D2.1.2 Progettazione e sviluppo di un modulo prototipale di rilevamento minacce su dispositivi e sensori nel campo.

(Namirial+UNIVPM)

**Formato di un *nome di dominio***

Un URL ha il seguente formato:

https://**www.ingegneria.univpm.it**

La parte in grassetto è quello che viene chiamato dominio.

Un dominio ha le seguenti caratteristiche:

* ha lunghezza massima di 253 caratteri;
* può contenere caratteri alfanumerici e il trattino [-] e l’underscore [\_];
* non vi è distinzione tra caratteri maiuscoli e minuscoli;
* ogni punto [.] separa due stringhe chiamate *label*;
* ciascuna label può avere lunghezza massima di 63 caratteri;
* non vi è un limite massimo al numero di label fintanto che non viene raggiunta la lunghezza massima di 253 caratteri (punti compresi);
* il trattino [.] e l’underscore [\_] non possono trovarsi né come primo né come ultimo carattere di una label.

Le varie label vengono così definite:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ... | corsi | . | ingegneria | . | univpm | . | it |
| ... | fourth-level[[1]](#footnote-2) | . | third-level | . | second-level | . | top-level (TLD) |

**Suffisso o eTLD**

Con suffisso si intende *“the part of a domain name which is not under the control of the individual registrant”* [1], ovvero la parte del nome di dominio che non può essere scelta arbitrariamente ma che deve fare parte di una lista definita dalle autorità. Esso viene anche chiamato *“effective top-level domain(eTLD)”.*È molto importante distinguere tra Top-Level-Domain (TLD) ed effective-Top-Level-Domain (eTLD) in quanto, dato un nome di dominio, è importante sapere quali label sono arbitrarie o meno.

**Definizioni**

* Nome di dominio o DN: ovvero l’intera stringa che rappresenta un dominio, sottodomini compresi. Es: **gmail.google.com**
* Label: ovvero ciascuna sottostringa separata dai punti, es: **gmail**, **google**, **com**
* Nome di dominio *mono-label*:ovvero quei DN composti, escludendo l’eTLD, da una sola label. Es: google.com, oxford.ac.uk.

**LSTM**

Le reti LSTM, Long Short-Term Memory, sono delle reti neurali aventi architettura RNN. Esse sono lo stato dell’arte per predizioni su sequenze temporali, quali ad esempio la scrittura e il linguaggio. Ad esempio, allenando queste reti con i testi estratti da dei libri, esse sono in grado di generare automaticamente delle frasi di senso compiuto. Similmente alle sequenze testuali, esse hanno ottenuto eccellenti risultati anche nel riconoscimento vocale. ~~Per cuiEsse hanno ottenuto buoni risultati anche nel riconoscere se un dominio è stato o meno generato da DGA.~~  
Esse sono nate in quanto le reti RNN normali non sono in grado di ricordare correttamente il contesto di una determinata sequenza, tanto meno quanto più la sequenza è lunga. Ad esempio, la frase “sono cresciuto in Francia [...] parlo bene il (?).”, mostra come la parola mancante sia probabilmente un linguaggio (*francese*), ma per risalire a quale linguaggio ho bisogno di conoscere il contesto, ovvero di memorizzare in qualche modo ciò che viene detto nella prima frase, cioè che sono cresciuto in Francia. Le reti RNN non erano in grado di capire come memorizzare un contesto, né come modificarlo aggiungendo o dimenticando informazioni, o più precisamente, esse memorizzavano un contesto, ma senza quei particolari accorgimenti che permettevano di mantenere o meno le informazioni utili, a causa della sostituzione delle informazioni contestuali troppo vecchie con quelle più nuove.  
Si guardi alla Figura 1 relativa alla frasi “le nuvole si trovano nel (?)”, dove ogni parola è un input di ciascuna cella, e le variabili indicano il contesto.

Essendo la frase relativamente corta, la rete neurale è in grado di effettuare una correlazione tra lo stato ottenuto dalla parola *nuvole* e lo stato relativo invece a *trovare,* la quale può permettergli di predire la parola corretta, ovvero *cielo*.

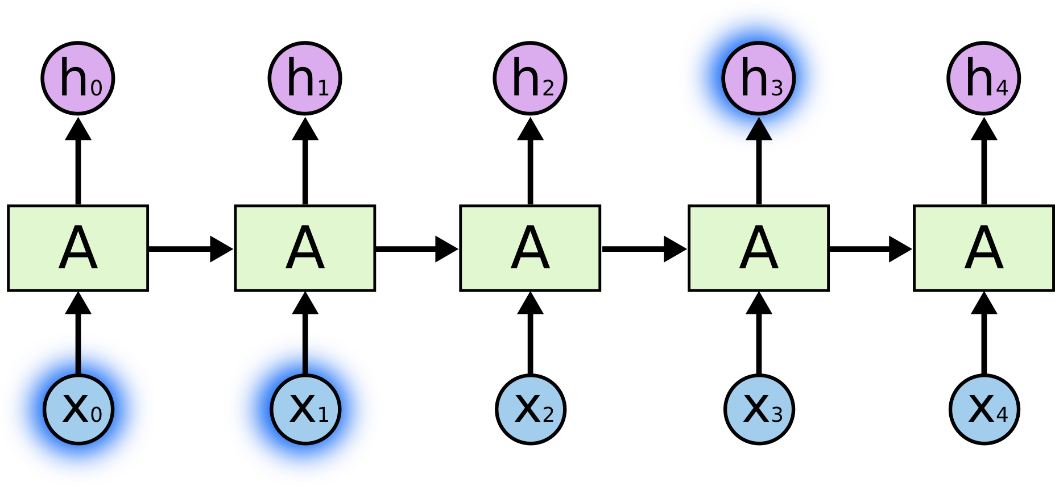


Figura 1: Dipendenza a breve termine di una rete RNN.

Nel caso invece della Figura 2, riprendendo la frase “*sono cresciuto in Francia [...] parlo bene il (?).*”, la correlazione tra il primo contesto e l’ultimo viene persa tanto più quanto aumenta la distanza tra le due frasi.

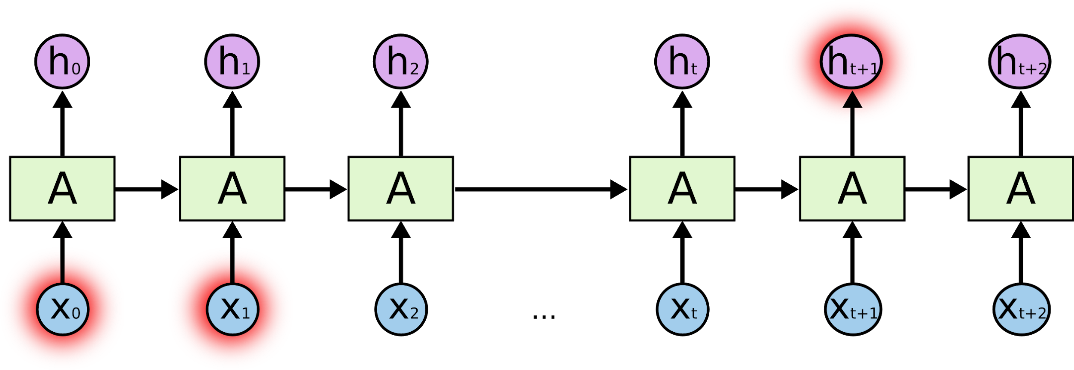


Figura 2: Dipendenza a lungo termine di una rete RNN.

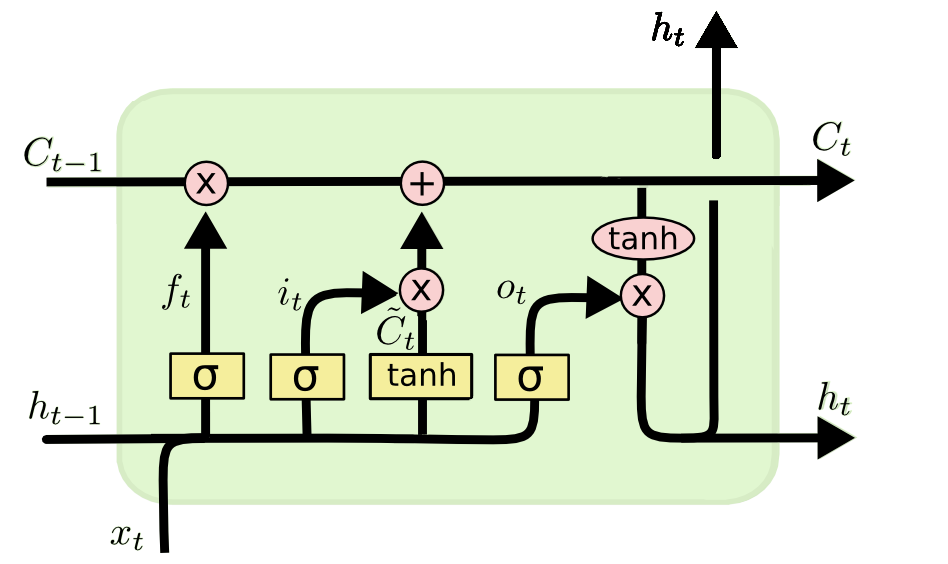
Il problema di ricordarsi informazioni contestuali a lungo termine è esattamente l’obiettivo prefissato delle reti LSTM. Una rete LSTM è una sequenza di celle (anche dette unità o moduli) come quelle mostrate in Figura 3.

Figura 3: Cella LSTM.

Le reti LSTM hanno introdotto il concetto di *gates*, ovvero delle particolari funzioni presenti in ogni modulo le quali hanno come compito l’estrazione di particolari informazioni per ciascun input, ed il concetto di cell state . Attenzione, l’output di ciascuna cella è definito con , da *hidden state*, allo steso modo delle reti RNN, ma come accennato è aggiunto un nuovo stato chiamato *cell state*. I gates hanno il compito di scegliere quali informazioni mantenere dello stato precedente e quali aggiungere. Ogni *gate* a seconda dell’implementazione LSTM può dipendere dall’input attuale , dall’output precedente e dallo stato precedente . Per ogni gate in fase di allenamento vengono calcolati quei parametri di ottimizzazione chiamati pesi. In figura [3](#bookmark=id.2et92p0) è possibile osservare una cella LSTM, dove i gates sono identificabili nelle lettere , , , e le operazioni di aggiornamento nei vari punti di addizione e moltiplicazione.

Queste funzioni possono essere definite come segue:

dove e sono i pesi allenati.

## **Input della rete LSTM**

Gli input della rete LSTM sono dei vettori di *n-*elementi dove ogni elemento è una entità unica appartentente ad un così detto *vocabolario*. Tale nome deriva dal frequente utilizzo di tali reti in applicazioni che hanno lo scopo di effettuare delle predizioni su input testuali. Negli esempi precedenti, dove si doveva predire quale fosse la parola mancante, le unità minime erano quindi le parole.

Dato che nel nostro caso gli input non sono frasi o testi bensí nomi di dominio, l’entità minima non sarà più una parola ma un singolo carattere. Sarebbe più corretto quindi *alfabeto*, ma per mantere la nomenclatura comune verrà utilizzato il termine *vocabolario[[2]](#footnote-3)*.

Naturalmente le reti neurali lavorano con numeri e non con lettere, per cui vi è il problema di convertire le entità del vocabolario in un formato compatibile con la rete.

Con  verrà indicato il vocabolario, ovvero un insieme di lettere.

*L’input nel nostro caso*

Esso è un nome di dominio, ovvero una sequenza di caratteri. Ogni carattere verrà quindi trasformato in un formato compatibile con la rete LSTM, ovvero subirà un processo di *codifica*. Dato però che gli input hanno lunghezza fissa, è necessario stabilire arbitrariamente una lunghezza di input fissa. Successivamente, quei domini di lunghezza inferiore verranno riempiti con un’entità del vocabolario neutra fintanto non venga raggiunta la lunghezza prestabilita. Nel caso la lunghezza fosse maggiore di , allora i nomi di dominio verranno troncati a destra.

Per cui indicando con il simbolo indicante il carattere codificato, un nome di dominio viene codificato nel seguente modo:

dove è la codifica del carattero neutro di riempimento.

I vari simboli possono essere numeri, vettori binari o reali a seconda della codifica.

Ora indicheremo con l’insieme di caratteri appartenenti al vocabolario, con il loro numero. Infine con l’insieme dei possibili vettori composti dai caratteri del vocabolario aventi lunghezza uguale a quella dell’input .

### **Integer encoding**

Essa è così definita:

dove è la rappresentazione nel dominio dei numeri interi del vocabolario.

Ad esempio:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ‘\_’ | ‘-’ | ‘0’ | ‘1’ | ‘2’ |  | ‘8’ | ‘9’ | ‘a’ |  | ‘x’ | ‘y’ | ‘z’ | % |  |  |
|  | 0 | 1 | 2 | 3 | 4 | ... | 10 | 11 | 12 | ... | 36 | 37 | 38 | 39 |  |  |

Il problema di questa codifica è che la caratteristica di essere naturalmente ordinata potrebbe non essere ignorata dalla rete neurale, la quale è probabilmente propensa a capire e sfruttare. Per questo motivo, per essere sicuri che non vi siano tali problemi è possibile utilizzare la one-hot encoding.

### **One-Hot encoding**

Essa è così definita:

dove è l’insieme dei numeri binari.  
Per cui ad ogni carattere viene associato un vettore binario sparso di dimensione la cui unicità è definita dalla posizione dell’unico 1 presente nel vettore. Ad esempio, per :

Anche questo tipo di codifica comporta degli svantaggi: si pensi ad un vocabolario formato non da elementi, bensì da . In questo caso ogni vettore di input sarebbe composto da vettori molto grandi e sparsi, introducendo un problema di memoria ed efficienza.

### **Embedding Layer**

Esso è definito come:

ovvero, un Embedding Layer, espande ogni possibile input in uno spazio multidimensionale dove è la grandezza massima dell’input ed è un numero arbitrario.

Semplificando, un Embedding Layer calcola la mappatura ottimale di ogni elemento del vocabolario in un vettore di numeri reali.

Esso dipende da tre parametri:

|  |  |
| --- | --- |
|  | dimensione del vocabolario |
|  | dimensione vettore di output |
|  | grandezza massima input[[3]](#footnote-4) |

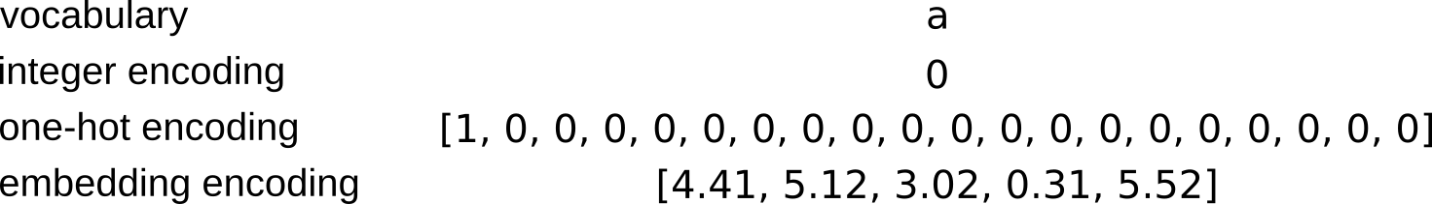
Nel nostro caso potrebbe essere: , numero di unità del layer LSTM, .  
Ad esempio, per univpm.it:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | u | n | i | v | p | m | . | i | t |
| int-encoding |  |  |  |  |  |  |  |  |  |  |
| embedded |  |  |  |  |  |  |  |  |  |  |

Quindi ogni dominio sarà rappresentato da un vettore di dimensione , poi convertito dall’Embedding Layer in un vettore che verrà poi processato dai layer successivi.  
Un altro vantaggio dell’Embedding Layer è che durante l’allenamento ottimizza i propri vettori a seconda delle similarità che riscontra durante l’allenamento dell’utilizzo di tali vettori.

Si guardi alle figure 4 e 5 per un esempio dei vari tipi di codifica. Si noti come al crescere di si fa maggiore anche la necessità di utilizzare un layer embedding. Inoltre si ricordi che la integer encoding non è utilizzabile i vari membri sono ordinati nauralmente, proprietà che si ripresa poi durante l’allenamento dalla rete.

|  |  |
| --- | --- |
| Integer enc | 0 |
| One-hot enc | [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] |
| Embedding enc | [4.41, 5.12, 3.02, 0.31, 5.52] |

****

**Modello**

La libreria utilizzata è Keras di Tensorflow. Qui di seguito vengono elencati in ordine i vari layer:

|  |  |  |
| --- | --- | --- |
| *Transformation* | transformation\_type |  |
| TextVectorization | output\_sequence\_length | Ovvero |
| Embedding | input\_dim | Ovvero |
| LSTM | stacked layers number |  |
| Dropout | rate |  |
| Dense | activation | Sigmoid, ovvero output compreso tra 0 ed 1 |

Il primo layer, Transformation, è stato scritto in corsivo perché non è un layer facente parte del modello, ma una manipolazione che viene effettuata ai nomi di dominio prima di essere dati in input al modello. È quindi indipendente dalle operazioni effettuate con la libreria Keras, differentemente da tutti gli altri layer. Inoltre *Transformation* può essere diverso a seconda che siamo in allenamento o meno.

**Trasformazioni**

Un nome di dominio è, come abbiamo detto, composto da diverse stringhe separate da punti dette *label*. Quest’ultime, eccezion fatta per quelle dell’eTLD, sono scelte arbitrariamente a patto che la *domain-label* insieme all’*eTLD* compongano un nome di dominio unico in tutto il mondo.

Data questa suddivisione, è quindi possibile estrarre dal nome di domino diverse informazioni, e allenare di conseguenza la rete LSTM. L’operazione che estrae una determinata parte di un nome di dominio è qui definita come trasformazione.

Per fare un esempio, un *tipo* di transformazione potrebbe essere omettere il suffisso. Un’altra potrebbe essere quella di omettere i punti.

Sono stati effettuati dei test con diversi tipo di trasformazioni, qui di seguito vengono descritte quelle che hanno dati risultati soddisfacenti (le quali corrispondono anche alle trasformazioni più ovvie da poter effettuare).

Trasformazione*DOMAIN:*

*S*i considera solo il dominio inteso come il livello successivo al suffisso, ad esempio: *med.edu.oxford.co.uk* diventa *oxford*

Quindi non contiene punti, è mono-label in quanto considera un solo livello ed ignora il suffisso.

***Trasformazione NOSUFFIX***

Considera l’intero nome di dominio, escludendo il suffisso, ad esempio: *med.edu.oxford.co.uk* diventa *med.edu.oxford*

***Trasformazione ANY***

Considera l’intero nome di dominiosenza effettuare modifiche.

**Test**

Il test è stato effettuato con dei pcap resi disponibili dal Stratosphere IPS Project. I vari pcap sono di tre diversi tipi: *i)* *normal*, ovvero di traffici registrati su computer non infetti; *ii)* *mixed*, ovvero di traffici registrati su computer inizialmente non infetti e poi infettati in un momento noto; *iii) malware* ovvero traffici registrati su computer infetti.

Bibliografia

1: Mozilla Foundation, Public Suffix List, 2020, https://publicsuffix.org/list/

1. È stato omesso ‘domain’ per ragioni di spazio, sarebbe: “fourt-level *domain”.* [↑](#footnote-ref-2)
2. Tra le entità uniche sono inclusi i simboli di punteggiatura. [↑](#footnote-ref-3)
3. Ovvero il numero massimo di elementi del vocabolario di cui è composto un input, utilizzato solamente nel caso in cui gli input abbiano lunghezza fissa. [↑](#footnote-ref-4)