%
  layer_dense(units = h_layer_out/2, activation = h_layer_activation) %>%
  layer_dropout(rate = d_dropout/2.0) %>%
  layer_dense(units = d_out, activation = o_layer_activation)
#summary(dbm)
```

```
# -------------------- DISCRIMINATORE RF
drf <- keras_model_sequential()
drf %>%
  layer_dense(units = h_layer_out, activation = i_layer_activation, input_shape = d_in) %>%
  layer_dropout(rate = d_dropout) %>%
  layer_dense(units = h_layer_out/2, activation = h_layer_activation) %>%
  layer_dropout(rate = d_dropout/2.0) %>%
  layer_dense(units = d_out, activation = o_layer_activation)
#summary(drf)

# -------------------- MODELLO AVVERSARIALE
adm <- keras_model_sequential()
adm %>%
  gen %>%
  drf
#summary(adm)

# -------------------- COMPILAZIONE MODELLI
gen %>% compile(
  optimizer = g_opt,
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
dbm %>% compile(
  optimizer = d_opt,
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
drf %>% compile(
  optimizer = d_opt,
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
adm %>% compile(
  optimizer = a_opt,
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

# Backup pesi sinaptici discriminatore
dbm_weights_bkp <- dbm$get_weights()


# -------------------- ADDESTRAMENTO
# Addestro il Discriminatore BM con Dataset BM Training (senza traffico maligno generato)
history <- dbm %>% fit(
  ds_bm_training,
  lbl_bm_training,
  epochs = d_maxepoch,
  verbose = 2,
  batch_size = d_batch_size,
  shuffle = TRUE,
  validation_split = d_validation_split
)
plot(history)
```

```
# Valutazione del Discriminatore BM con Dataset BM Test (senza traffico maligno generato)
dbm %>% evaluate(
  ds_bm_testing,
  lbl_bm_testing,
  batch_size = d_batch_size/8,
  verbose = 2
)

# Addestro il Modello Avversariale con Dataset M Training
history_epochs <- NULL
history_loss <- NULL
history_acc <- NULL
for(i in 1:a_maxsteps) {
  # DRF
  g_noise <- NULL
  for(k in 1:(a_batch_size/2)) {
    rand_val <- rnorm(n = g_in, mean = 0, sd = 1)
    rand_val <- t(rand_val)
    g_noise <- rbind(g_noise, rand_val)
  }
  fake_data <- gen %>% predict(g_noise)
  train_batch <- as.matrix(ds_m_train[sample(nrow(ds_m_train), a_batch_size/2), 1:N_FEATURES])
  dx_data <- train_batch
  dx_data <- rbind(dx_data, fake_data)
  dy_data <- matrix(data = REAL, nrow = a_batch_size/2, ncol = 1)
  dy_data <- rbind(dy_data, matrix(data = FAKE, nrow = a_batch_size/2, ncol = 1))
  d_loss <- drf %>% train_on_batch(x = dx_data, y = dy_data)

  # ADM
  g_noise <- NULL
  for(k in 1:a_batch_size) {
    rand_val <- rnorm(n = g_in, mean = 0, sd = 1)
    rand_val <- t(rand_val)
    g_noise <- rbind(g_noise, rand_val)
  }
  ay_data <- matrix(data = REAL, nrow = a_batch_size, ncol = 1)
  a_loss <- adm %>% train_on_batch(x = g_noise, y = ay_data)

  if((i %% 25) == 0) {
    history_epochs <- rbind(history_epochs, i)
    history_loss <- rbind(history_loss, a_loss[1])
    history_acc <- rbind(history_acc, a_loss[2])
    loss_val <- round(as.numeric(a_loss[1]), digits = 6)
    acc_val <- round(as.numeric(a_loss[2]), digits = 6)
    loss_str <- str_pad(loss_val, width = 10, pad = " ")
    acc_str <- str_pad(acc_val, width = 10, pad = " ")
    perc <- round(((i/a_maxsteps)*100), digits = 2)
    print(paste("ADV ", ": [ loss: ", loss_str, ", acc: ", acc_str," ] ", perc, "%"))
  }
}
plot(x = history_epochs, y = history_acc, type = "b", pch = 19, col = "red", xlab = "epochs", ylab = "acc")
```

```r
# ---------- Grafici PRIMA dell'addestramento con Augmented Set ---------- #
# ---------- ROC ---------- #
# Rete Discriminatoria BM
ds_bm_full <- ds_bm_training
ds_bm_full <- cbind(ds_bm_full, lbl_bm_training)
ds_bm_full_valid <- as.matrix(ds_bm_full[sample(nrow(ds_bm_full), size = 100),])
ds_bm_valid <- ds_bm_full_valid[, 1:N_FEATURES]
lbl_bm_valid <- ds_bm_full_valid[, LABELS_COL]

predictions_bm <- as.vector(dbm %>% predict_classes(ds_bm_valid))
predictions_bm <- prediction(predictions_bm, lbl_bm_valid)
performances_bm <- performance(prediction.obj = predictions_bm, measure = "tpr", x.measure = "fpr")
plot(performances_bm)

# ---------- Confusion Matrix ---------- #
# Rete Discriminatoria BM
slotpred_bm <- unlist(slot(predictions_bm, name = "predictions"), use.names = FALSE)
cm_bm <- confusionMatrix(table(slotpred_bm, lbl_bm_valid, dnn = c("Predictions", "Reference")))
fourfoldplot(cm_bm$table, color = c("#C55555", "#5B915B"), conf.level = 0, main = "D-BM", margin = 1)



# Costruisco un Dataset BM Aumentato
g_noise <- NULL
num_m_samples <- nrow(ds_m_train)
for(k in 1:(num_m_samples)) {
  rand_val <- rnorm(n = g_in, mean = 0, sd = 1)
  rand_val <- t(rand_val)
  g_noise <- rbind(g_noise, rand_val)
}
generated_m_samples <- gen %>% predict(g_noise)
generated_m_samples <- cbind(generated_m_samples, matrix(data = MALIGN, nrow = num_m_samples, ncol = 1))
ds_bm_aug <- rbind(ds_bm_aug, ds_b_train)
ds_bm_aug <- rbind(ds_bm_aug, ds_b_add)
ds_bm_aug <- rbind(ds_bm_aug, ds_m_train)
ds_bm_aug <- rbind(ds_bm_aug, generated_m_samples)

# Shuffle
ds_bm_aug <- ds_bm_aug[sample(nrow(ds_bm_aug)), ]

# Isolo etichette dai dati
lbl_bm_aug <- as.matrix(ds_bm_aug[, LABELS_COL])
ds_bm_aug <- as.matrix(ds_bm_aug[, 1:N_FEATURES])
```

```
# Addestro il discriminatore su Dataset BM Aumentato
dbm$set_weights(dbm_weights_bkp)
history <- dbm %>% fit(
  ds_bm_aug,
  lbl_bm_aug,
  epochs = d_maxepoch,
  verbose = 2,
  batch_size = d_batch_size,
  shuffle = TRUE,
  validation_split = d_validation_split
)
plot(history)

# Valutazione discriminatore su Dataset BM Aumentato dopo il re-training
dbm %>% evaluate(
  ds_bm_testing,
  lbl_bm_testing,
  batch_size = d_batch_size/8,
  verbose = 2
)



# ---------- Grafici DOPO l'addestramento con Augmented Set ---------- #
# ---------- ROC ---------- #
# Rete Discriminatoria BM
ds_bm_full <- ds_bm_aug
ds_bm_full <- cbind(ds_bm_full, lbl_bm_aug)
ds_bm_full_valid <- as.matrix(ds_bm_full[sample(nrow(ds_bm_full), size = 100),])
ds_bm_valid <- ds_bm_full_valid[, 1:N_FEATURES]
lbl_bm_valid <- ds_bm_full_valid[, LABELS_COL]

predictions_bm <- as.vector(dbm %>% predict_classes(ds_bm_valid))
predictions_bm <- prediction(predictions_bm, lbl_bm_valid)
performances_bm <- performance(prediction.obj = predictions_bm, measure = "tpr", x.measure = "fpr")
plot(performances_bm)

# ---------- Confusion Matrix ---------- #
# Rete Discriminatoria BM
slotpred_bm <- unlist(slot(predictions_bm, name = "predictions"), use.names = FALSE)
cm_bm <- confusionMatrix(table(slotpred_bm, lbl_bm_valid, dnn = c("Predictions", "Reference")))
fourfoldplot(cm_bm$table, color = c("#C55555", "#5B915B"), conf.level = 0, main = "D-BM", margin = 1)
```

BIBLIOGRAFIA

[1]
https://italiancoders.it/deep-learning-svelato-ecco-come-funzionano-le-reti-neurali-artificiali/

[2] http://www.intelligenzaartificiale.it/reti-neurali/

[3] Donald O. Hebb, “The organization of behavior; a neuropsychological theory”. Wiley, New York, 1949

[4] Ian Goodfellow, "Generative Adversarial Nets", 2014

[5]
https://www.matematicamente.it/approfondimenti/problem-solving/giochi-non-a-somma-zero-e-non-cooperativi-john-nash/

[6] Radford A., Metz L., Chintala S. - "Unsupervised representation learning with deep convolutional generative adversarial networks", 2015

[7]
https://www.spindox.it/it/blog/reti-neurali-convoluzionali-il-deep-learning-ispirato-alla-corteccia-visiva/

[8] C. Baur, S. Albarqouni, N. Navab - "Generating Highly Realistic Images of Skin Lesions with GANs", 2018

[9] Y. Bengio, A. Courville, P. Vincent - "Representation Learning: A Review and New Perspectives", 2014

[10] P. Bontrager, A. Roy, J. Togelius, N. Memon, A. Ross - "DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution", 2017

[11] U. Fiore, A. De Santis, F. Perla, P. Zanetti, F. Palmieri - "Using generative adversarial networks for improving classification effectiveness in credit card fraud detection", 2017

[12] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, V. R. Chandrasekhar - "Efficient GAN-Based Anomaly Detection", 2018

[13] S. Ioffe, C. Szegedy - "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2015

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov - "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", 2014

[15] X. Glorot, A. Bordes, Y.Bengio - "Deep Sparse Rectifier Neural Networks", 2011

[16] Stehman, Stephen V. (1997). "Selecting and interpreting measures of thematic classification accuracy". Remote Sensing of Environment. 62 (1): 77–89. doi:10.1016/S0034-4257(97)00083-7

[17] Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation" (PDF). Journal of Machine Learning Technologies. 2 (1): 37–63

[18] http://deeplearning.net/software_links/

[19] https://www.unb.ca/cic/datasets/ids.html

[20] https://github.com/ISCX/CICFlowMeter