Università Politecnica delle Marche

Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Quattro tecniche per costruire vettori di embedding nel rilevamento di DGA basato sull'apprendimento profondo: un confronto sperimentale

Four Techniques for Constructing Embedding Vectors in Deep Learning-based DGA Detection: An Experimental Comparison

Relatore:
Prof. Spalazzi Luca

Tesi di Laurea di:
Olivieri Lorenzo

Correlatore:
Prof. Cucchiarelli Alessandro

Anno accademico 2021/22

Indice

Elenco delle figure

Introduzione

La rapida diffusione di Internet e dei dispositivi elettronici degli ultimi decenni ha apportato notevoli miglioramenti all'interno dei nostri stili di vita. Molte operazioni della vita di tutti i giorni, a partire ad esempio dalle operazioni bancarie, possono essere effettuate con un semplice click attraverso i sistemi informatici. Inoltre, la pandemia di COVID-19 ha accelerato ulteriormente la digitalizzazione della nostra società. Se da un lato quest'ultima ha apportato dei benefici, dall'altro vi sono dei rischi a cui gli utenti della rete sono esposti. Il pericolo è rappresentato dai cyber-criminali che, nell'eventualità in cui riuscissero a infettare i sistemi informatici attraverso malware, avrebbero potenzialmente accesso a elevate quantità di dati sensibili che potrebbero utilizzare per scopi malevoli. A testimonianza della numerosa presenza di pericoli online, secondo Forbes, "nel 2021, 281,5 milioni di persone sono state coinvolte in una data breach mentre il crimine digitale è costato alle aziende 1,79 milioni di dollari per minuto" [25].

L'ENISA ha evidenziato che "nell'anno 2020 tra tutte le tipologie di malware, quelli basati su botnets sono stati i più diffusi (28%)" [3]. Essi possono configurare il dispositivo su cui sono installati come facente parte di una grande rete di dispositivi infetti (zombies), connettendolo a un Command & Control (C&C) server. Attraverso di esso, i cybercriminali (botmasters) riescono ad avere il controllo sui sistemi infetti, sfruttandoli per effettuare attacchi informatici. Per interrompere la connessione (o il suo tentativo) dei bot al C&C server, è possibile identificare il nome di dominio di quest'ultimo a partire dalle richieste DNS (Domain Name Service) effettuate dal dispositivo, reindirizzandole verso un host differente. Per evitare tale identificazione, oggigiorno la maggior parte delle botnets sono dotate di Domain Generation Algorithms (DGAs). Questi algoritmi consentono di generare una lista di nomi di dominio composti di caratteri generati casualmente o di parole di una

certa lingua. I bot tentano di connettersi al server inviando richieste DNS ai nomi di dominio presenti in questa lista fin quando la connessione non ha successo. I C&C server cambiano periodicamente il loro nome di dominio, quindi i bot devono riuscire a connettersi ad essi entro un intervallo di tempo di una certa durata.

Come si può intuire, l'introduzione dei DGAs ha reso più difficile la rilevazione dei malware nei sistemi infetti.

Per affrontare questa problematica, sono state sviluppate tecniche di Machine Learning e, più recentemente, di Deep Learning. Il seguente elaborato fa riferimento alla ricerca [7], che si propone di migliorare l'algoritmo basato su Deep Learning per la classificazione di DGA, utilizzando embedding di n-grams pre-addestrati ottenuti tramite apprendimento non supervisionato (Unsupervised Learning) in un contesto in cui sono disponibili dataset di dimensione contenuta. In particolare, l'obiettivo della seguente tesi è illustrare i risultati derivanti dalla combinazione di vettori di embedding relativi agli n-gram di un nome di dominio e fornire tale combinazione ad un classificatore posto a valle dello strato di embedding. Rispetto alla metodologia già adottata nel paper citato si possono evidenziare due principali differenze:

- Nel lavoro citato, la sequenza di vettori di embedding viene fornita cosi com'è in ingresso al classificatore. In questo lavoro di tesi la sequenza viene combinata e al classificatore viene fornito il vettore risultato della combinazione;
- Nell'elaborato menzionato, inoltre, il classificatore adottato, dovendo gestire una sequenza di vettori, è di tipo Long-Short Term Memory (LSTM). In questo lavoro di tesi, non avendo sequenze da gestire, si è preferito optare per un più semplice Multi Layer Perceptron (MLP).

Il resto del lavoro è organizzato come segue:

- Nel Capitolo 1 si illustrerà il funzionamento generale delle botnets e dei DGAs in esse utilizzati;
- Nel Capitolo 2 si illustreranno le diverse tecniche di Machine Learning più utilizzate nel tempo finalizzate alla rilevazione di malware DGA e l'approccio basato su Embedding;

- Nel Capitolo 3 si mostrerà l'architettura dei classificatori utilizzati, esporrà i dati di addestramento impiegati e i tipi di esperimenti effettuati;
- Nel Capitolo 4 si presenteranno i risultati dei vari esperimenti mediante indici di performance;
- Infine verranno tratte alcune conclusioni dal lavoro svolto e proposti alcuni sviluppi futuri.

Capitolo 1

Malware e DGA

Il termine botnet è l'abbreviazione di "robot network" e consiste in una serie di macchine infette da malware (bot), coordinate da un botmaster in maniera tale che esse compiano azioni illecite online per suo conto. Nella Figura 1.1 è illustrato un esempio di una tipica struttura di una robot network.

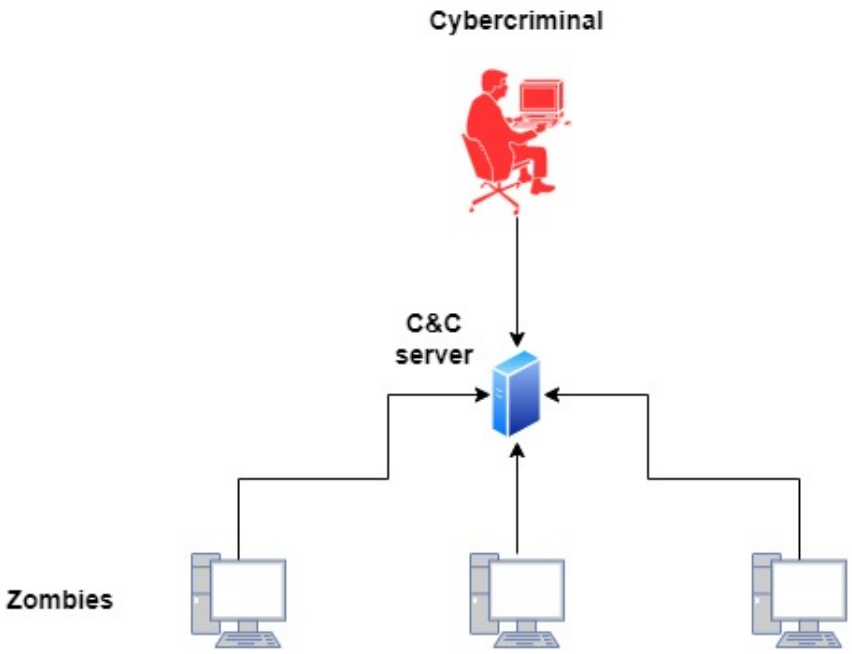

Figura 1.1: Botnet centralizzata [20]

1.1 Componenti di una botnet

Le botnets sono solitamente composte da quattro componenti:

- **Bot**: È un software malevolo installato su macchine vulnerabili che fa si che esse possano essere controllabili esternamente da malintenzionati senza il consenso del proprietario. Una delle modalità attraverso le quali questi virus possono essere installati sui sistemi vittima sono per il tramite di download di file da email di spam o siti infetti. Ogni volta che il sistema infetto viene riavviato, il software in questione viene rieseguito in automatico. Le azioni malevole sono effettuate dalle macchine infette grazie a una connessione ad un canale command and control.

- **Zombies**: Macchine infette dal malware bot che spesso hanno le caratteristiche del bersaglio tipico dei criminali informatici. Esse sono: disponibilità di alte velocità per la navigazione, bassi livelli di protezione, elevata distanza geografica rispetto al gestore della botnet. Infatti, l'utente proprietario del sistema infetto dotato di alta velocità per la navigazione in Internet, sarà meno propenso a sospettare che il proprio dispositivo lo sia. Questo perchè i volumi di traffico generati dal bot potrebbero non essere elevati in rapporto alla velocità di trasmissione della propria connessione di rete. Inoltre, per quanto riguarda l'elevata distanza geografica, vengono scelti host distanti rispetto alla posizione dell'amministratore della botnet per rendere più difficile la sua localizzazione da parte delle forze dell'ordine.

- **Botmaster**: Rappresenta colui che coordina la rete di zombies con finalità malevole. Stabilisce quali sono i comandi che le macchine infette devono eseguire e, successivamente, li comunica ad esse per il tramite di un intermediario chiamato C&C server. Non sempre coincide con il creatore della botnet poichè è possibile che quest'ultima venga venduta a clienti terzi rispetto allo sviluppatore. Indipendentemente da chi sia il soggetto che rappresenta il botmaster, la botnet viene di solito creata con lo scopo di procurargli guadagni finanziari. [13]

- **C&C server (Command and Control)**: Viene utilizzato per inviare i comandi atti a far compiere ai sistemi ad esso connessi azioni illegali. Da esso dipende la stabilità, la robustezza e il tempo di reazione della botnet. È possi-

bile individuare quattro tipi di strutture di C&C: centralizzata, decentralizzata, ibrida e casuale.

1.2 Architettura di una botnet

L'infrastruttura C&C può essere di diversi tipi: centralizzata, decentralizzata, ibrida, casuale. L'esistenza di diverse modalità di organizzazione di C&C testimonia l'evoluzione che le botnet hanno avuto nel tempo.

1.2.1 Centralizzata

La struttura centralizzata è simile a quella del modello di rete client-server. Di conseguenza, tutti i bot sono connessi a uno o un paio di punti rendezvous. La Figura 1.1 mostra come è costituita una architettura centralizzata. Le modalità di comunicazione utilizzate con questa architettura sono principalmente le seguenti:

- **Internet Relay Chat (IRC)**: gli zombies connessi a botnets che adottano questo protocollo attendono comandi dal canale IRC creato dal botmaster sul C&C. Permette ai membri della botnet di comunicare tra loro e al botmaster di comunicare con un sottogruppo della rete di sistemi infetti. Inoltre, questo protocollo si è evoluto nel tempo introducendo l'offuscamento delle comunicazioni sul C&C. È facilmente reperibile online e flessibile ma è possibile rilevarlo agevolmente e bloccare il traffico derivante dal suo utilizzo attraverso un firewall;
- **HyperText Transfer Protocol (HTTP)**: nasce a causa della semplice rilevazione delle botnets basate su IRC. Il suo principale vantaggio consiste nel fatto che il traffico HTTP è permesso nella maggior parte delle reti.

Il vantaggio dell'architettura in questione sta nei rapidi tempi di reazione e nella buona coordinazione degli zombies della botnet. Il botmaster è in diretto contatto con il C&C, e dunque, può essere a conoscenza di quanti zombies sono attivi e della loro distribuzione globale. Il problema principale di questo tipo di configurazione consiste nel fatto che rappresenta un SPOF (Single Point Of Failure - letteralmente "Unico punto di fallimento"). Ciò vuol dire che se la botnet venisse scoperta, potrebbe essere bloccata semplicemente localizzando il C&C. Questa problematica

ha fatto si che nascesse una diversa configurazione dell'infrastruttura, ossia quella decentralizzata.

1.2.2 Decentralizzata

Questa tipologia di architettura nasce dall'esigenza dei botmasters di far rimanere la botnet celata per più tempo possibile, in modo che egli possa continuare a trarne profitti. Infatti, le botnets che optano per questa organizzazione non hanno più un SPOF, il che vuol dire che non è più possibile individuare al loro interno, un singolo elemento che invia i comandi ai vari zombies. Le botnets decentralizzate sono basate su protocolli P2P. In poche parole, le istruzioni malevole vengono condivise tra i vari zombies e quindi ognuno di essi agisce sia come server che come client. Questo tipo di architettura è più vantaggiosa di quella centralizzata se si considera l'assenza di SPOF, ma per quanto riguarda i tempi di reazione, nella struttura decentralizzata un messaggio impiega più tempo ad essere condiviso tra tutti i partecipanti alla botnet. In Figura 1.2 è mostrata visivamente la struttura alla quale si sta facendo riferimento.

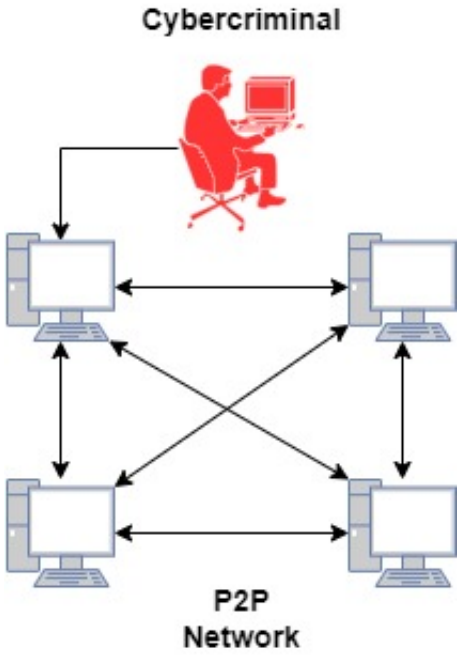

Figura 1.2: Botnet decentralizzata [20]

1.2.3 Ibrida

Combina le caratteristiche delle precedenti due architetture. Per velocizzare il passaggio di informazioni tra gli zombie, a differenza dell'architettura decentralizzata, qui ci sono alcuni zombies relegati al ruolo di "servant" (letteralmente "servi") e alcuni che agiscono da clients. I servant agiscono sia da client che da server e hanno un IP statico e pubblico. Rimangono in ascolto di connessioni in arrivo e per lo scambio di informazioni utilizzano una chiave privata auto-generata. Sono gli unici a possedere gli indirizzi IP dei client. Periodicamente, tutti gli zombies devono connettersi ai servant per ottenere nuove direttive. I client hanno un IP dinamico e privato. Solitamente si trovano dietro a un firewall, senza un accesso globale a Internet.

1.2.4 Casuale

In questo modello non vi è alcun mezzo intermedio nella comunicazione tra botmaster e zombie. Questo fatto costituisce un vantaggio poichè la difficile rilevazione della trasmissione di dati rende problematica la distruzione dell'intera botnet. Di contro, il botmaster non potrà effettuare attacchi di grossa taglia poichè è incapace, nel modello in questione, di coordinare più di uno zombie in contemporanea. Di fatto, attualmente nessuna botnet è basata su questa architettura, ed è considerata soltanto teorica.

1.3 Ciclo di vita di una botnet

Prima di poter diventare un membro della botnet, un host vulnerabile, cioè un possibile futuro zombie, deve attraversare sostanzialmente cinque fasi, come evidenziato dalla Figura 1.3:

- **Initial Injection**: questa prima fase consiste nell'infezione del sistema per il tramite di download o apertura di file da dispositivi removibili, siti o email infette.
- **Secondary Injection**: questa fase può essere iniziata soltanto se la prima è andata a buon fine. Nel momento in cui l'utente apre il file infetto, esegue uno

script che ricerca in rete i binaries del malware. Quando quest'ultimi sono stati scaricati e installati, il sistema infetto inizierà ad agire come uno zombie.

- **Connection o Rallying**: in questo stadio avviene la connessione con il C&C. Il software malevolo installato è programmato per far si che questa fase si verifichi nuovamente ad ogni riavvio della macchina infetta, in modo tale che il botmaster possa riprenderne il controllo. In questo punto del ciclo di vita la botnet è a rischio, dato che ogni volta che il computer infetto tenta di connettersi al server genera traffico che potrebbe essere identificato monitorando il servizio DNS.

- **Malicious activities** ("Attività malevoli"): questa fase consiste nell'inizio dell'attacco, che può essere di diverse tipologie: DDoS (Distributed Denial of Service), spam di email tramite server SMTP (Simple Mail Transfer Protocol) dell'utente infettato, phishing e furto di dati. In questo stadio la comunicazione tra C&C e zombie si fa più intensa ma la botnet potrebbe ancora rimanere celata ad alcune tecniche di rilevazione dato che non genera elevati volumi di traffico.

- **Maintenance and Update** ("Manutenzione e aggiornamento"): È compito del botmaster aggiornare il bot per renderlo continuamente non rilevabile e bypassare possibili nuove tecniche di rilevazione. Inoltre, questa fase è cruciale se esso vuole aggiungere nuove funzionalità al bot. Dal punto di vista degl esperti di cybersecurity, questo momento risulta di fondamentale importanza per la rilevazione della botnet, perchè ogni volta che c'è un aggiornamento del malware, vi è il reintensificarsi della comunicazione tra zombies e C&C.

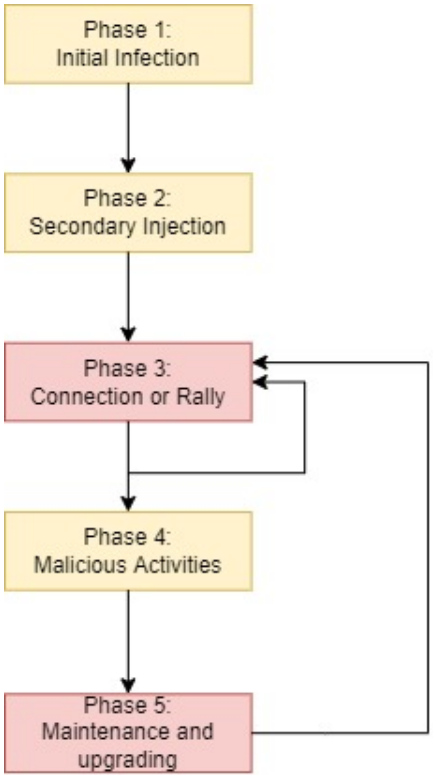

Figura 1.3: Ciclo di vita di una botnet [16]

1.4 Domain Generation Algorithm - DGA

I DGAs sono algoritmi per la generazione periodica di liste di nomi di dominio che, nel ciclo di vita della botnet, subentrano nella fase relativa alla Connection o Rallying, ossia alla connessione con il C&C. Una volta generato l'elenco dei nomi di dominio, lo zombie tenta di ad ognuno di essi fin quando non ha successo nel raggiungere il server. Le liste di nomi di dominio vengono generate sulla base di un seme. Quest'ultimo generalmente coincide con la data corrente, ma può essere basato su qualsiasi tipo di dato, proveniente da diverse fonti (es. data di determinati post di Twitter). Inoltre, esso è conosciuto sia dal bot che dal botmasters in modo da permettere agli zombies di connettersi al dominio generato entro un certo lasso di tempo. Quest'ultimo fatto rende inefficace il sinkholing e blacklisting del nome di dominio. Le tecniche in questione consistono rispettivamente nel redirigere il traffico diretto inizialmente verso un server, a uno di propria scelta e nell'inserire il nome di dominio all'interno di una lista nera che fa si che esso non venga più utilizzato. Il fatto che il seme possa dipendere da tipi di dato variabili nel tempo fa si che si

possa generare domini in maniera randomica e di difficile previsione da parte degli analisti del settore della cybersecurity. In [12] è emerso che è possibile differenziare i diversi tipi di DGAs nelle seguenti categorie:

- **Arithmetic-based**: calcolano una sequenza di valori che o hanno una rappresentazione ASCII utilizzabile per un nome di dominio oppure scelgono un offset in uno o più hardcoded array che costituiscono l'alfabeto del DGA. Sono il tipo più comune di DGA;
- **Hash-based**: utilizzano la notazione hexdigest di un hash per produrre un nome di dominio (es. MD5 e SHA256);
- **Wordlist-based**: concatenano una sequenza di parole da una o più liste di parole. Quest'ultime possono essere incorporate all'interno del binario del malware oppure ottenute da una fonte pubblicamente accessibile;
- **Permutation-based**: derivano tutte le possibili permutazioni delle lettere all'interno di un nome di dominio iniziale.

Capitolo 2

Machine learning per rilevazione di DGA

L'IA (Intelligenza Artificiale) è una scienza che ha la finalità di far agire i computers come esseri umani. Un suo sottogruppo è rappresentato dal ML (Machine Learning), che invece si pone l'obiettivo di permettere agli elaboratori di imparare a eseguire determinati compiti senza che essi siano stati esplicitamente programmati per farlo. Il DL (Deep Learning) consiste in una sottoclasse del ML ed è incentrato sull'utilizzo delle reti neurali profonde per estrarre automaticamente caratteristiche utili presenti nei dati grezzi, impiegandoli per auto addestrarsi. Ciò che distingue DL dal resto del ML è il fatto che nel ML tradizionale, le caratteristiche utili che dovevano essere individuate nei dati venivano definite esplicitamente dai programmatori, mentre nel DL tutto ciò avviene automaticamente. Nel ML in generale e nel DL in particolare esistono tre tipologie di apprendimento:

- **Supervisionato**: l'algoritmo osserva degli esempi di coppie input-output e impara una funzione che associa gli input agli output;
- **Non supervisionato**: a differenza del caso precedente, la rete neurale impara schemi presenti nei dati senza alcuna conoscenza a posteriori su di essi;
- **Rinforzato**: l'apprendimento avviene per mezzo di rinforzi o punizioni a seconda delle azioni o risultati che l'algoritmo fornisce.

L'ossatura del DL è stata già definita in passato ma attualmente questo campo è in particolare via di sviluppo come conseguenza dei miglioramenti avvenuti

nel campo dell'hardware, della programmazione e dell'elevata disponibilità di dati dovuta alla diffusione dell'informatica in ogni disciplina. Le tecniche di DL sono applicate ad ogni ambito e quindi anche a quello della cybersecurity, e, dunque, alla rilevazione di botnets e DGAs attraverso l'analisi di DNs. I modelli di reti neurali più utilizzati nel particolare settore sono: CNN [22, 5], (Convolutional Neural Networks), RNN (Recurrent Neural Networks), LSTM (Long-Short Term Memory) [2, 5], GRU (Gated Recurrent Units) [10], RNNs con attention mechanism [5].

Bisogna notare come, nel campo dell'individuazione di malware DGA, sia necessario l'utilizzo di tecniche di Embedding, per fare in modo che le parole siano trasformate in numeri interpretabili da un algoritmo. Una metodologia fondamentale utilizzata è quella dei Word Embeddings, che consistono in reti neurali che hanno il compito di trasformare la generica parola in un vettore numerico che tenga conto del suo significato.

2.1 Neurone e reti neurali

2.1.1 Perceptron

Il **neurone**, in inglese Perceptron, è stato ideato nel 1943 da McCulloch e Pitts [8]. L'idea nasce dall'ipotesi, nel campo delle neuroscienze, che l'attività mentale sia conseguenza di interazioni elettrochimiche in reti di cellule cerebrali chiamate neuroni. Come si è già detto, il DL fa utilizzo delle reti neurali profonde. Quest'ultime sono costituite da combinazioni particolari di blocchi, detti neuroni. Come è possibile notare dalla Figura 2.1, il j-esimo neurone riceve n input. Ognuno di essi viene moltiplicato per il rispettivo **peso** w_{ij}. Ogni unità j calcola prima di tutto una **somma pesata** dei suoi i input (2.1).

$$in_j = \sum_{i=0}^{n} w_{ij} a_i. \tag{2.1}$$

Dopodichè, si applica alla precedente somma una **funzione di attivazione** g per ottenere l'output del neurone, come mostrato nell'equazione 2.2.

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{ij} a_i) \tag{2.2}$$

La funzione di attivazione viene utilizzata per introdurre una non linearità all'interno del modello. Quest'ultima consente di approssimare funzioni arbitrariamente complesse. Le principali funzioni di attivazione sono la sigmoide, gradino, ReLU e tangente iperbolica.

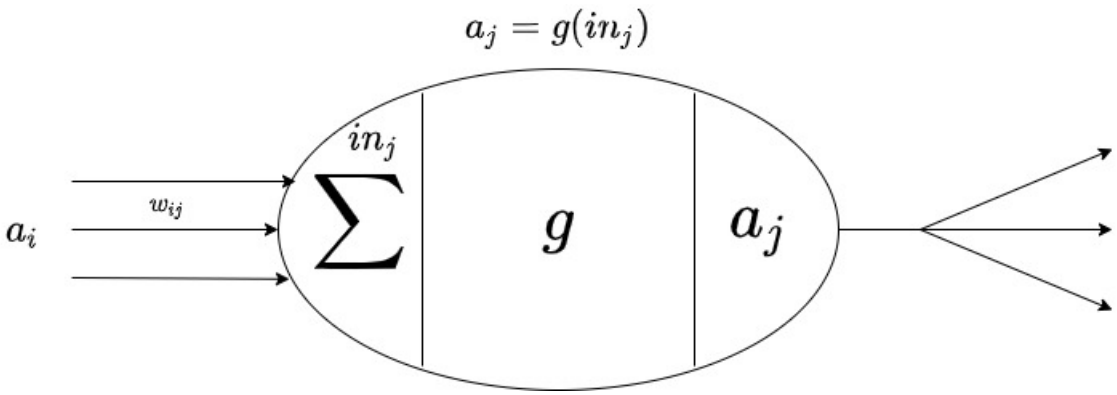

Figura 2.1: Modello del neurone [14]

Alcuni esempi di NN (Neural Network) che possono essere costituiti a partire dal perceptron sono:

- **Multi Output Perceptron**: anche chiamato **dense layer**, è costituito da un livello di input e uno di neuroni. Gli input vengono combinati nei neuroni attraverso i pesi. In ogni neurone, gli stessi input possono confluire con pesi diversi. Una volta applicata la funzione di attivazione, i neuroni producono output indipendenti l'uno dall'altro.

- **Single Layer Neural Network**: è costituita da un livello di input, un livello nascosto di n neuroni (anche detto hidden layer) e un livello di output. Gli input, come nel caso precedente, si combinano attraverso una matrice dei pesi $\mathbf{W}^{(1)}$ per poi immettersi nei neuroni dell'hidden layer. In ogni neurone, viene effettuata la somma della moltiplicazione tra pesi e input e, successivamente,

si calcola la funzione di attivazione della sommatoria. I valori che risultano da questa computazione si immettono negli output, combinati nuovamente con una matrice dei pesi $\mathbf{W}^{(2)}$. Quindi, ogni output risulterà essere una somma pesata delle funzioni di attivazione uscenti dai nodi intermedi della rete;

- **Deep Neural Network**: rete più complessa di quelle ai punti precedenti, caratterizzata dalla presenza di n hidden layer di n nodi.

2.1.2 Addestramento

Una volta che è stato stabilito il compito che la rete neurale deve eseguire, occorre addestrarla. Se verrà utilizzata senza averla sottoposta a quest'ultima fase, essa predirrà outputs che si discosteranno di molto da quelli attesi. Generalizzando a tutto il dataset, si dice che la NN genererà una **perdita empirica**:

$$J(\mathbf{W}) = \frac{1}{n}\sum_{i=0}^{n} \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)}) \tag{2.3}$$

A seconda della tipologia di problema che la rete neurale è stata incaricata di risolvere, vi sono specifiche funzioni perdita che possono essere utilizzate. Ad esempio, per la classificazione binaria si utilizza la Binary Cross Entropy Loss mentre per la regressione si utilizza la Mean Squared Error Loss.

Gradient Descent Algorithm

La fase di addestramento consiste nella minimizzazione della funzione perdita per il tramite del **GDA (Gradient Descent Algorithm)**. Se la perdita dipendesse da due pesi, quest'ultimo permetterebbe di spostarsi all'interno del disegno della funzione, cambiando i valori da cui essa dipende con la finalità di trovare il suo minimo globale. È praticamente impossibile, nelle applicazioni reali, ottenere una funzione perdita tridimensionale. Questo è ovvio guardando alla struttura delle reti neurali profonde. Infatti, la presenza di numerosi layer di altrettanto numerosi nodi tra livello di input e output, fa si che la rete sia caratterizzata da milioni, se non miliardi, di parametri. Di conseguenza, la funzione perdita dipenderà da ogni parametro del NN. Detto ciò, il GDA, consiste nell'eseguire i seguenti passi:

1. Inizializzazione casuale dei pesi;

2. Iterazione fino alla convergenza:

   - Calcolo del gradiente (2.4);

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} \tag{2.4}$$

   - Aggiornamento dei pesi (2.5);

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} \tag{2.5}$$

3. Restituzione dei pesi.

Questo processo viene messo in atto per ogni parametro nella rete. È possibile ottenere la variazione della perdita rispetto a qualsiasi peso della rete grazie alla **BP (Backpropagation)**. Quest'ultima consiste nell'applicazione della regola della catena (in inglese "chain rule"). Come si può osservare dalla Figura 2.2, per calcolare

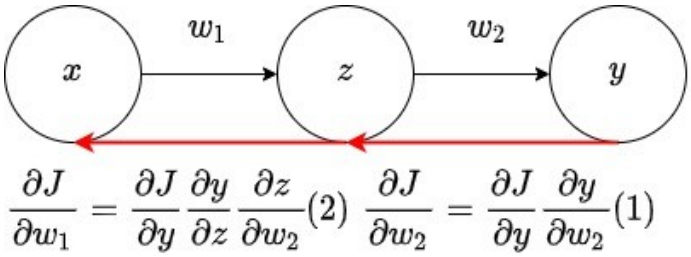

Figura 2.2: Esempio di Backpropagation

la variazione della perdita rispetto al peso tra output e hidden layer, si calcola la variazione della perdita rispetto al peso w_2. Applicando nuovamente la regola della catena, è possibile ottenere la variazione della perdita rispetto al parametro w_1.

Selezione del modello

La dimensione del modello viene scelta attraverso un processo iterativo, chiamato **CV (Cross Validation)**. Per fare in modo che il modello venga addestrato, occorre somministrargli dei dati in modo tale che esso impari caratteristiche in essi contenute. Il dataset dato in pasto all'algoritmo sarà suddiviso, in una certa percentuale ogni iterazione a seconda dell'algoritmo di CV scelto, in training set e validation set. Il training set rappresenta l'insieme di dati che verranno utilizzati per addestrare l'algoritmo di apprendimento, mentre il validation set contiene i dati impiegati per misurare le sue prestazioni attraverso il calcolo della funzione perdita. L'obiettivo è

quello di evitare il più possibile i fenomeni di **overfitting** e **underfitting**. Si dice che il modello è in overfitting quando esso si adatta troppo alle particolari caratteristiche dei dati del training set in maniera tale da avere cattive prestazioni su quelli che l'algoritmo non ha mai visualizzato. Si dice, invece, che il modello è in underfitting quando non riesce ad apprendere peculiarità dai dati con cui è stato addestrato e ciò fa si che non garantisca prestazioni sufficienti sul validation set. Nella Figura 2.3 è possibile osservare l'andamento dell'errore sui due insiemi di dati e le parti di grafico corrispondenti alle problematiche appena descritte. Le tecniche di **regularization** (regolarizzazione del modello) più comuni, che consentono di aggirarle sono:

- **Dropout**: imposta in maniera casuale le funzioni di attivazione di una certa percentuale di neuroni della rete, in modo tale da spronarla a non adattarsi troppo ai dati di addestramento;
- **Early Stopping**: consiste nel fermare il processo di addestramento nel momento in cui la dimensione del modello garantisce il minimo errore sul validation set.

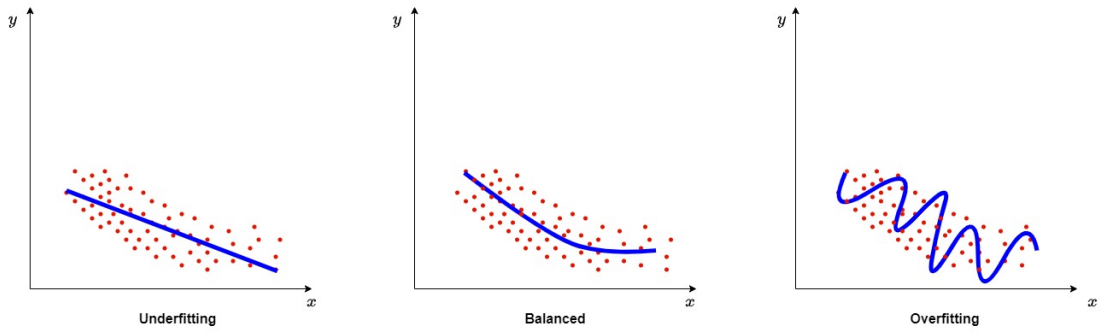

Figura 2.3: Underfitting e overfitting del modello

2.2 Tipologie di reti neurali

Per quanto riguarda la rilevazione di botnets dotate di DGAs, i modelli di rete neurale più utilizzati sono: CNN (Convolutional Neural Network), RNN (Recurrent Neural Networks), LSTM (Long-Short Term Memory), GRU (Gated Recurrent Units), Transformer. Di seguito, verranno illustrati uno per uno.

2.2.1 CNN - Convolutional Neural Network

La CNN è un network neurale a livelli multipli, basato sulla corteccia visiva animale. Riguardo alla sua architettura, i primi livelli sono utilizzati per riconoscere caratteristiche dei dati mentre gli strati successivi sono adibiti alla loro ricombinazione per formare attributi di alto livello associati agli input e alla loro classificazione. Nei livelli dedicati all'estrazione di peculiarità dai dati si effettuano prevalentemente, in maniera intervallata, operazioni di convoluzione e pooling, il quale consente la riduzione della loro dimensionalità. Le più comuni attività di pooling sono max e average pooling. Le CNNs hanno vasta applicabilità. Vengono utilizzate nel campo del riconoscimento vocale, delle immagini, applicazioni mediche e diversi task di modellazione del linguaggio naturale. Rappresenta una delle architetture più prestanti nel mondo del DL.

2.2.2 RNN - Recurrent Neural Network

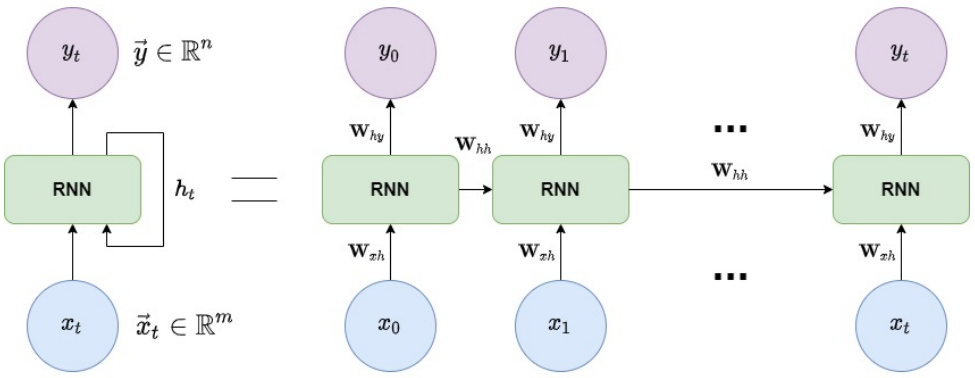

Figura 2.4: Architettura Recurrent Neural Network

Le RNN permettono di eseguire nel tempo operazioni su sequenze di vettori. Inoltre, sono dotate di memoria a lungo termine e mantengono l'informazione relative

all'ordine. Sono adatte a problemi di modellazione di sequenze quando queste sono dell'ordine delle centinaia. Il riquadro verde in Figura 2.4 rappresenta la RNN che riceve in input un vettore di m elementi e restituisce in output un vettore di n elementi. Inoltre, essa prende lo stato interno dell'istante precedente combinato con una matrice dei pesi e fornisce il suo stato interno dell'istante attuale alle RNN successive, ricombinandolo nuovamente con la stessa matrice. Anche input e output vengono combinate con le apposite matrici di pesi.

$$h_t = \tanh(\mathbf{W_{hh}^T} h_{t-1} + \mathbf{W_{xh}^T} x_t) \tag{2.6}$$

$$y_t = \mathbf{W_{hy}^T} h_t \tag{2.7}$$

L'equazione 2.6 evidenzia come viene aggiornato lo stato interno della rete ricorrente. La funzione di attivazione del network, in questo caso, è la tangente iperbolica che prende in input la somma della moltiplicazione della matrice dei pesi interna con lo stato relativo all'istante precedente e del prodotto tra matrice dei pesi tra input e RNN, e input attuale. Dalla 2.7 si può osservare come venga prodotto l'output, ossia con una moltiplicazione tra matrice dei pesi tra RNN e output, e lo stato attuale. La funzione di aggiornamento dello stato interno e tutte le matrici dei parametri del network rimangono immutate tra un istante di tempo e l'altro.

Backpropagation Through Time e problematiche

Per addestrare questa tipologia di reti, viene utilizzata la **BPTT** (**Backpropagation through time**). Il calcolo della perdita totale della rete viene effettuato sommando tutte le singole perdite. Per allenarla occorre applicare la tecnica della BPTT, che in questa tipologia di rete, viene effettuata per ogni istante di tempo e tra gli istanti di tempo. A causa delle ripetute computazioni del gradiente e delle conseguenti moltiplicazioni della matrice dello stato del network, nel momento in cui vi sono parametri molto maggiori o molto minori di uno, possono verificarsi, rispettivamente, le seguenti problematiche:

- **Gradiente esplodente**: ripetute moltiplicazioni della matrice dei pesi dello stato, causa la divergenza dell'algoritmo GDA, con conseguente impossibilità di ottimizzazione. Può essere risolto con la tecnica del **Gradient clipping**, con il quale si va a ridimensionare valori di gradiente elevati;

- **Gradiente evanescente**: allo stesso modo, il prodotto reiterato di numeri molto minori di uno causa la convergenza a zero del gradiente. Se quest'ultimo diventa troppo piccolo man mano che si effettua la backpropagation, i pesi relativi a istanti precedenti saranno corretti di una quantità insufficiente, tale da fermare il processo di ottimizzazione o da renderlo troppo lento. È possibile adottare tre soluzioni a questa problematica:
  - **Funzione di attivazione**: utilizzando particolari funzioni di attivazione, come ad esempio la ReLU (Rectified Linear Unit), è possibile evitare il restringimento del gradiente aumentandone il valore prima che inizi la sua eccessiva diminuzione;
  - **Inizializzazione dei parametri**: consiste nell'inizializzazione della matrice dei pesi e dei bias alla matrice identità;
  - **Gated cells**: vengono utilizzate delle unità ricorrenti più complesse che consentono di aggiungere informazioni rilevanti e rimuovere quelle irrilevanti dallo stato della RNN. Ciò viene effettuato per il tramite di particolari funzioni di attivazione e operazioni matematiche. Esempi di gated cells sono LSTM (2.2.3) e GRU (2.2.4).

2.2.3 LSTM - Long-Short Term Memory

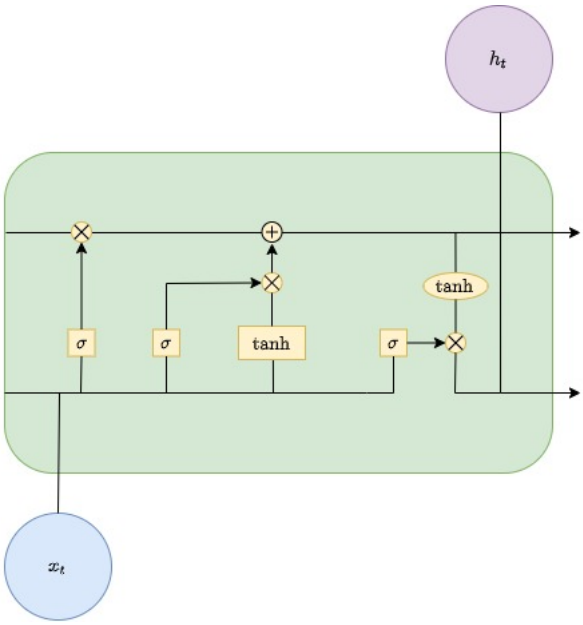

Figura 2.5: Architettura Long-Short Term Memory

La rete LSTM utilizza una unità di memoria chiamata cella che è costituita da tre porte, anche chiamate gates. Quest'ultime hanno le seguenti funzioni:

- La porta di input gestisce il flusso delle nuove informazioni che entrano nella memoria;
- La seconda porta è utilizzata quando una certa informazione viene dimenticata e aiuta la rete nella memorizzazione di nuovi dati;
- La terza porta controlla le informazioni presenti nel network che verranno fornite in output.

L'LSTM rappresenta un'alternativa alle RNN e la sua particolarità rispetto ad esse consiste nell'effetto mitigante contro la problematica del gradiente evanescente.

2.2.4 GRU - Gated Recurrent Cells

A differenza delle reti LSTM, questa versione di network ricorrente è formata da due porte:

- porta di aggiornamento: il suo compito è quello rivedere i contenuti dell'output della cella precedente;

- porta di ripristino: consente di portare all'interno della cella le informazioni di quella precedente e i nuovi input.

Le reti GRUs sono versioni più leggere delle RNNs e delle LSTM in termini di topologia, costo di computazione e complessità. Essendo appunto, leggera, richiede pochi parametri e dunque il suo addestramento risulta rapido. Se si ha a disposizione abbastanza dati e potenza computazionale è più conveniente utilizzare un'architettura LSTM.

2.2.5 Transformer network

Le RNNs non sono adatte per i problemi di modellazione di sequenze quando quest'ultime sono dell'ordine di mille o diecimila. Inoltre, esse non consentono nessuna parallelizzazione, non hanno nessuna memoria a lungo termine e fanno si che nel processo dell'encoding vi siano colli di bottiglia. I Transformer network nascono con la finalità di risolvere tutte le problematiche delle RNNs e si basano sul **meccanismo di attenzione**. Quest'ultimo imita il confronto di una query con delle chiavi contenute all'interno di un database, per poi estrarre un valore.

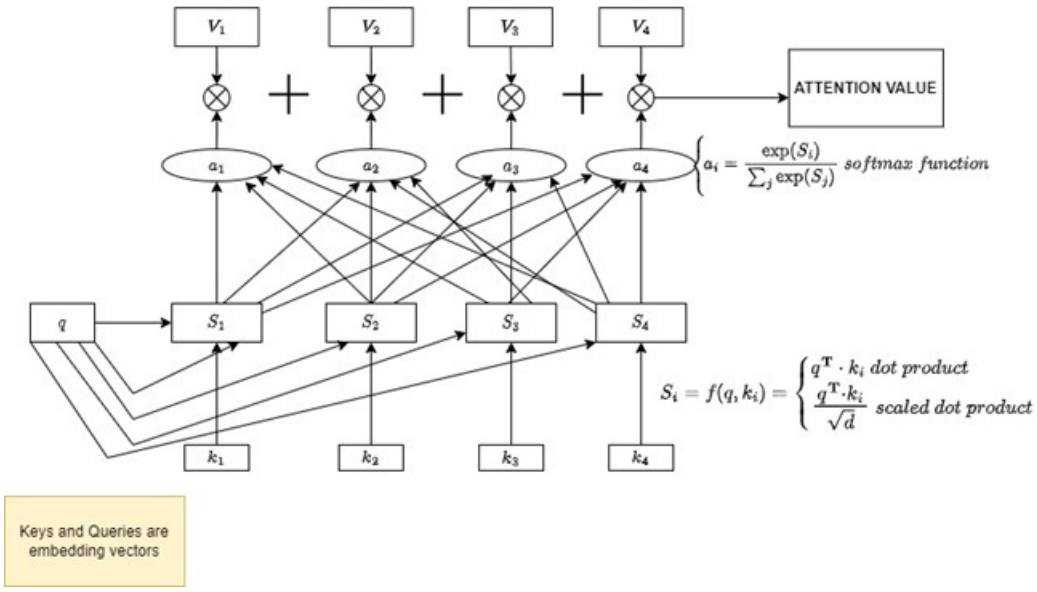

Figura 2.6: Funzionamento del meccanismo di attenzione

Il valore estratto, in questo caso, non è quello corrispondente alla chiave coincidente con la query ma è risultante dalla somma pesata dei valori corrispondenti alle chiavi più simili alla query richiesta. Dalla Figura 2.6, è possibile notare come

inizialmente si effettui la codifica dei vettori relativi alle chiavi e alla query. Come si vedrà in seguito, l'embedding consente di mappare una parola o un insieme di caratteri in un vettore di numeri reali che tiene conto della loro semantica. Successivamente, viene effettuato un prodotto scalare (che può essere di diversi tipi) per calcolare la somiglianza tra la query e le varie keys. Si avranno tanti scalari quante erano le keys da confrontare con la query. Dopodichè, la funzione softmax li utilizza per calcolare dei coefficienti che rappresentano dei pesi. Quest'ultimi vengono combinati con i corrispondenti valori per il tramite dell'operazione moltiplicazione e successivamente sommati per produrre il **valore di attenzione**.

L'architettura del modello Trasformer è costituita da un **encoder** e un **decoder**. In generale, l'encoder converte una sequenza di simboli $(x_1, ..., x_n)$ in una sequenza di rappresentazioni continue $\mathbf{z} = (z_1, ..., z_n)$. Noto $\mathbf{z}$, il decoder genera successivamente in output una sequenza di simboli $(y_1, ..., y_m)$, un elemento alla volta. Ad ogni passo, il simbolo generato precedentemente viene utilizzato come input addizionale nel momento in cui deve essere generato il successivo. I componenti sopra enunciati, nella particolare architettura sono implementati:

- **Encoder**: è costituito da una pila di sei livelli identici. Ogni livello ha due sottolivelli. Il primo è un meccanismo di multi-head self attention e il secondo è un semplice Feed Forward network. L'input dei due livelli viene riproposto nel successivo livello di addizione e normalizzazione.
- **Decoder**: anch'esso composto da una pila di sei livelli identici. Vi è un layer aggiuntivo rispetto all'encoder, che esegue una multi-head attention sull'output della pila di encoder. Nel layer di addizione e normalizzazione confluiscono gli input del layer e l'output del layer precedente. Vi è inoltre, un meccanismo di masked multi-head attention, che assicura che le predizioni per l'i-esima posizione possano dipendere soltanto da output relativi a posizioni precedenti.

2.3 Embeddings

I network neurali non hanno la capacità di intendere le parole. Essi sono degli operatori funzionali che vanno ad effettuare delle operazioni sugli inputs. Se si volesse tentare di rendere interpretabile il linguaggio naturale alla rete, occorrerebbe

trovare un modo di convertire le parole in numeri. Gli embedding permettono di codificare le parole in matrici o vettori di numeri, tale che il network possa operare matematicamente su di esse.

2.3.1 Modelli classici

I modelli classici sono sempre stati utilizzati in passato per la classificazione testuale. In questi metodi, un testo è trasformato in un vettore di dimensione fissata. Un esempio di metodologia è quello basato sulla frequenza delle parole, dove il vettore della singola parola è rappresentato dalla frequenza con cui essa appare in ogni documento. Di seguito si analizzeranno brevemente i metodi di rappresentazione categorica (Categorical representation method) e i metodi di rappresentazione pesata delle parole.

Categorical Word Representation

Figura 2.7: Illustrazione dei modelli One Hot e BoW

È la metodologia più semplice per rappresentare il testo. Le parole sono rappresentate come zero o uno. I principali metodi di rappresentazione categorica sono:

- **One Hot encoding**: i vettori di parola hanno dimensione pari al numero di parole contenute nel vocabolario (n). L'i-esima parola del vocabolario è rappresentata, quindi, con un vettore n-dimensionale, avente l'i-esimo valore posto a uno;
- **Bag of Words (BoW)**: BoW è un estensione del One Hot encoding. Esso aggiunge semplicemente una rappresentazione one-hot delle parole in una frase.

Weighted Word representation

I modelli di rappresentazione pesata delle parole provvedono una rappresentazione numerica delle parole basata sulla loro frequenza nei documenti. Essi sono:

- **Term Frequency (TF)**: calcola quante volte si ripete una determinata parola in un documento. La frequenza di apparizione della parola in un documento viene poi divisa in base al totale dei termini in esso contenuti;
- **Term Frequency-Inverse Document Frequency (TF-IDF)**: la metodologia in questione è stata introdotta per ridurre l'impatto dei termini che si presentano più frequentemente nelle frasi, come ad esempio congiunzioni e articoli. Il vettore TF-IDF di un termine t in un documento d è definito matematicamente come segue:

$$TF - IDF(t, d) = TF(t, d) \cdot IDF(t) \tag{2.8}$$

dove

$$TF(t, d) = \frac{frequency\ of\ t\ in\ d}{total\ number\ of\ terms\ space\ in\ d}$$
$$IDF(t) = \log(\frac{N}{n})$$

Non ha la capacità di mantenere l'ordine delle parole in un documento, il significato e le sue informazioni sintattiche.

2.3.2 Distributed Word Representation

I metodi classici hanno fallito nel catturare la sintattica e la semantica delle parole, oltre al fatto che soffrono del problema dell'elevata dimensionalità nella sua rappresentazione. I modelli classici sono stati sostituiti da metodi basati sulle reti neurali, in particolare, nell'area del NLP (Natural Language Processing), da algoritmi di apprendimento non supervisionato come i word embeddings. Per **word embedding** si intende "una funzione $V \to \mathcal{R}^D : w \to \vec{w}$ che associa ogni parola w di un vocabolario V a un vettore di valori continui $\vec{w}$ che appartiene a uno spazio di embedding di dimensione D" [15]. Essi si dividono in modelli non contestuali e contestuali. La differenza tra i due sta nel fatto che i secondi generano un vettore di embedding

differente per la stessa parola a seconda del contesto in cui essa è inserita, mentre i primi utilizzano sempre lo stesso vettore per ogni contesto. Per quanto riguarda le rappresentazioni contestuali, di seguito saranno elencate soltanto alcune delle tante metodologie.

Non Contextual Embeddings

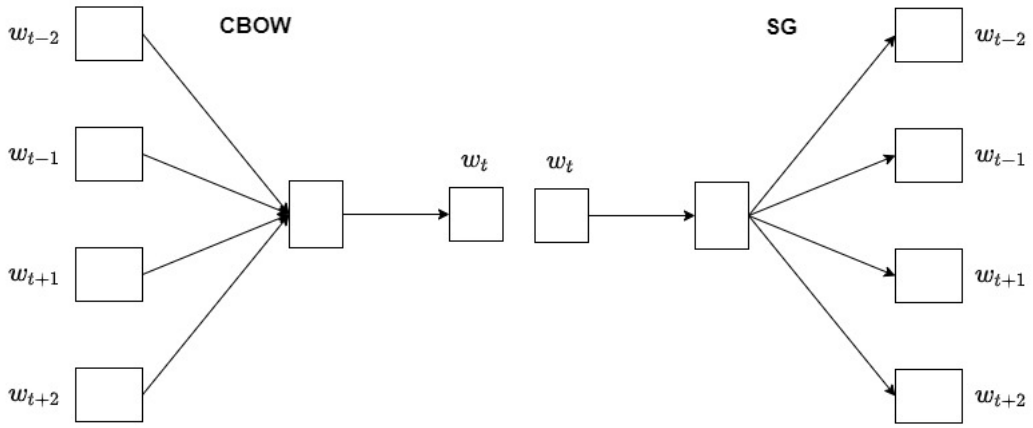

Figura 2.8: Architettura CBOW(sinistra) e SG(destra) [21]

I modelli principali utilizzati dalle varie tecniche per la generazione di embedding non contestuali sono:

- **CBOW (Contextual Bag of Words)**: costituito da tre livelli, predice la parola centrale della finestra di contesto basandosi su quelle circostanti;
- **SG (Skip Gram)**: rappresenta l'inverso del precedente modello. Ha lo stesso numero di livelli ma predice le parole circostanti prendendo in input la parola centrale della finestra di contesto;
- **Global Vector**: basato sulla costruzione di una matrice di co-occorrenza X_{ij} nella quale ogni elemento rappresenta la frequenza con cui la parola w_i compare accoppiata con w_J in una determinata finestra di contesto.

Contextual Embeddings

Alcuni dei modelli utilizzati dalle varie tecniche di generazione di embedding contestuali sono:

- **Context2Vec (Generic Context word representation)**: utilizza l'achitettura Bi-LSTM. Per la sua implementazione è stato utilizzato un ampio corpo di testo per imparare un modello neurale che incorpora il contesto della frase e delle parole target in uno spazio di bassa dimensione che successivamente viene ottimizzato per riflettere l'interdipendenza tra i termini selezionati e il relativo contesto;
- **CoVe (Contextualized word representations vectors)**: utilizza l'architettura Bi-LSTM in combinazione con i global vectors per ottenere una nuova rappresentazione;
- **ELMO (Embedding from language Model)**: anch'esso basato su architettura Bi-LSTM. Introduce una rappresentazione profonda delle parole in base al contesto, per risolvere il problema della natura dinamica dell'utilizzo delle parole nella semantica e nella grammatica.

Capitolo 3

Materiali e metodi

Il paper [7] si propone di migliorare l'architettura LSTM.MI [18] ponendole a monte embedding di n-grams pre-addestrati ottenuti mediante l'apprendimento non supervisionato di FastText utilizzando il modello Skip-gram. Nel momento in cui viene passato un nome di dominio all'architettura del lavoro citato, lo si converte in vettori di n-grams da fornire direttamente in input alla rete LSTM. Nel lavoro in questione invece, si vuole combinare i vettori di n-grams attraverso quattro operazioni illustrate nel paper [11]. I metodi di fusione di vettori di embedding che verranno utilizzati sono: somma, somma pesata, media, media pesata. Il vettore risultante rappresentante l'intero nome di dominio viene somministrato a un MLP che successivamente ne effettua la classificazione binaria.

3.1 Dataset

Il dataset utilizzato negli esperimenti è stato costruito attraverso la composizione di nomi di dominio di diverse classi di dga fornite dal dataset UMUDGA [23] che contiene 52 famiglie di nomi di dominio algoritmicamente generati. Il dataset impiegato è costituito da un totale di 100000 nomi di dominio, di cui la metà appartiene alla classe dei domini legittimi (legit) e l'altra metà a quella dei domini illegittimi (DGA). Per comporlo, sono state effettuate le seguenti operazioni:

- Lettura dei nomi di dominio: è stata effettuata la lettura del file 100000.txt contenuto all'interno del dataset UMUDGA per ottenere i nomi di dominio permessi, interrompendola nel momento in cui si tenta di leggere la riga n.50000.

Successivamente viene effettuata la lettura dei file 5000.txt dello stesso dataset per ogni famiglia di DGA, fermandola nel momento in cui si tenta di leggere la riga n. 1000;

- Formattazione e creazione: in questa fase viene creato e formattato il file .csv dal quale si attingerà per effettuare la classificazione.

Il .csv risultante dalle precedenti operazioni risulta organizzato secondo i seguenti campi:

- **dga/legit**: è un'etichetta che indica se il nome di dominio è lecito o è generato algoritmicamente;
- **legit/nome classe dga**: attributo che indica il nome della famiglia di malware che ha generato il nome di dominio se esso è illecito, altrimenti riporta il valore "legit";
- **Dominio senza punto**: colonna utilizzata per indicare il nome di dominio senza il punto che precede il TLD (Top Level Domain);
- **Dominio**: proprietà utilizzata per indicare il nome di dominio comprensivo di punto;
- **Dominio in monogrammi**: attributo utilizzato per indicare il nome di dominio senza punto come insieme di monogrammi suddivisi da spazi;
- **Dominio in bigrammi**: attributo utilizzato per indicare il nome di dominio senza punto come insieme di bigrammi suddivisi da spazi;
- **Dominio in trigrammi**: attributo utilizzato per indicare il nome di dominio senza punto come insieme di trigrammi suddivisi da spazi;

Prima di scrivere il file .csv è stato effettuato uno shuffle della lista che conteneva le sue righe. Come è possibile osservare dalla tabella 3.1, il dataset è bilanciato.

Dataset			
Legit / Famiglia di DGA	N. legit / N. per famiglia di DGA	Legit / Famiglia di DGA	N.legit / N. per famiglia di DGA
legit	50000	pizd	1000
alureon	1000	proslikefan	1000
banjori	1000	pushdo	1000
bedep	1000	pykspa	1000
ccleaner	1000	pykspa_noise	1000
chinad	1000	qadars	1000
corebot	1000	qakbot	1000
cryptolocker	1000	ramdo	1000
dircrypt	1000	ramnit	1000
dyre	1000	ranbyus_v1	1000
fobber_v1	1000	ranbyus_v2	1000
fobber_v2	1000	rovnix	1000
gozi_gpl	1000	shiotob	1000
gozi_luther	1000	simda	1000
gozi_nasa	1000	sisron	1000
gozi_rfc4343	1000	suppobox_1	1000
kraken_v1	1000	suppobox_2	1000
kraken_v2	1000	suppobox_3	1000
locky	1000	symmi	1000
matsnu	1000	tempedreve	1000
murofet_v1	1000	tinba	1000
murofet_v2	1000	vawtrak_v1	1000
murofet_v3	1000	vawtrak_v2	1000
necurs	1000	vawtrak_v3	1000
nymaim	1000	zeus-newgoz	1000
padcrypt	1000		

Tabella 3.1: Composizione del dataset

3.2 Implementazione

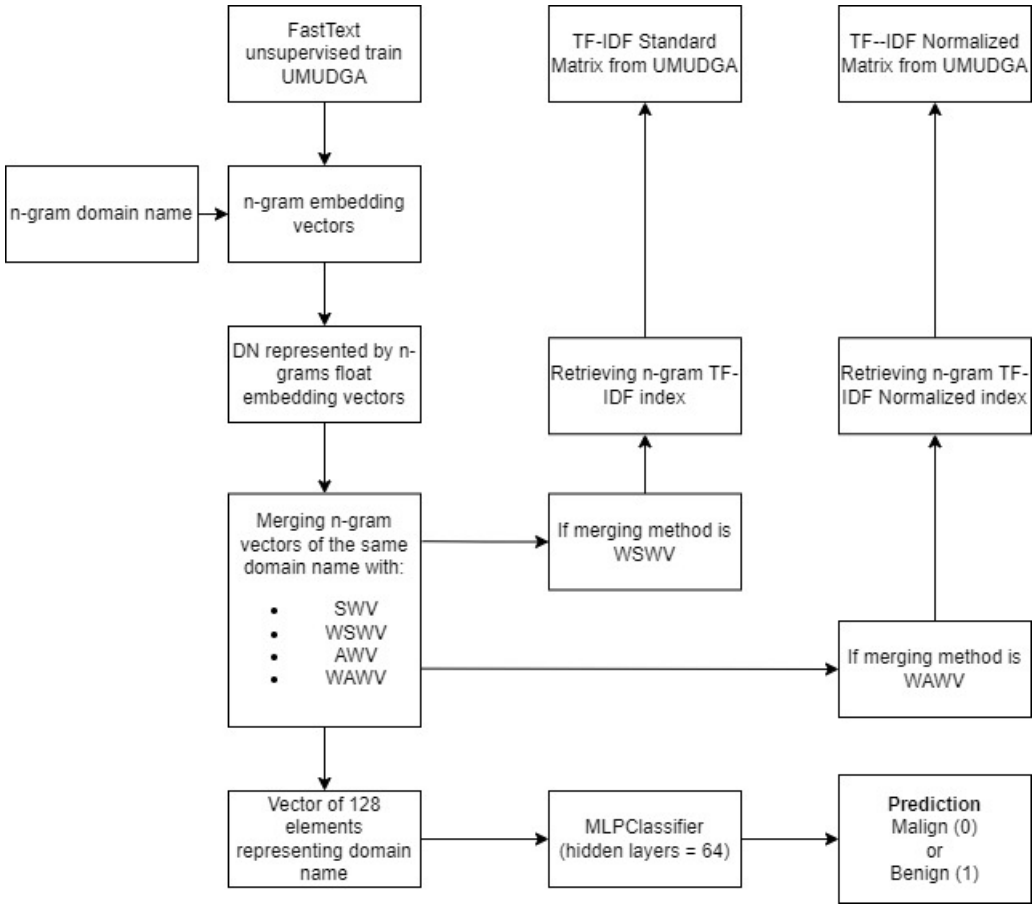

Figura 3.1: Architettura del sistema

3.2.1 FastText

La prima fase dell'implementazione riguarda l'addestramento non supervisionato di FastText utilizzando il modello SG (Skip-gram). L'addestramento è avvenuto con parametri di default, tranne per quanto riguarda maxn, minn e dim. I primi due sono campi che riguardano la lunghezza massima e minima degli n-grams. Dato che si vuole generare monogrammi, bigrammi e trigrammi, si è impostato i campi in questione rispettivamente a 3 e 1. Per quanto riguarda il campo dim, esso riguarda la dimensione del word vector generato da FastText che è stato quindi settato pari a 128. I dati forniti in pasto all'algoritmo sono quelli relativi alla colonna del nome di dominio senza punto del file .csv di cui si parla nel precedente paragrafo. Una volta che è avvenuta la suddivisione dei nomi di dominio in ngrams, si è scritto i

word vectors relativi ai monogrammi, bigrammi, trigrammi distinti all'interno dei rispettivi file .vec. Ad esempio, una riga del file monograms_trained.vec è formato dal monogramma e dopo uno spazio è presente il relativo vettore di embedding a esso associato.

Figura 3.2: Esempio di nome di dominio espresso in monogrammi, bigrammi, trigrammi

3.2.2 Matrice TF-IDF

La matrice TF-IDF illustra, per ogni nome di dominio (documento) l'indice term frequency inverse term frequency di ogni n-gram presente al suo interno. La matrice in questione viene calcolata per il tramite di un algoritmo separato rispetto a quello del classificatore. In particolare, le operazioni effettuate in esso sono le seguenti:

- Lettura dei dati: i dati vengono letti dal .csv, ottenendo soltanto le colonne relative ai nomi di dominio rappresentati con monogrammi, bigrammi, trigrammi.
- Formattazione: nel .csv gli n-gram relativi a un nome di dominio sono separati da spazi. Nell'algoritmo essi vengono elaborati in modo tale da avere una lista di nomi di dominio rappresentati come liste di n-gram.
- Processo di creazione matrice TF-IDF:
  - Creazione TF-IDF standard: viene contata la frequenza di ogni n-gram distinto in tutto il dataset e memorizzato all'interno di un dizionario. Ogni valore del dizionario viene poi sostituito con il logaritmo del rapporto tra

il numero totale di nomi di dominio e la frequenza di apparizione del singolo n-gram nel dataset. Successivamente viene contata la frequenza di apparizione di ogni n-gram in ogni nome di dominio, suddivisa per il numero di n-grams in esso contenuti e moltiplicata per il valore risultante dal logaritmo di cui sopra. Alla fine dell'operazione si avrà un documento che contiene un oggetto json per ogni nome di dominio. Quest'ultimo sarà rappresentato in n-gram e ogni n-gram avrà un suo indice term-frequency-inverse-term-frequency;

– Creazione TF-IDF normalizzata: il procedimento per il calcolo della matrice normalizzata è lo stesso di quella standard. L'unica differenza è che ogni indice term-frequency-inverse-term-frequency relativo a ogni n-gram del singolo nome di dominio viene diviso per la somma degli indici tf-idf degli n-gram dello stesso.

3.2.3 Metodi di fusione

Prima di poter applicare le tecniche di fusione, occorre associare a ogni monogramma, bigramma, trigramma di ogni nome di dominio, un embedding vector che è possibile recuperare dai file monograms_trained.vec, bigrams_trained.vec, trigrams_trained.vec. Successivamente vengono applicate le seguenti trasformazioni a ogni nome di dominio rappresentato in monogrammi, bigrammi, trigrammi:

- **SWV - Sum of Word Vectors**: viene effettuata la somma tra tutti i vettori di embedding associati a monogrammi, bigrammi, trigrammi per ogni nome di dominio;
- **WSWV - Weighted Sum of Word Vectors**: per ogni monogramma, bigramma, trigramma di ogni nome di dominio, viene preso il rispettivo embedding vector e moltiplicato per il relativo termine term-frequency inverse-term-frequency presente all'interno della matice TF-IDF. Dopodichè, i vettori di embedding pesati di monogrammi, bigrammi, trigrammi di ogni nome di dominio vengono sommati tra loro.
- **AWV - Average of Word Vectors**: viene effettuata la media tra tutti i vettori di embedding associati a monogrammi, bigrammi, trigrammi per ogni

nome di dominio;

- **WAWV - Weighted Average of Word Vectors**: per ogni monogramma, bigramma, trigramma di ogni nome di dominio, viene preso il rispettivo embedding vector e moltiplicato per il relativo termine term-frequency inverse-term-frequency presente all'interno della matrice TF-IDF normalizzata. Successivamente, viene effettuata la media dei vettori di embedding pesati di monogrammi, bigrammi, trigrammi relativi a ogni nome di dominio.

Se un nome di dominio può essere rappresentato come una sequenza di vettori di embedding di monogrammi, bigrammi, trigrammi, attraverso questi metodi di unione essi vanno a sommarsi-mediarsi in maniera pesata o non, in modo da ottenere alla fine un vettore da 128 elementi per ogni nome di dominio. Al termine di questa fase si avranno quindi quattro insiemi di vettori, ognuno rappresentante un nome di dominio, relativi ai quattro metodi di fusione.

3.2.4 MLP - Multi Layer Perceptron

I vettori da 128 elementi vengono forniti in input al classificatore MLPClassifier, implementazione della libreria open source sklearn[1]. Ad esso vengono passati parametri personalizzati per quanto riguarda il numero di hidden layer, funzione di attivazione e random_state. Il numero degli hidden layer è pari uno che ha dimensione 64. La funzione di attivazione utilizzata è la sigmoide. Il parametro random_state rappresenta l'inizializzazione randomica di pesi e bias. Ponendo quest'ultima a un numero intero, la si tiene costante per ogni chiamata del classificatore. L'algoritmo di ottimizzazione utilizzato è Adam con learning rate pari a 0.001 e iterazioni massime (Epochs) pari a 200. L'addestramento e suddivisione del dataset in test set e training set viene effettuato per il tramite di una KFold con $k = 5$. Il dataset viene equamente suddiviso tra dga e legit in ogni iterazione. Al termine dell'addestramento, ad ogni ciclo, viene effettuata la predizione su di un sottogruppo di 20000 nomi di dominio del dataset. Il nome di dominio è considerato maligno se etichettato come 0, altrimenti benigno.

[1] https://scikit-learn.org/stable/

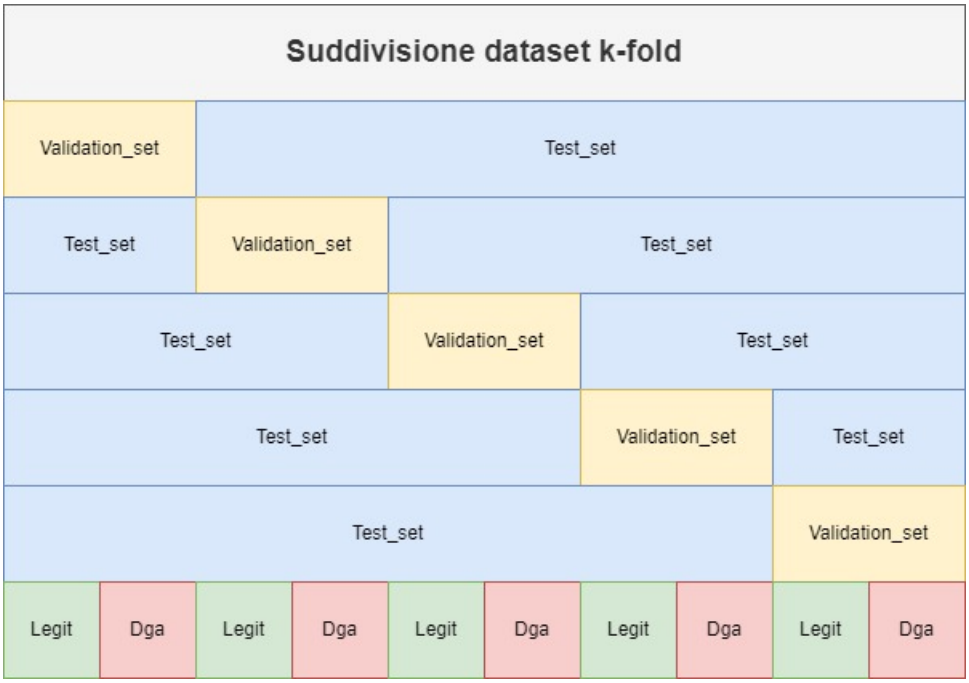

Figura 3.3: Suddivisione dataset

3.3 Parametri di valutazione

Per valutare le prestazioni dei vari metodi di fusione verranno utilizzati le seguenti metriche: Accuracy, Precision, Recall e F1 Score.

3.3.1 Accuracy

L'accuracy misura il numero delle predizioni corrette delle due classi rispetto al totale delle predizioni.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

3.3.2 Precision

La precision calcola il rapporto fra gli elementi che sono stati classificati in maniera corretta come positivi e il totale degli elementi classificati come positivi.

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

3.3.3 Recall

La recall determina il rapporto fra gli elementi classificati in maniera corretta come positivi e tutti gli elementi positivi del dataset.

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

3.3.4 F1 Score

L'F1 Score rappresenta una media armonica di precision e recall.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

Per ogni metrica è possibile calcolare la **macro-average**. Quest'ultima consiste nel calcolo della metrica in maniera indipendente per ciascuna classe per poi fare la media. Si prende in considerazione questa tipologia di media perchè il dataset è bilanciato.

Capitolo 4

Risultati sperimentali

In questo capitolo si illustreranno le prestazioni relative ai quattro metodi di fusione di vettori di embedding provati. Le performance verranno illustrate tramite grafici e tabelle. Sarà possibile inoltre visionare come il metodo di fusione somma garantisca prestazioni migliori rispetto agli altri.

4.1 Risultati monogrammi

A primo impatto è possibile osservare dalle Figure 4.1, 4.2, 4.3 e 4.4 che il metodo più prestante per quanto riguarda i nomi di dominio rappresentati con monogrammi, è quello basato sulla somma mentre il secondo più performante è basato sulla somma pesata. I metodi basati su somma e somma pesata raggiungono il picco nella quarta fold ed è possibile visualizzarlo numericamente all'interno della tabella 4.1 e 4.2.

k-fold	SWV			WSWV			AWV			WAWV		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
1	0.872	0.873	0.872	0.851	0.852	0.851	0.797	0.797	0.797	0.786	0.786	0.786
2	0.874	0.875	0.874	0.851	0.853	0.851	0.796	0.796	0.798	0.785	0.785	0.785
3	0.879	0.879	0.878	0.848	0.851	0.852	0.799	0.800	0.798	0.785	0.785	0.785
4	0.882	0.883	0.882	0.852	0.853	0.852	0.798	0.799	0.798	0.792	0.793	0.792
5	0.876	0.878	0.876	0.850	0.851	0.850	0.796	0.798	0.796	0.792	0.794	0.791

Tabella 4.1: Precision Macro, Recall Macro e F1 Macro relativi ai monogrammi per ogni metodo di fusione

k-fold	SWV Accuracy	WSWV Accuracy	AWV Accuracy	WAWV Accuracy
1	0.872	0.852	0.797	0.786
2	0.874	0.851	0.796	0.785
3	0.878	0.848	0.798	0.785
4	0.882	0.852	0.798	0.792
5	0.876	0.850	0.796	0.792

Tabella 4.2: Accuracy relativa ai monogrammi per ogni metodo di fusione

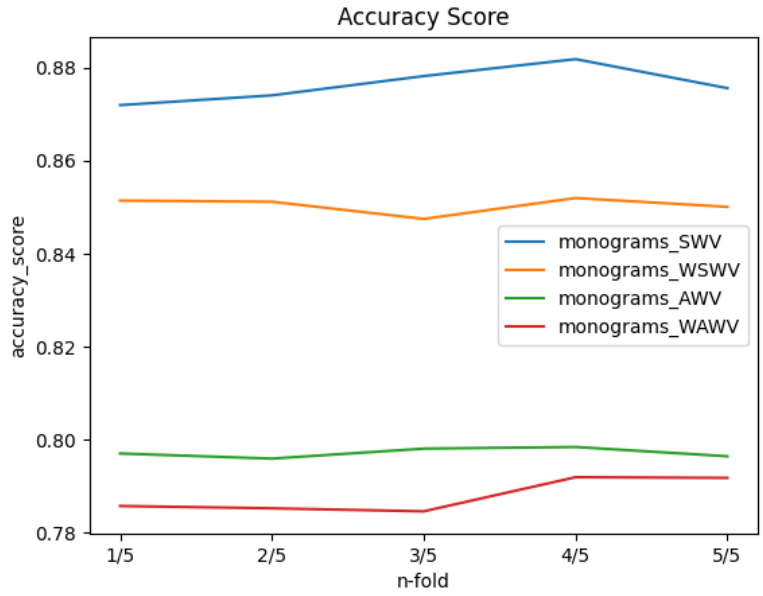

Figura 4.1: Accuracy Score Monogrammi

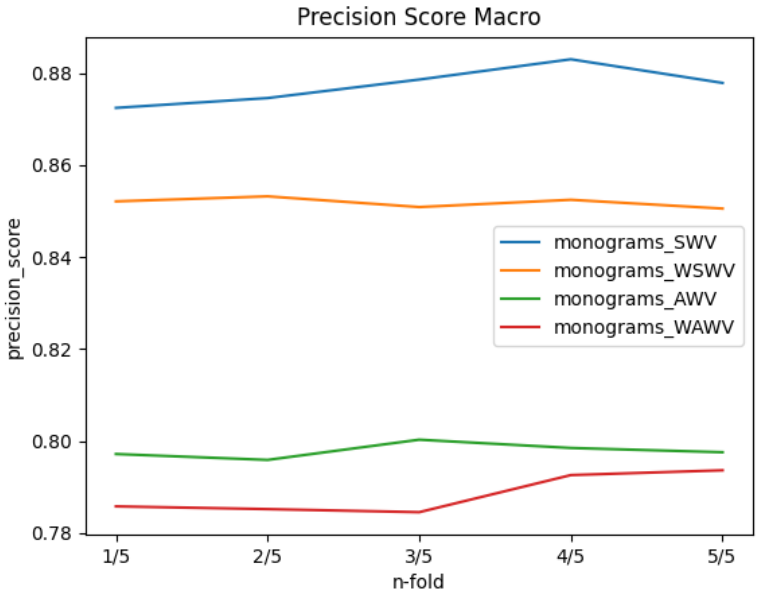

Figura 4.2: Precision Score Macro Monogrammi

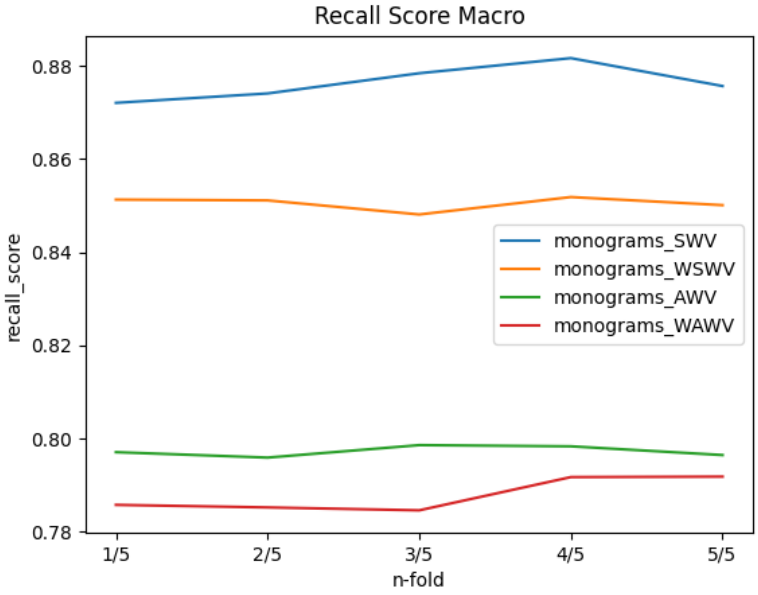

Figura 4.3: Recall Score Macro Monogrammi

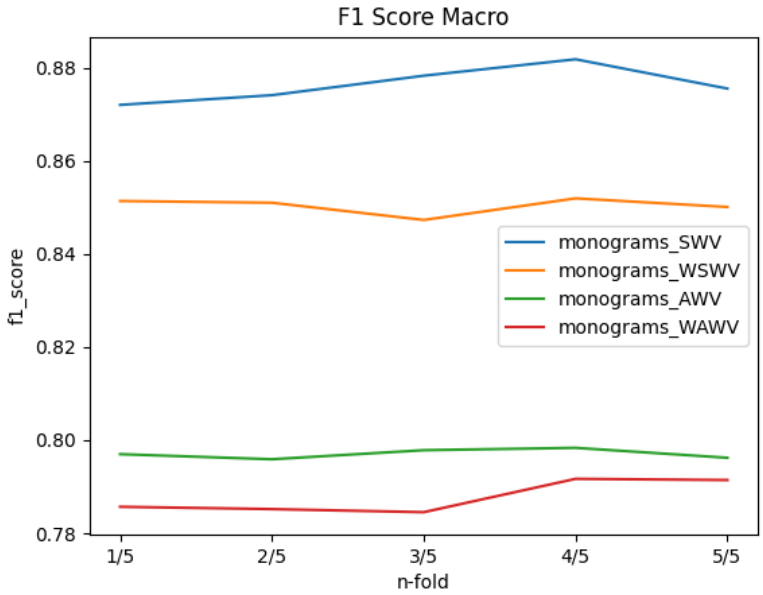

Figura 4.4: F1 Score Macro Monogrammi

4.2 Risultati bigrammi

Anche per quanto riguarda i bigrammi è possibile osservare nelle figure 4.5, 4.6, 4.7 e 4.8 come i metodi basati sulla somma siano i più prestanti, in particolar modo la somma non pesata. Il picco per i metodi somma viene sempre raggiunto nella quarta fold. Per visualizzare precisamente i valori delle metriche nelle varie fold è possibile fare riferimento alle tabelle 4.3 e 4.4.

k-fold	SWV			WSWV			AWV			WAWV		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
1	0.881	0.885	0.881	0.850	0.851	0.850	0.808	0.809	0.808	0.822	0.822	0.822
2	0.879	0.879	0.879	0.847	0.848	0.847	0.814	0.814	0.814	0.827	0.827	0.827
3	0.884	0.885	0.884	0.852	0.852	0.852	0.817	0.818	0.816	0.826	0.825	0.825
4	0.884	0.884	0.884	0.852	0.854	0.852	0.815	0.816	0.816	0.830	0.830	0.830
5	0.885	0.886	0.885	0.852	0.854	0.852	0.816	0.818	0.816	0.827	0.829	0.827

Tabella 4.3: Accuracy relativa ai bigrammi per ogni metodo di fusione

k-fold	SWV Accuracy	WSWV Accuracy	AWV Accuracy	WAWV Accuracy
1	0.881	0.850	0.808	0.822
2	0.879	0.847	0.814	0.827
3	0.884	0.852	0.816	0.826
4	0.884	0.852	0.816	0.830
5	0.885	0.852	0.816	0.827

Tabella 4.4: Accuracy relativa ai bigrammi per ogni metodo di fusione

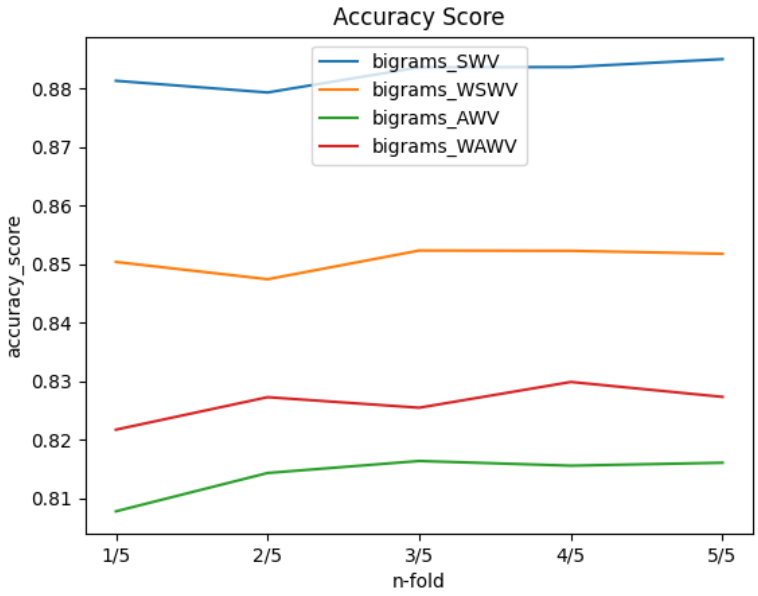

Figura 4.5: Accuracy Score Bigrammi

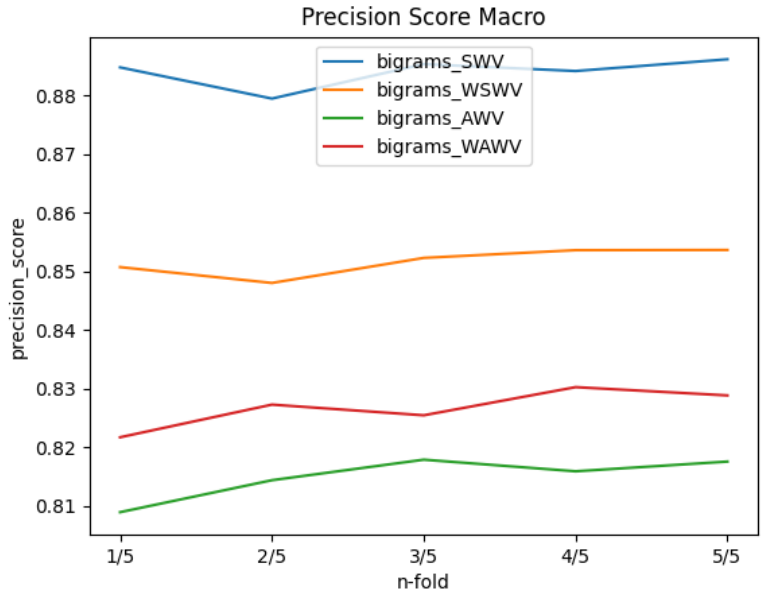

Figura 4.6: Precision Score Macro Bigrammi

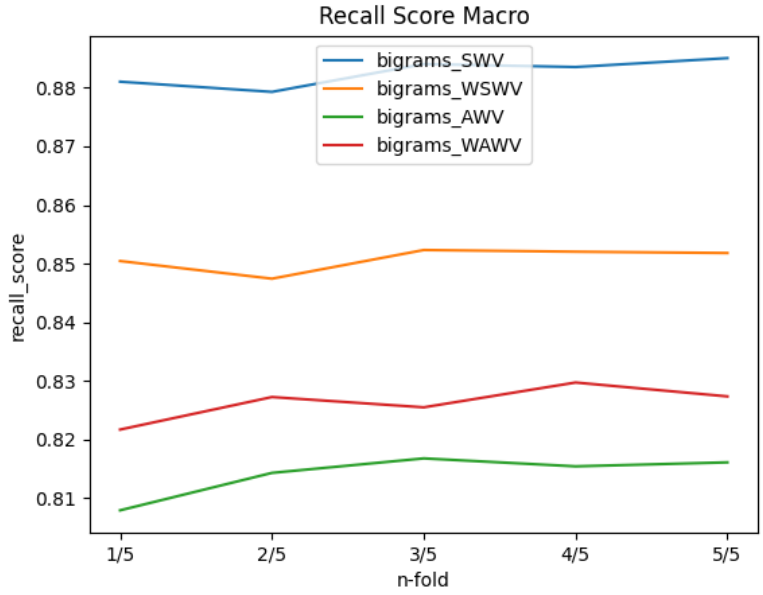

Figura 4.7: Recall Score Macro Bigrammi

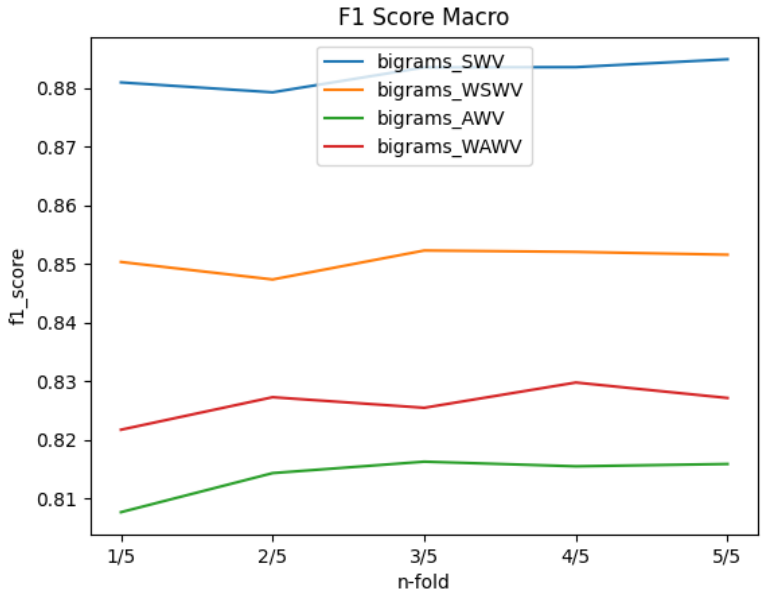

Figura 4.8: F1 Score Macro Bigrammi

4.3 Risultati trigrammi

In ultimo è possibile osservare dai grafici 4.9, 4.10, 4.11 e 4.12 come ancora una volta il metodo somma si rivela il più prestante. Il metodo basato su somma pesata raggiunge questa volta performance minori rispetto al caso del nome di dominio rappresentato con bigrammi, tanto che nella terza fold viene superato in tre metriche su quattro dal metodo di fusione basato sulla somma pesata. Nelle altre iterazioni la somma pesata rimane comunque un metodo di fusione migliore rispetto alla media pesata. Le tabelle alle quali fare riferimento per quanto riguarda i trigrammi sono le 4.5 e 4.6.

k-fold	SWV			WSWV			AWV			WAWV		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
1	0.880	0.885	0.880	0.841	0.842	0.841	0.814	0.815	0.813	0.830	0.830	0.830
2	0.883	0.884	0.883	0.839	0.842	0.838	0.820	0.820	0.820	0.839	0.839	0.839
3	0.886	0.887	0.885	0.828	0.838	0.828	0.824	0.825	0.824	0.836	0.836	0.836
4	0.886	0.886	0.886	0.846	0.846	0.846	0.819	0.820	0.819	0.837	0.837	0.837
5	0.887	0.887	0.887	0.844	0.848	0.844	0.820	0.822	0.820	0.836	0.837	0.836

Tabella 4.5: Precision Macro, Recall Macro e F1 Macro relative ai trigrammi per ogni metodo di fusione

k-fold	SWV Accuracy	WSWV Accuracy	AWV Accuracy	WAWV Accuracy
1	0.880	0.841	0.813	0.830
2	0.883	0.839	0.820	0.839
3	0.885	0.829	0.824	0.836
4	0.886	0.846	0.820	0.837
5	0.887	0.844	0.820	0.836

Tabella 4.6: Accuracy relativa ai trigrammi per ogni metodo di fusione

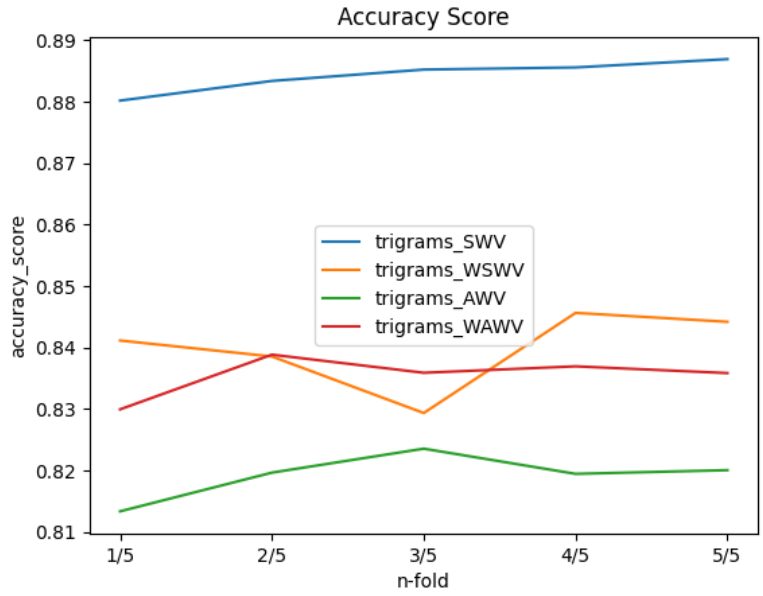

Figura 4.9: Accuracy Score Trigrammi

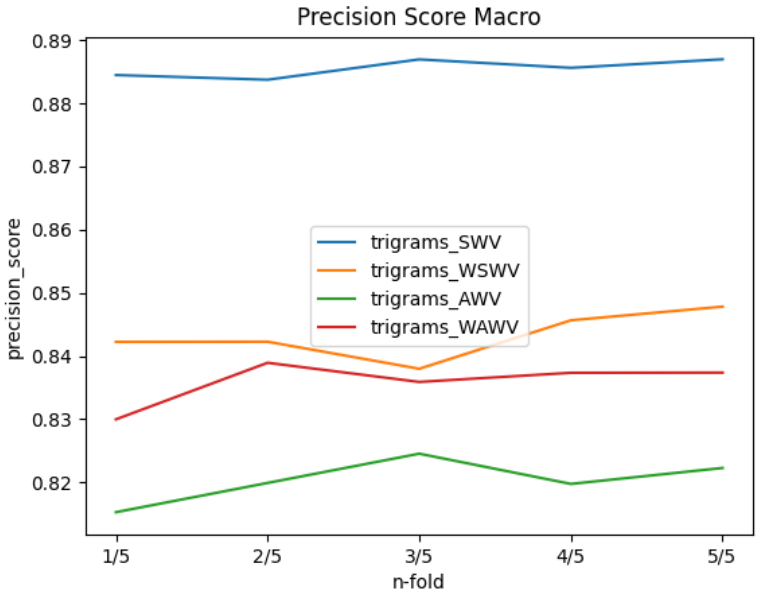

Figura 4.10: Precision Score Macro Trigrammi

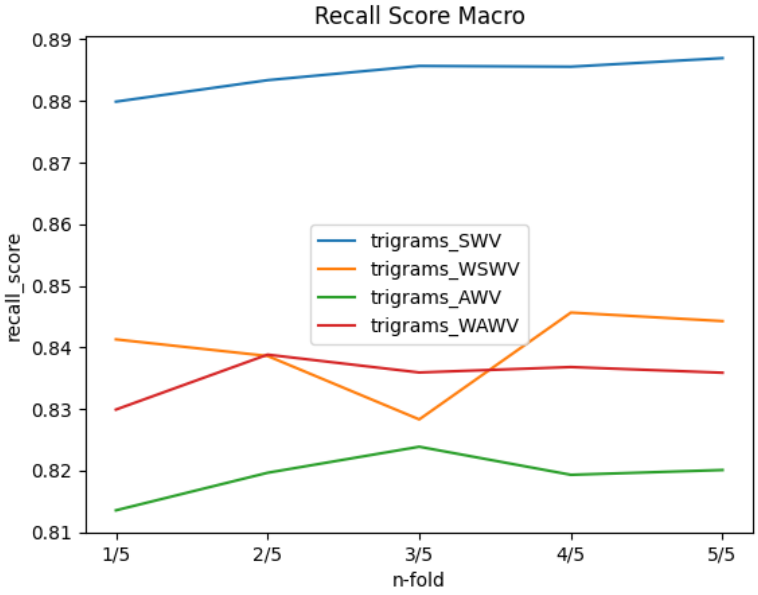

Figura 4.11: Recall Score Macro Trigrammi

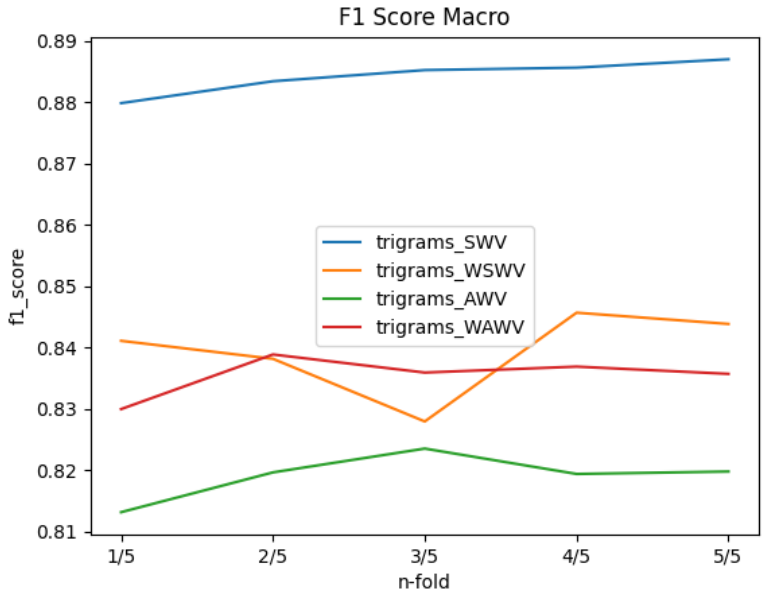

Figura 4.12: F1 Score Macro Trigrammi

n-gram	SWV				WSWV				AWV				WAWV			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
1-gram	0.876	0.877	0.878	0.876	0.851	0.850	0.852	0.851	0.797	0.797	0.798	0.797	0.788	0.788	0.789	0.788
2-gram	0.883	0.883	0.884	0.883	0.851	0.851	0.852	0.851	0.814	0.815	0.815	0.814	0.826	0.826	0.827	0.826
3-gram	0.884	0.884	0.886	0.884	0.840	0.840	0.843	0.839	0.819	0.819	0.820	0.819	0.836	0.836	0.836	0.836

Tabella 4.7: Confronto della media di tutte le metriche su 5-fold di tutti gli n-gram

Conclusioni

Nel seguente elaborato si è illustrato i risultati delle diverse tecniche di combinazione dei vettori di embedding come alternativa al dare in pasto al classificatore sequenze di vettori di embedding relativi a n-grams dello stesso nome di dominio.

I metodi che garantiscono prestazioni maggiori sono risultati essere quelli basati sulla somma e somma pesata dei vettori di embedding.

Potrebbe essere interessante effettuare un esperimento analogo utilizzando un multi classificatore MLP oppure qualche altra rete neurale e confrontarne le prestazioni. Altra variazione che potrebbe essere utile fare è variare la dimensione dei vettori di embedding relativi a n-grams variando il parametro dim di FastText. Vi sono inoltre dei parametri come learning rate e epochs di quest'ultimo che sono state lasciate impostate ai valori di default e dunque eventualmente si potrebbe impostare diversamente anche quest'ultimi.

Bibliografia

[1] Md Zahangir Alom et al. «A State-of-the-Art Survey on Deep Learning Theory and Architectures». *Electronics* (2019).

[2] H.S. Anderson, J. Woodbridge e B. Filar. «DeepDGA: Adversarially-tuned domain generation and detection». *AISec 2016 - Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2016* (2016).

[3] ENISA - European Union Agency for CyberSecurity. «ENISA Threat Landscape Report 2021» (2021), p. 45. DOI: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021/@@download/fullReport.

[4] Shaveta Dargan, Munish Kimar, Maruthi Rohit Ayyagari e Gulshan Kumar. «A Survey of Deep Learning and Its Applications: A New paradigm to Machine Learning» (2019).

[5] F.Ren, Z. Jiang, X. Wang e J. Liu. «A DGA domain names detection modeling method based on integrating an attention mechanism and deep neural network». *Cybersecurity 3* (2020).

[6] Faiza Khan Khattak et al. «A survey of word embeddings for clinical text». *Journal of Biomedical Informatics* (2019).

[7] Spalazzi L., Cucchiarelli A., Morbidoni C. e Teti A. «Leveraging n-gram neural embeddings to improve deep learning DGA detection». *The 37th ACM/SIGAPP Symposium on Applied Computing* (2022). DOI: https://doi.org/10.1145/3477314.3507269.

[8] Warren S. McCulloch e Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». *The bulletin of mathematical biophysics* (1943). DOI: https://doi.org/10.1007/BF02478259.

[9] Usman Naseem, Imran Razzak, Shah Khalid Khan e Mukesh Prasad. «A Comprehensive Survey on Word Representation Models: From Classical to State-of-the-Art Word Representation Language Models». *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* (2021).

[10] P.Lison e V.Mavroeidis. «Automatic detection of malware-generated domains with recurrent neural models.» (2017).

[11] Marcelo Pita e Gisele L. Pappa. «Strategies for Short Text Representation in the Word Vector Space». *Brazilian Conference on Intelligent Systems* (2018). DOI: https://doi.org/10.1109/BRACIS.2018.00053.

[12] Daniel Plohmann, Khaled Yakdan, Micheal Klatt e Elmar Gerhards-Padilla. «A Comprehensive Measurement Study of Domain Generating Malware». *25th USENIX Security Symposium* (2016).

[13] Rafael A. Rodríguez-Gómez, Gabriel Maciá-Fernández e Pedro García-Teodoro. «Survey and Taxonomy of Botnet Research through Life-Cycle». *ACM Computing Surveys* (2013).

[14] Stuart Russel e Peter Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 2009.

[15] Tobias Schnabel, Igor Labutov, David Mimno e Thorsten Joachims. «Evaluation methods for unsupervised word embeddings». *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics* (2015).

[16] Sérgio S.C. Silva, Rodrigo M.P. Silva e Ronaldo M. Salles Raquel C.G. Pinto. «Botnets: A survey». *Computer Networks* (2013).

[17] Raaghavi Sivaguru et al. «An Evaluation of DGA classifiers». *IEEE International Conference on Big Data (Big Data)* (2018).

[18] Duc Tran et al. «A LSTM based framework for handling multiclass imbalance in DGA botnet detection». *Neurocomputing* (2017). DOI: https://doi.org/10.1016/j.neucom.2017.11.018.

[19] Ashish Vaswani et al. «Attention Is All You Need». *31st Conference on Neural Information Processing Systems (NIPS)* (2017).

[20] Simon Nam Than Vu et al. «A survey on Botnets: Incentives, Evolution, Detection and Current Trends». *Future Internet* (2021).

[21] Shirui Wang, Wenan Zhou e Chao Jiang. «A survey of word embeddings based on deep learning». *Springer Nature* (2020).

[22] Y.Li, K.Xiong, T.Chin e C.Hu. «A Machine Learning Framework for Domain Generation Algorithm-Based Malware Detection». *IEEE Access 7* (2019).

[23] Mattia Zago, Manuel Gil Pérez e Gregorio Martínez Pérez. «UMUDGA: A dataset for profiling DGA-based botnet». *Computers & Security* (2020). DOI: https://doi.org/10.1016/j.cose.2020.101719.

Sitografia

[24] Alexander Amini. *MIT - Introduction to Deep Learning*. URL: https://www.youtube.com/watch?v=7sB052Pz0sQ.

[25] Young Entrepreuner Council - Forbes. *Cybersecurity Trends for 2022: Why staying ahead of the threat has never been so critical*. URL: https://www.forbes.com/sites/theyec/2022/02/01/cybersecurity-trends-for-2022-why-staying-ahead-of-the-threat-has-never-been-so-critical/?sh=5dd31ffa8b08. (accessed: 01.09.2016).

[26] *Scikit learn - Machine Learning in python*. URL: https://scikit-learn.org/stable/.

[27] Ava Soleimany. *MIT - Recurrent Neural Networks and Transformers*. URL: https://www.youtube.com/watch?v=QvkQ1B3FBqA.