Universidad ORT Uruguay

Facultad de Ingeniería

Comparison of LSTM and Transformer Neural Network on multiple approaches for weblogs attack detection

Entregado como requisito para la obtención del título de Master en Ingeniería

Nicolás Martínez Varsi - 186580

Tutor: Sergio Yovine

2022

Declaración de Autoría

Yo, Nicolás Martínez Varsi, declaro que el trabajo que se presenta en esta obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba el Proyecto;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Nicolás Martínez

10-03-2022

Agradecimientos

Agradezco a mi tutor, Dr. Sergio Yovine, por el tiempo, paciencia y guía bridada en el desarrollo de este proyecto.

Dedico y agradezco a mi familia, pareja y amigos, quienes me han dado su apoyo incondicional, no solo en esta etapa, sino en toda la maestría.

Abstract Español

El presente trabajo realiza una discusión y comparación de diferentes enfoques y redes neuronales para la clasificación de secuencias, en un contexto de detección de ataques en servicios web.

El primer enfoque para la detección de ataques mediante clasificación de logs es la creación de modelos de clasificación basados en caracteres. El segundo enfoque parte de la creación de modelos de lenguaje que predicen la probabilidad del siguiente caracter en una secuencia, que en conjunto con una técnica de cálculo de umbrales para las probabilidades, clasifican los logs para detectar ataques.

Estos enfoques fueron trabajados con redes neuronales de tipo LSTM, comunes en el procesamiento de secuencias, así también como con redes neuronales Transformer. Las redes Transformer han tenido muy buenos resultados en sistemas de traducción de máquina y problemas similares en cuanto a procesamiento de lenguaje natural, pero no ha sido explorado su uso en detección de ataques en base a logs.

Para presentar las comparaciones de enfoques y redes neuronales, se realizó un análisis del estado del arte, de los enfoques a aplicar y se realizaron múltiples experimentos. Estos experimentos implicaron el desarrollo de códigos para el análisis, transformación y preparación de los data sets, así como el entrenamiento y evaluación de los modelos y clasificaciones.

Finalmente se plantean conclusiones sobre el uso de cada enfoque y red neuronal, así como el planteo de futuros trabajos que puedan mejorar y responder cuestiones encontradas en el proyecto.

Abstract

This work discusses and compares different approaches and neural networks for sequence classification, in a context of attack detection in web services.

The first approach to attack detection with log classification is to create character-based classification models. The second approach is based on the creation of language models that predict the probability of the next character in a sequence, which, together with a probability threshold calculation technique, classify the logs to detect attacks.

These approaches were worked with LSTM neural networks, common in sequence processing, as well as with Transformer neural networks. Transformer networks have had very good results on machine translation systems and similar natural language processing problems, but their use in weblogs attack detection has not been explored.

To present the comparisons of approaches and neural networks, an analysis of the state of the art and the approaches to be applied was carried out, in addition to multiple experiments. These experiments involved the development of codes for the analysis, transformation and preparation of the data sets, as well as the training and evaluation of the models and classifications.

Finally, conclusions are drawn about the use of each approach and neural network, as well as the proposal of future works that can improve and answer questions found in the project.

Palabras clave

Redes Neuronales, Aprendizaje profundo, Transformer, LSTM, Detección de ataques

Key words

Neural Network, Deep Learning, Transformer, LSTM, Attack Detection

Contents

1 Introduction

Sequence classification is a common technique used in information security to classify logs and detect attacks or unusual behaviours. Nowadays it is common to implement sequence classification with machine learning systems that can be updated and improved throughout time with new data.

There are some works, like [1] and [2], that use Recurrent Neural Networks [3] and, in particular, Long short-term memory [4] (LSTM) like in the second case, to implement intrusion detection systems based on classifying sequences. These neural network implementations have proven to be better than traditional machine learning models in most cases. There are other works, like [5], that use Convolutional Neural Networks [6] at character-level with the same purpose of classifying sequences as malicious web requests with good results.

In this thesis we will be discussing and evaluating different methods relying on character-level sequence classification using neural networks models, as the state of the art suggests that they perform better than other techniques. These methods involve two approaches to classifying sequences. The first one consists in training a *classifier*, that is, a network that outputs the probability of a sequence to be an attack. The second one seeks training a *language model* [7] which produces next-character probabilities. In this case, the classification is done by comparing the probability of the whole sequence (product of next-symbol probabilities) with an appropriate threshold. For each approach, the models are built using two different kinds of neural networks: LSTM and Transformer [8]. In order to experiment and evaluate, these methods will be applied to detecting attacks to web services by analyzing weblogs. For this, two known data sets will be used. Each line of a weblog will be treated as a character sequence and classified as attack or no-attack.

It is worth to mention that the language model approach has not been previously applied for attack detection, as well as the use of LSTM at character level.

The same observation holds for Transformers. These methods and neural networks will be compared in terms of classification performance, complexity, training time and ease of tuning.

The work carried out in this thesis is also motivated by data privacy, an area of worldwide interest, where the AI research team at the University is indeed working on in the context of two research projects funded by ANII. In such projects, there is a need of reference models trained on non-privatized data to compare against the performance of other models built on privatized sequences. In this setting, because of the nature of the ideas explored in those projects that seek generating privatized data from non-privatized one, it is important to evaluate language models as they are the basis for building sequence privatizers. That approach is thoroughly explored in [9], which studied several methods based on generative sequence-to-sequence neural networks, such as, for example, training a language model with a differential privacy mechanism to privatize the output sequence.

The rest of this thesis is organized as follows.

Chapter 2 Neural networks, presents basic concepts on neural networks that are brought into discussion on this thesis. It is an introduction to Recurrent Neural Networks, Long short-term memory and Transformers.

In Chapter 3 Weblogs Attack Detection, there's a study of the state of the art in the context of attack detection. Then, the approaches studied in this thesis are presented.

Chapter 4 Data preparation, describes the data sets and processes needed to prepare data for experiments.

Chapter 5 Experimental results, contains the experiments carried on in this project, as the discussions on the results.

Finally, Chapter 6 Conclusion and future work, presents the conclusion on the analysis of the experiments and results. It also provides the basis of future work.

2 Neural networks

2.1 Recurrent neural networks

Recurrent Neural Networks are a family of neural networks specialized in processing sequences of values, as described in [3]. This is not the only way of processing sequential data, but this specialized neural networks can scale to longer sequences than non-specialized neural networks.

As the name of this type of network suggests, there are concepts of recurrent and recursive computation. The idea is to share a state across the layers of the network, in this case, as the hidden units of the network. This is defined in [3] as follows:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta), \tag{2.1}$$

The function in equation 2.1 defines the state recursively ($h^{(t)}$ depends on $h^{(t-1)}$) and with input $x^{(t)}$, being t the time or positional index of the sequence. θ represents the network coefficients (or parameters). This can be visualized in figure 2.1:

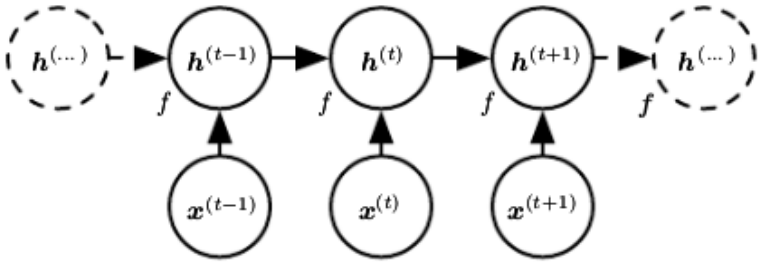

Figure 2.1: Simple unfolded RNN obtained from [3] and describes equation 2.1

This recurrent neural network takes the data from input x into the state h, but it processes it for each time step and feeds the next step until the whole sequence is processed. This is done for a finite number of steps τ. In this example there is no output, which in practice it should have some output layer that uses the last state h to make predictions.

In [3], they propose three main design patterns for recurrent neural networks:

1. RNN with ouputs at each time step and connected hidden units like it is shown in figure 2.2
2. RNN with outputs at each time step that are used as the only input for the next hidden unit as it can be seen in figure 2.3
3. RNN with connected hidden units but with only a single output at the end like the one in figure 2.4.

In figures 2.2, 2.3 and 2.4, o is the output mapped from input x, y is the target value for input x, L is the loss that measures how different is o from target y.

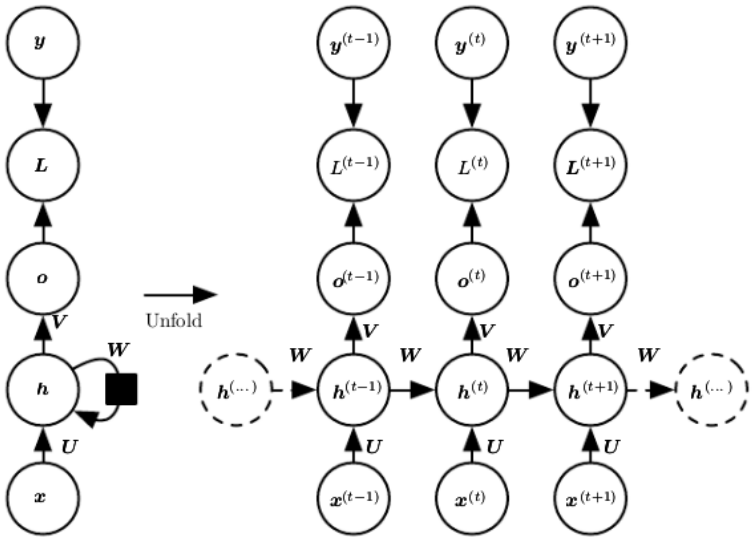

Figure 2.2: Unfolded RNN with hidden units connected at each step. Figure obtained from [3].

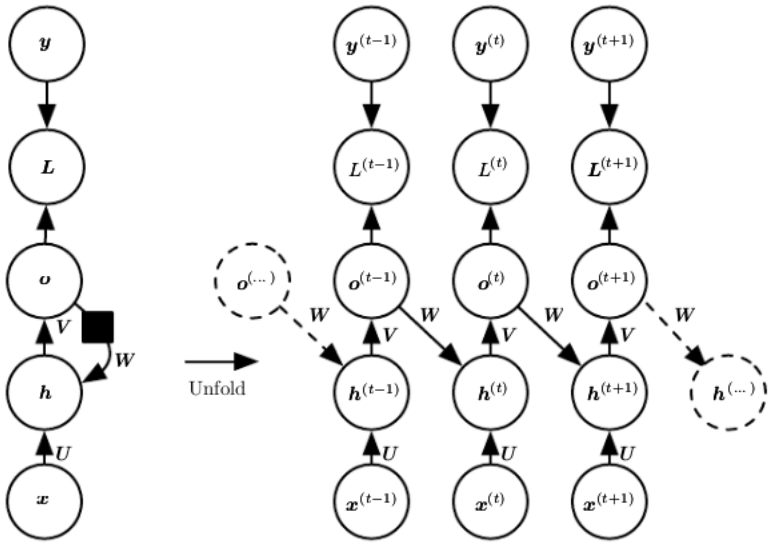

Figure 2.3: Unfolded RNN with each step output as the input for the next step hidden unit. Figure obtained from [3].

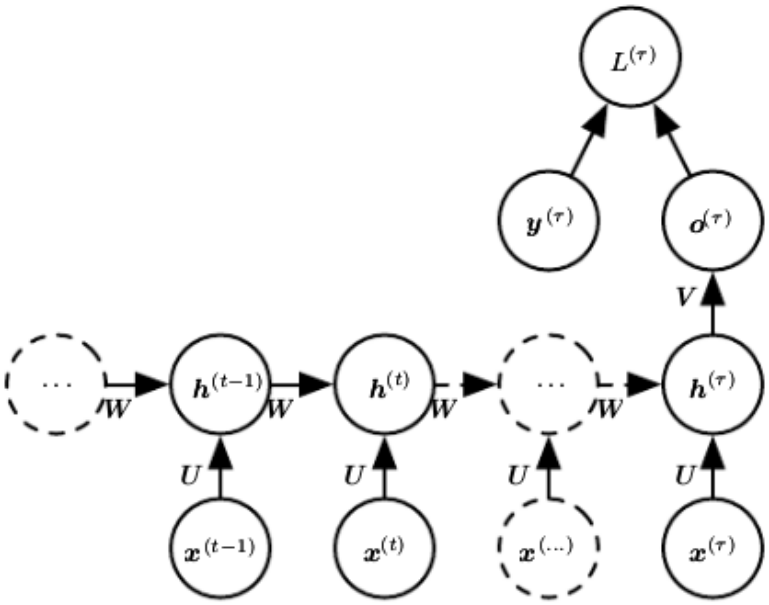

Figure 2.4: Unfolded RNN that only calculates output and loss after the last step. Figure obtained from [3].

2.1.1 Long short-term memory

Long short-term memory, or LSTM, is a type of RNN that was first introduced in [4] as a solution to some issues common RNN have, like gradient vanishing on many stages. It is explained in [3], that LSTM implements self-loops to produce paths so that the gradient can prevail for long duration and because of that, it has been used successfully in applications like handwriting recognition, speech recognition, machine translation and parsing, between many others. LSTM networks are better than ordinary RNN in learning long-term dependencies.

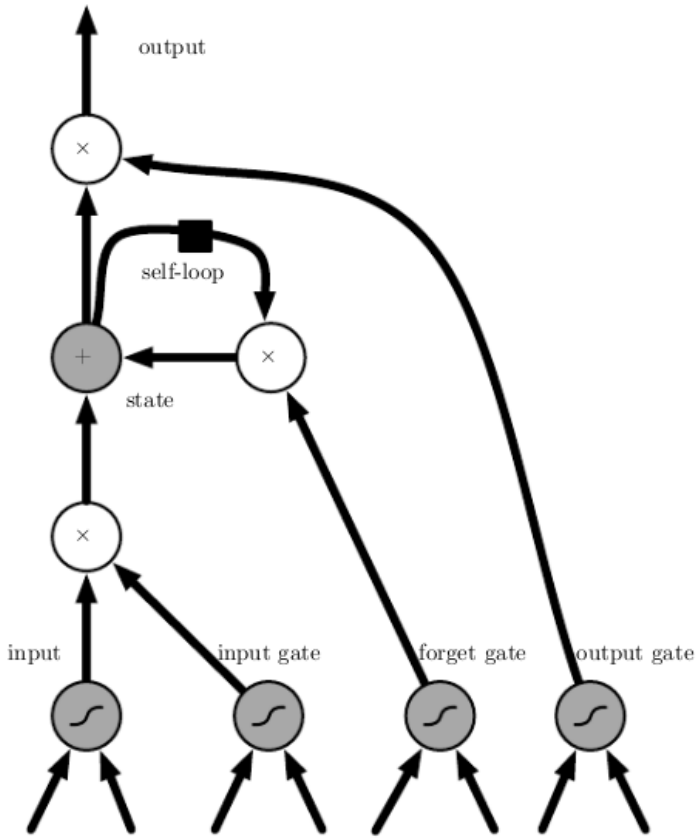

Figure 2.5: LSTM cell. Figure obtained from [3].

Figure 2.5 shows what is commonly named “LSTM cell” or “LSTM unit”. This cells can be compared to normal RNN hidden state units, with the addition

that they have an internal recurrence, the self-loop, that with a gate unit system control the flow of information. On the outer recurrence, this cells are connected recurrently in the same RNN hidden units are [3].

2.2 Transformers

The Transformer network architecture was presented in 2017 in [8], as a new simple architecture for sequence modeling and transduction problemas, like language modeling and machine translation, that were commonly based on complex recurrent or convolutional neural networks. They defined it based on attention mechanisms, without the need of convolutional or recurrent networks.

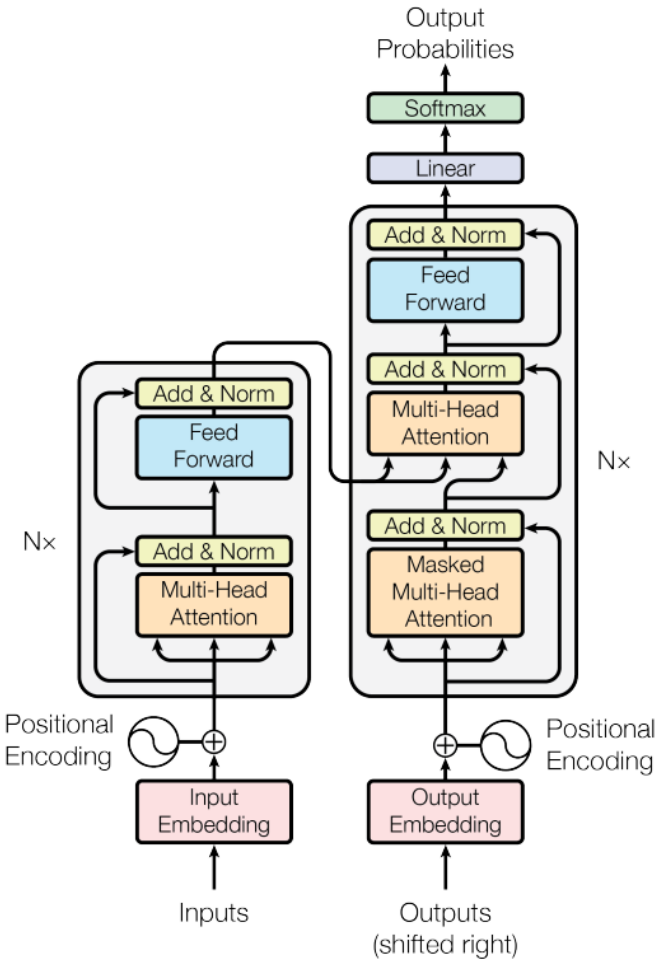

Figure 2.6: Transformer model architecture. Figure obtained from [8].

The model architecture can be visualized in figure 2.6. Because it was original intended for sequence transduction models, it's architecture is based on an encoder-decoder structure. The block on the left is the encoder and the block on the right the decoder. Both of them are composed of the same sub-layers: multi-head self-attention mechanism and simple, position-wise fully connected feed-forward network [8].

Attention mechanisms are meant to process sequences focusing on specific parts of the input at each step [10]. The multi-head attention proposed in [8] linearly projects the computations multiple times and perform the attention function in parallel. The fully connected feed-forward network is applied to each position and consists of two linear transformations with ReLU activation in between. Because the model is not based on recurrent or convolutional networks, the information about positions and order of the tokens in the sequence is given using a positional encoding in the embedding before the Transformer block.

3 Weblogs Attack Detection

3.1 State of the art

Throughout the initial investigation, multiple papers and projects related to attack, anomaly or intrusion detection using neural network methods were analyzed.

In "A deep learning approach for intrusion detection using recurrent neural networks" [1], they explore deep learning based systems for intrusion detection, proposing one built with RNNs. Intrusion detection is used to identify attacks in networks and they studied binary classification to detect if there is or isn't an attack, as well as multiclass classification. In this case, the input data is in form of a network record that contains tens of attributes. After some experimental results they conclude that RNNs perform better than traditional machine learning classification methods.

Similar to [1], in "Long short term memory recurrent neural network classifier for intrusion detection" [2], they confirm that the deep learning approach is effective for this kind of situations using LSTM RNN, by doing performance tests with optimal hyper-parameters.

LSTM networks were also used in "One-class Collective Anomaly Detection based on LSTM-RNNs" [11], in which they treat it as an anomaly detection problem in networks to distinguish between normal behavior and illegal or malicious events in network systems. They focus on detecting an anomaly event by considering data as time-series and at a collective level observing prediction errors for a sequence of time-steps. Their RNN architecture is principally defined by an LSTM and the experiments results indicate that the model is capable of detecting anomalies in an efficient way.

On all those projects mentioned above, RNNs, and particularly LSTMs, solve the problem with good performance. They also have in common the way input data is treated: each register is a matrix of numeric features.

Another anomaly detection project is presented in [12], "Deeplog: Anomaly detection and diagnosis from system logs through deep learning", in which they propose a deep neural network model utilizing a LSTM to model a system log as a natural language sequence. The objective was to learn patterns from normal executions and to detect when data deviate from what the model learned from these patterns. What's interesting of this work is that log data is viewed as elements of a sequence that follows patterns and grammar rules. The anomaly detection is done at a per log entry level instead of usual session level. After an extensive evaluation on large system logs (HDFS, OpenStack and others), results demonstrated a better effectiveness compared to previous methods.

A different approach is given in [5], "Malicious web request detection using character-level CNN", where data is treated as character-level and instead of using RNNs, they present a Convolution Neural Network solution for detecting malicious HTTP web requests. This kind of neural network is commonly used for image processing, but on this work they build a character-level embedding layer before the convolution layer to process web requests. They found that this solution has a better false positive rate than some traditional machine learning models. This can be attributed to the way the model pays more attention to the relationship between the characters of a request.

On the same line of character-level processing, there's CECoR-Net [13]. This solution for web attack detection combines CNN with LSTM techniques. The input data are HTTP requests processed by character-level simplifying the pre-processing. Convolution and pooling operations of CNN record some local features of the web requests and are then provided to the LSTM. CECoR-Net provides better results than other CNN models, detecting known featured attacks and also unknown attacks with higher precision.

All of the works discussed above conclude that neural network solutions perform better than other traditional machine learning techniques used for this type of situations. Inspired by that, this project focus on continuing the experimental investigations on RNN, particularly LSTM, but also comparing with Transformer neural networks. This kind of network is commonly used for natural language processing, but not for attack or anomaly detection. The experiments, described in chapter 5, are done by building LSTM and Transformer architectures in a character-level way, using them in two approaches that will be described next, and are different from the ones analyzed in this section.

One of those approaches was taken into account because of university colleagues work "Property Checking with Interpretable Error Characterization for Recurrent Neural Networks" [14]. This paper is about sequence classification using RNNs but with main focus on property-checking for verifying requirements, using a tool for learning probably approximately correct deterministic finite automata. One of the experiments they executed involved an RNN for classifying Hadoop File System Logs, using the Deeplog dataset of [12]. This RNN was built to predict the probability distribution of symbols at each position in the sentence of the log. For a sentence of length L, for every position $t \in [0, L-1]$, the network computes a probability vector of size equal to the number of symbols in the whole symbols set. This probability vector contains the predicted probability of each symbol to be the t-th symbol in the sequence. This kind of RNN can be used together with a threshold to build a classifier. This inspired the language model approach described in the next section.

3.2 Approaches

The experiments can be categorized in two approaches:

1. Using classification models. This means that a neural network is trained to directly classify weblogs.
2. Using Language Models. This means building a language model and finding thresholds to classify weblogs.

On both approaches, processing is done at a character level on each line of a weblog. In this way, the models are trained by learning how characters relate with each other inside a log entry. Every line of a log is parsed to a URI path including query parameters:

```
/uri/.../...?param1=value1&param2=value2...
```

For example:

```
/tienda1/publico/pagar.jsp?modo=insertar&precio=4110&B1A=Confirmar
```

An other aspect on both approaches is that the neural network architecture is either implemented with LSTM or Transformer on each test. This gives the opportunity to compare these architectures.

The main focus of this work is to compare all these different techniques an approaches in terms of classification performance, parameters needed, training time and ease of tuning.

3.2.1 Using classification models

This approach involves training a neural network whose outcome is a probability vector of size 2, as the classification is either attack or no-attack.

```
trainedModel(input) -> [prabilityOfNoAttack, probabilityOfAttack]
```

This probability vector is then used with an argmax function to determine the class.

```
argmax([prabilityOfNoAttack, probabilityOfAttack]) -> 0 or 1
```

The architecture of the neural network model is based on LSTM or Transformer and can be described on a high level with the following layers:

1. Embedding layer
2. LSTM or Transformer related layers
3. Dense layer of output size 2 with a softmax activation function

The input for training is each log line labeled as attack or no-attack.

Note that "LSTM or Transformer related layers" is expanded on each case and can vary from one experiment to another. Besides the layers, models are trained with categorical cross-entropy loss function, Adam optimizer as the optimization algorithm and early stopping as a form of regularization.

Experimental tests for classification models are based on this architecture.

3.2.2 Using language models

This approach is more complex compared to the previous one, because this involves not only training a model but also finding the best threshold to classify with the language model.

The language model built with the neural network training can be seen as a function that calculates the probability of all other characters in vocabulary as the next token of each sub-sequence of input log.

For example, if input log is "/example", the output is a probability vector of vocabulary size for each sub-sequence:

- Sub-sequence "/" -> [probability vector of vocabulary size]
- Sub-sequence "/e" -> [probability vector of vocabulary size]
- ...
- Sub-sequence "/examp" -> [probability vector of vocabulary size]
- Sub-sequence "/exampl" -> [probability vector of vocabulary size]

Each probability vector can be queried for the next symbol probability. In the example above, the second vector is queried by position of character x: *having "/e", what's the probability of character 'x'?.*

This function can be named `p(s, c)`, the probability of character c after sequence s.

$$p(s, c) = P[s_t = c | s_0 ... s_{t-1}] \tag{3.1}$$

To calculate a single probability for a log line, this function needs to be called for all sub-sequences. The sub-sequence function can be defined as `subseq(l,i)`, the sub-sequence of line l starting at position 0 to position i included.

There are two options to calculate this single probability:

1. Product:

$$probability(line) = \prod_{i=0}^{length(line)-2} p(subseq(line, i), i + 1) \tag{3.2}$$

2. Sum of logarithms:

$$probability(line) = \sum_{i=0}^{length(line)-2} \log(p(subseq(line, i), i + 1)) \tag{3.3}$$

Due to probability vectors being of vocabulary size, each probability value tend to be small. There's also the fact that iterations of the sum or the product depend on the length of the sequence. This makes the probability of option 1, using product, to be smaller on each multiplication. The threshold search becomes more difficult and this is improved by using option 2 instead. Option 2, sum of logarithms, is used in language model experiments.

The architecture of the neural network model is based on LSTM or Transformer and can be described on a high level with the following layers:

1. Embedding layer
2. LSTM or Transformer related layers
3. Time distributed layer composed of a Dense layer of output size equal to vocabulary size with a softmax activation function

The input for training is each log line (windowed and padded) and the target is the same line but shifted one position. For example, for input `a b c d`, target is `b c d e`.

Note that "LSTM or Transformer related layers" is expanded on each case and can vary from one experiment to another. Besides the layers, models are trained with categorical cross-entropy loss function, Adam optimizer as the optimization algorithm and early stopping as a form of regularization.

Experimental tests for language models classification are based on this architecture.

Because the language model behaviour is to learn if a sequence belongs to the language, the model must be trained only with one class of data. In this work, attack logs were used to train the language models.

3.2.2.1 Threshold calculation

The purpose of the threshold is to determine if given the probability of the language model for some input, it is classified as attack or no-attack.

To find this value, the probabilities of all test inputs are calculated first. With these probabilities and the correct class of each input, ROC curve is built. ROC

curve computes true positive rates (TPR) and false positive rates (FPR) for each threshold that is used to determine one class or the other. In this scenario, what matters is the threshold that generates the best TPR and FPR, i.e, the TPR that is closest to 1 and the FPR that is closest to 0.

If i is the index of a threshold, this can be represented as the following distance function:

$$threshold\_distance(i) = \sqrt{(fpr[i] - 0)^2 + (tpr[i] - 1)^2} \tag{3.4}$$

The previous function must be used with all thresholds i and the best threshold is the one that has the minimum distance. Once the threshold with the minimum distance is found, it can be used to classify inputs in attack or no-attack.

Because probabilities that language model calculates depend on the sequence length, as described in equation 3.3, a single threshold for every input might not be the best solution. To improve this algorithm, the minimum distance and best threshold is calculated for each input sequence length. This results in a key-value list, in which each sequence size has a threshold to decide whether an input of that size is classified as attack or no-attack.

4 Data preparation

As stated in section 3.2, two approaches were taken to classify weblogs in attack or no-attack. Independently of the approach, it was necessary to prepare the data first, including cleaning, transforming and managing text encoding.

4.1 Data sets

Two data sets were chosen to carry out the experiments:

1. A data set used in [15], which we will refer as "Good and bad queries data set".

   Attack example:

   ```
   /javascript/mod.exe
   ```

   No-attack (normal query) example:

   ```
   /javascript/forums.xml
   ```

   This data set contains a total of 1342657 access logs, being 1294531 non-attack and 48126 attack.

2. A data set used in [16], which we will refer as "CSIC data set".

   Attack example:

   ```
   GET
   http://localhost:8080/tienda1/publico/pagar.jsp?
   modo=insertar&precio=4110&B1A=Confirmar HTTP/1.1
   ```

No-attack (normal query) example:

```
GET
http://localhost:8080/tienda1/publico/carrito.jsp HTTP/1.1
```

This data set contains a total of 1339776 access logs, being 984000 non-attack and 355776 attack.

On all of them, the data preparation process removes parts of the weblogs like host names, http protocol, http verbs and version.

This results in clean data sets containing only the URL path and query parameters, e.g., "/tienda1/publico/carrito.jsp". This helps the rest of the data preparation process and other processes that come after to remain agnostic to the data set.

4.2 Text encoding

Before reading and processing the data sets, it was necessary to decide how the text was going to be encoded. The two approaches to build the weblogs classifiers are character-based, so, the encoding defines the vocabulary size for our models.

After exploring common alternatives like UTF-8 and ASCII, the most convenient solution was to use ASCII with backslash replace for errors.

The extended ASCII encoding supports 256 values, while UTF-8 supports more than 1 million values. For the simplicity of experiments and model building, 256 is a more adequate size for the vocabulary. It also adds the simplicity of converting character to indices (0-255), and vice versa, using simple Python functions like ord and chr.

```
>>> ord('A')
65
>>> chr(65)
'A'
```

In case the text contains characters that can not be represented by ASCII characters, the data preparation process replaces them with the backslash representation.

The next simple code gives an example of this transformation.

```
with open('non-ascii.txt',
        mode='r',
        encoding="ascii",
        errors="backslashreplace") as f:
    for line in f:
        print(line)
```

If a non-ASCII character is present in the file non-ascii.txt, it will fallback to the backslashreplace handler.

Given the following example that contains an emoji 😮 (non-ASCII character), the output contains \xf0\x9f\x98\xaf that is the literal UTF-8 backslash representation of that non-ASCII character.

Input:

```
<img class="emoji" alt="😮" src="x" />

<svg onload=prompt(document.domain)>
```

Output:

```
<img class="emoji" alt="\xf0\x9f\x98\xaf" src="x" />

<svg onload=prompt(document.domain)>
```

Using this backslashreplace error handler, the result text in the data set can be processed in a character-level with the 256 maximum ASCII values. The above example was taken from “Good and bad queries” data set presented in 4.1.

4.3 Data sets log lines length

For testing time performance, it was necessary to decrease data sets size and log lines length. For that purpose, it was convenient to remove the longest lines.

In the figures 4.1 and 4.2 it is shown that less than 7% of the data for CSIC and Good and bad queries was longer than 200 characters, resulting in an acceptable threshold to start filtering long data.

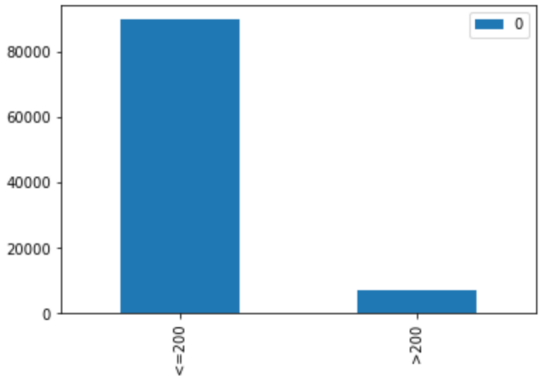

Figure 4.1: CSIC data length histogram

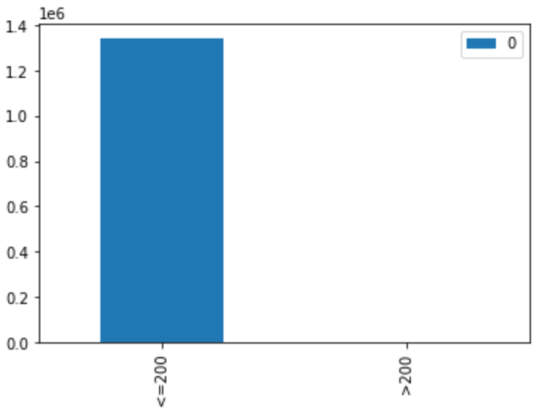

Figure 4.2: Good and bad queries data length histogram

With that filter, the data used in the experiments were lines of logs with less than 200 characters.

4.4 Windowing

Using the approach that classifies a web log with a Language Model and a threshold, more information can be added to the model training using windows.

For example, given a sequence "`a b c d e f g`" with a window size of 4, it can be converted to the following 7 sequences:

```
a
a b
a b c
a b c d
b c d e
c d e f
d e f g
```

As it can be seen in the example above, there are new sequences that add information of the relationship of the symbols that form the whole sequence.

This technique is useful, for example, to calculate the probability of the next symbol given a sequence: "What's the probability of `d` given `a b c`?" and "What's the probability of `e` given `b c d`?"

4.5 Padding

Training neural network models needs that every input has the same length. The padding technique can be used in case of sequence processing, where inputs are of different lengths. Zero padding adds the necessary amount of zeroes to make every input be of a specified length. This is mostly done in two ways, pre-padding or post-padding.

Pre-padding was used in the experiments, because, as stated in [17], it shows better results in RNN models.

In the experiments developed in this project, padding was applied with different lengths depending on the approach. In case of directly classifying a web log in attack or no-attack, padding was made to the maximum size described in 4.3. On the other hand, to build the Language Model where windows were applied to the inputs, padding is done to the windows size.

For example, the sequence "`a b c d e f g`", would be converted to 7 windows of size 4 and then padded to length 4 as follows:

```
0 0 0 a
0 0 a b
0 a b c
a b c d
b c d e
c d e f
d e f g
```

5 Experimental results

This chapter presents a discussion on how the experiments were built and what were the results obtained.

5.1 Technical aspects

The experiments were carried out with the following technical characteristics:

- Code was developed with Python 3.7.6 and Keras 2.4.0 in Jupyter Notebooks.
- Weights & Biases [18] was used to track and visualize experiments metrics.
- Jupyter Notebooks were executed in a JupyterHub cluster providede by the University.
- JupyterHub Notebook instances are based on Ubuntu 18.04.4 .
- JupyterHub Notebook instances run on nodes with 36 CPUs, 512GB of RAM and powered by NVIDIA RTX 2080 TI GPUs.
- JupyterHub cluster is shared with other students.

5.2 Experiments

This section discusses some of the experiments done and their outcomes. In order to keep the amount of experiments adequate to the size of this project, some parameters and variables were fixed for all experiments. All models were trained with categorical cross-entropy loss function, Adam optimizer as the optimization algorithm, early stopping and dropout as forms of regularization. The early stopping

patience was fixed at 20 epochs and the dropout rate at 0.1. Because the purpose of the experiments was to compare approaches and architectures, and not to build models with the highest performance, there was no automated hyper-parameters tuning performed.

All the experiments were based on the approaches and models described in 3.2, with variations in neural networks layers and parameters configuration.

This section describes experiments in terms of:

- Approach: using classification models or language models + threshold.
- Neural network base architecture: LSTM or Transformer.
- Configuration: Common configuration like learning rate, windows size in case of LM, number of layers (LSTM or Transformer blocks) and embedding size. For LSTM layers, LSTM units. For Transformer layers, hidden layer size in feed forward network inside transformer (ff-dim) and number of attention heads.
- Number of trainable parameters: The configuration and complexity of the neural network impacts in the amount of trainable parameters.
- Results:
  - Training duration: hours required to train the neural networks.
  - Accuracy: it is a measure of how correctly sequences were classified. It is calculated as number of correctly classified sequences / total number of sequences [19].
  - Recall: it is defined es total number of documents retrieved that are relevant / total number of relevant documents in the database [20]. In this case, it is the total number of attack sequences classified as attack / total number of attack sequences. It is also known as sensitivity or true positive rate. In attack detection this is an adequate metric to take into account because it is more important to correctly detect attacks than no-attacks.

Both accuracy and recall are common measures in classification problems. Accuracy gives a general picture of the performance of the classifier, but in some cases like attack detection, correctly classifying attacks is more important than classifying non attacks. So, recall is also a practical measure to take into account.

5.2.1 Classification models

Table 5.1 shows a comparison of experiments done with good and bad queries data set using the classification model approach.

Neural network	Configuration	# of params	Training duration (hours)	Classification Accuracy	Classification Recall
LSTM	learning-rate=0.001 layers=2 lstm-units=500 embedding-size=500	4133502	17.9	0.99	0.99
Transformer	learning-rate=0.001 layers=1 ff-dim=32 num-heads=2 embedding-size=32	21790	4.1	0.99	0.98

Table 5.1: Comparison of experiments for good and bad queries data set using a classification model approach.

It can be observed that the accuracy is the same between LSTM and Transformer and the recall has a difference of 1 point in favour of LSTM. This difference will probably not be noticeable in practice. The main differences between these two experiments are in training time and number of trainable parameters. Both of them can be explained looking at the complexity of the LSTM. To get to those results, the LSTM needed some tuning and testing the Transformer experiment did not. This involved increasing units, embedding size and adding a second layer, resulting in more than 4 million of trainable parameters and a training time more than 4 times the Transformer training time. In the case of the Transformer experiment, it was easier to build a model with almost the same accuracy and recall performance, as non parameters tuning was necessary.

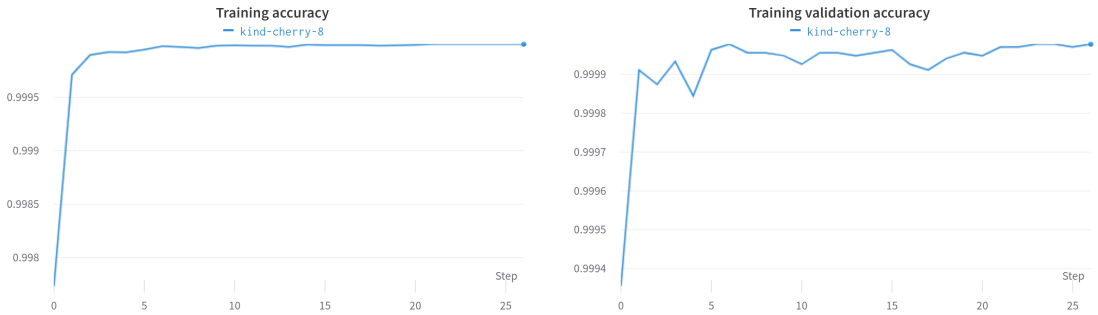

Figure 5.1: Training and validation accuracy with an LSTM classifier for good and bad queries data set

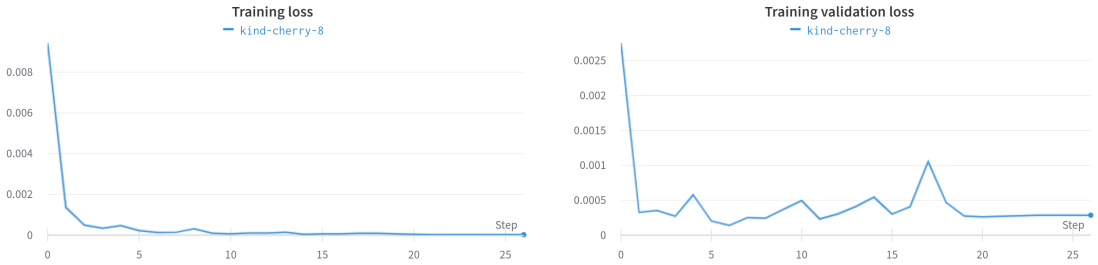

Figure 5.2: Training and validation loss with an LSTM classifier for good and bad queries data set

Comparing figures 5.1 and 5.2, with 5.3 and 5.4, it can be seen that LSTM stopped after 25 epochs approximately and Transfomers after 120 epochs. Despite that difference, because the Transformer experiment was less complex, each epoch was faster and that is why LSTM still needed more time to finish training.

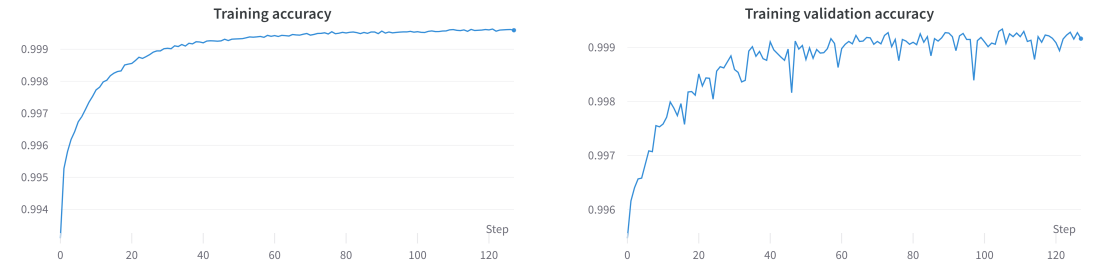

Figure 5.3: Training and validation accuracy with a Transformer classifier for good and bad queries data set

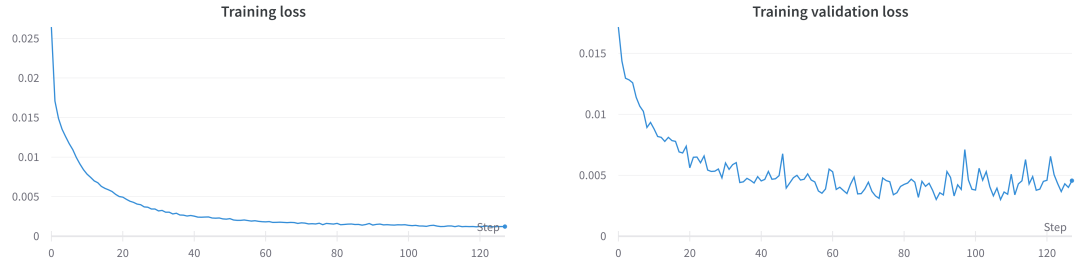

Figure 5.4: Training and validation loss with a Transformer classifier for good and bad queries data set

Moving on to CSIC data set, with classification models, table 5.2 shows interesting results on the experiments with LSTM and Transformer. The accuracy and recall obtained are almost the same, but the situation of the number of parameters and training duration is also the same to the one with "good and bad queries" data set. Transformer model is less complex and in less than 1 hour got the same performance as the LSTM model in almost 3 hours.

Neural network	Configuration	# of params	Training duration (hours)	Classification Accuracy	Classification Recall
LSTM	learning-rate=0.001 layers=3 lstm-units=800 embedding-size=800	15576802	2.9	0.81	0.99
Transformer	learning-rate=0.001 layers=1 ff-dim=128 num-heads=8 embedding-size=128	160574	0.6	0.82	0.99

Table 5.2: Comparison of experiments for CSIC data set using a classification model approach.

Despite that 81% and 82% of accuracy might not be considered an acceptable result, the 99% of recall surely is. This behaviour of getting almost the same results, but with easier tuning and less training time with the Transformer neural network is consistent in both data sets using the classification model approach.

5.2.2 Language models

Making a similar analysis to the one made with the classification model approach, table 5.3 shows a comparison of experiments done with good and bad queries data set using the language model approach with the threshold calculation technique for classification.

Neural network	Configuration	# of params	Training duration (hours)	Classification Accuracy	Classification Recall
LSTM	learning-rate=0.001 window-size=10 layers=2 lstm-units=500 embedding-size=500	4261257	5	0.89	0.83
Transformer	learning-rate=0.001 window-size=10 layers=1 ff-dim=128 num-heads=8 embedding-size=512	1434025	6.7	0.63	0.61

Table 5.3: Comparison of experiments for good and bad queries data set using language models and thresholds for classification.

The behaviour on this approach is different from the one analyzed before. The

LSTM experiment, with a more complex neural network with more than 4 million trainable parameters, took 5 hours and got 89% of accuracy and 83% of recall. It is much better than the Transformer experiment, with 26 and 22 points over in accuracy and recall respectively. This experiment trained 1.4 million parameters in almost 7 hours. The duration may be explained because of the early stopping technique, but the interesting fact is how low accuracy and recall were.

Neural network	Configuration	# of params	Training duration (hours)	Classification Accuracy	Classification Recall
LSTM	learning-rate=0.001 window-size=10 layers=2 lstm-units=800 embedding-size=800	10657857	3	0.78	0.84
Transformer	learning-rate=0.001 window-size=10 layers=1 ff-dim=128 num-heads=8 embedding-size=128	165929	1.4	0.82	0.62

Table 5.4: Comparison of experiments for CSIC data set using language models and thresholds for classification.

Comparing LSTM and Transformer language model approach for CISC data set in table 5.4 shows similar aspects. In this case the low score for Transformer was on recall, with 62% vs. 84% for LSTM.

The expectations on Transformer performance with language model approach were higher than what it was expected from LSTM experiment, because of the how it outperformed other recurrent or convolutional based neural networks in [8].

This situation, why language model together with threshold calculation are not performing as expected, specially with Transformer, is analyzed further in section 5.2.4.

5.2.3 Comparing approaches

The discussion above was about comparing LSTM and Transformer within each approach, but, looking at tables 5.1 and 5.3, for "good and bad queries", each approach can be compared with each other. At a performance level, using a classification model is better than using language model approach according to the experiments done on this data set.

In the case of CSIC data set, that difference is not that big from the accuracy perspective. But the 62% and 84% of recall in language model approach are far from the 99% of recall in classification approach. Therefore, if one solution needs to be chosen, classification models performed better on this data set.

Throughout the experiments execution, it was easier to use and tune the classification models to get to comparable performance values. It might be possible to improve the language model approach results, but it surely will need more tuning and training hours.

5.2.4 More on language models with Transformer

The goal of this project was not to build the best performing solutions. Nevertheless, as discussed previously, language model approach, mainly with Transformer, did not perform as expected.

This section describes some experiments done, making changes on the complexity of the neural network configuration in order to improve language model performance and understand why it did not perform as expected. Table 5.5 contains those experiments, using the approach of language models in combination with thresholds calculation and Transformer neural network on "good and bad queries" data set. To distinguish each experiment, the column "Experiment name" was added using the auto-generated name from Weights and Biases tool.

Experiment name	Configuration	# of params	Training duration (hours)	Classification Accuracy	Classification Recall
restful-disco-8	learning-rate=0.001 window-size=10 layers=1 ff-dim=128 num-heads=8 embedding-size=512	1434025	6.7	0.63	0.61
clear-dream-9	learning-rate=0.001 window-size=30 layers=1 ff-dim=128 num-heads=8 embedding-size=512	1434025	1.4	0.6	0.67
effortless-wave-12	learning-rate=0.001 window-size=30 layers=1 ff-dim=1024 num-heads=64 embedding-size=512	2352425	9.9	0.55	0.62
stoic-snow-15	learning-rate=0.001 window-size=30 layers=2 ff-dim=1024 num-heads=64 embedding-size=512	4455209	17.4	0.65	0.63
treasured-vortex-21	learning-rate=0.0001 window-size=30 layers=3 ff-dim=1024 num-heads=64 embedding-size=512	6557993	11.3	0.5	0.68
divine-rain-22	learning-rate=0.0001 window-size=10 layers=3 ff-dim=1024 num-heads=64 embedding-size=512	6557993	12.8	0.61	0.58

Table 5.5: Comparison of experiments for good and bad queries data set using LM + threshold approach in combination with Transformer.

The first experiment in the table, **restful-disco-8**, as well as the other experiments done with language models in different data sets, is based on windows of size 10. This size is common between sequence processing solutions, but thinking in a character-level situation, this size could be increased to improve the data obtained from the relationship of the characters in a sequence.

The experiment **clear-dream-9** was done with that purpose, having the same configuration from **restful-disco-8**, but with a window size increased to 30. The first difference in the output is the training time, improving from 5.1 hours to 1.4. This is due to the amount of sub-sequences (windows) that the model is trained with. Increasing the windows size, decreases the total amount of sub-sequences. Looking at the accuracy and recall, it can be seen that accuracy slightly decreased, 63% to 60%, but recall improved from 61% to 67%.

Keeping the windows size at 30, because of the improve at the recall, the experiment **effortless-wave-12** is done with more complexity at the Transformer block, increasing ff-dim and number of heads. This new complexity can be seen in the number of trainable parameters. This variation resulted in worse performance, mainly seen at the 55% accuracy.

On the other hand, with **stoic-snow-15**, adding only one extra transformer block layer, accuracy and recall improved, but there is no big difference with the first experiment, **restful-disco-8**. In terms of training duration, it is more than three times compared to that experiment.

The experiment **treasured-vortex-21** added a third transformer block layer and decreased the learning rate. This resulted in the worst accuracy (50%) and the best recall (68%) of all these experiments. Training duration was 11.3 hours, less than the previous experiment, due to early stopping finding the best epoch earlier, but 10 times the duration of **clear-dream-9**, which has better accuracy and almost the same recall.

The last experiment, **divine-rain-22**, took the same configuration as **treasured-vortex-21**, but went back to a windows size of 10, to have a second test on this windows size difference. Again, the experiment with windows size 10 performed better in terms of accuracy, 61%, but worse in terms of recall with 58%.

Despite the efforts to improve this language model classifier, none of the experiments performed better than 65% of accuracy and 68% of recall. If one of these classifiers needs to be chosen, **clear-dream-9** seems to have the best combination of accuracy, recall and low training duration.

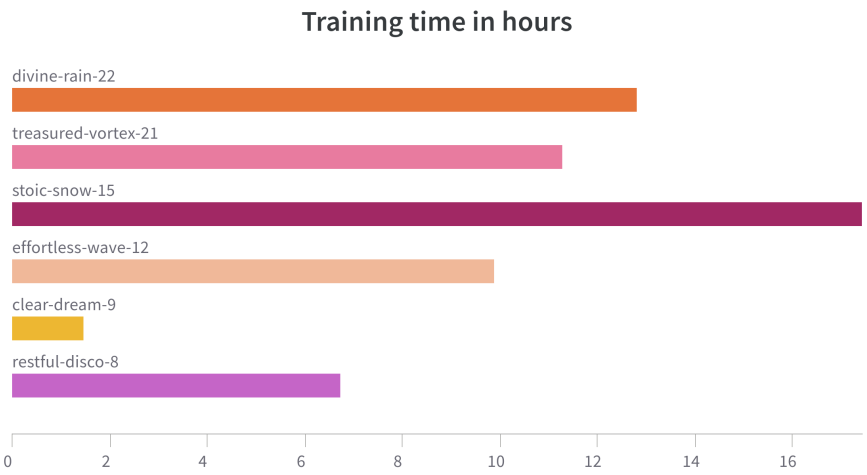

Figure 5.5: Experiments language model (next symbol predictor) training duration with LM + threshold and Transformer for good and bad queries data set

Figure 5.5 shows that the more complex the neural network becomes, the longer it needs to be trained. But, as it can be seen in figures 5.6 and 5.7, accuracy and recall of the classifiers did not improve significantly, and in some cases are worse.

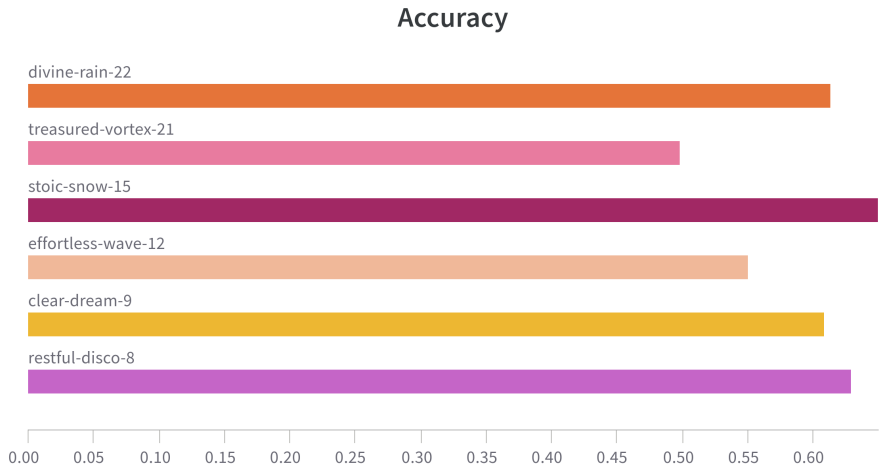

Figure 5.6: Experiments classification accuracy with LM + threshold and Transformer for good and bad queries data set

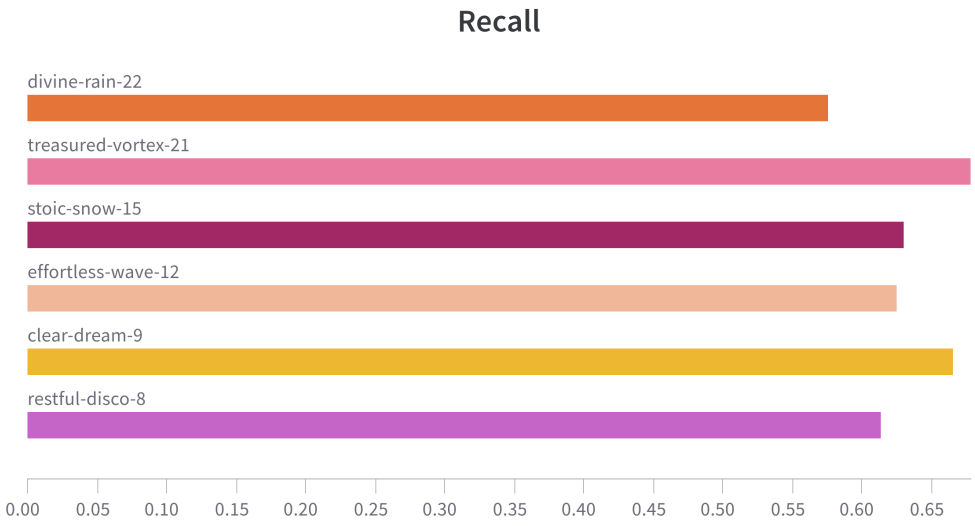

Figure 5.7: Experiments classification recall with LM + threshold and Transformer for good and bad queries data set

In order to keep investigating this behaviour, let's recall that this classification approach has two main components:

1. The language model based on a neural network (next symbol predictor).
2. The thresholds calculated to classify based on the language model probabilities, as described in 3.2.2.1.

Digging in the first one, figures 5.8 and 5.9, that show the language model training validation accuracy and validation loss respectively, indicate that the neural network models behind the classifiers seems to be performing well.

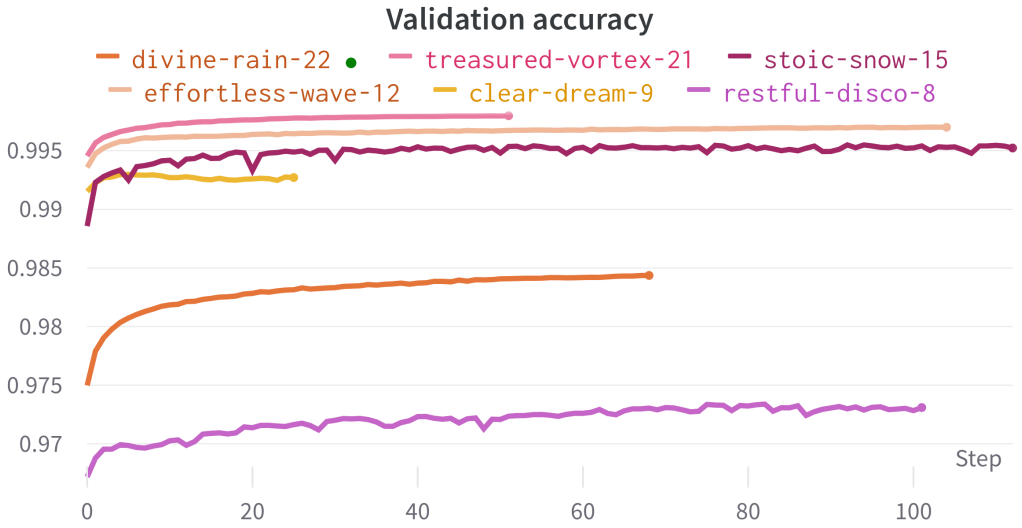

Figure 5.8: Experiments training validation accuracy with LM + threshold and Transformer for good and bad queries data set

All the experiments reached values over 97% of accuracy and even four of them over 99%. In the case of loss, those four experiments were lower than 0.04.

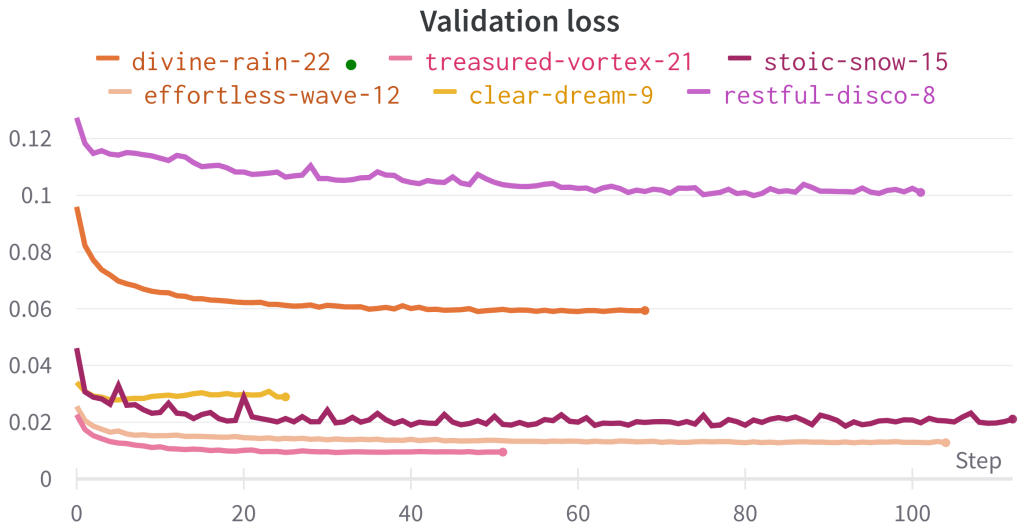

Figure 5.9: Experiments training validation loss with LM + threshold and Transformer for good and bad queries data set

Taking into account the order in which the experiments appear in 5.8 and 5.9, they can be ranked from worst to best as follows:

1. restful-disco-8
2. divine-rain-22
3. clear-dream-9
4. stoic-snow-15
5. effortless-wave-12
6. treasured-vortex-21

If we recall that **divine-rain-22** was the most complex configured model, but with windows size of 10, it makes sense that it is the second on the list, because the tests indicated that longer windows performed better. Taking out that experiment, the rest are mostly in the order of improvement they would have been expected to be. **stoic-snow-15** and **effortless-wave-12** should be swapped, but their values in the figures are close, so it is an understandable difference.

According to the values seen on the figures, this could indicate that the first component of this approach, the neural network language model that predicts the next symbol, is performing well. Hence, all the efforts on improving the classifier did not improve it at a whole, but it did improve the neural network model behind it.

In summary, according to validation metrics, the neural network language model makes next symbol predictions with high accuracy, but when the threshold component is added to use it in a classification problem, it does not perform well.

This is not the case with LSTM, if we recall the values in 5.3, a language model with threshold for classification got 89% of accuracy and 83% of recall, 20% higher than the Transformer version. Analyzing the LSTM language model training validation accuracy figure, 5.10, it reached approximately 75% in validation, a lower performance than the Transformer language model.

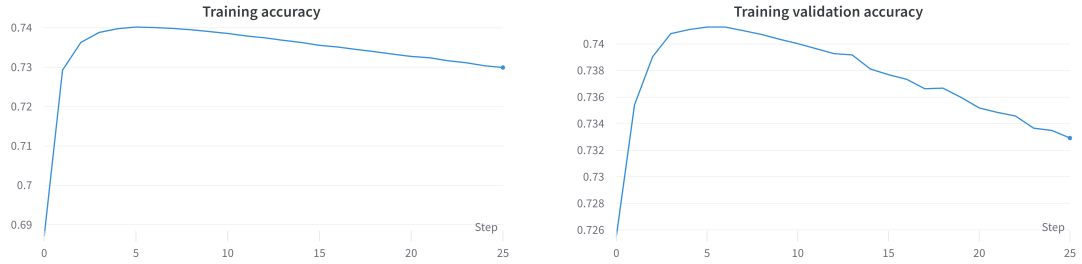

Figure 5.10: Training and validation accuracy with an LSTM language model for good and bad queries data set

This could indicate that the LSTM language model is not an excellent predictor of the next symbol, nevertheless, using it with thresholds to classify gets a 89% of accuracy and 83% of recall. On an other point of view, it might be that the Transformer language model is such an excellent predictor of the next symbol that when it is used with the threshold technique, it can not find a suitable threshold for each sequence length to make classifications properly.

Apart from that, let's not rule out that the way the thresholds are being calculated, using minimum distances and ROC curve values, might be improved or it could even be changed to a different technique.

This investigation and testing to improve Transformers language model for the classification problem, or also checking other possible techniques, needs further investigation and can be addressed in future work.

6 Conclusions and future work

This project contributes an analysis and comparison of approaches and neural networks in the context of weblogs attack detection. In addition, it presents first applications of language models approach, LSTM at character level and the use of Transformers in a weblogs attack detection problem.

This also involved developing the code that prepared the data sets, trained and test the classification systems to execute the experiments. To compare the two approaches (language model and classification model) and the neural networks (Transformers and LSTM), experiments were done on two weblogs data sets. The output of the experiments were performance metrics (accuracy and recall) and training duration, but it was also used in the comparison the complexity and ease of tuning of the neural networks.

In the case of classification models, there were no major differences in accuracy or recall between LSTM and Transformer. But, in order to get to those values, LSTM needed more tuning and a more complex configuration, that led to more training time.

On the other hand, in language model approach, the LSTM performed better on the tests done. Regarding complexity and tuning, Transformers were simpler to train, but they did not obtained good performance metrics. As to training hours, there were no clear differences.

In respect of comparing the classification model and the language model approaches, there are no doubts that classification models did the job better in terms of performance. Concerning complexity, tuning and training time, it depends on the neural network used, but as language model approach did not get high accuracy or recall, it will probably need more tuning and training to get something comparable to the 98-99% classification models got in recall.

If the only important aspect is the classification of weblogs to detect an attack,

then the classification model with Transformer neural network should be the best option to start. It reached high performance metric values, needed little tuning, low complexity and low training time.

The low training time characteristic is important in the context of costs. For example, 100 experiments like the ones described in table 5.1, in a virtual machine in Amazon Web Services with 16 vCPUs, 64 GB of memory and 1 GPU, would cost approximately 350 USD for the Transformer experiments and approximately 1570 USD for the LSTM experiments.

In the case a language model is needed to fulfill some requirements, for example, to use the next symbol prediction probabilities, then LSTM should be the choice. On the experiments done, it obtained higher accuracy and recall than the Transformer's experiments. Nevertheless, this values were not that high as classification models experiments.

That last conclusion motivates some of the future work described next, to keep investigating on language models for classification and to give another try to Transformers to increase the performance of this approach.

On the experiments with language model approach and Transformers it could be seen that this combination could not perform like the LSTM based language model or the classification approach with either neural network. Looking at the experiment of language model approach and LSTM, the language model neural network performance on predicting next symbol was not as good as the Transformer language model performance on that same prediction. The theory proposed is that in order to make more accurate classifications with thresholds, the language model neural network should not be highly accurate on the next symbol prediction. This needs to be analyzed and tested to validate the behaviour.

In addition, finding and analyzing alternatives to the threshold calculation method could lead to better results and new insights on the comparison of approaches and neural networks.

An other topic to be addressed in future work is including other modern neural network architectures besides LSTM and Transformer to the analysis. Transformer can be categorized as a modern neural network, but in order to keep the comparison updated, other types of neural network can be added to the comparison.

Finally, the language model approach taken in this project stated that the neural networks were trained with attack-class weblogs, aiming to model the attack domain. This could be implemented on the opposite direction, training the language models with non-attack weblogs and modeling normal access logs. The

results of that investigation could be compared with the ones obtained in this project to determine which method is more adequate. If results are acceptable, both methods could even be used together, using the two predictions to make the classification.

7 Bibliographic references

[1] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[2] J. Kim, J. Kim, H. L. Thi Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*, Jeju, Korea, 2016, pp. 1–5.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ch. 10 Sequence Modeling: Recurrent and Recursive Nets. [Online]. Available: http://www.deeplearningbook.org

[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[5] W. Rong, B. Zhang, and X. Lv, "Malicious web request detection using character-level CNN," *CoRR*, vol. abs/1811.08641, 2018. [Online]. Available: http://arxiv.org/abs/1811.08641

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ch. 9 Convolutional Networks. [Online]. Available: http://www.deeplearningbook.org

[7] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., vol. 13. MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[9] R. Visca, "Estudio de modelos de privacidad de datos," *Publicaciones de ANII*, 2021. [Online]. Available: https://hdl.handle.net/20.500.12381/463

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ch. 12 Applications. [Online]. Available: http://www.deeplearningbook.org

[11] N. N. Thi, V. L. Cao, and N. Le-Khac, "One-class collective anomaly detection based on long short-term memory recurrent neural networks," *CoRR*, vol. abs/1802.00324, 2018. [Online]. Available: http://arxiv.org/abs/1802.00324

[12] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[13] X. Gong, J. Lu, Y. Wang, H. Qiu, R. He, and M. Qiu, "Cecor-net: A character-level neural network model for web attack detection," in *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, Tokyo, Japan, 2019, pp. 98–103.

[14] F. Mayr, S. Yovine, and R. Visca, "Property checking with interpretable error characterization for recurrent neural networks," *Machine Learning and Knowledge Extraction*, vol. 3, no. 1, pp. 205–227, 2021. [Online]. Available: https://www.mdpi.com/2504-4990/3/1/10

[15] J. Li, H. Zhang, and Z. Wei, "The weighted word2vec paragraph vectors for anomaly detection over http traffic," *IEEE Access*, vol. PP, pp. 1–1, Aug. 2020.

[16] C. Torrano-Giménez, A. Pérez-Villegas, and G. Álvarez Marañón, "An anomaly-based approach for intrusion detection in web traffic," 2010.

[17] M. Dwarampudi and N. V. S. Reddy, "Effects of padding on lstms and cnns," *CoRR*, vol. abs/1903.07288, 2019. [Online]. Available: http://arxiv.org/abs/1903.07288

[18] L. Biewald, "Experiment tracking with weights and biases," 2020. [Online]. Available: https://www.wandb.com/

[19] C. Sammut and G. I. Webb, Eds., *Accuracy*. Boston, MA: Springer US, 2010, pp. 9–10. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_3

[20] K. M. Ting, *Precision and Recall*. Boston, MA: Springer US, 2010, pp. 781–781. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_652