URLNet : Learning a URL Representation with Deep Learning for Malicious URL Detection

Summary

Previous Methods

blacklist, whitelist

- cannot be exhaustive, cannot detect newly generated malicilous URLs

by machine learning

- Bag-of-Words like features, with SVM
- inable to capture semantic and sequential patterns
- require substantial manual feature engineering
- inable to handle unseen features and generalize to test data

Our method

- Deep learning with CNN
  - learn to classify
  - learn word/char embedding jointly
  - advanced word-embedding to solve too many rare words problem

Malicious URL detection

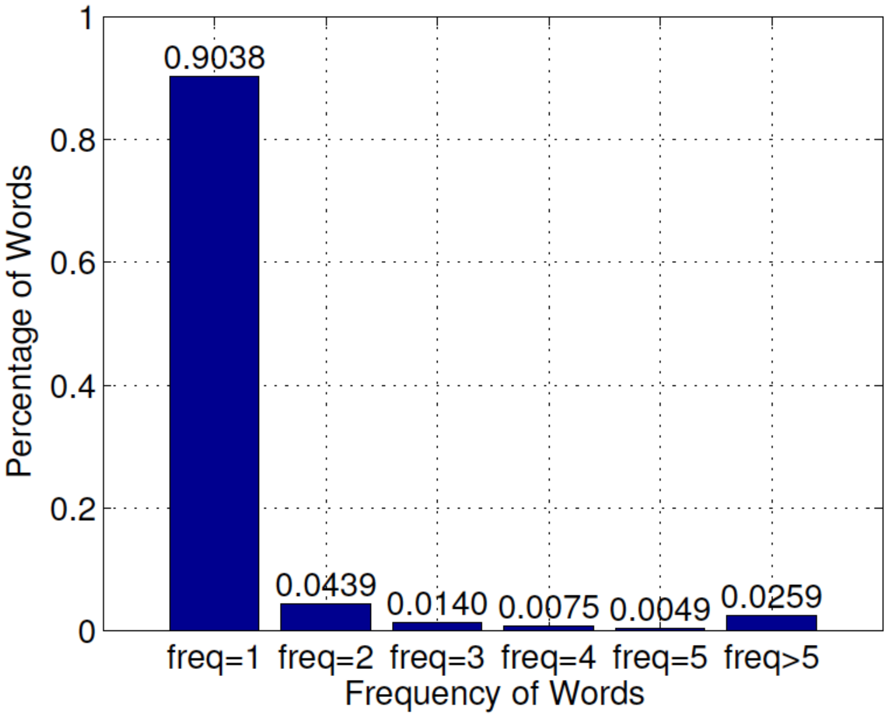
Problem setting

Consider a set of T URLs, $\{(\mathbf{u}_1, y_1), \ldots, (\mathbf{u}_T, y_T)\}$, where $\mathbf{u}_t$ for $t = 1, \ldots, T$ represents a URL, and $y_t \in \{-1, +1\}$ denotes the label of the URL, with $y = +1$ being a malicious URL, and $y_t = -1$ being a benign URL. The first step in the classification procedure is to obtain a feature representation $\mathbf{u}_t \rightarrow \mathbf{x}_t$ where $\mathbf{x}_t \in \mathbb{R}^n$ is the $n-$dimensional feature vector representing URL $\mathbf{u}_t$. The next step is to learn a prediction function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}$ which is the score predicting the class assignment for a URL instance $\mathbf{x}$. The

Lexical Features

- URL splitted into words which are delimited by special characters
- dictionary constructed by unique words in tranining set
- features

- Bag-of-Words features : occurance in dictionary list
- length of URL, lengths of different segments in URL, number of dots
- lack of sequential info => create a seperate dict for every segments of the URL



- inable to obtain information from rare words
  - most of words appears only once
  - in training => become unknown
  - in test => become unknown

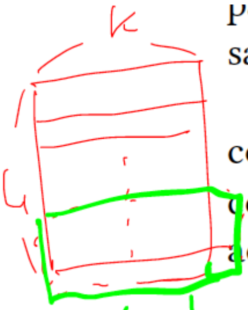
URLNet

embedding

A URL **u** is essentially a sequence of characters or words (delimited by special characters). We aim to obtain its matrix representation $\mathbf{u} \rightarrow \mathbf{x} \in \mathbb{R}^{L \times k}$, such that the instance **x** comprises a set of contiguous components $x_i, i = 1, \ldots, L$ in a sequence, where the component can be a character or a word of the URL. Each such component is represented by an embedding such that $x_i \in \mathbb{R}^k$, is a k-dimensional vector.

CNN convolution

parallelization, usually all sequences are padded or truncated to the same length L.



kxh filter.

A CNN would convolve over this instance $\mathbf{x} \in \mathbb{R}^{L \times k}$ using a convolutional operator. A convolution operation $\otimes$ of length h consists of convolving a filter $\mathbf{W} \in \mathbb{R}^{k \times h}$ followed by a non-linear activation f to produce a new feature:

$$c_i = f(\mathbf{W} \otimes \mathbf{x}_{i:i+h-1} + b_i)$$

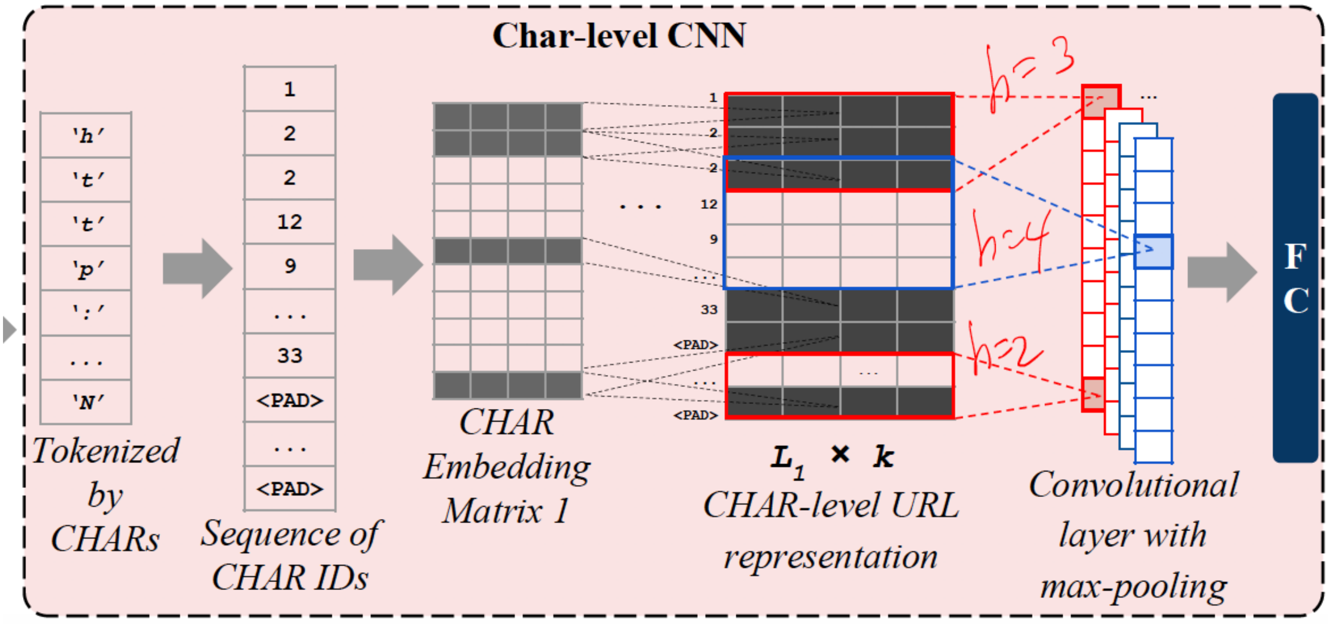
where b_i is the bias. This convolution layer's output applies a filter $\mathbf{W}$ with a nonlinear activation to every h–length segment of its input, each of which is separated by a pre-defined stride value. These outputs are then concatenated to produce output $\mathbf{c}$ such that:

1개의 W에서 나온 것 → $$\mathbf{c} = [c_1, c_2, \ldots, c_{L-h+1}]$$ 이런것이 W 여러개에 대해 각각

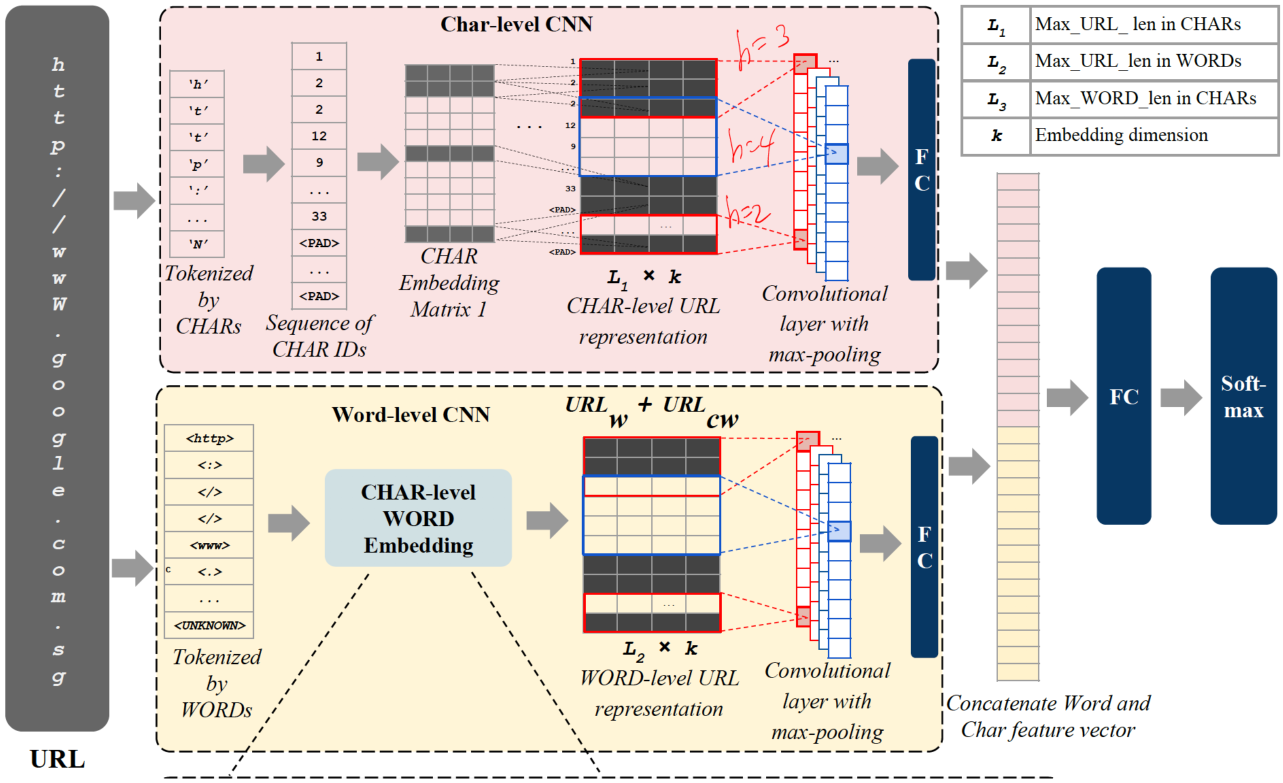
After the convolution, a pooling step (either max or average pooling) is applied to reduce the feature dimension and to identify the most important features.

CHAR embedding and Detection

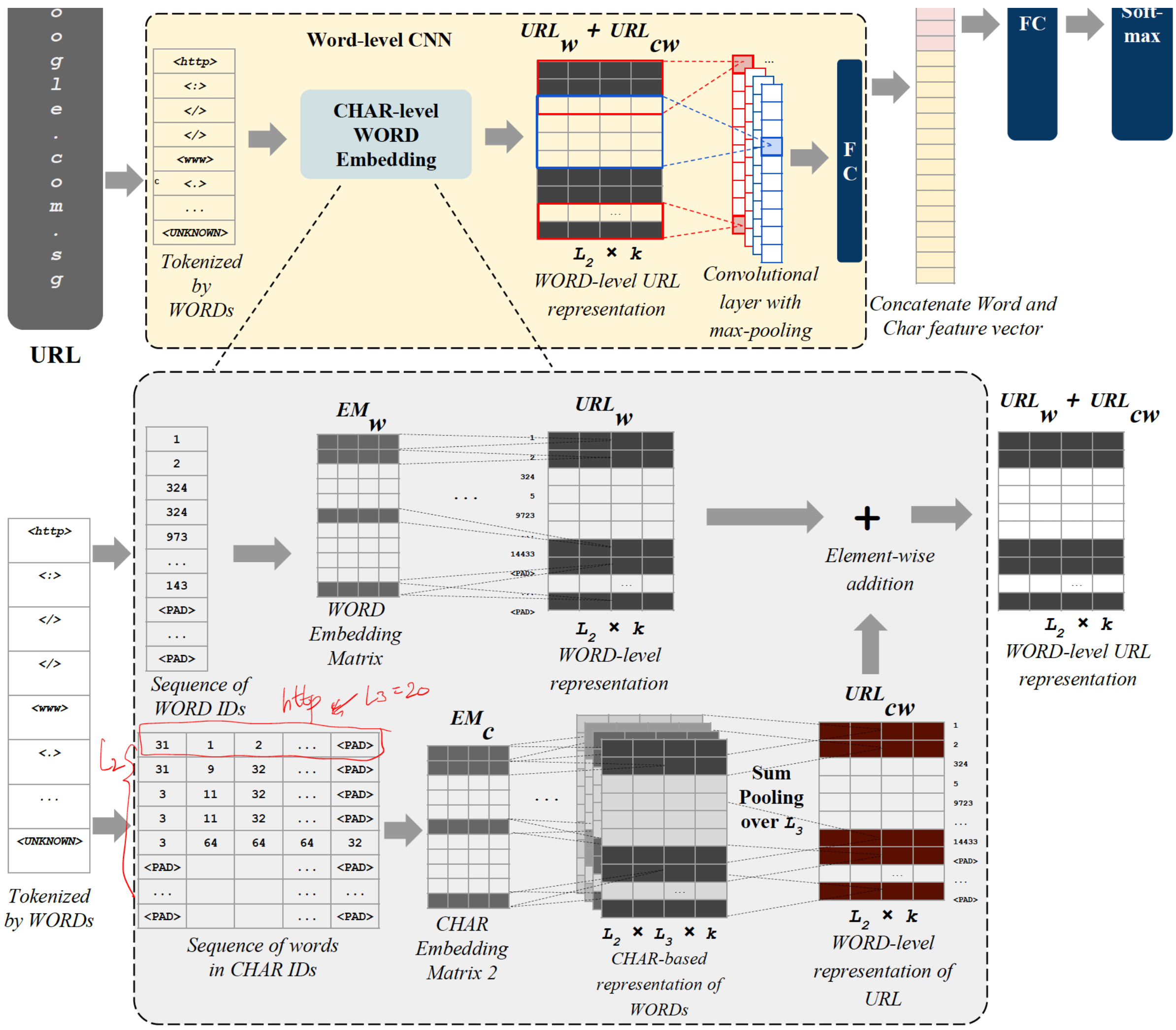
- but limitations : ignore word boundary, weak to attak of minor modification



Word-level embedding and Detection



Improved Word Embedding Using Character-level Word Embedding\



Experiments

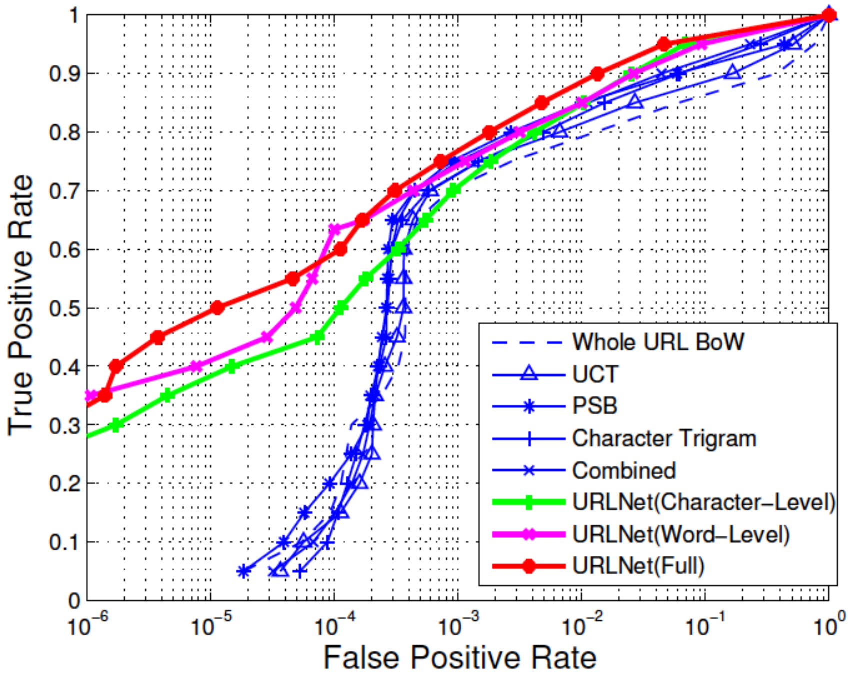
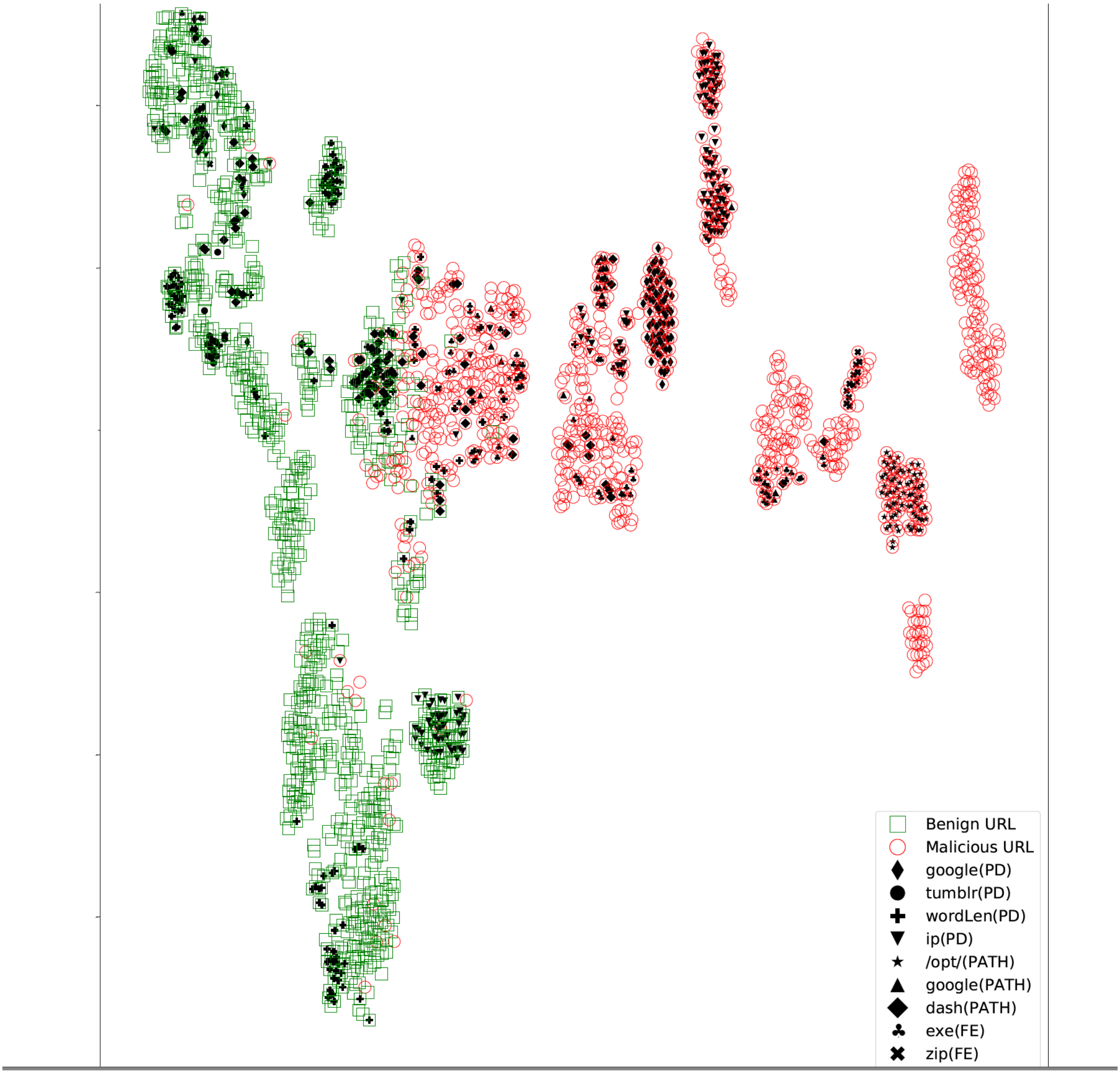


Figure 4: Area Under ROC Curve (Trained on 1m, Tested on 10m). URLNet(Full) is slightly worse than URLNet(Word-level) at FPR = 10^{-4}, but better otherwise. URLNet(Full) is consistently better than URLNet(Character-level). All URLNet variants outperform baselines.

Table 5: Examples of lexical patterns in URLs and example URLs. The lexical patterns are extracted at different parts of the URL string: primary domain, URL path, and file extension.

URL Component	Lexical Pattern	Example URL
Primary Domain	contains 'tumblr'	http://exampledomain.tumblr.com/
	contains 'google'	http://www.google.com/urlpath/... http://abcd123googlexyz456.com/urlpath/...
	contains IP	http://192.168.0.1/ http://192.168.0.1/urlpath/...
	has average word length >10	http://a1ds2dce0b33fdgd425d8fsgg9836c4234d0.exampledomain.net/
Path	contains 'google'	http://www.exampledomain.com/filename?f=GOOGLEEARTH... http://exampledomain.net/urlpath/googledrive/sub_dir/...
	contains '/opt/'	http://www.exampledomain.com/opt/...
	contains the *dash* pattern in the last path token	http://exampledomain.com/urlpath/abc-123-fff-456-...
File Extension	Includes a file with extension 'exe'	http://exampledomain.net/urlpath/filename.exe
	Includes a file with extension 'zip'	http://exampledomain.com/urlpath/filename.zip


Benign URL
Malicious URL
google(PD)
tumblr(PD)
wordLen(PD)
ip(PD)
/opt/(PATH)
google(PATH)
dash(PATH)
exe(FE)
zip(FE)

In [ ]: