UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento di Scienze
Matematiche, Informatiche e Fisiche

TESI DI LAUREA IN
INTERNET OF THINGS, BIG DATA E WEB

Classificazione di utenti web mediante ispezione superficiale dei pacchetti e tecniche di Deep Learning

CANDIDATO
Mattia D'Urso

RELATORE
Prof. Claudio Piciarelli

CORRELATORE
Prof. Marino Miculan

Anno accademico 2019-2020

Contatti dell'Istituto
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

© 2020 Mattia D'Urso
This work is shared under the Creative Commons 4.0 License Attribution-NonCommercial-ShareAlike.

A mio nonno Danilo,
per avermi sempre spronato a dare di più.

Abstract

In questa tesi si affronta il problema della classificazione di utenti web mediante un'ispezione superficiale di pacchetti TCP/IP. Si è utilizzato un dataset, formato da circa 6 milioni di pacchetti, comprendente il traffico generato da circa 150 utenti. Per eseguire l'analisi si è deciso di utilizzare Python come linguaggio e la libreria Pytorch la quale implementa tecniche di Deep Learning per la classificazione, in particolare si è partiti da una rete e da un dataset per poi ottimizzare i risultati. Attraverso i numerosi test svolti si è riusciti a raggiungere una precisione del 70%.

Indice

1

Introduzione

Oggigiorno la crittografia è in grado di codificare milioni e milioni di bit in pochi istanti per mezzo di algoritmi e chiavi in continua evoluzione rendendo così impossibile decifrare, nel breve periodo, la maggior parte delle informazioni. Il lavoro svolto in questa tesi parte da una semplice domanda basata sulla consapevolezza che non sia possibile accedere al contenuto dei singoli pacchetti.

È possibile classificare degli utenti web con una sola ispezione superficiale dei loro pacchetti IP, senza aver accesso al contenuto? [4]

Ciò permetterebbe di risalire al profilo degli utenti e di conseguenza capire chi nello specifico genera un determinato traffico. Per fare ciò è necessario recupera alcune informazioni. Queste informazioni sono presenti in ogni pacchetto ed è fondamentale che rimangano in chiaro per il corretto funzionamento della rete poiché comunicano all'infrastruttura di rete i dati necessari per essere consegnati. Un'ispezione superficiale consiste nella raccolta di queste informazioni.

Per risolvere il problema è necessario analizzare più pacchetti possibili cosi da riconoscere dei pattern di utilizzo della rete. Come si vedrà in seguito ogni pacchetto ha molte informazioni in chiaro, quelle considerate in questa tesi sono:

- Timestamp
- Indirizzo MAC del mittente
- Indirizzo IP di destinazione
- Dimensione del pacchetto
- Porta del mittente
- Porta del destinatario

Per effettuare l'analisi si è scelto di usare un modello di intelligenza artificiale cosi da ricercare i pattern in maniera efficiente.

Nel capitolo 2 si introducono al lettore alcune nozioni generali sul funzionamento delle reti di calcolatori fino ad arrivare ai protocolli che le regolano e alla spiegazione di cosa sono i pacchetti TCP/IP. Inoltre vengono fornite anche una breve introduzione al machine learning e una rappresentazione matematica della tecnica usata per risolvere il problema di questa tesi.

Nel capitolo 3 si affrontano i dettagli del problema, si parlerà del dataset e del modello da cui si è partiti, degli imprevisti incontrati e dei risultati iniziali.

Nel capitolo 4 si affronta la risoluzione degli imprevisti e l'ottimizzazione del modello spiegando, per ogni iper-parametro in cosa impatta e quale importanza ha.

Nel capitolo 5, infine, si traggono le conclusioni del lavoro svolto e si danno degli spunti per migliorare i risultati ottenuti.

2

Reti di Calcolatori e Machine Learning

In questo capitolo si vuole introdurre il lettore ad alcune nozioni sul funzionamento delle reti e ad alcuni termini e meccanismi del machine learning, inoltre si illustrerà com'è composto un pacchetto TCP/IP cosa sono significa eseguire lo *sniffing*, quindi si darà una descrizione matematica di cosa succede all'interno di un modello deep learning.

2.1 Breve introduzione alle Reti di Calcolatori

Le reti di computer sono un sistema di computer interconnessi allo scopo di condividere informazioni digitali.[7] Dal vecchio modello basato sulla presenza di un solo potente elaboratore (mainframe) e terminali collegati si è oggi passati a modelli che prevedono la presenza di un numero sempre maggiore di elaboratori autonomi e interconnessi. La diffusione di questi sistemi ha diverse utilità, come permettere di condividere risorse, mettere in comunicazione persone, raccogliere informazioni in tempo reale, ecc.

2.1.1 Breve introduzione al web

Uno dei principali servizi offerti dalla rete mondiale di calcolatori, detta *Internet*, è il World Wide Web. Il web nasce nel 6 agosto 1991[24], data in cui Tim Berners-Lee pubblicò il primo sito web. L'idea alla base era quella di riuscire a trasmettere informazioni velocemente all'interno del CERN. Negli anni successivi il web inizia a diffondersi fino arrivare a quella che viene definita l'era del web (2000-2010). La maggior parte delle persone conosce Internet attraverso le sue applicazioni: siti web, e-mail e contenuti in streaming. Gli utenti visualizzano pagine piene di oggetti testuali e grafici, fanno *clic* sugli oggetti di cui vogliono saperne di più e viene visualizzata una nuova pagina corrispondente.

2.1.2 Il modello ISO/OSI

Il modello ISO/OSI è la rappresentazione teorica di come funziona una rete di calcolatori. Le informazioni vengono trasmesse sotto forma di pacchetti e viaggiano attraverso l'infrastruttura di rete dal nostro computer al server e viceversa. Ad ogni livello del modello ISO/OSI le informazioni vengono incapsulate e preparate per il layer sottostante fino ad arrivare al livello fisico che immette l'impulso elettrico nella rete e questo viaggia fino a destinazione per poi percorre il percorso a ritroso e visualizzare una risposta

sul monitor. I pacchetti possono essere di vario tipo TCP / IP, UDP / IP, ARP, SNMP, RIP etc. Si assume che la maggior parte delle azioni dell'utente generino del traffico TCP / IP.

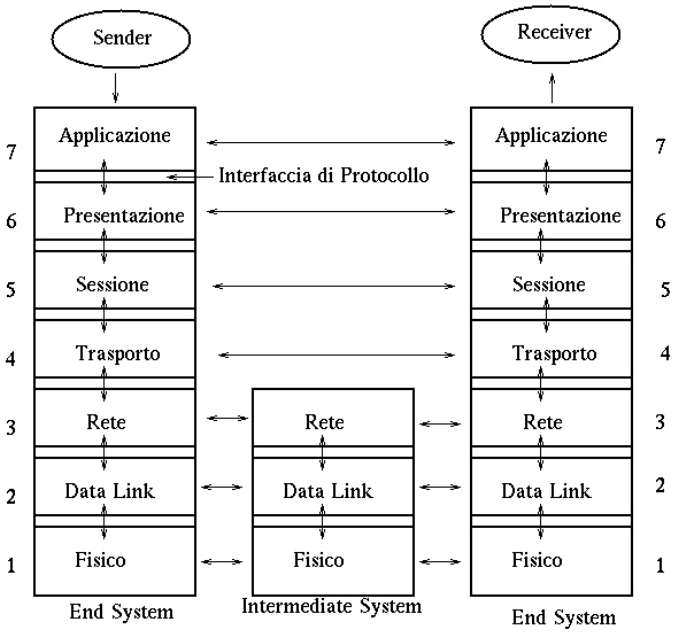


Figura 2.1: Rappresentazione del modello ISO/OSI

La figura 2.1 è una rappresentazione del modello ISO/OSI in cui si vedono i sette livelli che lo compongono e uno stack intermedio quale potrebbe essere un router che inoltra i pacchetti all'interno della rete. Ogni livello comunica con un livello nella medesima posizione, cosi da non doversi preoccupare di cosa succede nei livelli sottostanti. In contrapposizione al modello teorico ISO/OSI c'è la sua applicazione: lo stack TCPI/IP. Questo è il modello realmente usato su internet.

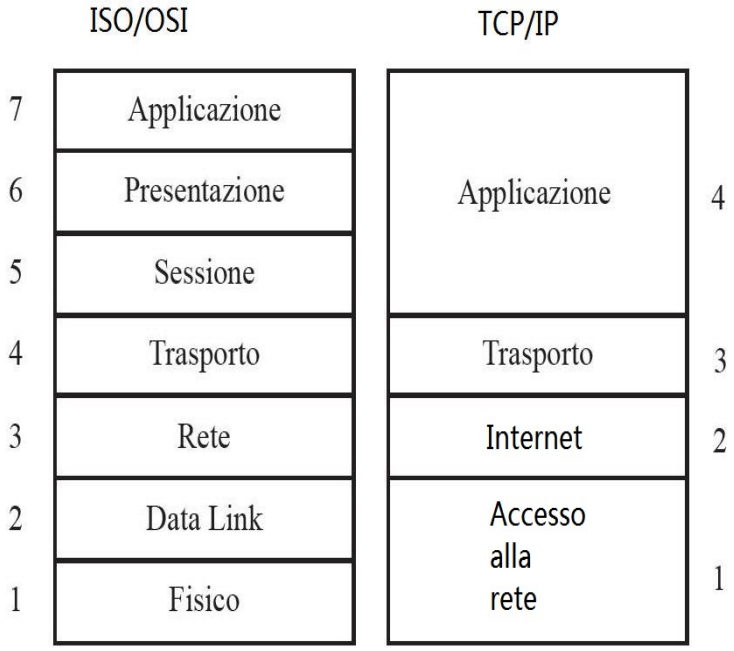


Figura 2.2: Rappresentazione dello stack TCP/IP in confronto con il modello ISO/OSI

2.1.3 I protocolli di rete

Tutte queste informazioni sono regolate da dei protocolli che ne determinano la direzione, la sicurezza e l'integrità. I protocolli più importanti tra questi sono:

- Transmission Control Protocol o *TCP*
  si trova al livello 4 del modello ISO/OSI e 3 dello stack TCP/IP, o di trasporto, e ha il compito di controllare la trasmissione, ovvero di garantire la consegna in sicurezza e l'integrità dei pacchetti.

- User Datagram Protocol o *UDP*
  si trova al livello 4 del modello ISO/OSI e 3 dello stack TCP/IP, o di trasporto, e si occupa dell'invio dei pacchetti tramite un servizio *best effort*, quindi senza offrire garanzie sulla consegna.

- Internet Protocol o *IP*
  si trova al livello 3 del modello ISO/OSI e 2 dello stack TCP/IP, o di rete, ed è il protocollo chiave per costruire reti interconnesse scalabili ed eterogenee. Viene eseguito su tutti i nodi della rete e definisce l'infrastruttura che permette alle reti di funzionare come se fosse una singola rete interconnessa.

Questi tre protocolli sono la base su cui poggia una qualsiasi rete di calcolatori.

2.1.4 Pacchetto TCP/IP

I dati trattati in questa tesi sono dati ricavati da dei pacchetti TCP/IP. Questi pacchetti sono fondamentali nel processo di scambio di dati poiché il protocollo che li regola garantisce il loro arrivo. Il peso di un pacchetto TCP/IP è determinato dai primi 20 byte, che sono normalmente occupati dallo header IP, dai successivi 20 byte, occupati dallo header del protocollo TCP, e infine da 1460 byte, appartenenti al payload (o carico utile), per un peso di 1500 byte massimi.[6]

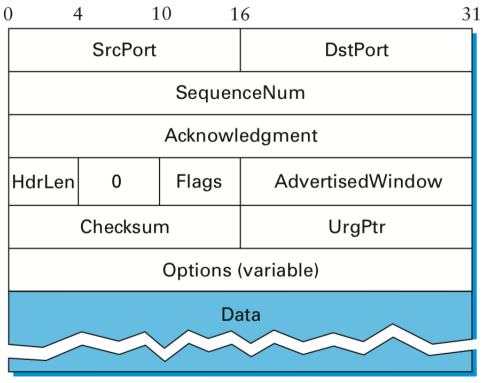


Figura 2.3: Header di un pacchetto TCP/IP

Un pacchetto TCP/IP è composto dai seguenti campi[22]:

- **Source port** [16 bit] - Identifica il numero di porta sull'host mittente associato alla connessione TCP.

- **Destination port** [16 bit] - Identifica il numero di porta sull'host destinatario associato alla connessione TCP.

- **Sequence number** [32 bit] - Numero di sequenza, indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo, a partire dall'Initial Sequence Number (ISN), deciso all'apertura della connessione.
- **Acknowledgment number** [32 bit] - Numero di riscontro, ha significato solo se il flag ACK è impostato a 1, e conferma la ricezione di una parte del flusso di dati nella direzione opposta, indicando il valore del prossimo Sequence number che l'host mittente del segmento TCP si aspetta di ricevere.
- **Header length** [4 bit] - Indica la lunghezza dell'header del segmento TCP; tale lunghezza può variare da 5 dword[20] (20 byte) a 15 dword (60 byte) a seconda della presenza e della lunghezza del campo facoltativo Options.
- **Reserved** [4 bit] - Bit non utilizzati e predisposti per sviluppi futuri del protocollo, dovrebbero essere impostati a zero.
- **Flags** [8 bit] - Bit utilizzati per il controllo del protocollo.
- **Advertise window** [16 bit] - Indica la dimensione della finestra di ricezione dell'host mittente, cioè il numero di byte che il mittente è in grado di accettare a partire da quello specificato dall'acknowledgment number.
- **Checksum** [16 bit] - Campo di controllo utilizzato per la verifica della validità del segmento. È ottenuto facendo il complemento a 1 della somma a 16 bit dell'intero header TCP (con il campo checksum messo a zero), dell'intero payload, con l'aggiunta di uno pseudo header composto da: indirizzo IP sorgente (32bit), indirizzo IP destinazione (32bit), un byte di zeri, un byte che indica il protocollo e due byte che indicano la lunghezza del pacchetto TCP (header + dati).
- **Urgent pointer** [16 bit] - Puntatore per dati urgenti, ha significato solo se il flag URG è impostato a 1 ed indica lo scostamento in byte a partire dal Sequence number del byte dei dati urgenti all'interno del flusso.
- **Options** - Opzioni (facoltative) per usi del protocollo avanzati.
- **Data** - Rappresenta il carico utile o payload da trasmettere.

2.1.5 Sniffing

Con *sniffing*, in informatica e nelle telecomunicazioni, si definisce l'attività di intercettazione passiva dei dati che transitano in una rete telematica.[21] Tramite la tecnica dello sniffing si può praticare un'ispezione superficiale dei pacchetti così da recuperare le informazioni in chiaro necessarie viste in precedenza.

I software utilizzati per condurre l'intercettazione sono chiamati analizzatori di rete o sniffer. Si tratta di prodotti software o hardware specifici per questo genere di attività. Essi sono in grado di memorizzare i dati ottenuti e di gestire le configurazioni delle interfacce di rete. Gli sniffer spesso integrano anche funzionalità di gestione dei dati basata sui protocolli dei vari livelli dello stack come HTTP, TCP, IP, 802.11. Lo sniffer usato in questa tesi è Wireshark[23], un software open source che può essere eseguito da tutti i principali sistemi operativi desktop.

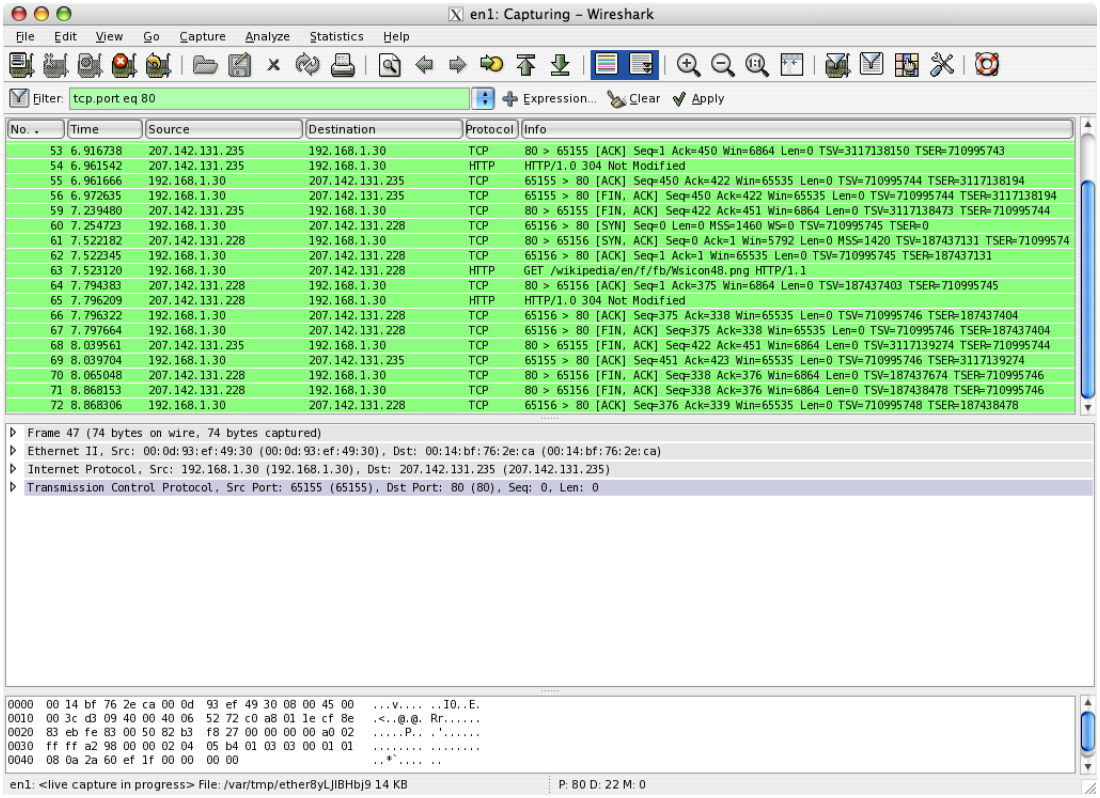


Figura 2.4: Screenshot di una schermata di Wireshark

2.2 Breve introduzione al Machine Learning

Con il termine "Machine Learning" si intende un software con cui i computer imparano a fare azioni ed attività basandosi sull'esperienza. In sostanza, gli algoritmi di Machine Learning, usano metodi matematico-computazionali per apprendere informazioni direttamente dai dati, senza modelli matematici ed equazioni predeterminate, ma creandosi le "regole" per lo svolgimento del compito da soli. Gli algoritmi di Machine Learning migliorano le loro prestazioni mano a mano che gli "esempi", ovvero il *dataset*, e la qualità di essi aumentano.

A forgiare il termine fu Arthur Lee Samuel nel 1959, scienziato americano pioniere nel campo dell'Intelligenza Artificiale, anche se, la definizione più accreditata dalla comunità scientifica, è quella fornita da un altro americano, Tom M. Mitchell, attualmente direttore del dipartimento Machine Learning della Carnegie Mellon University:

> "*Si dice che un programma apprende dall'esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E.*"[17]

2.2.1 Supervised, Unsupervised e Reinforcement Learning

Con la tecnica del supervised learning il programmatore possiede sia il dataset per allenare il modello (*input*) sia i rispettivi risultati (*output*).[13] I problemi che ricadono nel supervised learning sono i problemi di regressione, il cui scopo è formulare una predizione in un insieme potenzialmente infinito, e i problemi di classificazione, in cui si conosce a priori l'insieme finito delle possibili predizioni. Il problema che viene affrontato in questa tesi è di tipo supervised learning, in particolare di classificazione. Nel dataset viene fornito anche il *label* (il risultato), ossia il MAC del mittente, cosi da allenare il modello a riconoscerne le abitudini.

Nel unsupervised learning si affrontano problemi pur non sapendo il risultato.[13] Si possono ricavare delle predizioni basandosi sui dati in continuo input, nonostante non ci sia una predizione corretta. I problemi che affronta il unsupervised learning si dividono in raggruppamento (detto anche *clustering*), si utilizza quando è necessario raggruppare i dati con caratteristiche simili, e associazione, in cui si ricercano regole che descrivono relazioni nei big data (quantità enormi di dati), come la relazione che intercorre tra l'acquisto di un bene X e l'acquisto di un bene Y.

Il reinforcement learning punta a sviluppare modelli in grado di scegliere quale azione compiere per il realizzare determinati obiettivi basandosi sull'interazione con l'ambiente circostante.[15] A differenza delle tecniche viste sopra, questa tecnica si occupa di risolvere problemi di decisioni sequenziali, in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro. La qualità di un'azione è data da un valore numerico di "ricompensa', che ha lo scopo di incoraggiare comportamenti corretti del sistema. Questo tipo di apprendimento è solitamente modellato con l'ausilio tre funzioni che si occupano di scegliere gli output sulla base degli input ricevuti, valutare l'efficacia degli output rispetto ad un preciso parametro di riferimento e modificare la prima funzione per massimizzare la valutazione di efficacia effettuata dalla seconda funzione.

2.2.2 Deep Learning

In questa tesi si è scelto di usare una categoria particolare di algoritmi predittivi: le reti deep learning. Queste reti, sono dei modelli ispirati alla struttura e alla funzione del cervello.[11] Essi, di fatto, si presentano come un sistema "adattivo" in grado di modificare la loro struttura (i nodi e le interconnessioni) basandosi sia sul dataset sia su informazioni interne che passano attraverso la rete neurale durante la fase di *training*. L'unità di base di queste reti è il neurone, spesso chiamato anche nodo o unità. Queste reti sono formate da dei neuroni di input e dei neuroni di output, nel mezzo ci sono gli hidden-layer. I neuroni ricevono input I da altri nodi o da una sorgente esterna e calcolano un output O. I vantaggi di questa architettura si riassumono nella capacità di estrapolare dei pattern dai big data, operazioni troppo complesse per l'uomo. Questi software vengono impiegati per il riconoscimento automatico di immagini, audio, testo, ecc.
Si è scelto di usare quest'ultimo tipo di modello poiché è il più adatto a classificare molti utenti ed elaborare una predizione.

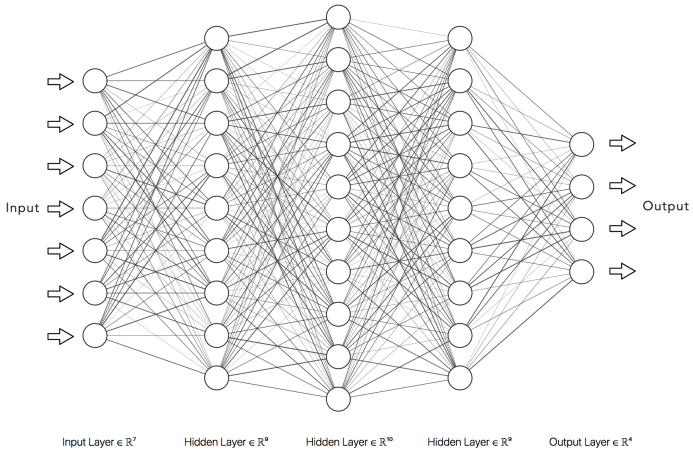


Figura 2.5: Rappresentazione di una rete neurale con 7 input, 3 hidden layer e 4 output

2.2.3 Rappresentazione interna del Supervised Learning

I modelli deep learning, come già detto, sono ispirati al cervello umano, il quale è formato da neuroni che hanno un corpo, dei dendriti e un assone. In esso il segnale di un neurone viaggia lungo l'assone e si trasferisce ai dendriti del neurone successivo. La connessione in cui passa il segnale è chiamata sinapsi. Questa è l'idea alla base di un algoritmo deep learning: ottenere degli input e darli in pasto al livello successivo che a sua volta diventa l'input per il livello successivo e così via. Questo accade ancora e ancora fino all'output finale.[1] Nei modelli deep learning ogni neurone, almeno inizialmente, è connesso a tutti i neuroni del successivo layer, un po' come succede nel cervello umano. A ciascuna delle sinapsi vengono assegnati dei pesi, che all'istante della creazione vengono inizializzati casualmente. Questi pesi sono il modo in cui la rete impara e regolandoli si decide in che misura i segnali vengono trasmessi. Allenare la rete significa ottimizzare i pesi. Ogni nodo ha una funzione di attivazione con relativo valore soglia, dei pesi assegnati ad ogni collegamento ed eventualmente un bias[18]. Quando le informazioni arrivano al neurone esso calcola il risultato della funzione di attivazione applicata sulla somma dei valori in ingresso x moltiplicati per i relativi pesi w e addizionati al bias b, ovvero

$$f(\sum_{i=1}^{n} w_i x_i + b)$$

il valore risultate, se sufficiente ad attivare il neurone, viene propagato verso i neuroni del livello successivo e cosi via fino a produrre un output.

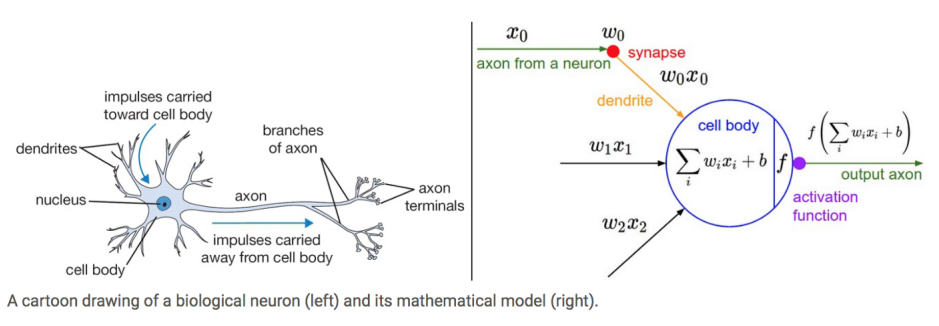


Figura 2.6: Rappresentazione di un neurone

Dopo aver ottenuto degli output, la rete si avvale di una funzione di costo per calcolare la loss confrontando l'output e l'output previsto effettivo. Il valore generato dalla funzione di costo è espresso come la differenza tra il valore effettivo e il valore previsto. Alcune delle funzioni di costo più usate nei problemi di classificazione sono[8]:

- Cross Entropy
- Categorical Cross Entropy Cost Function
- Binary Cross Entropy Cost Function
- Hinge Loss Function

A questo punto le informazioni viaggiano a ritroso nella rete con l'obiettivo di ridurre al minimo la funzione di costo modificando i pesi. Questo processo è chiamato *back propagation*, ovvero il modo in cui si calcolano i gradienti. Durante questo processo i pesi vengono regolati tutti contemporaneamente. Per fare ciò si usa un una funzione $X_n = F_n(W_n, X_{n-1})$ dove X_n è il vettore rappresentante dell'output del modulo, W_n è il vettore dei parametri ottimizzabili nel modello e X_{n-1} è il vettore di input.[25] Se la derivata parziale E^p rispetto a X_n è conosciuta allora la derivata parziale E^p rispetto a W_n e X_{n-1} può essere calcolata usando la ricorsione all'indietro:

$$\frac{\partial E^p}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1})\frac{\partial E^p}{\partial X_n}$$

$$\frac{\partial E^p}{\partial X_{n-1}} = \frac{\partial F}{\partial W}(W_n, X_{n-1})\frac{\partial E^p}{\partial X_n}$$

dove $\frac{\partial F}{\partial W}(W_n, X_{n-1})$ è la Jacobiana di F rispetto a W valutata al punto (W_n, X_{n-1}) e $\frac{\partial F}{\partial X}(W_n, X_{n-1})$ è la Jacobiana di F rispetto a X. La Jacobiana di un vettore funzione è una matrice contenente la derivata parziale di tutti gli output rispetto agli input. Quando le sopracitate equazioni sono applicate al modello, in ordine inverso, dal layer N al layer 1 tutte le derivate parziali delle funzioni costo e i relativi parametri possono essere calcolati. Le reti neurali tradizionali sono un caso particolare del relativo sistema dove dove i moduli alternano livelli di moltiplicazioni tra matrici (pesi) e di funzioni sigmoid (i nodi):

$$Y_n = W_n X_{n-1}$$
$$X_n = F(Y_n)$$

dove W_n è la matrice con i numeri delle colonne di dimensione X_{n-1} e i numeri delle righe di dimensione X_n. F è il vettore funzione che applica la sigmoid ad ogni componente del suo input. Y_n è il vettore del totale dei pesi del layer n. Applicando le regole all'equazione sopra la classica è propagation ha questa forma:

$$\frac{\partial E^p}{\partial y_n^i} = f'(y_n^i)\frac{\partial E^p}{\partial x_n^i}$$

$$\frac{\partial E^p}{\partial w_n^{ij}} = x_{n-1}^j \frac{\partial E^p}{\partial y_n^i}$$

$$\frac{\partial E^p}{\partial x^k_{n-1}} = \sum_i w^{ik}_n \frac{\partial E^p}{\partial y^i_n}$$

La procedura più semplice in tale contesto è l'algoritmo di gradient descend dove W si aggiusta iterativamente come segue:

$$W(t) = W(t-1) - \eta \frac{\partial E}{\partial W}$$

3

Il problema

In questo capitolo si illustrano i dettagli su come verrà affrontato il problema. Saranno indicati, nello specifico, il dataset e il modello di partenza e i problemi che si è reso necessario risolvere prima di iniziare l'addestramento del modello.

3.1 Il problema

Ci si è chiesti se sia possibile classificare deli utenti web con una sola ispezione superficiale dei loro pacchetti IP, senza aver accesso al payload. Per rispondere a questo quesito si è partiti da uno *sniffing* eseguito sulla rete dell'Università degli Studi di Udine, in particolare presso la sede dei Rizzi. Ciò ha permesso di intercettare circa 6 milioni di pacchetti e condensarli in un unico dataset. Da questi pacchetti si è scelto di considerare solamente alcune informazioni, in particolare:

- Timestamp
- Indirizzo MAC del mittente (*label*)
- Indirizzo IP di destinazione
- Dimensione del pacchetto
- Porta del mittente
- Porta del destinatario

Vengono inseriti nel modello 5 input, mentre l'output è uno solo, ovvero il label. Quindi si è scelto un modello *deep learning* come base da cui partire per svolgere l'analisi dei dati raccolti. Il modello è in grado di analizzare ogni singolo pacchetto e predire, con una certa precisione, a quale utente appartiene.

3.2 Nozioni utili

Come detto in precedenza si è scelto di usare un modello deep learning per effettuare l'analisi del dataset. Questa tipologia di modelli viene valutata con due metriche principali:

- Loss
  È un valore calcolato dalla loss function, o funzione di costo, e indica a quanto corrisponde l'errore medio nelle predizioni. È un valore che assume i valori $[0, \infty]$ e migliora al diminuire di tale valore.

- Accuracy
  Corrisponde al rapporto tra predizioni corrette e valori attesi. È un valore compreso $[0, 1]$ e migliora all'aumentare di tale valore.

3.3 Hardware utilizzato

I test sono stati svolti utilizzando un computer fisso con il seguente hardware:

CPU	Intel 8700K
GPU	AMD Radeon RX 580
RAM	32 GB DDR4
OS	macOS 10.13.6

I valori nei vari test, eseguiti con gli stessi parametri, potrebbero differire di qualche punto percentuale quando ripetuti poiché, all'inizio, l'ordine del dataset e i pesi dei singoli neuroni vengono randomizzati. Nonostante si sia utilizzata la libreria Pytorch, la quale utilizza CUDA per parallelizzare il calcolo, non si è usata una GPU NVIDIA, ciò ha influenzato i tempi di addestramento e validazione della rete.

3.4 Dataset di partenza

Dopo la raccolta dei dati, il dataset è stato pseudonimizzato e sono stati estratti solo i dati di interesse, infine è convertito in *.csv*. Il dataset risultante ha la seguente struttura:

No., Time, MAC source, Destination, Length, Source port, Destination port
0, 0.000000, f2:25:d6:94:70:17, 41.198.61.0, 66, 65233, 443
1, 0.000110, f4:25:c6:aa:64:12, 42.192.34.4, 66, 32456, 345
2, 0.001581, c4:42:87:b3:23:a9, 42.192.34.4, 66, 54387, 264
...
6156724, 3599.394397, f2:c1:29:73:64:b5, 192.168.0.125, 1412, 80, 50604

Questo dataset viene preparato per essere dato in input al modello associando ai campi "MAC source" e "Destination" con un id numerico progressivo e salvato nel formato *.npy*, dopodiché il dataset viene processato per sostituire il campo "Time" con la distanza, sempre in millisecondi, dal pacchetto precedente della stesso mittente e non più dal pacchetto zero. Il risultato è questo:

No., Time, MAC source, Destination, Length, Source port, Destination port
0, 0.000000, 0, 0, 66, 64763, 443
1, 1.1000e-04, 1, 1, 66, 32456, 345
2, 1.4710e-03, 2, 1, 66, 54387, 264
...
6156724, 2.4414e-04, 149, 1342, 1412, 80, 50604

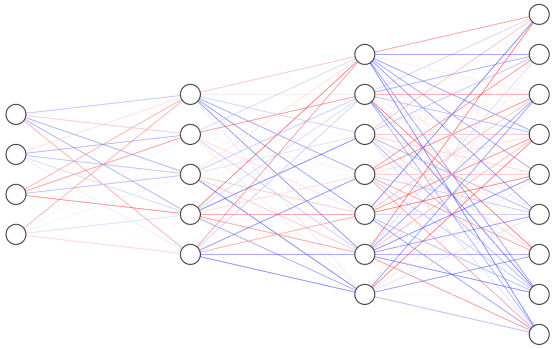
Infine, il dataset, viene suddiviso in *trainig set* e *test set*, rispettivamente l'80% e il 20% del dataset iniziale.

3.5 Modello di partenza

La rete di partenza ha restituito, in un primo momento, dei risultati incoraggianti. La rete accetta in input il timestamp e la destinazione del pacchetto, la porta del mittente e la porta del destinatario e presenta le seguenti caratteristiche:

Params	Value
Architecture	4/50/100/150
Activate function	ReLU
Learning rate	1e-2
Batch	64
Epochs	10

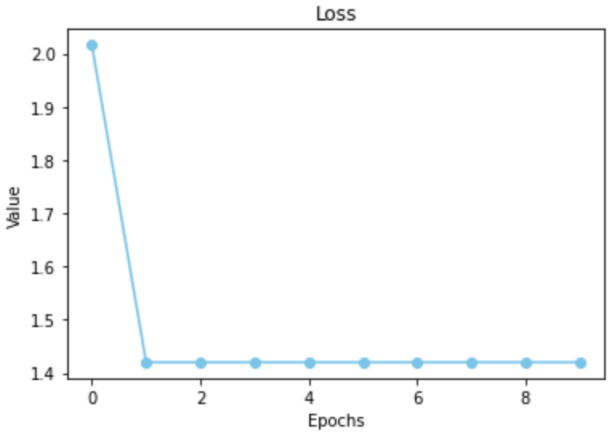
(a) Descrizione del modello di partenza



(b) Rappresentazione di una rete neurale

3.6 Risultati di partenza

Il primissimo test sul modello di partenza, allenato per 10 epoche, si è concluso con una precisione del 84.679%. Di seguito il grafico della loss con il dataset originario:




```
Accuracy test: 84.67963409423828 %
```

Figura 3.1: Loss di partenza

Il grafico evidenzia come dalla all'epoca 1 il valore inizia a ripetersi, ciò indica che la rete né migliora né peggiora, ma resta stabile. Questo fenomeno si chiama *overfit* e verrà spiegato in seguito in cosa consiste. In questo caso si è scoperto che più della metà dei pacchetti (3.549.597/6.156.726, circa il 57.7%) provengono dall'utente numero 2. Ciò comporta che la rete si abitua a dare in output sempre lo stesso un risultato, ovvero 2. A questo è dovuta la precisione cosi alta raggiunta dalla rete di partenza.

3.7 Approccio utilizzato

Si è partiti con una validazione statica del codice seguita da una validazione dinamica, in seguito si è analizzato il dataset a disposizione. Quindi ci si è resi conto che era necessaria una ristrutturazione del dataset. Dopodiché si è passati alla fase di ottimizzazione della rete testando un iper-parametro dopo l'altro fino ad ottenere il miglior risultato possibile.

4

Test svolti

Si ritiene che allenare una rete neurale sia un metodo di forza bruta poiché la rete viene inizializzata con uno stato casuale e addestrata fino ad ottenere un modello accurato con un dataset estremamente ampio.[19] I ricercatori devono fare attenzione con la progettazione del modello, la progettazione dell'algoritmo e la corrispondente selezione di iper-parametri. È sufficiente che uno solo degli iper-parametri non sia ottimizzata a dovere per peggiorare sensibilmente le performance della rete. In questo capitolo si parlerà dei test svolti per ottimizzare i parametri delle rete, spiegando quale ruolo e impatto abbia ognuno di essi. Si ricorda che si è partiti da una rete neurale con architettura a quattro layer (4/50/100/150), ReLu come funzione di attivazione e 1e-2 come learning rate.

4.1 Validation test

Si definisce *validation set* il campione di dati utilizzato per fornire una valutazione imparziale rispetto all'adattamento del modello sul training set durante l'ottimizzazione degli iper-parametri del modello, solitamente è molto più piccolo del training set.[16] Il validation set viene utilizzato per test frequenti (chiamati *validation test*), in particolare, in questa tesi, ad ogni epoca. Il modello viene ciclicamente testato con questi dati, ma non allenato. Il validation set influisce sul modello indirettamente in quanto indicatore dell'andamento dell'allenamento. Come primo passo della fase di ottimizzazione si è scelto di introdurre un validation test per monitorare al meglio l'evoluzione del modello e il grado di overfitting. Per fare ciò si è dovuto rivedere le proporzioni di divisione del dataset. Le proporzioni aggiornate sono:

- 60% training set
- 20% validation set
- 20% test set

L'overfitting è un errore di modellazione che si verifica quando il grafico delle predizioni del modello è troppo vicina a quello dei dati di training.[14] L'overfitting del modello assume generalmente la forma di un modello eccessivamente complesso per spiegare il comportamento dei dati in esame, perdendo così la capacità di generalizzare. Generalmente è causato da un dataset sbilanciato. Il fenomeno opposto è l'underfitting in cui il modello allenato ha una forma troppo semplice e quindi generalizza troppo. Di seguito un grafico esplicativo dei due fenomeni.

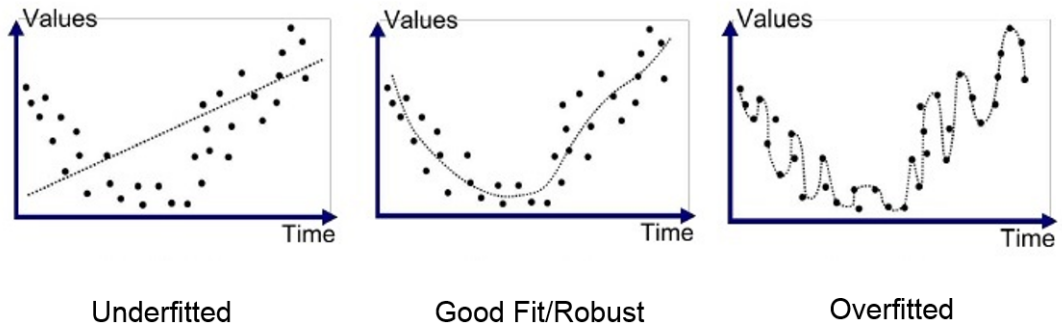


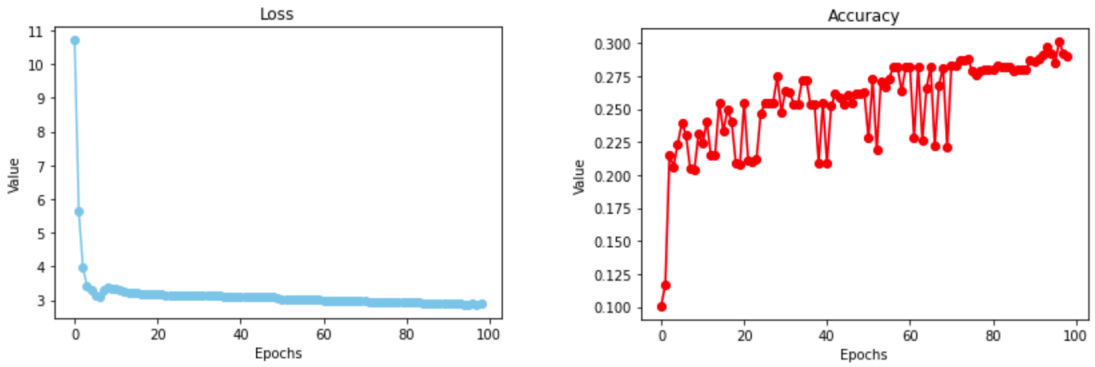
Figura 4.1: Esempio di Underfit e Overfit

4.2 Ristrutturazione del dataset

Come detto in precedenza, il dataset necessita di una ristrutturazione. Per fare ciò si è ricercato il numero di pacchetti per ogni utente, questo ha evidenziato non solo lo sbilanciamento del dataset verso l'utente 2 ma anche il fatto che ci fossero altri 3 utenti con più di 150.000 pacchetti e 58 con meno di 5.000, mentre la media dei pacchetti per utente è 40.773. Quindi si è scelto di escludere tali utenti, diminuendo il numero di pacchetti a 2.054.165 e 90 utenti, con una media di 22.824 pacchetti per utente. Dopo aver ristrutturato il dataset i risultati non sono cambiati quindi si è proceduto con una variazione di alcuni parametri, oltre che all'introduzione della destinazione tra gli input.

Params	Before	After
Architecture	4/50/100/150	5/50/100/90
Lr	1e-2	1e-4
Epochs	10	100

Ora i grafici della funzione di costo e la precisione si presentano in questo modo:



(a) Loss (b) Accuracy

La loss è pari a 2.90307, mentre la accuracy è pari a 28.93704%. Come si può vedere dal precedente grafico i risultati sono inferiori a quelli del capitolo 3 ma sicuramente più attendibili, conseguenza del dataset migliorato, e quindi un buon punto di partenza per la fase di ottimizzazione.

4.3 Ottimizzazione della rete

Dopo aver studiato a fondo il punto di partenza e ristrutturato il dataset si è proceduto con l'ottimizzazione degli iper-parametri, o *hyper-parameter tuning*, per migliorare ulteriormente il modello. Sono diversi i parametri da affinare, nelle successive sezioni si spiega in cosa impattano e si mostrano i risultati dei test svolti.

4.3.1 Architettura

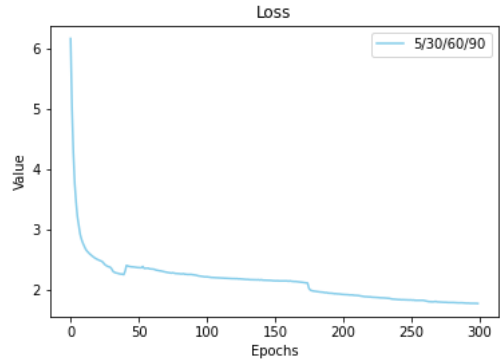
I livelli, o *layer*, sono l'architettura stessa del modello, è il parametro che definisce:

- Il numero di input che accetta il modello
- Il numero di livelli nascosti, o *hidden-layers*, e i loro neuroni
- Il numero di output, ovvero l'insieme dei valori che può assumere una predizione

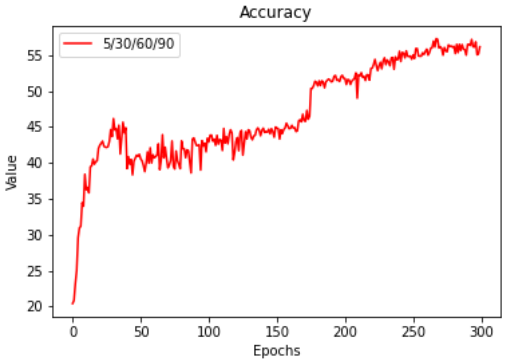
Quello che succede negli hidden-layer è oscuro al programmatore, il calcolatore esegue dei calcoli in parziale autonomia (il programmatore definisce solo le funzioni di attivazione, il numero e la struttura dei layer) per arrivare allo stato ottimale, ovvero la condizione più vicina allo stato dell'arte.

Si è partiti dal modello visto in precedenza, formato da quattro layer, e si sono svolti dei test variando la forma e il numero di essi. Si sono scelte diverse forme e non tutte hanno dato buoni risultati, di seguito sono riportare le migliori. La seconda struttura si è rivelata essere, anche se di poco, la migliore e d'ora in poi verrà usata per gli altri test. Da notare che al variare della struttura alcuni modelli tendono a migliorare più velocemente di altri per poi rallentare.

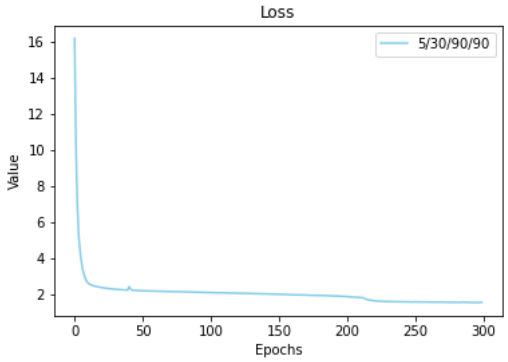
Architecture	Epochs	Mean Loss	Train Acc %	Validation Acc %
5/30/60/90	100	2.213	43.590	43.447
5/30/60/90	200	1.924	54.026	51.779
5/30/60/90	300	1.771	56.219	56.169
5/30/90/90	100	2.127	48.389	48.185
5/30/90/90	200	1.907	51.153	51.112
5/30/90/90	**300**	**1.590**	**61.175**	**61.156**
5/60/120/90	100	2.269	46.200	46.088
5/60/120/90	200	2.143	49.132	48.980
5/60/120/90	300	1.991	47.555	47.349
5/60/150/90	100	2.100	49.712	49.639
5/60/150/90	200	1.892	53.573	53.504
5/60/150/90	300	1.729	59.700	59.608



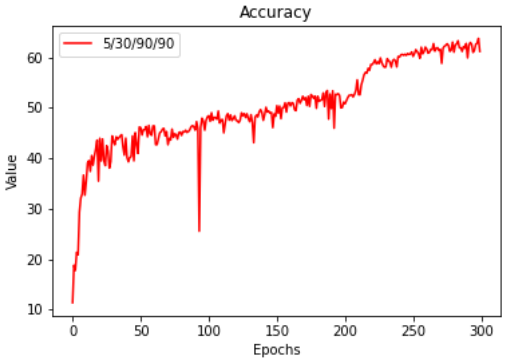
(a) Loss per 5/30/60/90



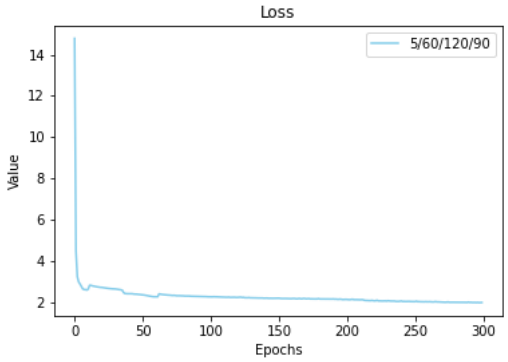
(b) Accuracy per 5/30/60/90



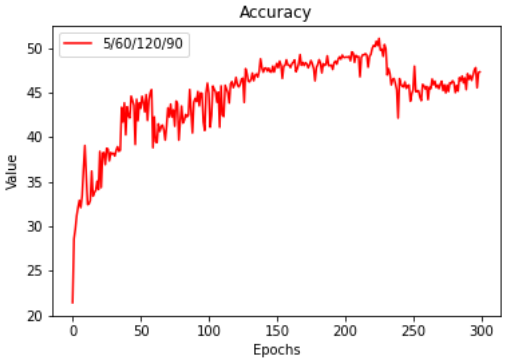
(c) Loss per 5/30/90/90



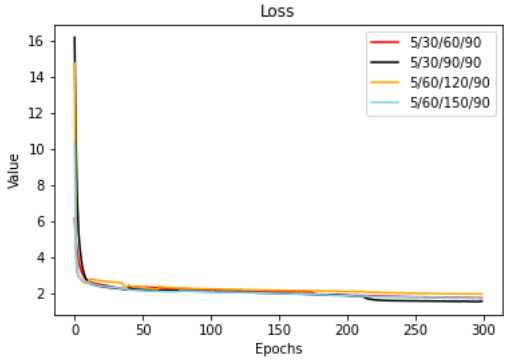
(d) Accuracy per 5/30/90/90



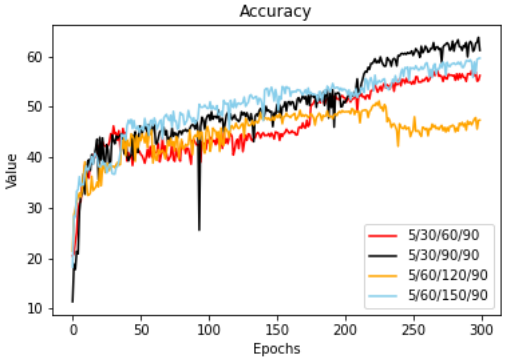
(e) Loss per 5/60/120/90



(f) Accuracy per 5/60/120/90



(g) Confronto Loss

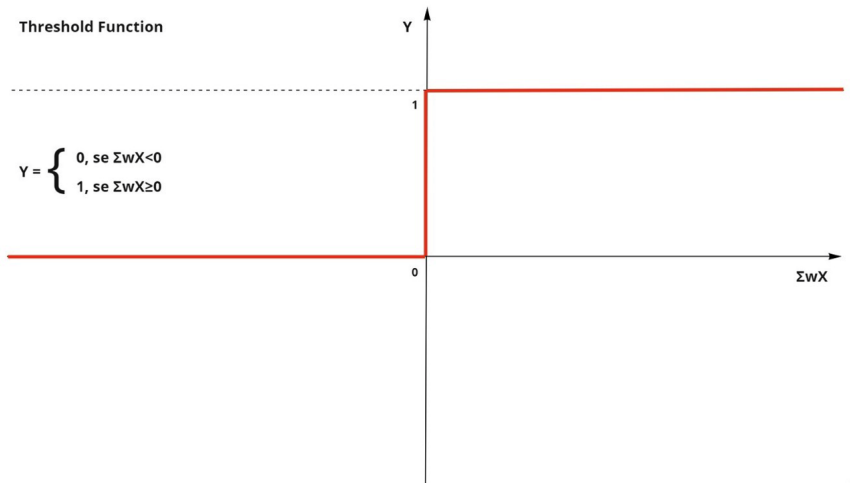


(h) Confronto Accuracy

4.3.2 Funzioni di attivazione

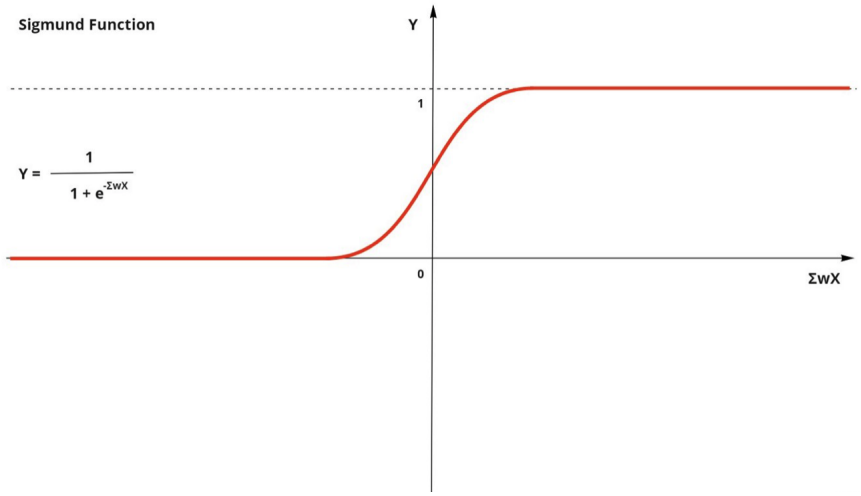
La funzione di attivazione f, o *activation function*, è determinante per il valore in uscita da ogni neurone. Al cambiare della funzione varia il codominio e di conseguenza l'insieme degli output. Ad ogni applicazione corrisponde una funzione di attivazione ottimale. Di seguito le funzioni più utilizzate[10] e i test condotti su esse.

- La funzione "soglia" (Threshold Function)



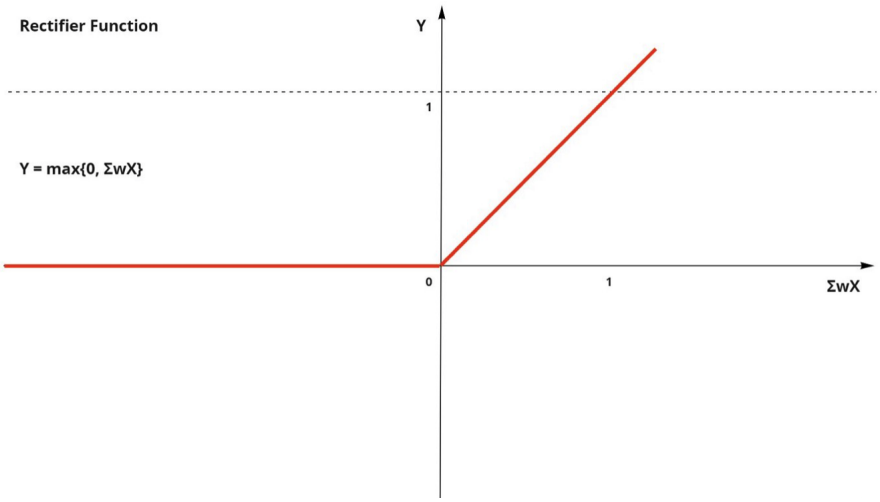
Il suo funzionamento è molto semplice, la funzione restituisce 1 nel caso in cui la somma pesata dei segnali in input è maggiore o uguale a zero, e 0 negli altri casi. Il suo utilizzo è molto utile qualora si avesse bisogno di un segnale in output binario.

- Funzione di Sigmoid (Sigmoid Function)



È una funzione monotona crescente asintotica che può essere utilizzata al posto della Threshold Function considerando il valore in uscita come probabilità che O sia uguale ad uno, ovvero Prob(O=1). In questo caso il codominio è C[0,1].

- Funzione di Rettificazione Lineare (ReLU)

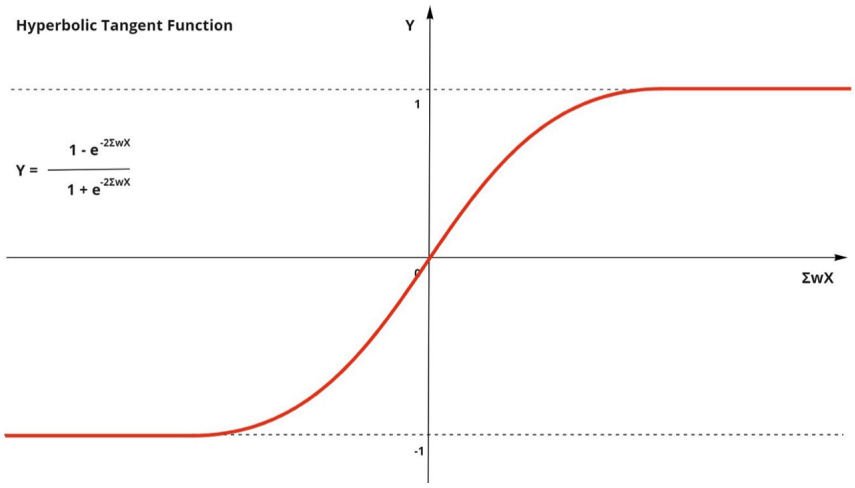


La ReLU è la funzione di attivazioni più utilizzata. Restituisce 0 qualora la somma pesata dei segnali in input è minore o uguale a zero, oppure

$$\sum_{i=1}^{n} w_i I + b$$

negli altri casi.

- Funzione Tangente Iperbolica (Hyperbolic Tangent Function)



Questa funzione è molto simile alla Sigmoid ma differisce per il codominio, infatti, in questo caso, è pari a C[-1,1.]

Di seguito i test, eseguiti per 300 epoche, con le varie funzioni.

Function	Mean Loss	Train Acc %	Validation Acc %
Threshold	1.811	53.473	53.387
Sigmoid	3.602	15.336	15.383
ReLU	**1.380**	**66.736**	**66.818**
Hyperbolic Tangent	3.673	13.879	13.939

I grafici sono stati omessi in quanto in questo caso non è di interesse il trend ma l'accuracy finale.

4.3.3 Learning rate

Il learning rate, abbreviato *lr*, è un parametro molto importante poiché da questo dipende il percorso di *gradient descend*. Il gradient descend è una tecnica di ottimizzazione comunemente usata negli algoritmi di machine learning e serve a trovare il valore del parametro lr tale che minimizza la funzione di costo. In un grafico del percorso di apprendimento del modello il gol è avvicinarsi il più possibile alla croce rossa sul fondo della V che rappresenta il minimo reale della funzione.

Se il lr viene impostato ad un valore troppo basso il percorso convergerà lentamente poiché si necessiterà di molto tempo per compiere tutti i piccoli passi e non è detto che ci arrivi, viceversa se associato ad valore troppo alto divergerà senza mai arrivare la minimo.

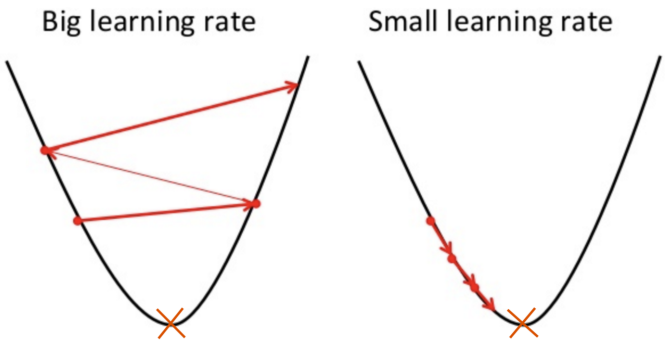
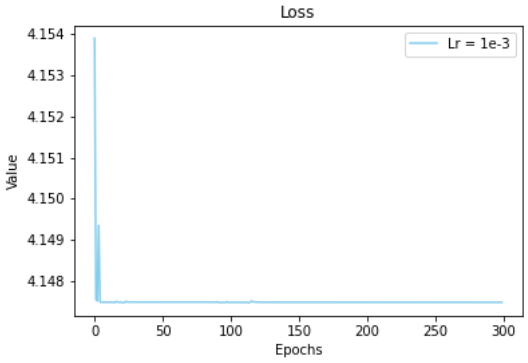


Figura 4.2: Esempio di *gradient descend*

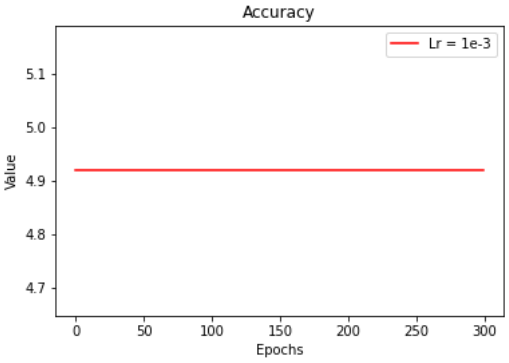
Tra i vari algoritmi per ottimizzare il gradient descend, *Adam*[5] è ritenuto il migliore, quindi si è scelto di eseguire dei test manuali usando solo questo algoritmo variando il parametro *lr*.

lr	Loss	Train Acc%*	Validation Acc%
1e-3	4.147	4.879	4.918
1e-4	**1.563**	**60.538**	**60.039**
1e-5	2.528	39.030	38.698
1e-6	3.463	23.463	23.635
1e-7	4.687	23.988	24.069

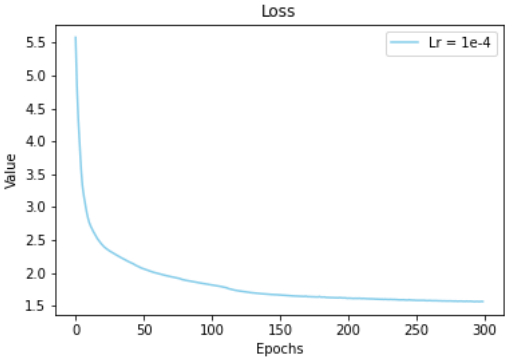
I risultati ottimali sono stati riscontrati con 1e-4 poiché gli altri valori dei test sono troppo piccoli o troppo grandi. Se ne deduce che la scelta del lr ottimale è un compromesso tra stabilità e velocità della convergenza. Di seguito si possono vedere i grafici dei test. Solo il grafico del secondo test si avvicinano ad un comportamento ideale. In questo test il valore della accuracy è marginale, in quanto potrebbe essere basso poiché necessita di più epoche, mentre il trend è più indicativo del comportamento della rete.



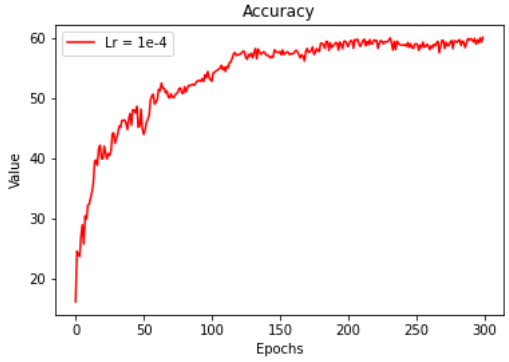
(a) Loss con lr = 1e-3



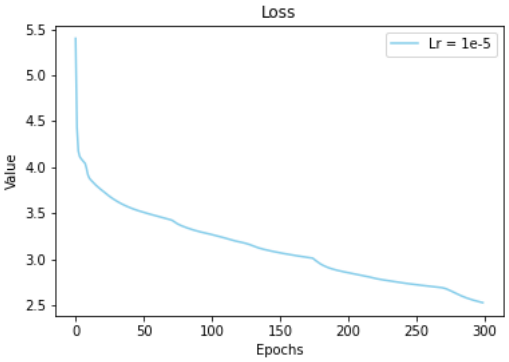
(b) Accuracy con lr = 1e-3



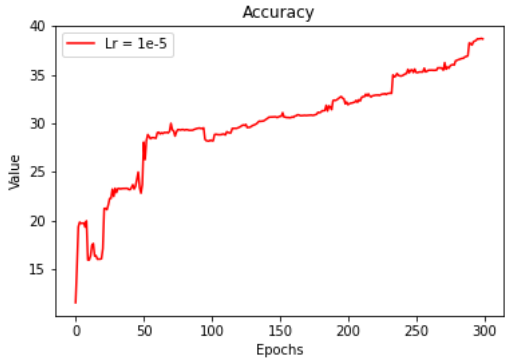
(c) Loss con lr = 1e-4



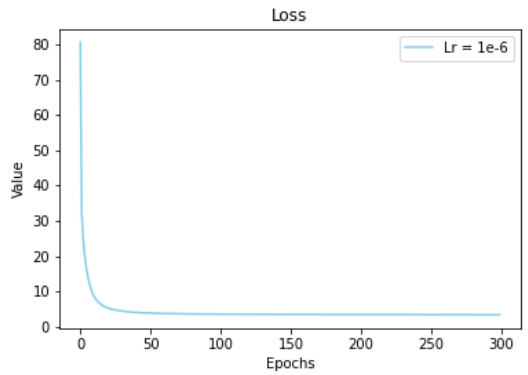
(d) Accuracy con lr = 1e-4



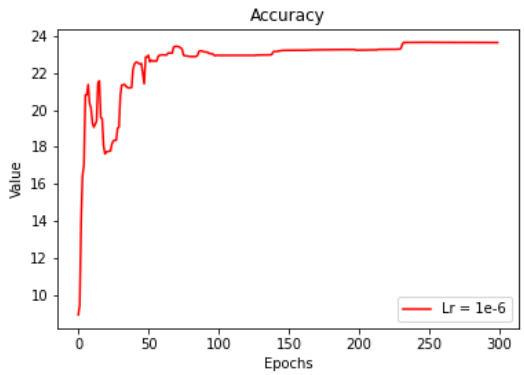
(e) Loss con lr = 1e-5



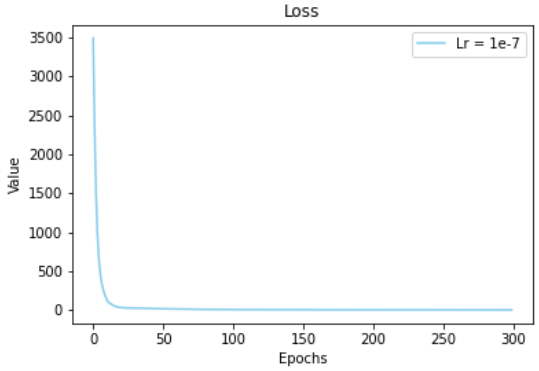
(f) Accuracy con lr = 1e-5



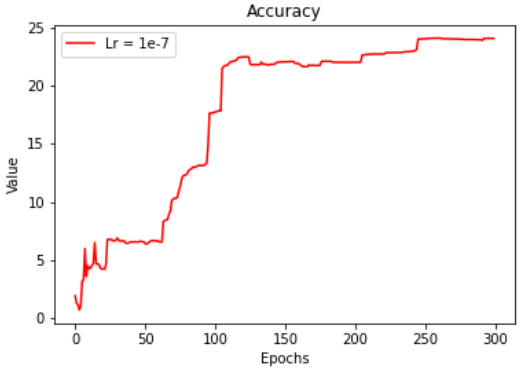
(g) Loss con lr = 1e-6



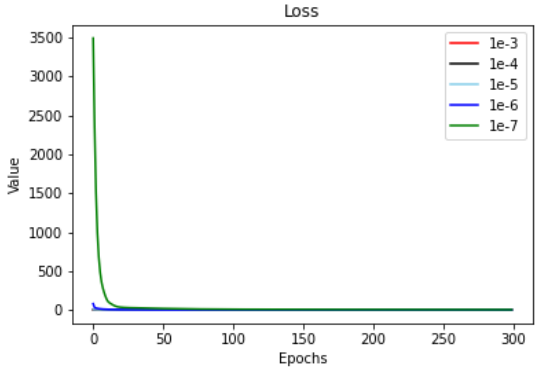
(h) Accuracy con lr = 1e-6



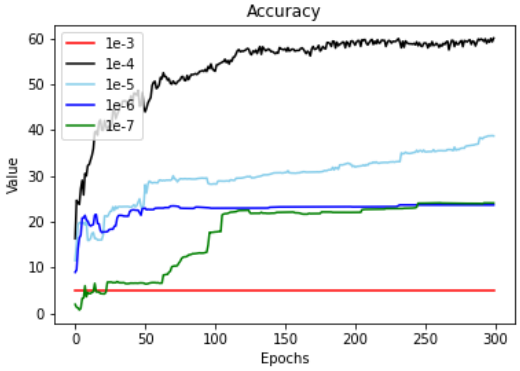
(a) Loss con lr = 1e-7



(b) Accuracy con lr = 1e-7



(c) Comparazione loss



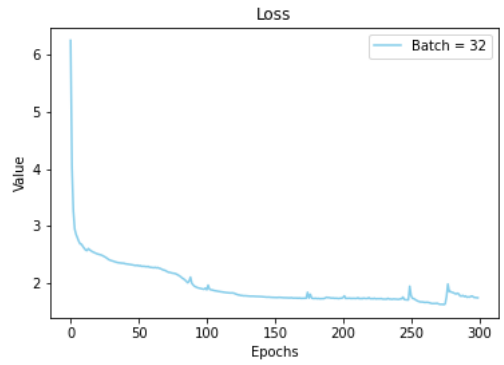
(d) Comparazione accuracy

4.3.4 Batch

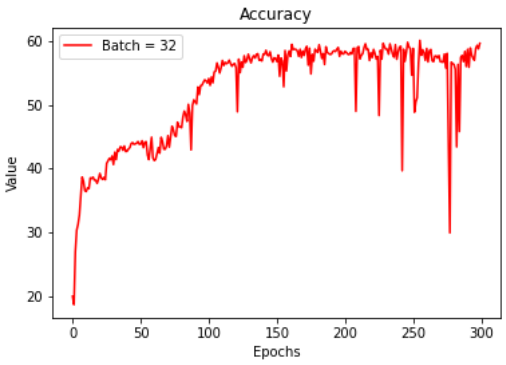
Solitamente, ad un algoritmo di apprendimento automatico, non si da un input alla volta ma dei *batch*, un insieme di input solitamente da 16, 32 o 64 elementi in base alla rete e alla memoria a disposizione. In questo test si è partiti da un batch di 32 elementi e si è arrivati fino a 128.

Batch	Epochs	Loss	Validation Acc %	Time for Epoch
32	100	1.912	53.430	0:00:34.343947
32	200	1.736	57.983	0:00:34.534274
32	300	1.743	59.650	0:00:37.893339
64	100	2.034	49.547	0:00:18.218019
64	200	1.780	57.000	0:00:17.925686
64	300	1.753	**55.148**	0:00:18.064700
128	100	2.177	48.264	0:00:09.408328
128	200	2.043	51.795	0:00:09.537429
128	300	1.934	52.966	0:00:09.527373

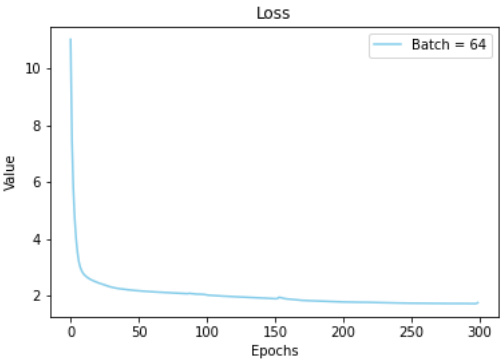
Si nota che la variazione della dimensione dei batch influenza solo marginalmente la precisione della rete, mentre il parametro più influenzato è il tempo di esecuzione. L'algoritmo di back propagation, che migliora la prestazioni della rete, viene utilizzato più volte durante ogni epoca, in particolare dopo l'elaborazione di ogni batch. Aumentare la dimensione dei batch significa utilizzare meno volte la back propagation. Nei test successivi, nonostante la precisione leggermente inferiore, si è scelto di utilizzare 64 come dimensione dei batch poiché un buon compromesso tra velocità di allenamento e precisione finale.



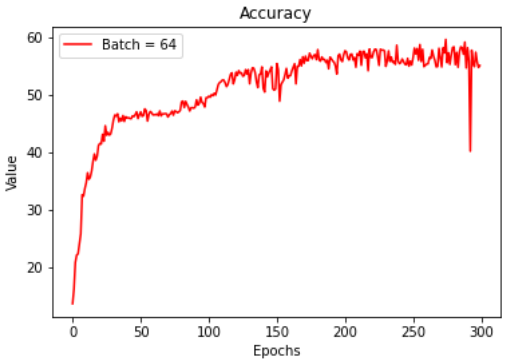
(a) Test con batch = 32



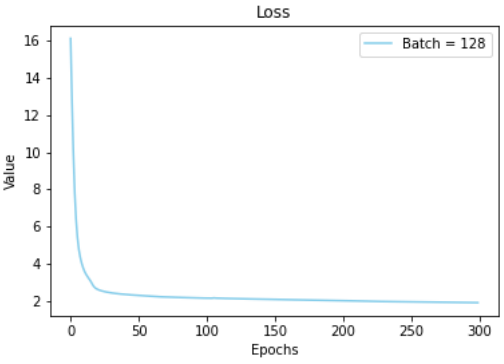
(b) Test con batch = 32



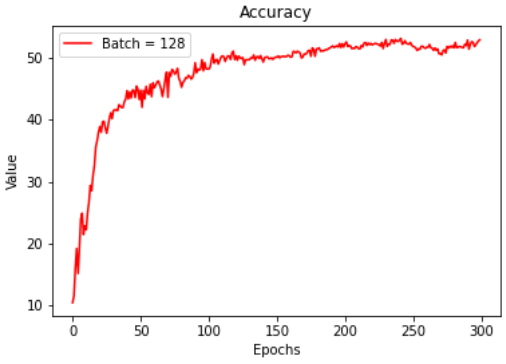
(c) Test con batch = 64



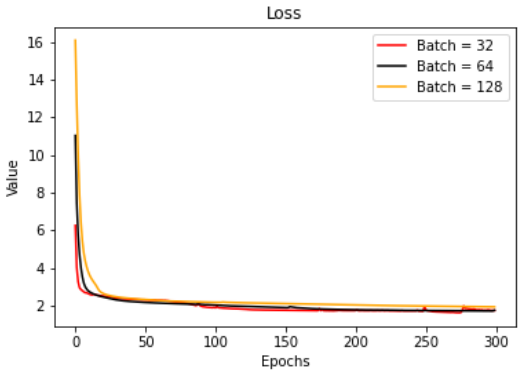
(d) Test con batch = 64



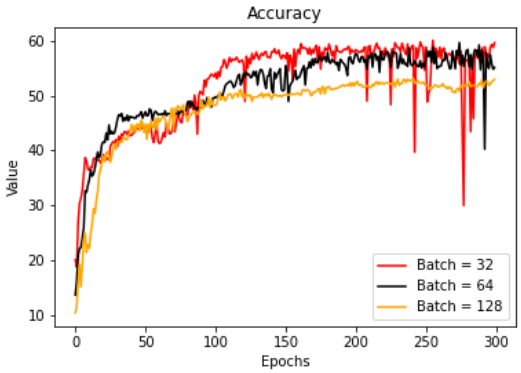
(e) Test con batch = 128



(f) Test con batch = 128



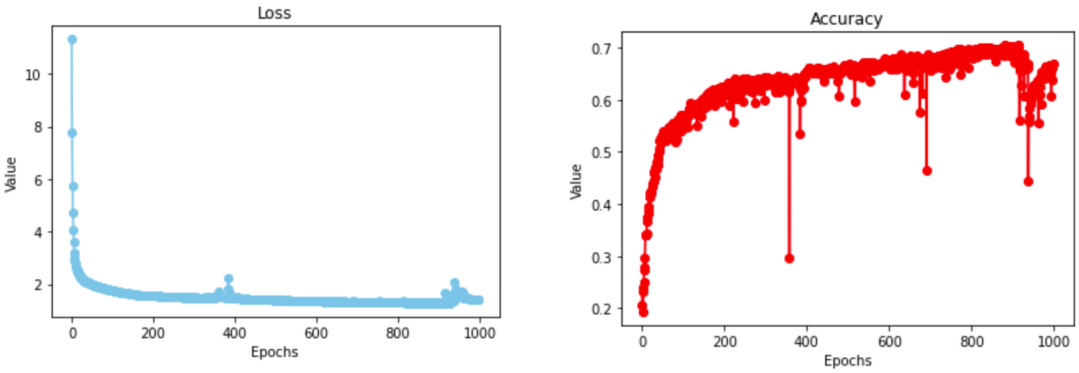
(g) Comparazione



(h) Comparazione

4.3.5 Epoche

Per epoche, o *epochs*, si intendono i cicli di addestramento del modello. Ad ogni epoca il modello riceve in input tutto il training set, modifica il valore dei suoi pesi attraverso la funziona di *back propagation*, e si prepara per l'epoca successiva. Il numero delle epoche è legato al valore del lr e della accuracy calcolata durante l'addestramento.



(a) Loss (b) Accuracy

Dal grafico si evince che il numero ideale di epoche per questo modello è intorno alle 900, oltre non c'è più margine di miglioramento.

4.4 Riepilogo e risultati ottenuti

Dopo aver svolto il *tuning* degli iper-parametri il risultato è una rete con una buona precisione ma che può essere migliorata con l'utilizzo dell'architettura LSTM. Di seguito le caratteristiche del miglior risultato ottenuto.

Params	Value
Architecture	5/30/90/90
Activate function	ReLU
Learning rate	1e-4
Epochs	900
Batch	64
Loss	1.279380
Accuracy	70.286%

5

Conclusioni e possibili sviluppi

Lo scopo di questo lavoro è stato ottenere un modello basato su tecniche di deep learning in grado di eseguire, con maggior precisione possibile, il compito di classificare degli utenti web basandosi sul loro traffico TCP/IP. È stato fornito un dataset, ottenuto da una cattura Wireshark del traffico web della sede dei Rizzi dell'Università degli Studi di Udine, dal quale si sono estratte le seguenti informazioni: timestamp, indirizzo MAC del mittente, indirizzo IP di destinazione, dimensione del pacchetto, porta del mittente e porta del destinatario. A questi dati, appartenenti ai relativi pacchetti, è stato aggiunto il numero di sequenza, da 0 a 6.156.726, in base all'ordine con cui sono stati catturati.

Si è partiti da un risultato iniziale condizionato del 84%. Dopo un'approfondita ispezione del dataset, esso è stato ristrutturato per essere usato durante l'addestramento del modello, poiché inizialmente sbilanciato. Questa operazione ha ridotto sensibilmente la quantità di dati utilizzabili, è stato rimosso circa il 67% dei dati poiché appartenente a utenti con un numero enorme (il 47% solo dell'utente 2) o un numero eccessivamente esiguo di pacchetti. Successivamente la rete ha raggiunto una precisione pari al 28.937% avendo aggiunto la destinazione tra gli input, diminuito il lr e aumentato le epoche. Dopodiché si è eseguita l'ottimizzazione dell'architettura (5/30/90/90), la scelta della funzione di attivazione migliore (ReLu), la scelta del learning rate più adatto (1e-4), la scelta della dimensione migliore per i batch (64) e infine si è calcolato il numero di epoche ideale (900). Si è raggiunta una precisione finale del 70.286%.

Sicuramente ampliare il dataset con una nuova cattura Wireshark aiuterebbe a classificare gli utenti con maggiore precisione. Con un nuovo dataset, possibilmente di dimensioni maggiori a quelle ottenute dopo aver ristrutturato il dataset (2.054.165 pacchetti appartenenti a 90 utenti distinti), si potrebbe sicuramente avere dei risultati migliori.

Per migliorare ulteriormente questo risultato si potrebbero utilizzare tecniche più evolute di addestramento come la previsione dell'idoneità delle reti neurali[9] o l'utilizzo di tecniche avanzate di ottimizzazione[2].

Un altro metodo per migliorare la performance si può trovare nell'utilizzo dell'architettura LSTM, ovvero una tipologia di reti ricorrenti (RNNs). In queste reti l'output dell'istante *t-1* influisce sulla decisione che raggiungerà all'istante t[3]. Se ne deduce che le RNN hanno due fonti di input, il presente (l'input inserito) e il recente passato (lo stato della rete all'istante *t-1*), che si combinano per elaborare una predizione. In una rete neurale tradizionale gli input sono processati indipendentemente gli uni dagli altri e generano degli output egualmente indipendenti. Per il lavoro svolto in questa tesi non è l'ideale. In una RNN, invece, i dati vengono processati contemporaneamente come sequenze di vettori e non singoli elementi cosi che l'output sia dipendente dalle predizione passate.[12] Una LSTM potrebbe essere ideale per svolgere il compito prefisso all'inizio di questa tesi poiché è in grado di tener conto anche l'ordine con cui il traffico TCP/IP viene generato dagli utenti. Considerando anche la dimensione temporale la precisione sicuramente aumenterebbe.

Bibliografia

[1] Anne Bonner
What is Deep Learning and How Does it Work?
`https://towardsdatascience.com`, ultima consultazione 03-09-2020

[2] Bergstra James, Bardenet Rémi, Béngio Yoshua, Kégl Balázs
Algorithms for hyper-parameter optimization
Advances in Neural Information Processing Systems, 2011

[3] Chris Nicholson
A Beginner's Guide to LSTMs and Recurrent Neural Networks
`https://pathmind.com`, ultima consultazione 10-09-2020

[4] Claudio Piciarelli, Marino Miculan, Gianluca Foresti
Towards User Recognition by Shallow Web Traffic Inspection
Italian Conference of Cyber Security, Pisa, 2019

[5] Diederik P. Kingma, Jimmy Ba
Adam: A Method for Stochastic Optimization
International Conference for Learning Representations, San Diego, 2015

[6] Larry L. Peterson, Bruce S. Davie
Reti di calcolatori
Apogeo Education, 2012

[7] M. Benaiah Deva Kumar, B. Deepa
Computer Networking: A Survey International Journal of Trend in Research and Development, Volume 2(5), 2015

[8] MLK
Cost functions for Classification problems
`https://machinelearningknowledge.ai`, ultima consultazione 03-09-2020

[9] Oliver Hennigh
Automated Design using Neural Networks and Gradient Descent
International Conference for Learning Representations, Vancouver, 2018

[10] Paolo Piacenti
Le 4 funzioni di attivazione neuronale più usate negli artificial neural network
`https://medium.com`, ultima consultazione 01-09-2020

[11] Purnasai Gudikandula
A Beginner Intro to Neural Networks
`https://medium.com`, ultima consultazione 01-09-2020

[12] Purnasai Gudikandula
Recurrent Neural Networks and LSTM explained
`https://medium.com`, ultima consultazione 10-09-2020

[13] R. Sathya, Annamma Abraham
Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification
International Journal of Advanced Research in Artificial Intelligence, 2013

[14] Rich Caruana, Steve Lawrence, Lee Giles
Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping
Conference of Advances in Neural Information Processing Systems 13, 2000, Denver, CO, USA

[15] Robert Moni
Reinforcement Learning algorithms - an intuitive overview
`https://medium.com`, ultima consultazione 09-09-2020

[16] Tarang Shah
About Train, Validation and Test Sets in Machine Learning
`https://towardsdatascience.com`, ultima consultazione 01-09-2020

[17] Tom M. Mitchell
Machine Learning
McGraw-Hill Science, 1997

[18] Tom M. Mitchell
The need of biases in learning generalizations
Rutger CS tech report CBM-TR-117, 1980

[19] Tong Yu, Hong Zhu
Hyper-Parameter Optimization: A Review of Algorithms and Applications
`https://it.arXiv.org`, ultima consultazione 01-09-2020

[20] Wikipedia
Dword
`https://it.wikipedia.org`, ultima consultazione 11-09-2020

[21] Wikipedia
Sniffing
`https://it.wikipedia.org`, ultima consultazione 16-09-2020

[22] Wikipedia
Transmission Control Protocol
`https://it.wikipedia.org`, ultima consultazione 02-09-2020

[23] Wikipedia
Wireshark
`https://it.wikipedia.org`, ultima consultazione 16-09-2020

[24] Wikipedia
World Wide Web
`https://it.wikipedia.org`, ultima consultazione 01-09-2020

[25] Yann LeCun, Leon Bottou, Genevieve Orr., Klaus-Robert Müller
Efficient Backpropagation
Published in "Neural Networks: Tricks of the Trade", Springer, 1998