FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Classification of URLs Using Deep Neural Networks
Student:	Matyáš Skalický
Supervisor:	doc. Ing. Pavel Kordík, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

Many URLs have some kind of semantic meaning - an URL may indicate that user is reading an article, viewing product page or making a purchase. The thesis should tackle the task of classification of URLs into a predefined set of semantic classes. Survey methods that are used for this purpose and put special attention to character level neural networks such as URLNet. Compare the performance of URLNet to trivial baseline models on a real-world dataset. Discuss the results and explain added value of deep learning preprocessing if any.

References

https://arxiv.org/abs/1802.03162

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 17, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Classification of URLs Using Deep Neural Networks

Matyáš Skalický

Department of Applied Mathematics
Supervisor: doc. Ing. Pavel Kordík, Ph.D.

May 16, 2019

Acknowledgements

I would like to thank my family for supporting me in all my interests and hobbies and for providing me with a home, where I'm always welcome. For the formal leadership, I would like to thank doc. Ing. Pavel Kordík, Ph.D. Special thanks goes to Ing. Ondřej Pluskal for many of the great tips during the development of my thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 16, 2019 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Skalický, Matyáš. *Classification of URLs Using Deep Neural Networks*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstract

My work explores the field of automatic URL classification with particular attention to character-level deep neural networks. It summarizes recent advancements in the field and proposes a working model which outperforms the enterprise baseline on a real world dataset.

Keywords url classification, character-level model, natural language processing, convolutional neural networks, machine learning, deep learning

Abstrakt

Tato bakalářská práce se zabývá problémem automatické klasifikace internetových adres. Důraz je kladen na hluboké neuronové sítě, konkrétně na modely, které pracují se vstupem na úrovni jednotlivých znaků. V práci je shrnutý současný stav řešení a je navržen model vhodný pro nasazení do produkce reálného firemního prostředí.

Klíčová slova klasifikace url, neuronové sítě, zpracování přirozeného jazyka, konvoluční neuronové sítě, strojové učení, hluboké účení

Contents

List of Figures

List of Tables

Introduction

Most of the URLs used in the World Wide Web carry a semantic meaning. This meaning helps users to identify the content of a website just by looking at the URL. Automatic detection of website category without the need of loading the site can be potentially very useful by a number of real life use cases: malicious URL detection, crawlers, parental control, network traffic logging and many more.

The dataset used in this thesis comes from a real production environment. The digital intelligence company which provided the dataset uses URL clickstream (sequence of URL addresses) to provide marketing analysis and insights into online consumer behavior. To extract valuable data from sequences of URLs, each URL is assigned a category. Hand written regular expressions are created and matched to the URLs to obtain a category describing the site's content. An automatic URL classification system is used as a tool which assists the employees developing the regular expressions when searching for URLs of a particular category. Currently, a traditional machine learning solution is deployed in the production.

First chapter describes a theoretical background behind the strategies of deep learning. It starts with a gentle introduction to the neural network models and then continues with some of the promising concepts in context of the URL classification task. A particular attention is put to the understanding of convolutional neural networks in the field of natural language processing. The theoretical section could be used as a summary by anyone looking for a comprehensive introduction to the character-level convolutional networks.

Second chapter first describes the dataset, and then surveys methods, which might be useful for URL classification. The most promising methods are implemented and their performance is compared in chapter 3. Some of the interesting experiments related to the hyperparameter selection are described in chapter 4. Finally, the best performing model is described in depth and evaluated against the industrial baseline in chapter 5.

Chapter 1

State of the Art

This chapter is meant to summarize some of the basic terms used in the field of machine learning and to introduce reader into some of its jargon.

1.1 Unique Resource Locator

Unique Resource Locator, abbreviated as URL, is the address of a resource on the World Wide Web. An URL is composed of the main two components (separated by a colon and two toward dashes): protocol identifier and a resource name. Protocol identifier specifies used internet protocol, while the resource name is an address or domain name where the resource is located. Example of an URL and its structure is shown on the Figure 1.1.

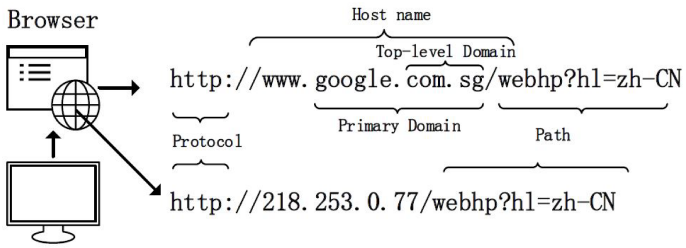

Figure 1.1: Structure of Uniform Resource Locator. Adopted from [1].

URLs are often meant to be easily understood by humans, and websites with good URL encoding design often include useful information into the URL itself [2]. Unique resource locators therefore impose extremely good features for machine learning [3]. They are easy to extract and can be read without the need of downloading the website which allows for fast classification.

1.2 Machine Learning

Traditional approach to solving a problem with help of a computer was to write a program with hardcoded instructions. This required a skilled engineer with knowledge of the domain and usually did not scale well with the task's complexity. *Machine Learning* (ML) uses a different approach to solving a problem. "A machine-learning system is trained rather than explicitly programmed." [4] Machine learning models gain all of their inference ability from data presented to them.

The rise of a new millennium did not only bring us powerful computers and the World Wide Web, but also lots of data to process. Machine learning takes an advantage of the amounts of data available to date. Modern machine learning models are often able to outperform some of the most complex hand crafted computer programs of the past. In some areas [5, 6, 7], highly specialized ML models have shown being able to exceed humans.

Based on the kind of input data and expected outputs, machine learning algorithms are traditionally divided [8] into 3 categories:

Supervised learning makes the model learn from input (image, text, array of numbers…) and a desired output (value or a class). The goal of the model is to learn the mapping between the input and output pairs. Ideally, the algorithm learns to generalize well and predicts correctly even on unseen data.

Unsupervised learning does not rely on labels but it instead tries to find similarities in data. Outputs of an unsupervised learning model are typically clusters (groups) of similar data.

Reinforcement learning falls, in a sense, between supervised and unsupervised learning. Some form of supervision exists, but the model receives its feedback from the interactions with the environment. This feedback indicates how well the model fulfills its tasks.

1.3 Deep Learning

The term *Deep Learning*, nowadays often an overused buzzword, is a subfield of machine learning. The word deep refers to a high number of consecutive layers used in a neural network. For complex tasks, it is not uncommon to see tens to hundreds [4] of hidden layers stacked together. This allows the network to obtain a great generalization ability and has proven its capability by obtaining state of the art results [9] in a wide variety of applications (computer vision, speech recognition, natural language processing, etc.).

1.4 Classification task

In supervised learning, the algorithm learns from pairs of inputs and desired outputs. The main goal of the classification task is to construct a function f that based on a set of inputs $X_0, X_1, \ldots, X_{n-1}$ predicts a variable Y, such that

$$Y \approx f(X_0, X_1, \ldots, X_{n-1}).$$

Opposed to the regression problem, results of the classification fall into a discrete set of labels obtained during the training phase.

Binary classification is a task which consists of classifying data into one of two distinct classes. Examples of binary classification models include Decision Trees, Random Forests or Neural Networks. Some of the classical machine learning algorithms (i.e. Support Vector Machine, Naive Bayes or Logic Regression) support, by default, only binary classification. However, there are several techniques which allow us to utilize a binary classifier on a multiclass classification task as well.

Multiclass classification consists of classifying the input data into three or more discrete classes. In terms of deep neural networks this is usually done by having a neural network with same number of output neurons as the number of possible classes. We apply a softmax activation on the last layer which yields the predicted probability distribution over each possible output class.

1.5 Natural Language Processing

Natural Language Processing (NLP) is an area of machine learning specialized in understanding and processing of natural language by computers. In a wide sense, it covers any kind of computer manipulation of the natural language.

In the past few years, natural language processing has undergone a massive growth [10]. NLP use cases range from machine translation [11], all the way to the text categorization, spam filtering and information extraction [12].

1.5.1 Tokenization

In order to feed the text into a machine learning model, the raw textual input must be first processed and converted into a sequence of tokens. There are several token representations, varying in the size granularity of the input chunk captured in the token.

The input dictionary usually needs to be predetermined before the start of learning in order to figure out the dimension of the input space. This, in terms of word-level models, means that all of the data needs to be loaded and preprocessed before the training of a model starts.

1.5.1.1 Word-Level Tokens

Word-level tokenization approach is the most commonly used approach among many of the NLP tasks. Words are basic building blocks of our language, so extracting them is a logical step. Words can be easily obtained by splitting sentences by spaces or other special characters. However, this method conveys several possible drawbacks. Human language often contains up to hundreds of thousands of words, and without stemming (process of reducing a word to its base form) the number of words the machine learning model can encounter might grow very high. When using a model with bag-of-words encoding, a large number of the possible words could cause the embedding dimension to explode.

1.5.1.2 Character-Level Tokens

Character-level tokenization encodes each character as a separate token. This approach yields a small embedding dimension and also has a low chance of seeing an out-of-dictionary character on the input.

1.5.1.3 N-gram Tokens

N-Gram is a group of N (or fewer) consecutive characters or words [4]. This helps us with the fact, that the bag-of-words encoding does not preserve any information about the order of individual tokens. By using the n-gram model, the order of N words or characters can be captured. We distinguish two n-gram types: character-level and word-level n-grams.

Character-level n-grams are groups of N consecutive letters. A benefit to using character n-grams is the limited number of tokens which might occur on the input. Lowercased english alphabet only consists of 26 characters. With 10 numbers, 16 punctuation marks, 1 character for space and 1 Out Of Vocabulary (OOV) token, we have a set of 54 possible characters which might occur in the input text. $54^3 = 157,464$ possible combinations of 3 consecutive characters can therefore occur on the input. The length of the feature vector would be $157,464$ and could be determined prior to training without the need of preprocessing of the whole input dataset.

Word-level n-grams are groups of N consecutive words. Since the number of possible words appearing in a language is usually high, the word n-grams might yield an embedding which is magnitudes larger than the single word embedding itself.

We will use URL *https://www.nic.cz/page/351/* as an example. In order to obtain words, we would split the URL by special characters. Example of tokens extracted from this url is shown in the following table:

Table 1.1: Examples of different token representations of an URL

type	token representation
word	*https*, *www*, *nic*, *cz*, *page*, *351*
character	*h*, *t*, *t*, *p*, *s*, *:*, */*, */*, *w*, *w*, *w*, *.*, *n*, *i*, *c*, ...
word-level 3-gram	*https www nic*, *www nic cz*, *nic cz page*, ...
character-level 3-gram	*htt*, *ttp*, *tps*, *ps:*, *s:/*, *://*, *//w*, */ww*, ...

An alternative encoding can be called character n-gram over words, where the input is first split into words, and character n-grams are then generated only from words not including spaces or other special characters.

1.5.2 Bag-of-words Model

Text is one of the most common forms of sequence data. It can be understood as sequences of characters, or sequence of words [4]. Juravsky and Martin [14] define the *bag-of-words* (BOW) model as "an unordered set of tokens with their position ignored, keeping only their frequency in the document." For example, let's assume the model's vocabulary (set of possible tokens) is:

{ good, drink, restaurant, beer, people, stairs }.

A sentence "Good people drink good beer." would be then converted to a term vector $\vec{v}$ of word-level tokens as:

$$\vec{v} = (\ 2 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0\ ).$$

By keeping the word count in the term vector, we can later use it to construct a weight for each word or even easily select top-k most occurring words. Other approach to the bag-of-words model neglects the word counts and only marks whenever the word was present. Such representation of the above sentence would be:

$$\vec{v} = (\ 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0\ ).$$

1.5.3 Feature Embeddings

When dealing with natural language or other text structures, we first use a bag of words model to obtain a set of tokens, also called model's vocabulary. Every tokenized input sequence can then be be represented as an array of token indices in the bag of words model. Storing input as an integer array is an efficient way to store tokenized sequences. This representation can be easily converted to a one-hot representation.

1.5.3.1 One-hot Representation

One-hot encoding is the most common and basic way of turning a token into a vector representation. Each of the tokens obtained in a bag of words can be expressed in so called *one-hot representation* - a zero vector of length equal to the size of the vocabulary with a single 1 on the i-th position for token of index i. The one-hot encoding is completely uninformed - it does not account for any similarities.

1.5.3.2 Dense Embedding

One-hot representation is a very sparse encoding. Depending on the size of the input vocabulary, one-hot vectors can have a very high dimension. When using a neural network, using high dimensional inputs can lead to a large number of trainable parameters. Instead, we can represent the inputs as dense representations of lower dimensions. The reduced input dimension is not the only feature of dense embeddings. As [13] states "The main benefit of the dense representations is in generalization power: if we believe some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities."

Embedding layer in the Keras library can be understood as "...a dictionary that maps integer indices (which stand for specific words) to dense vectors. It takes integers as input, it looks up these integers in an internal dictionary, and it returns the associated vectors. It's effectively a dictionary lookup." [4] In background, the layer is implemented as a multiplication of a one-hot vector with a trained embedding matrix. The embedding matrix is initialized randomly and trained with the rest of the network via backpropagation. As [13] states "it may be worthwhile to let the network figure out the correlations and gain some statistical strength by sharing the parameters."

Pretrained word embeddings like Word2Vec [15] or GloVe [16] are an alternative to embeddings trained jointly with the model. Such embeddings are generally pretrained in unsupervised manner. Output dense vectors generally reflect the word co-occurrence statistics. Pretrained embeddings can often be downloaded for different languages.

1.6 Neural Networks

The basic unit of the neural network, the perceptron, was described by Frank Rosenblatt [17] in 1958. The structure of such a unit is vaguely inspired by the natural structure of the human brain. Since the discovery of perceptron, the humanity has gone through multiple machine learning revolutions, all fueled by advancements in the field of neural network research.

The deep learning (1.3) has shown to be able to outperform most of the classical machine learning algorithms in cases, where there is enough data and computational power available.

1.6.1 Perceptron

Perceptron denotes the simplest part, a single cell of the neural network. The original Rosenblatt perceptron used a non-differentiable step activation function and worked as a simple, single-layer binary classifier.

1.6.1.1 Structure

Perceptron maps an array of inputs $x = [1, x_1, x_2, \ldots, x_n]$ to an output $\hat{Y}$ using an array of weights $w = [w_0, w_1, w_2, \ldots, w_n]$. Weight w_0, usually called bias, is always multiplied by 1 and therefore does not change with different input data. To obtain the inner potential ξ, each input is multiplied by a corresponding weight and then summed.

$$\xi = \sum_{i=0}^{n} w_i x_i = w^T x \tag{1.1}$$

The output $\hat{Y}$ is obtained by applying a non-linear activation function f to the inner potential ξ as follows:

$$\hat{Y} = f(\xi) = f\Big(\sum_{i=0}^{n} w_i x_i\Big) = f(w^T x) \tag{1.2}$$

1.6.1.2 Activation function

The Rosenblatt perceptron originally used a step activation function, but for simplicity, we will use the unit step activation function f defined as:

$$\mathrm{f}(x) := \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{otherwise} \end{cases} \tag{1.3}$$

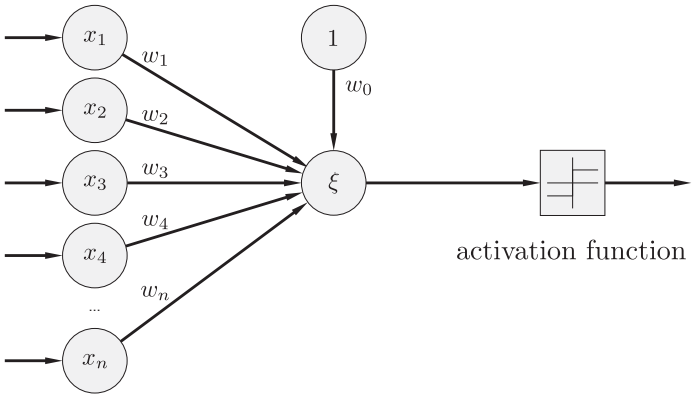

Figure 1.2: Structure of a perceptron

1.6.1.3 Learning

The perceptron automatically tries to adjust its weights to mimic the expected output data. The training [18] happens as follows:

1. In the beginning, the weights are initialized randomly.
2. For each data sample:
   a) calculate the output $\hat{y}^i$ for current input x^i
   b) update the weights $w_0, w_1, \ldots w_n$ to match the result $\hat{y}^i$ with expected output y^i. The update $\triangle w_i$ is calculated as:

$$\begin{aligned} \triangle w_0 &= \eta(y^i - \hat{y}^i) \cdot 1 \\ \triangle w_1 &= \eta(y^i - \hat{y}^i) \cdot x_1^i \\ &\ldots \\ \triangle w_n &= \eta(y^i - \hat{y}^i) \cdot x_n^i \end{aligned} \tag{1.4}$$

It is important to note, that the convergence of a Rosenblatt perceptron is guaranteed only if the training data is linearly separable.

1.6.2 Multi Layer Perceptron

The true power of the neural networks comes from stacking multiple layers of perceptrons into a *multi layer perceptron* (MLP) model. We distinguish input, hidden and output layers of the neural network. Following mathematical notation is inspired by [19].

- Lets describe a l-layers deep neural network where $n_1, n_2, \ldots, n_l$ are the numbers of neurons in each layer.

- Output of the j-th neuron in an i-th layer can be described as a function $g_j^{(i)} = \mathbb{R}^{n_{i-1}} \to \mathbb{R}$ which has on its input the outputs of n_{i-1} neurons from a previous layer.

- $g_j^{(i)}(x) = f(w^T x)$, where f is an activation function, $w = [w_0, w_1, \ldots, w_n]$ are the layer weights and $x = [1, x_1, \ldots, x_{n_{i-1}}]$ are the inputs from the previous layer.

- To obtain the output value in a fully connected MLP, we calculate the values of each individual neuron, layer by layer, from the input to the output layer. The whole forward pass can be described as a function $g : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ which is created by stacking the layers as

$$g = g^{(1)} \circ g^{(2)} \circ \ldots \circ g^{(l-1)} \circ g^{(l)}. \tag{1.5}$$

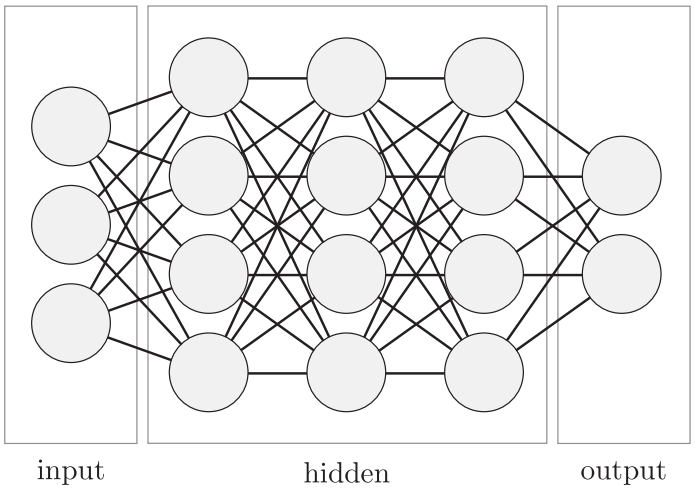

Figure 1.3: Structure of a fully connected neural network

1.6.2.1 Learning

Neural network is basically a function which maps n inputs to m outputs as:

$$f(x_1, x_2, \dots x_n) = y_1, y_2, \dots, y_m \tag{1.6}$$

To learn the network, we adjust the network's parameters (weights). To adjust the weights, we utilize a loss function such as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1.7}$$

where $\hat{y}_i$ is the predicted value and y_i the expected value. We then try to minimize the output of such loss function.

An algorithm, which enabled effective learning of multi layer percetron networks, *backpropagation* [20] was popularized by Rumelhart, Hinton, and Williams in 1986. The algorithm requires that the neural network is a differentiable function. Weights are initialized as small random numbers in the beginning. Training of a single batch can be described [19] as follows:

- We initialize the error $J(w)$ to 0.
- For each training sample (x_i, Y_i) in the batch:
  - we calculate the predicted output

$$\hat{Y_i} = f(x_i) \tag{1.8}$$

  - and update the error as

$$J(w) \leftarrow J(w) + \frac{1}{n}(Y_i - \hat{Y_i})^2. \tag{1.9}$$

- Then, the gradient is calculated as

$$\nabla_w J = \Big(\frac{\partial J}{\partial w_1}, \dots \frac{\partial J}{\partial w_m}\Big)^T \tag{1.10}$$

- and weights are updated based on the learning rate η and gradient as

$$w \leftarrow w - \eta\nabla_w J. \tag{1.11}$$

1.6.3 Activation Function

Activation function's main purpose is to introduce non-linearity into the network and to control the information passed to the next layer. Different activation functions vary in convergence speed, as well as resources required for the computation. In order to compute the gradient vector during backpropagation, activation function should be differentiable. "The weight initialization and the activation function of deep neural networks have a crucial impact on the performance of the training procedure." [21] As [13] states, ReLU units work usually better than TanH and TanH works better than sigmoid in deep learning applications.

1.6.3.1 Sigmoid

Sigmoid activation function has been used widely in the past. However, sigmoid suffers from several drawbacks. Major one being the so-called *vanishing gradient* [21]. This problem is caused by the small derivative of a sigmoid function when training deep neural networks. When calculating the weight change during backpropagation, we work with derivatives of activation functions, which in case of sigmoid, are very small numbers. This can result in model only doing very small updates when training.

$$f(x) = sigmoid(x) = \sigma(x) = \frac{e^x}{1 + e^x} \tag{1.12}$$

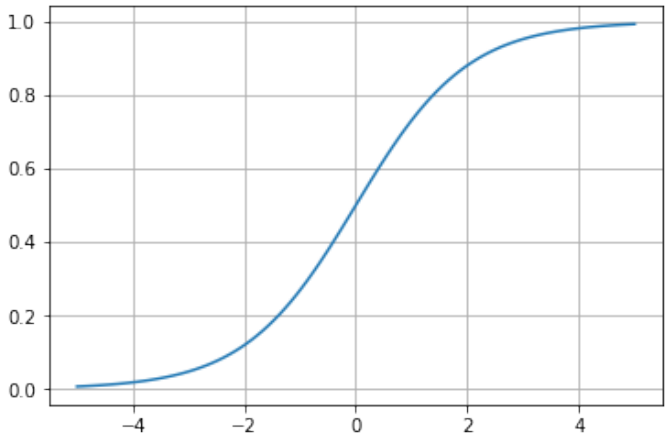

Figure 1.4: Sigmoid activation function

1.6.3.2 Hyperbolic Tangent

Hyperbolic tangent (TanH) is a zero–centered function with the output range between -1 and 1. The TanH function if often preferred over sigmoid activation function thanks to its better training performance for multi-layer neural networks. Hyperbolic is also known to suffer from the vanishing gradient problem. The main advantage is that it produces zero centered output thereby aiding the back-propagation process [22].

$$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{1.13}$$

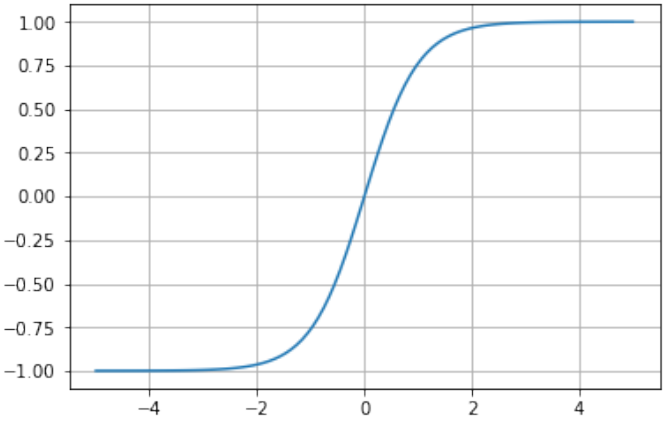

Figure 1.5: Hyperbolic Tangent activation function

1.6.3.3 Rectified Linear Unit

Thanks to its simplicity and effectiveness when training deep neural networks, *rectified linear unit* (ReLU) has become the default activation function across the deep learning community [22]. The benefits include good gradient propagation, fast computation and scale invariance. Also, the ReLU is one-sided (biologically plausible) and introduces sparsity into the network. (Only 50% of hidden units have non-zero output after random initialization.) ReLU is not differentiable at 0, however we can manually define the value of derivation to be either 0 or 1. Contrary to classical activation functions (sigmoid, TanH etc…), ReLU does not suffer from the gradient vanishing problem [21, 13].

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{1.14}$$

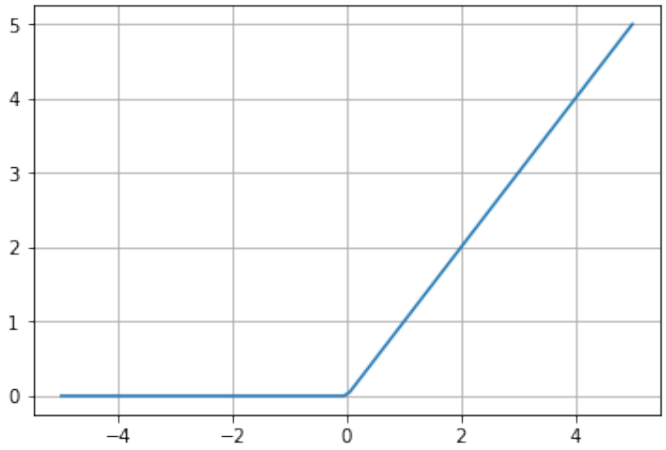

Figure 1.6: Rectified Linear Unit activation function

1.6.3.4 Exponential linear unit

Exponential linear unit (ELU) [23], described only recently in 2015, is an activation function, which has shown to be able to speed up learning of deep neural networks. Unlike ReLu, ELU can produce negative outputs and therefore has shown to push the mean activations closer to zero which aids the training process.

$$f(x) = \begin{cases} e^x - 1 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{1.15}$$

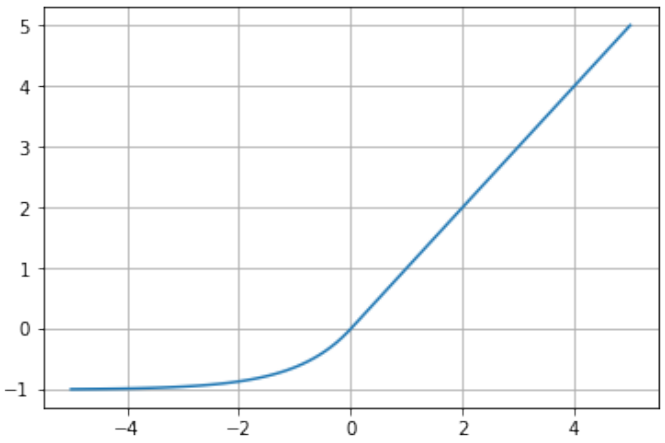

Figure 1.7: Exponential Linear Unit activation function

1.6.3.5 SoftMax

Contrary to other previously described activation functions, *SoftMax* requires outputs of the whole output layer in order to be computed. The output lies between 0 and 1 and sums to 1 for the whole layer. SoftMax is often used as a last layer of neural networks in multi-class classification tasks. The output represents the predicted probability of input belonging to the i-th class [13].

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{1.16}$$

1.6.4 Regularization

When training neural networks, especially on small datasets, we might see the neural network to learn the outputs for the training data really well. But when it is presented with unseen data, it fails to generalize. Behavior of having small bias on training data, but high variance on test data is called overfitting. Best solution to fight the overfitting is to get more data, a process, that is not always possible. "Regularization is any supplementary technique that aims at making the model generalize better, i.e. produce better results on the test set." [24] This sections describes some of the most popular techniques used to deal with overfitting.

1.6.4.1 Reducing the Size of a Neural Network

The idea behind this approach is, that reducing the size of a neural network also reduces its capacity to learn perfect mappings between specific inputs and outputs. As [4] notes: "Deep learning models seem to be very good at fitting to the training data, but the real challenge is generalization, not fitting."

1.6.4.2 Weight Regularization

With high number of parameters in a neural network, there might be many possible settings of weights to produce the same training error. Models with large weights can signalize an unstable neural network, where small changes in the input can lead to significant differences on output. This might be a sign of overfitting. To reduce the variance we can add a regularization term [4], which will be added to our loss function and will encourage the network to keep the weights small.

L1 regularization — The cost added is proportional to the sum of absolute weight coefficients. (L1 norm)

$$loss = L(Y, \hat{Y}) + \lambda \sum_i |w_i| \tag{1.17}$$

L2 regularization — The cost added is proportional to the sum of squared weight coefficients. (L2 norm)

$$loss = L(Y, \hat{Y}) + \lambda \sum_i w_i^2 \tag{1.18}$$

1.6.4.3 Dropout

Dropout, proposed by Geoffrey Hinton et al. [25], is one of the most widely used and most effective regularization techniques for neural networks. "During training, dropout, applied to a layer, randomly changes some of the layer weights to zero." [4] The core idea is that by dropping out some of the weights we break some of the insignificant patterns and motivate the network to generalize well.

1.6.4.4 Batch Normalization

Batch normalization, proposed by Sergey Ioffe and Christian Szegedy in 2015 [26] is a technique which improves stability, performance and training speed of neural networks. The goal is to transform the layer inputs to have a mean output activation of zero and standard deviation of one [27]. It works by adding two trainable parameters to each layer. The normalized output is multiplied by a "standard deviation" added a "mean" based on the trained parameters [28].

1.7 Convolutional Neural Networks

Convolutional neural networks (CNN) are a category of neural networks which gained its popularity mainly with the computer vision and image-driven pattern recognition tasks [29].

Convolutional layers work by applying the convolution operation by sliding an n-dimensional window (also called filter) over the input matrix, multiplying its weights element-wise with the input values and summing the product. This process is shown on the Figure 1.8. To introduce non-linearity, an activation function (1.6.3) is applied to each individual summed output value. The filter weights are randomly initialized at start and are trained by backpropagation.

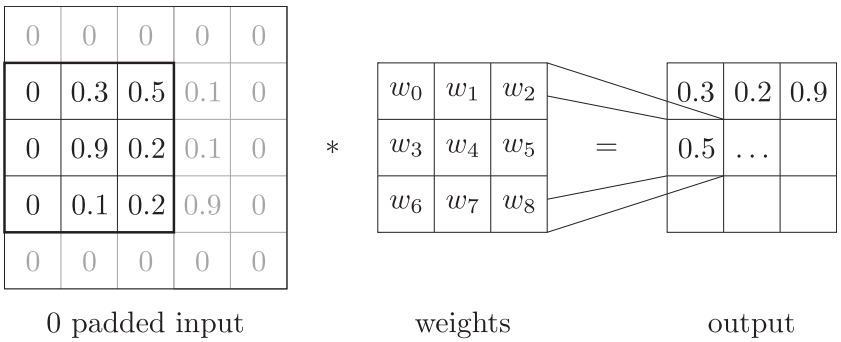

Figure 1.8: 2D convolution on a zero padded input with 3×3 filter size

1.7.1 Padding

Padding consists of adding an appropriate number of rows and columns of zeros on each side of the input matrix [4]. There are several reasons to use the padding with CNNs:

size of the output: by adjusting the padding size, we can control the size of the output

information loss: the data close to the border of the input matrix would be seen less by the convolutional filter and therefore would have smaller effect on the output

1.7.2 Stride

Other parameter influencing the size of the output of a convolutional layer is stride. Stride basically means the distance between 2 consecutive applications of the convolution window. By controlling the size of stride, we can effectively control the size of the output. Since the stride has effect on the output size, pooling layers can sometimes be replaced [30] by convolutional layers with stride larger than one.

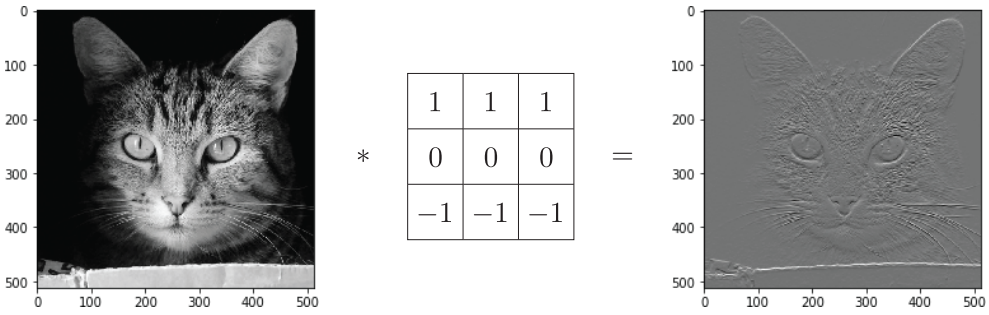

Figure 1.9: Example of convolutional filter detecting horizontal lines

1.7.3 Max Pooling

Max pooling is the most popular technique of downsampling the input into a smaller number of features. By reducing the size of the convolutional output, we can connect the convolutional layers to a fully connected layer of a reasonable size. Max pooling works similar to convolution operation and also utilizes a sliding window but with stride equal to the size of a window. Contrary to the convolution, the output is simply a maximum value of the sliding window applied to the input matrix.

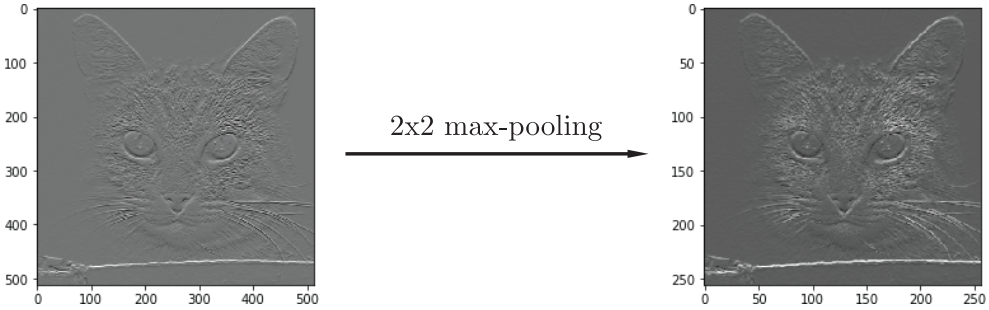

Figure 1.10: Example of 2x2 max pooling applied on the output of 1.9, the output is downsampled by a factor of 2.

1.7.4 Global Max Pooling

Global max pooling is described as an operation, which performs a max pooling operation over the whole input. Result of a max pooling operation is a single value. This allows us to take an input from a convolutional filter of arbitrary size and downsample it into a single value.

1.7.5 Usage in Text Classification

The problem with using a bag of words model is that the order of tokens is not preserved. To tackle this issue, convolutional filters of different sizes can be used on top of input embeddings to learn to identify useful patterns from certain groups of tokens appearing together.

This process can be seen on the Figure 1.11. As [31] points out, CNNs can be used effectively for text classification task. They can achieve great results even without excessive parameter tuning. A description of model using a CNN for text classification is visualized on the Figure 1.12 and can be summarized [13] as follows:

1. Important n-grams are detected by Convolution1D filters, each learning a specific category of n-grams.

2. MaxPooling1D is used to detect the most activated filters.

3. Fully connected layers perform the decision making.

The function of convolutional filters in text classification is researched and explained in [32]: "Current common wisdom posits that filters serve as ngram detectors: each filter searches for a specific class of ngrams, which it marks by assigning them high scores. These highest-scoring detected ngrams survive the max-pooling operation. The final decision is then based on the set of ngrams in the max-pooled vector (represented by the set of corresponding filters). Intuitively, ngrams which any filter scores highly (relative to how it scores other ngrams) are ngrams which are highly relevant for the classification of the text."

The paper [32] also mentions that "...filters are not homogeneous, i.e. a single filter can, and often does, detect multiple distinctly different families of ngrams. Filters also detect negative items in ngrams – they not only select for a family of ngrams but often actively suppress a related family of negated ngrams."

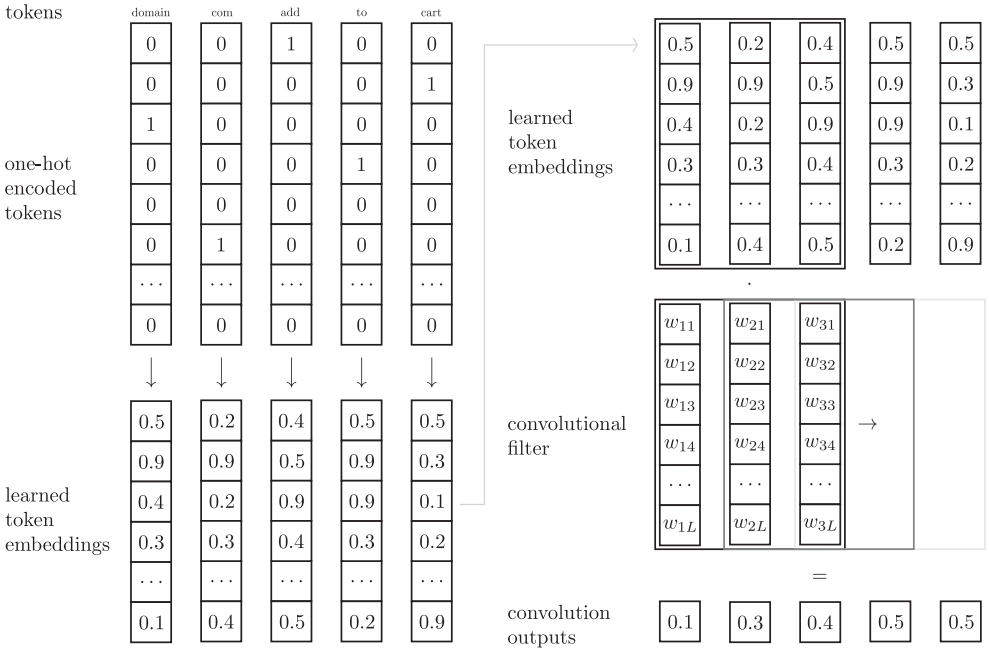

Figure 1.11: Sparse one-hot feature vectors are transformed into dense k-dimensional embedding vectors and a convolutional filter is applied.

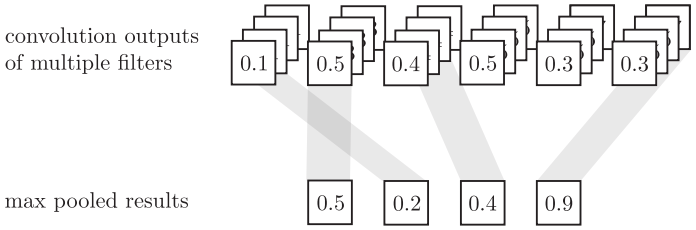

Figure 1.12: Global max pooling over the filter outputs produces a fixed-size output equal in size to the number of used filters.

1.8 Recurrent Neural Networks

One of the limitations of the feed forward neural networks is their lack of "memory". By processing each input separately, the network has no knowledge of previous inputs when making a decision [4]. In order to process sequences with such networks, we must present the whole sequence on the input at the same time. *Recurrent Neural Networks* (RNN) work by sequentially iterating through the input and keeping an internal state. This is achieved by using the output from the previous state as one of the inputs to the network. This is visualized on the Figure 1.13.

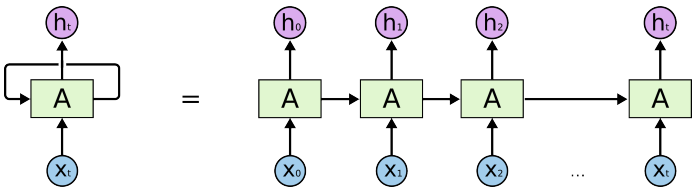

Figure 1.13: Unrolled recurrent neural network. Adopted from [33].

One of the features of RNNs is also that they designed to process inputs of arbitrary length [34].

1.8.1 Limitations of Recurrent Neural Networks

While theoretically being very capable, RNNs pose several limitations. Recurrent neural networks can be sensitive to the vanishing and exploding gradient problem [35]. RNNs are not effective when learning dependencies of 5 to 10 discrete time steps or more. The propagated gradients can become too small, or blow up [36]. The other limitation of recurrent neural networks is the demanding computational requirements of training RNN models [37].

1.8.2 Long-Short Term Memory

Long Short Term Memory [38] networks (LSTM), proposed by Hochreiter and Schmidhuber (1997), are a type of a recurrent neural network capable of learning long-term dependencies. LSTM tries to solve the issue of standard RNN which struggles to model long-range dependencies.

In addition to using an internal state, LSTM cell utilizes specialized "gates" to control the flow of information. Typical LSTM cell [39] contains:

input gate – controls the flow of input information

forget gate – controls whether to keep or forget data in the hidden state

output gate – allows or prevents the internal state to be seen from outside

Since texts are basically sequences of tokens (characters, words), RNNs and especially LSTMs are often used model natural language processing tasks. Structure of a typical LSTM cell and gates used is shown in the Figure 1.14.

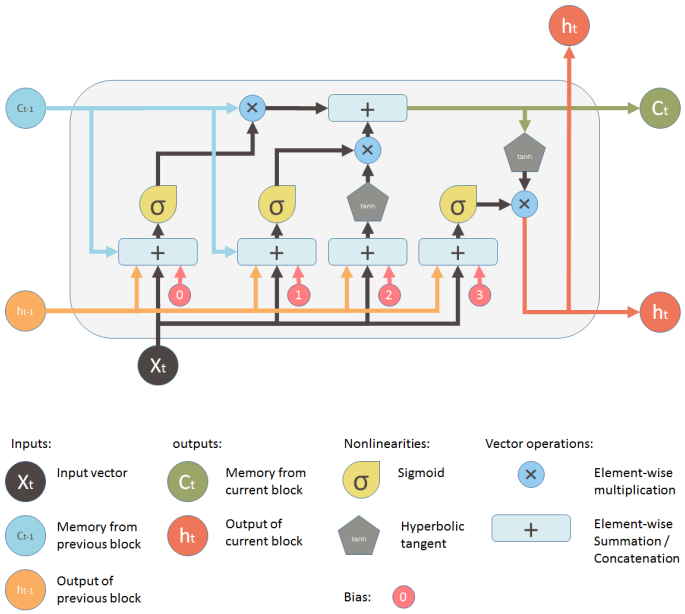

Figure 1.14: Structure of a LSTM cell. Adopted from [40].

1.9 Measuring Performance

During the development of a machine learning model, the developer needs to have a feedback about the performance and capability of a model. The development process often involves testing many different architectures and possible features.

A common good practice is to determine the goals of the model and choose an appropriate numerical metric which reflects the desired outcome. After the metric is chosen, a simple baseline model is created to obtain the first score on validation data. A baseline then serves as a springboard for further model development and as a comparison to further, more advanced models.

The upcoming sections examine different metrics that can be used to evaluate binary classification with possible extension to the multi-class classification problem.

1.9.1 Accuracy

Accuracy is the simplest metric, with the easiest interpretability, but with a huge drawback when working on unbalanced datasets.

$$\text{Classification accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{1.19}$$

As the above equation 1.19 shows, accuracy can be very easily applied even to a multi-class classification problem returning the percentage of correctly classified samples. However, in cases when dealing with skewed datasets, the tested model can show very high accuracy even though it simply returns the most common class.

1.9.2 Precision and Recall

A common performance measurement visualization for machine learning classification tasks is a confusion matrix. In binary case, it is a table containing 4 different combinations of numbers of predictions (true/false) and actual values (positive/negative):

Table 1.2: Confusion matrix

	meaning	description
TP	True Positive	data labeled positive correctly predicted
TN	True Negative	data labeled negative correctly predicted
FP	False Positive	data labeled negative incorrectly predicted as positive
FN	False Negative	data labeled positive incorrectly predicted as negative

Terms *precision* and *recall* come from the data retrieval field, where they are used to describe performance of a query. Precision describes the number of data correctly predicted as positive out of all data labeled as positive. Or in the data retrieval terms, the fraction of correctly retrieved documents out of all retrieved. "It answers the question: 'When it predicts the positive result, how often is it correct?' " [41]

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{\#\text{ correctly retrieved documents}}{\#\text{ all retrieved documents}} \tag{1.20}$$

Recall, on the other hand, is described as the fraction of data correctly predicted as positive out of all data that should have been predicted as positive. In data retrieval, the fraction of correctly retrieved documents out of all relevant documents. "It answers the question: 'When it is actually the positive result, how often does it predict correctly?' " [41]

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{\#\text{ correctly retrieved documents}}{\#\text{ all relevant documents}} \tag{1.21}$$

Both precision precision (1.20) and recall (1.21) return a number x from range $0 \leq x \leq 1$. Ideally, we would like them both to be one (everything correct), but usually there is a tradeoff between having a high precision and a high recall.

By maximizing precision we motivate our model to classify a smaller number of classes as a positive yielding a low number of false positives (FP). By maximizing recall, we maximize how sensitive the model is. It is often used to limit the number of false negatives (FN).

1.9.3 F-Score

Instead of keeping track of both precision and recall, it is often simpler to monitor a single metric. F_β-score is defined as:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{1.22}$$

When $\beta = 1$ this (F_1) score represents a harmonic mean of precision and recall. In order for model to obtain a high F_1-score, the precision and recall both need to be high values.

The beta parameter determines the weight of precision in the combined score. $\beta < 1$ lends more weight to precision, while $\beta > 1$ favors recall ($\beta \to 0$ considers only precision, $\beta \to \text{inf}$ only recall) [42].

In case of a multiclass classification problem, a composite score can be computed by two different types of average: *micro-average* and *macro-average*. To calculate the micro-averaged F_1 score, we first compute the global precision (p) and recall (r) as [43]:

$$p = \frac{\sum_{i \in M} TP_i}{\sum_{i \in M}(TP_i + FP_i)}, \; r = \frac{\sum_{i \in M} TP_i}{\sum_{i \in M}(TP_i + FN_i)}. \tag{1.23}$$

The micro-averaged F_1 score is then calculated as:

$$F_1(\text{micro-averaged}) = \frac{2pr}{p + r}. \tag{1.24}$$

To obtain the F_1 macro-averaged score, the F_1 score is computed separately for each class $i \in M$ and divided by the number of classes $|M|$ [43]:

$$F_1(\text{macro-averaged}) = \frac{\sum_{i \in M} F_{1i}}{|M|}. \tag{1.25}$$

"It is understood, that the micro-averaged scores (recall, precision and F_1) tend to be dominated by the classifier's performance on common categories and that the macro-averaged scores are more influenced by the rare categories." [44]

1.9.4 ROC Graph

A receiver operating characteristics (ROC) graph is a technique for visualizing performance of classifiers [45]. It is a useful measure especially on unbalanced datasets where accuracy can often be a poor metric.

To draw a point in ROC space, we first need to measure the classifier's tp-rate (true positive rate, also called recall, defined in 1.21) and fp-rate (false positive rate, defined in 1.26). These values are then plotted in a two-dimensional graph where tp-rate is displayed on the Y axis and fp-rate is displayed on the X axis.

$$\text{fp-rate} = \frac{FP}{FP + TN} \tag{1.26}$$

The following description of ROC analysis is distilled from the paper *An introduction to ROC analysis* [45] by Tom Fawcett:

ROC graph shows relative tradeoffs between benefits (true positives) and costs (false positives). A trained binary classifier produces a point in the graph. By looking at the ROC graph, we can compare multiple classifiers by looking at the visualization. Informally, a point in ROC space is better than another of it is to the northwest (closer to the top left corner). The diagonal $y = x$ represents the strategy of random guessing.

1.9.5 ROC Curve

Some of the classifiers such as Naive Bayes classifier or neural networks naturally predict a probability of instance belonging to a class. Such classifier returning a relative score can be thresholded (a static threshold can be selected) to produce a discrete (binary) classifier. By trying all of the possible thresholds and projecting the ROC points into a graph *ROC curve* is constructed.

1.9.6 Area Under ROC Curve

By calculating an area under ROC curve (abbreviated as *ROC AUC*) a score can be obtained. "The AUC has an important statistical property: the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance." [45]

In case of multi-class classification, ROC AUC score is measured separately for each class. A single composite AUC score can be then calculated for example as an average of AUC scores for each class (macro averaged ROC AUC score). In cases, when the classes are not balanced, each AUC score can also be weighted by the classes prevalence in data.

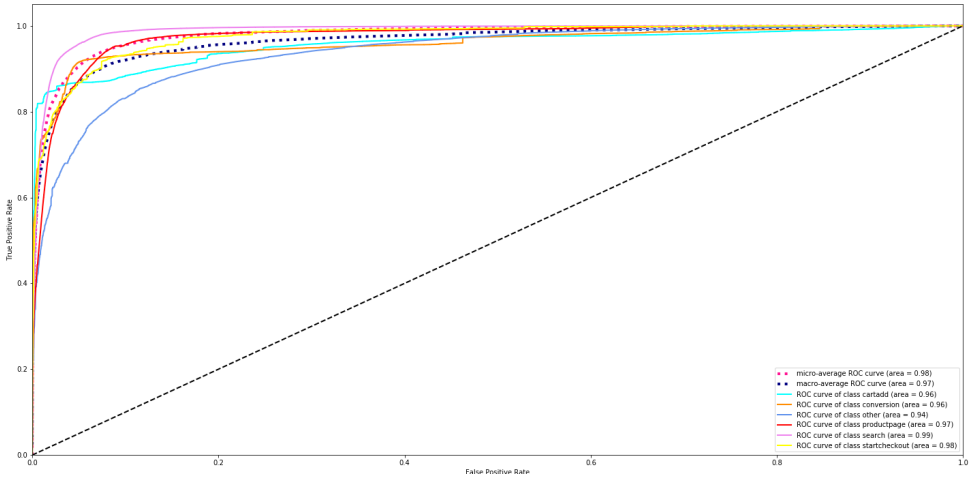

Figure 1.15: Example of a ROC curve plotted separately for each class.

Chapter 2

Analysis

Following chapter first analyzes the dataset and then explores the algorithms and current approaches related to the URL classification task.

2.1 Dataset

The dataset was provided by a big-data marketing company specializing in customer behavior analysis. For legal reasons, the dataset is not publicly available. It consists of $2,794,909$ sample URLs from 1494 unique domains belonging to one of the six categories:

Table 2.1: Categories present in the dataset

category	description	# of samples
cartadd	product was added to cart	119,511
startcheckout	checkout has started	144,140
search	searched for an item	624,521
conversion	purchase was completed	55,837
productpage	product page	1,036,758
other	any other category	814,142

To save space while preserving information, the dataset was preprocessed and duplicate URLs were removed. This led to a compression from $14,138,338$ to $2,794,909$ samples. The number of identical URLs was stored for each sample as url_count. To motivate the model to obtain the best generalization ability, a sample_weight was computed for each sample as:

$$\text{sample\_weight} = \frac{\text{\# of identical URLs}}{\text{\# of URLs in the same category and domain}}$$

Purpose for this weight is to motivate the model to learn every category per each domain with the same importance since the numbers of URLs in each category of different domains can be vastly different.

The dataset was split by domain into train, validation and test datasets. It is important that the samples in the test and validation sets do not contain domains from the train dataset as it is relatively easy for model to learn a representation which works on a particular domain.

Table 2.2: Dataset train, validation and test split

type	# of domains	# of samples	# of unique URLs
train	895	7,169,883	1,473,485
validation	300	3,693,060	688,167
test	300	3,275,395	633,257

Training dataset was used to train the models, while the role of the validation dataset was to tune hyperparameters of the proposed models. A holdout test dataset was kept until the final model was selected and then used to compare performance against the already-implemented production baseline.

2.2 Related Work

The most popular use case of URL classification is for the malicious internet address detection. This task has been traditionally tackled by manually created domain blacklists. This approach, however, requires time exhaustive human labor. This led to usage of automated machine learning techniques. The go to method being logistic regression or SVMs with bag of words features. These models are fast, but suffer from several limitations: they often fail to capture the semantic meaning of the URL, they often require manual feature engineering and fail to generalize well on unseen data.

Topic of this thesis falls under the field of natural language processing, specifically text classification. This a common and widely used practice with usage ranging from automatic social media monitoring, sentiment analysis, spam detection [46] all the way to the automated malicious (phishing) or spam URL detection [9, 47].

Related work can be divided into 4 sections: first section (2.2.1) summarizes the usage of *classical machine learning models* for the topic of automatic URL classification. The second (2.2.2) and third (2.2.3) sections describe the usage of *character-level* convolutional neural networks for text and respectively for URL classification. In the fourth section (2.2.4), *other* character-level URL classification techniques are presented.

2.2.1 Classical Machine Learning Models in URL Classification

The single most popular usage of URL classification systems is the binary task of malicious URL detection. Traditional approach was to compare the URL to a phishing database like PhishTank[1] and check, if it has been tagged as malicious before [1]. The process of adding malicious URLs to the database is time-consuming, and the resulting entry to the database covers only a specific website. This approach is still the most commonly used [1] by the most antivirus solutions.

To be able to detect unseen, untagged websites as malicious, machine learning learning solutions were developed and the process of URL detection was automated. Any general classification model can be used with an URL represented as a feature vector created of bag-of-word features. Most popular models among the automatic website detection field are Support Vector Machines (SVMs), Logistic Regression, Naive Bayes and Random Forests [1, 48].

Several studies compared use of the traditional models mentioned above. As [49] shows, Logistic regression and SVMs outperform the Naive Bayes model on several URL classification datasets. [50] compared kNN (k-Nearest Neighbors), SVM and a Random Forest algorithm on the UCI Phishing Website dataset. Random Forest classifier was found to be the best performing algorithm for this task. Random Forest algorithm also performed best [51] on a Twitter spam URL dataset.

2.2.2 Deep Learning Methods for Text Classification

The following section describes 3 pioneering works in the field of text classification using convolutional neural networks. The usage CNNs for text classification was first proposed by Kim Yoon [31] (2014) in paper *Convolutional deep learning methods for text classification.* Yoon utilized convolving filters on top of the existing pretrained word2vec vectors. Paper published a year later by Zhang et al. [52] (2015) explores the use of convolutional neural networks solely with character-level inputs. Conneau et. al [53] (2016) takes this idea further by proposing a deep convolutional architecture for text classification.

2.2.2.1 Convolutional Neural Networks for Sentence Classification

Convolutional Neural Networks for Sentence Classification, a paper by Kim Yoon [31] (2014) shows, that convolutional neural networks, widely used in the computer vision community, can be effectively used for the text classification task. A simple CNN with one convolutional layer is trained on top of pretrained word vectors obtained from an unsupervised language model Word2Vec.

[1] `https://www.phishtank.com/`

Sentences are padded with zeros or truncated to a fixed length of 300 tokens. The model utilizes filters of sizes 3, 4, 5 with 100 feature maps each, dropout rate of $p = 0.5$ and L_2-norm constraints on weight vectors of size 3.

This simple model achieves excellent results on multiple benchmarks. Paper points out, that pre-trained vectors are "universal" feature extractors and can be utilized for various classification tasks. Also that using dropout constantly improves the classification performance by 2–4 %.

2.2.2.2 Character-level Convolutional Networks for Text Classification

Zhang, X.; Zhao, J.; et al. [52] (2015) propose the use of character-level convolutional networks for text classification. Their work shows comparison of character-level deep learning models against traditional models utilizing bag of words, n-grams and their TFIDF variants. Several different sizes of datasets are used. It is shown, that traditional methods like n-gram TFIDF remain strong candidates for datasets of size up to several hundreds of thousands samples, but are surpassed by CNNs on large scale datasets (over 500,000 samples). LSTMs seem to outperform the traditional methods on the large scale datasets as well, but are (by a small margin) worse than CNNs in all of the experiments conducted. Two character-level models (large and small) are constructed utilizing a lowercased alphabet of 70 characters.

Texts are padded with zeros or truncated to feature vectors of a fixed length. The model intended for large datasets consists of 6 convolutional layers of 1024 features each with consecutive kernel sizes of 7, 7, 3, 3, 3 and 3. Stride is set to one and max pooling of size 3 is applied in the first and last two layers. Output of the convolutional layers is connected to a sequence of two fully connected layers of 2048 units each. The fully connected layers are regularized by dropout.

The most important conclusion of the experiments conducted is that convolutional neural networks with character-level inputs are an effective method which works without the need of word-level input.

2.2.2.3 Very Deep Convolutional Networks for Text Classification

In this work inspired by the success of deep learning models in computer vision tasks, Conneau, Schwenk et al. [53] (2016), propose a deep learning approach for text classification. As the paper states, by increasing a depth of a model, better results can be obtained.

4 models of depth 9, 17, 29 and 49 layers are compared. Whole network is composed of a series of convolutional blocks as described in 3.2 with optional shortcuts between the blocks followed by K-MaxPooling (top-k most activated filters are preserved).

2.2.3 Deep Learning Methods for URL Classification

Usage of deep convolutional networks solely for the purpose of malicious URL detection was proposed by Saxe & Konstantin in the eXpose [47] (2017) paper. A year later, URLNet [9] (2018) used simultaneously both the character and word-level features with convolutional neural networks to achieve great results in the binary task of malicious URL detection.

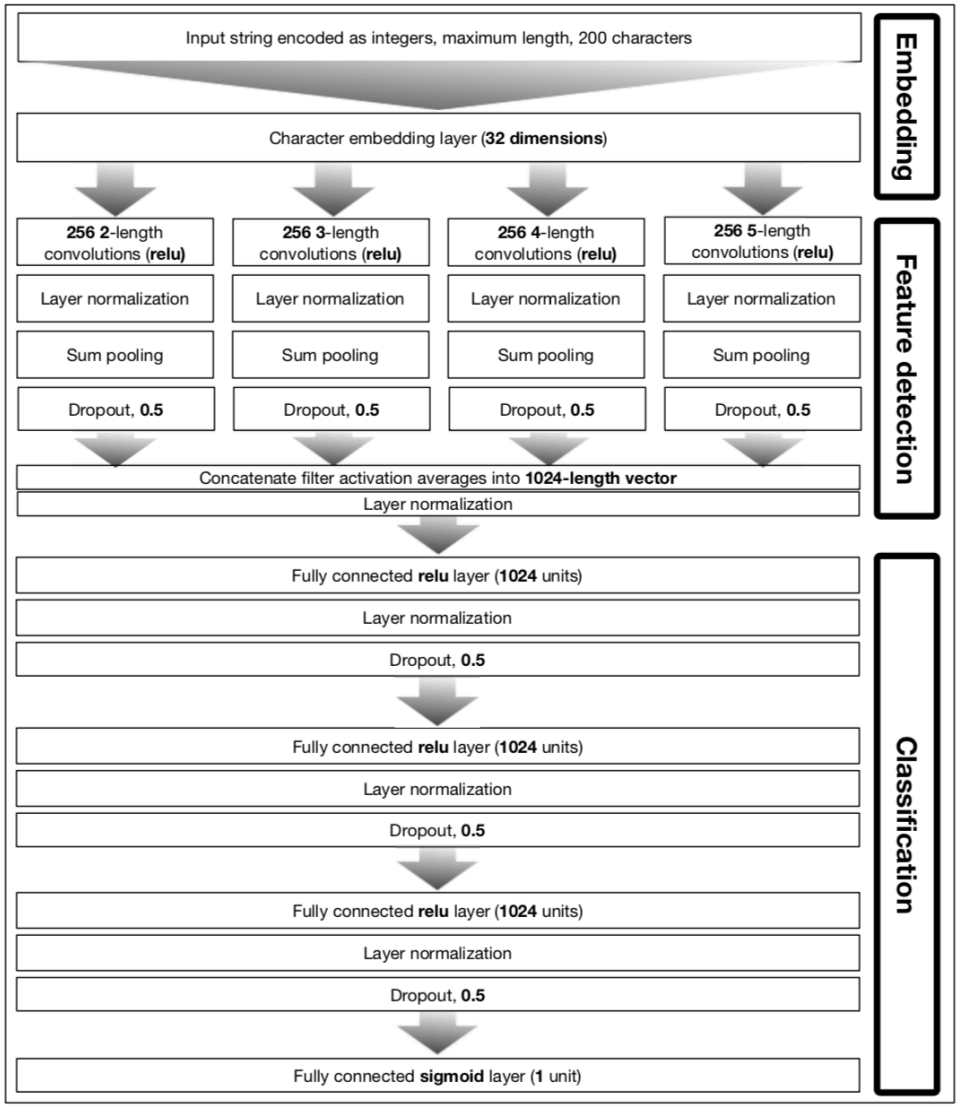

Figure 2.1: Structure of the eXpose neural network

2.2.3.1 eXpose

"While similar approaches have been suggested in NLP, eXpose is the first approach which demonstrates the usage of deep-learning method for cybersecurity problems." [47]

eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys [47] explores the idea of using the character-level convolutional neural networks to identify malicious URLs, file paths and registry keys. As it states, applying a neural network directly to the raw input of short character strings provides better classification accuracy than previous approaches that rely on hand-manufactured features.

Input character sequences are first padded with zeros or truncated to a fixed length of 200. A vocabulary of 87 URL-valid characters is used. Embedding vectors of size 32 are trained jointly with a convolutional network. To detect the features from learned embeddings, 4 convolutional blocks each consisting of 256 filters of sizes 3, 4, 5, 6 are used. Dropout of $p = 0.5$ and layer-wise batch normalization is used in the convolutional as well as in the fully connected layers. ReLu activation function is utilized. As opposed to other text classification papers [31, 9], eXpose utilizes the *global sum pooling* operation. All 4 convolutional blocks are concatenated to a joint output of size 1024 (4×256 filter outputs). The classification part of the network consists of 3 fully connected layers of size 1024 connected to a sigmoid output of size 1.

2.2.3.2 URLNet

URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection [9] aims to improve the detection of malicious URLs. Compared to eXpose, both word and character-level features are used simultaneously. The paper describes results of experiments on large-scale datasets and shows a performance gain over the existing methods.

The input URLs are at first preprocessed to remove the internet protocol. URL is then divided into primary domain, path, last path token, and top level domain. For each component, a bag of words is constructed. Several other features such as token bigrams in path are added along with character n-grams of the domain. These features help to detect URLs with slightly modified domain names. In addition, special features such as length of hostname and number of dots in the URL are used as well. Only the top-k appearing tokens are kept to keep the number of weights bearable. The preprocessed URLs are transformed to sequences of integer tokens and zero-padded or concatenated to a constant length of 200.

Token arrays are fed into an *embedding layer* to create 32-dimensional dense embeddings. 256 convolutional filters of lengths 3, 4, 5 and 6 are used to capture the order of the embedding layer inputs. *Global max pooling* is ran over all of the filters producing 1024 outputs in each branch connected to a FC layer of 512 units. Both character and word-level branches are then concatenated together and connected to fully connected layers of 1024, 512, 256 and 128 units each. Last layer consists of 1 neuron with a softmax activation function.

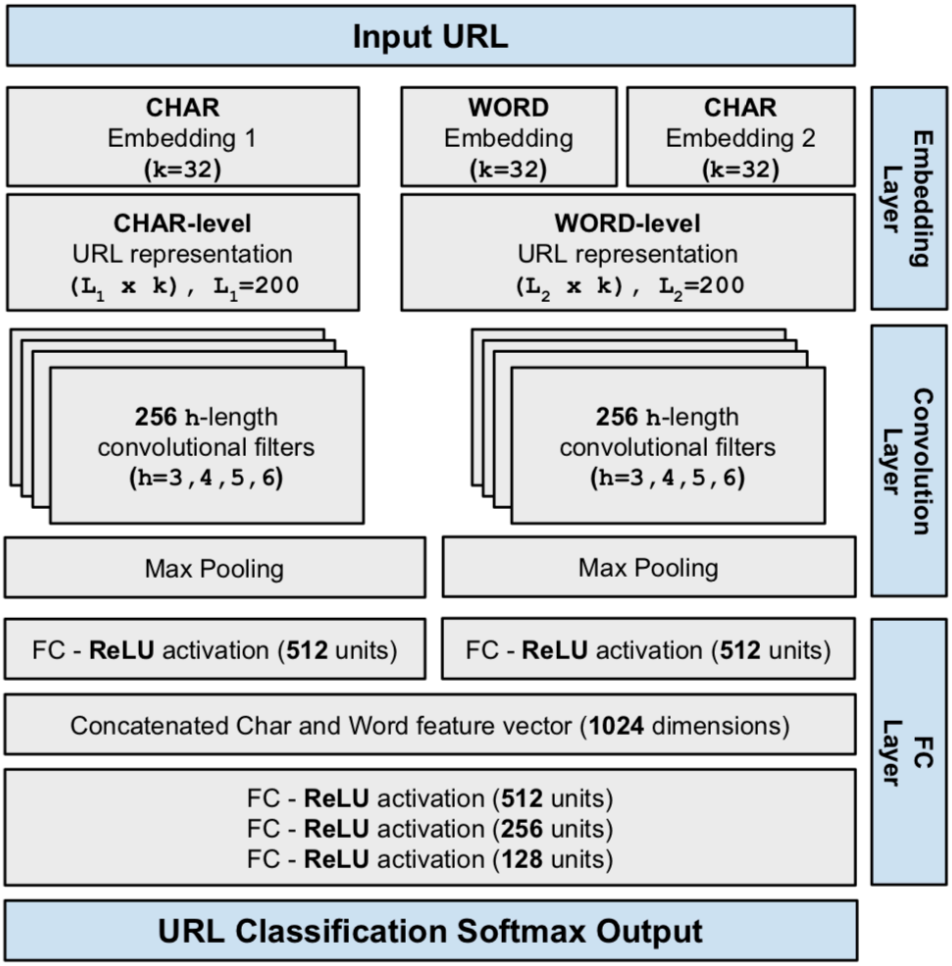

Figure 2.2: Structure of the URLNet neural network. URLNet is composed of 2 parts: character and a word-character level branches which are concatenated and fed into a series of fully connected layers.

2.2.4 Other Character-level Deep Learning Methods for URL Classification

The previously described paper *Character-level convolutional deep learning methods* also offers a comparison of convolutional methods to a vanilla recurrent long-short term memory (LSTM) model. This comparison is done on the pretrained word-level word2vec vectors. The results of LSTM model are worse than the CNN implementation in all of the test cases by a small margin. Usage of LSTM for URL classification is further researched by Bahnsen et al. in [39] and described below.

2.2.4.1 Classifying Phishing URLs Using Recurrent Neural Networks

Bahnsen et al. [39] (2017) explore the idea of using the recurrent neural networks, and more particularly a long-short term memory (LSTM) networks for binary classification of malicious URLs. The approach of using a recurrent neural network is compared to a random forest (RF) classification algorithm with manually created features. The LSTM network has shown over 5% higher accuracy than the feature-engineered model with RF. It has also shown, that the LSTM network improves its own performance faster than the RF as the number of training samples increase.

The LSTM and RF models were compared in terms of training time on a consumer grade PC as well. The results described in Table 2.3 show, that the LSTM network took around 100 times longer to train. Interestingly, the LSTM predictions done on GTX 860M GPU were slower only by a factor of 3 compared to the RF predictions done on a consumer grade processor. The RF model required almost 500 times the memory to store.

Table 2.3: Comparison of LSTM and RF methods

Method	Training Time minutes	Evaluation Time URLs per second	Memory consumption MB
RF	2.95 ± 0.11	942.12 ± 95.02	288.7
LSTM	238.7 ± 0.79	280.90 ± 64.48	0.581

As the authors state, the LSTM model showed an overall better prediction performance without the need of manual expert feature extraction needed in a RF model. However, the LSTM model lacks the interpretability of a random forest model. Also, the neural network models require far more training data, time and expertise to achieve results comparable to a traditional models such as RF. Figure 2.3 describes the structure of the classifier.

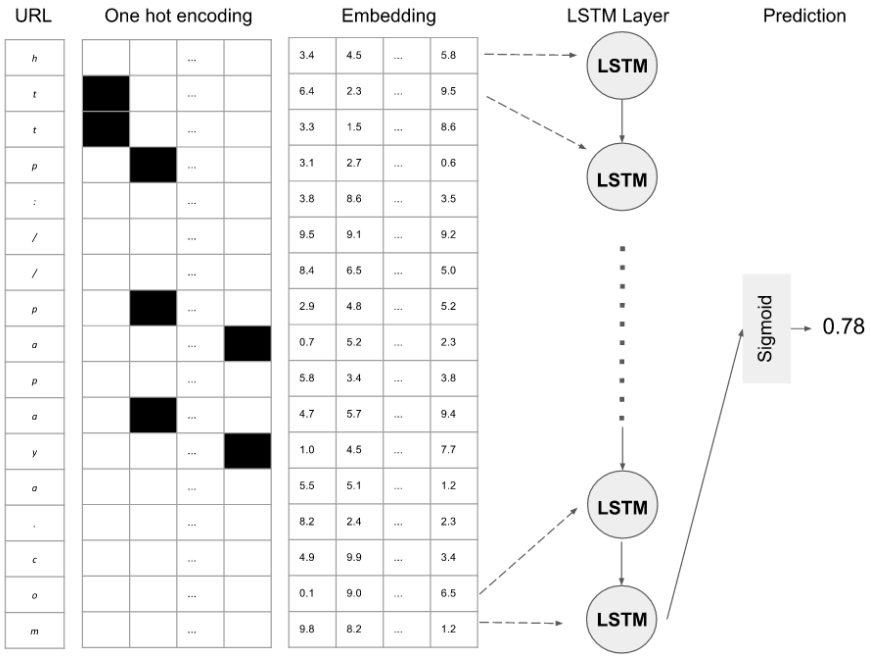

Figure 2.3: Structure of a LSTM classifier proposed by Bahnsen et al. One-hot input vectors are translated to a dense 128-dimensional embedding. Learned embeddings are fed to a LSTM layer as a 150-step sequence.

2.3 Discussion

Current trend in natural language processing seems to be in using very deep models. While the pioneering work by Kim Yoon [31] (2014) utilized only a shallow convolutional neural networks (just one convolutional layer), the papers on character-level text classification published more recently seem to be all benefitting from increased depth CNNs. In case of Zhang, X.; Zhao, J.; et al. [52] (2015), 6 convolutional layers were used, and in case of Conneau, Schwenk et al. [53] (2016) model of up to 49 consecutive layers was proposed.

Current research in deep URL classification only utilizes shallow convolutional neural networks [9, 47]. An alternative approach to the URL classification is to utilize recurrent neural networks. However the results do not seem to be in par with usage of CNNs.

The upcoming chapter compares different approaches and models on a real world dataset. The main question is, whether very deep models can outperform the shallow character-level models in the URL classification task. The primary goal is to obtain the model with best generalization ability, while also keeping in mind that to utilize the model in real-world production environment, the model is required to work even with limited computational resources.

Chapter 3

Experimental Model Comparison

Based on the prior analysis, numerous different architectures were developed and evaluated. A dummy classifier which generates predictions based on the distribution of the training set was evaluated as a simplest baseline.

All of the compared models based on CNNs were evaluated with URLs truncated or padded to a constant length of 320 tokens with the "post" strategy (described in 4.1) and all of them utilized same alphabet and dense embeddings (described in 1.5.3.2) of dimension 32.

Table 3.1: Validation scores of researched models. Random forest macro F1 score was ill-defined and therefore omitted in the table.

model	$F1_{macro}$	$F1_{micro}$	AUC_{macro}	AUC_{micro}
Dummy classifier	0.1287	0.3914	0.4999	0.6723
Logistic Regression	0.7685	0.8899	0.9752	0.9856
Naive Bayes	0.6172	0.8273	0.8809	0.9560
Random Forest	–	0.6210	0.9350	0.9352
CNN – 1 layer word	0.7754	0.8660	0.9573	0.9823
CNN – 1 layer character	**0.8382**	**0.9163**	0.9785	**0.9921**
CNN – 2 layer character	0.8191	0.9108	0.9782	0.9905
CNN – 3 layer character	0.7764	0.8985	0.9716	0.9896
CNN – VDCNN	0.7462	0.8781	0.9676	0.9849
CNN – URLNet	0.8153	0.9012	**0.9805**	0.9917
LSTM – character	0.7413	0.8799	0.9571	0.9820

Following section describes the structure of models from the table 3.1.

3.1 Traditional Machine Learning Models

The Logistic Regression, Naive Bayes, and Random Forest algorithms were all implemented using SKLearn [54] ML library on top of character 3-grams generated from the input URLs. The inputs were preprocessed by removing the protocol, domain and also lowercased. Logistic regression has proven to be the best, with very fast prediction speed and training. Random Forest and Naive Bayes models both did not perform well.

3.2 Models Based on Convolutional Neural Networks

Current state of the art among the NLP community is the usage of convolutional neural networks on top of learned dense embeddings. Several promising architectures varying in depth were implemented and compared.

3.2.1 Shallow (1 Layer) CNN Model

Shallow convolutional model is composed of a single convolutional layer followed by global max pooling and 2 fully connected layers. The tests included *character* and *word-level* inputs. As the best performing solution, this model is further described in the section 5.2.

3.2.2 Deeper (2 and 3 Layer) CNN models

One of the limitations of the single layer convolutional models like [9, 47] is that the filters learn sequences of tokens of length limited by the longest filter. Any information about relative positions of detected features is lost during the global max pooling operation. Solution to this problem can be to stack multiple convolutional and pooling layers effectively increasing the receptive field.

Deeper convolutional models used a network similar to the architecture utilized in 3.2.1. With the only difference that each convolutional branch was repeated 2 or 3 times with varying number of filters in between.

3.2.3 URLNet Inspired Model

Implementation of this model is based on the architecture of the URLNet (2.2.3.2). Character-level and Word-level feature vectors are created from each input URL and used simultaneously. The model takes character-level inputs of length of 320 characters and word-level inputs of length 160 words. The word tokens are generated by splitting the URL by special characters. Only words that appear more than once in the training dataset are preserved.

Both character and word-level branches of 512 filters are each concatenated to a separate fully connected layer of 512 units. The outputs of these branches are then concatenated together and fed into 2 fully connected layers of size 1024. Last fully connected layer of size 6 utilizes a SoftMax activation function.

3.2.4 VDCNN Inspired Model

A very deep model based on the architecture of the shallowest model (9 convolutional layers deep, proposed by [53]) was implemented and evaluated. The architecture of the model was previously explained in section 2.2.2.2.

Tested model consists of character-level embedding layer of dimension 32 followed by 4 ConvPool blocks. The ConvPool blocks utilizes (listed from input) 64, 128, 256 and 512 filters and is visualised in Table 3.2.

Table 3.2: Structure of a single ConvPool block

i) ii) iii)	Conv1D(kernel_size = 3, filters) BatchNormalization() Activation("ReLu")
i) ii) iii)	Conv1D(kernel_size = 3, filters) BatchNormalization() Activation("ReLu")
	MaxPooling1D(2)

Output of the stacked ConvPool blocks is finished with K-Max pooling layer and fed into 2 fully connected layers followed by a SoftMax layer of size 6.

3.3 Other Character-Level Models

An alternative approach to the character-level CNN models is to utilize recurrent neural networks. Particularly a long-short-term-memory network.

3.3.1 Vanilla LSTM

A simple "vanilla" LSTM network was implemented and tested on inputs of 120 characters. Outputs from the embedding layer of dimension 32 are fed into a 64 cell LSTM network. The LSTM is followed by 2 fully connected layers of 64 units each and a softmax layer of size 6.

Training the LSTM network was a challenging task, since the network failed to converge when long input sequences were presented. Experiments with utilizing longer input sequences and also increasing the number of LSTM units only led to decrease in performance. Also, the training of a single batch was around 4 times slower than in the case of shallow CNN models. The model converged very slowly during training.

3.4 Conclusion

Table 3.1 describes the validation performance of implemented models. Out of classical machine learning models, Logistic Regression was observed to be the best performing. Several different depths and architectures of convolutional models were implemented and compared. It seems, that the task of URL detection does not necessarily improve with deeper convolutional models.

Several different input features were used. While the classical models used character 3-grams, a shallow CNN model with word-level inputs was implemented and evaluated as well.

A simplified model inspired by the structure of URLNet has shown to be able to obtain the highest macro AUC score, however only by a very small margin compared to the shallow character-level model. Also, the URLNet model is twice the size, slower in training and prediction and most importantly requires a complicated extraction of word features prior to training. Slightly better performance can be explained by a larger receptive field due to the use of the word-level inputs.

A "vanilla" LSTM was observed to be effective on inputs of smaller length. It did not scale well with long sample lengths.

Shallow character-level convolutional neural networks seems to be the best choice for an URL classification task. Apart from tokenization and padding, they require no preprocessing, are stable during training and the inference speed is better than in case of larger models.

Chapter 4

Other Experiments

In order to understand the role of different hyper-parameters, several experiments were concluded. The results are presented below.

4.1 Input length

Although 1D convolution operation followed by global max pooling should support inputs of variable lengths, it seems that all of the current research [52, 47, 9] done in the area of text CNNs uses texts of constant size as inputs. This is done by truncating the string if it is longer than the maximum number of tokens, or padding it from one side with zeros to match the constant length.

This is most likely due to a fact that using very long input sequences proportionally slows down the training and is not necessary when comparing different approaches.

The experiment concluded tested how the shallow CNN model's performance differs on URLs of different lengths with different padding/truncate options:

- **pre** – pad before the sequence starts, remove values from sequence longer than the maximum length at the beginning
- **post** – pad after sequence ends, remove values from sequence longer than the maximum length at the end

The results are shown in the Figure 4.1. After using the lengths of input URLs longer than 300 characters, the type of padding seems to have no serious effect. This is likely due to the fact, that more than 95% of samples are less than 300 characters in length. More valuable conclusion of this experiment is that the information needed for the correct classification is contained towards the start of the URL.

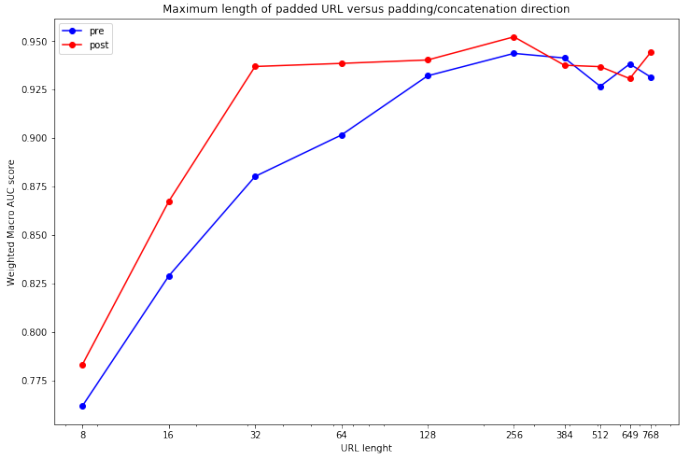

Figure 4.1: Comparison of padding settings.

4.2 Learned Embeddings

Convolutional filters are used on the top of dense learned embeddings as described in 1.5.3.2. To visualize the learned embeddings, a 32-dimensional embedding matrix was projected to 2D using t-SNE and to 3D using PCA. The <*p*> denotes a padding character and <*u*> the OOV (unknown) character. The projections indicate that the model learned that numbers have a similar semantic meaning, whereas special characters are far less similar. The results are visualized on the Figure 4.3. Different embedding dimensions were tested for the input of 120 characters. The results are shown in the Figure 4.2.

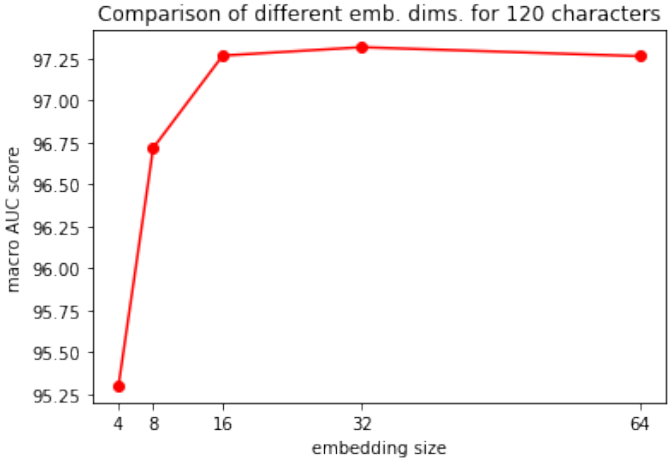

Figure 4.2: Comparison of different embedding dimensions for 120 characters

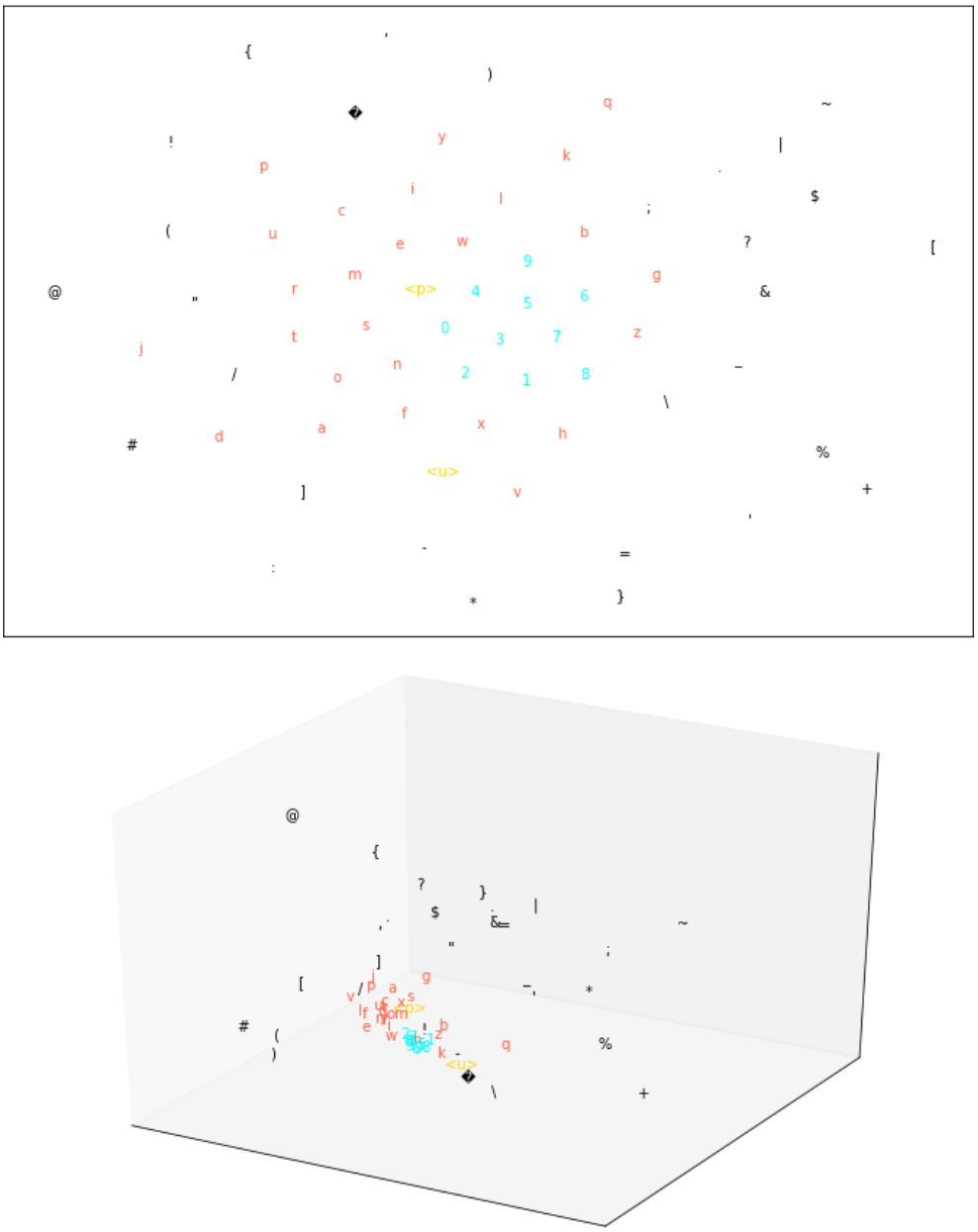

Figure 4.3: Projection of 32-dimensional embedding matrix to 2D using t-SNE (top) and to 3D using PCA (bottom). Numbers are visualized in blue, characters in red, special characters in black and special tokens yellow.

Chapter 5

Implementation

Goldberg [13] defines the general structure of a NLP classification system as:

1. "Extract a set of core linguistic features $f_1, \ldots, f_k$ that are relevant for predicting the output class.
2. "For each feature f_i of interest, retrieve the corresponding vector $v(f_i)$.
3. "Combine the vectors (either by concatenation, summation or a combination of both) into an input vector x.
4. "Feed x into a non-linear classifier (feed-forward neural network)."

The following chapter describes the best performing model selected as a final solution from the implemented and compared architectures in greater detail. The performance of the final model is also compared to the industrial baseline.

5.1 Library

For the implementation of the model itself, Keras[2] python deep learning library was used:

> "Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research."

[2] `https://keras.io/`

5.2 Model Architecture

Figure 5.1 shows the visualization of the selected model. First, the tokenized input is fed into a dense embedding layer of size 32. The embedding layer is randomly initialized and trained simultaneously with the rest of the model.

Convolutional filters of kernel sizes 3, 5, 7 and 9 each with 128 features are utilized. Global max pooling is applied to the output of each filter. Outputs of the convolutional branches are regularized by dropout.

The max pooled outputs are concatenated to a vector of size 512 and fed into 2 fully connected layers each of size 512. Both fully FC layers are regularized by batch normalization and dropout.

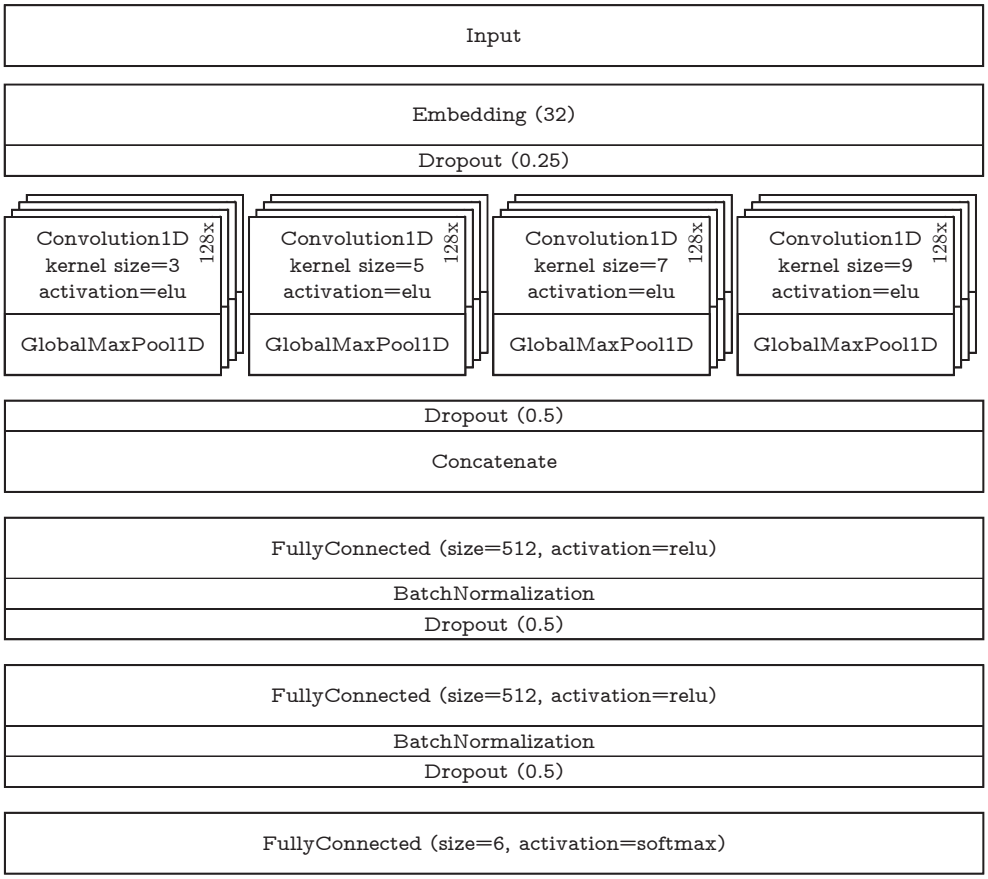

Figure 5.1: Architecture of the final shallow character-level CNN model

Unlike [9, 47] which were designed for a binary classification task, developed model is meant to predict one out of n classes on its output. The size of the last fully connected layer is therefore equal to the number of predicted classes. SoftMax activation is applied to predict a probability of sample belonging to class.

5.3 Training and Preprocessing

The model was trained using 4 nVidia Tesla P100 graphical processing units (GPU) using the Google Cloud[3] virtual machine. Using a cloud solution allows for quick development without the need of purchasing expensive hardware. Batch size of 4096 samples has shown to be the best compromise between the training speed and learning performance.

Since related work only proposes use of inputs of fixed length, one of the challenges to be solved during the development was the usage of variable length inputs. When training the model, all of the samples in the batch are required to be of same length. To solve this task, each batch is preprocessed separately. Batches are zero padded to the longest sample present in each batch. Keras Generator object allows for preprocessing to be done in parallel with the training or prediction which further improves the overall prediction speed. Padding utilizes the "post" strategy as described in 4.1.

Prior to prediction, input URLs are first sorted by length. After sorting, average length of padded sequence that needs to be processed by the model is smaller and the prediction is therefore faster. Whole process of batch preprocessing is visualized on the Figure 5.2.

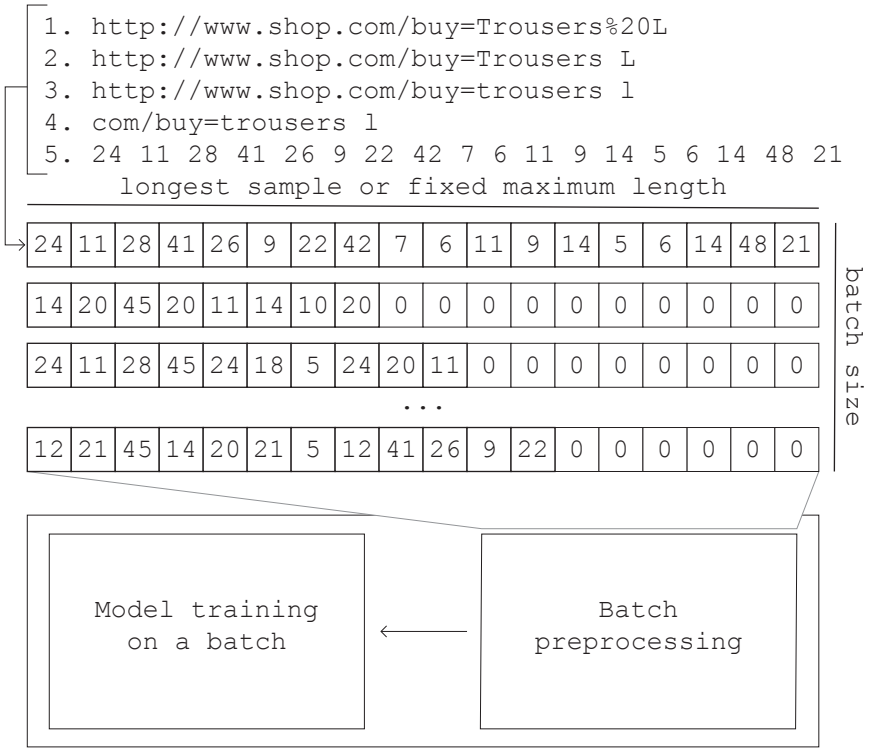

Figure 5.2: Example of preprocessing of a single batch

[3] https://cloud.google.com/

5.3.1 Tokenization

The URL is first decoded from the ASCII codes, lowercased and then stripped of the protocol and domain name. The top level domain is kept in order to provide information for the model which language might be used. Domain name can contain valuable information about the type of website, however, it does not directly correspond to the structure of URLs used for pages of different categories. Since the main goal is to create a model with the best generalization ability on any domain, the domain name is removed.

```
\1qwertzuiopasdfghjklyxcvbnm1234567890
\\-/=&_.?+ :,|%[]#\"' ;~\n!@$*(){}
```

Figure 5.3: Characters used as the model's input alphabet

If a character is not present in the given alphabet, it is replaced by a special OOV (out of vocabulary) character "\1". The vectorized inputs are padded with zeros ("\0" character) or truncated to a vector of fixed length of 320 tokens. Example of the preprocessing is shown in the table 5.1.

Table 5.1: Example of preprocessing and tokenization of an URL: (i) URL is decoded, (ii) lowercased, (iii) protocol and domain name are removed, (iv) the URL characters are tokenized

	https://alza.cz/CartAdd.htm?id=170&pkitem=544&iname=nVidia%20V100
i.	https://alza.cz/CartAdd.htm?id=170&pkitem=544&iname=nVidia V100
ii.	https://alza.cz/cartadd.htm?id=170&pkitem=544&iname=nvidia v100
iii.	cz/cartadd.htm?id=170841881&pkitem=544&iname=nvidia v100
iv.	23, 7, 40, 23, 12, 5, 6, 12, 14, 14, 44, 17, 6, 27, 45, 9, 14, 41, 28, 34, 37, 35,...

5.3.2 Optimizer

RMSprop optimizer has shown to obtain high validation scores in fewer epochs compared to the Adam optimizer, which usually took longer but the training was more stable and converged to slightly better results.

5.3.3 Loss Function

Since ROC AUC score is not differentiable, it cannot optimized directly as a loss function. A common approach is to use a proxy-objective, such as categorical cross entropy, but it does not necessarily improve the ROC AUC score. An alternative differentiable approximation based on Wilcoxon-Mann-Whitney statistic [55] implemented in the tflearn[4] library was used.

[4] `https://github.com/tflearn/tflearn/`

5.4 Evaluation Metrics

The selection of the right metric was one of the most challenging tasks during the development of the model. A simple accuracy score does not provide a good metric on unbalanced datasets. Instead, 4 scores were recorded: macro F1, micro F1, macro AUC and micro AUC. Primary goal was optimizing the macro AUC metric. All the scores were weighted by the url_count (described in 2.1) to take in mind that duplicate URLs were removed prior to training.

5.4.1 Monitoring Training Performance

Custom callback function was developed to both automatically monitor the metrics and save the results during the experiments for further examination. The dashboard used to monitor the training performance is shown on the Figure 5.4.

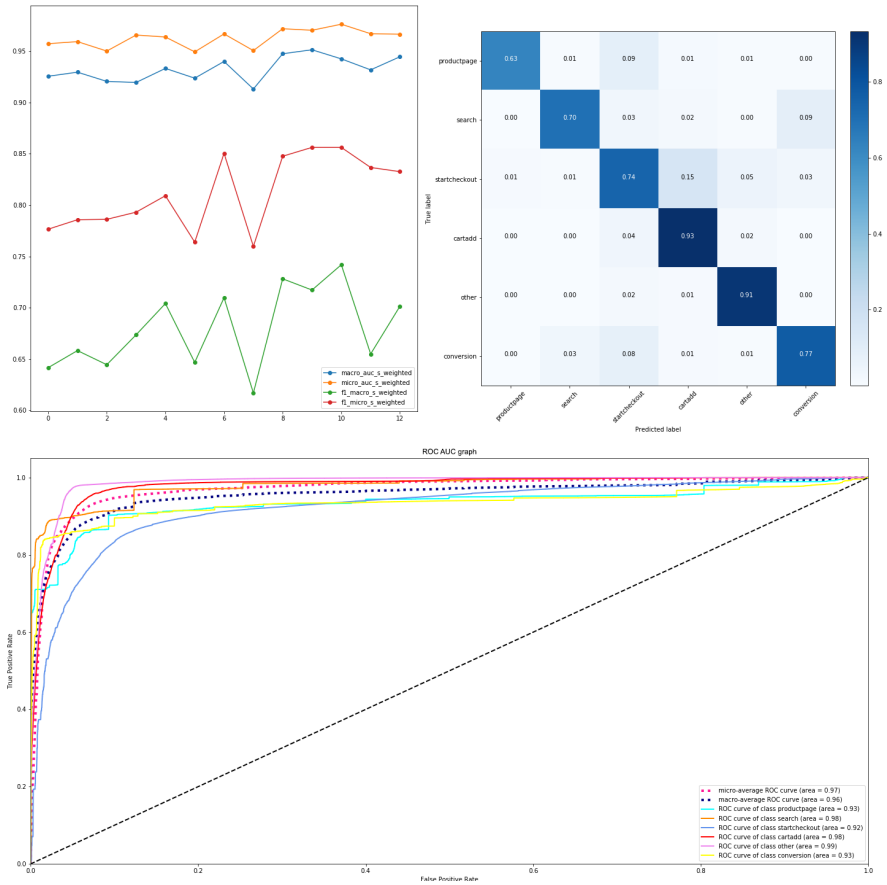

Figure 5.4: Overview of the training dashboard. Validation score is shown on the top left, an average of confusion matrices calculated separately for each class on the top right, ROC graph on the bottom.

5.5 Model Evaluation

Before the training and development started, a holdout "test" dataset was put away. A purpose of this dataset was to show a real, unbiased performance gain over the existing baseline model which is currently utilized in production.

Current production "autodetection" system uses a logistic regression classifier built on top of character trigrams using VowpalWabbit[5] library. VowpalWabbit (VW) is a machine learning library developed by Microsoft. It was originally selected for its emphasis to inference and training speed. Logistic regression is one of the most popular approaches used in text classification. It is fast to train yet effective. The major limitation is that it is unable to easily detect dependencies of characters longer than the length of the n-gram.

The final model trained on the inputs of a fixed length of 320 characters is compared to the baseline, and also to a model of the same architecture, but retrained and used with inputs of arbitrary length.

Table 5.2: Test score comparison of a final model

input length	$F1_{macro}$	$F1_{micro}$	AUC_{macro}	AUC_{micro}
VW baseline	0.7889	0.8888	0.9695	0.9820
CNN fixed length	0.8438	**0.9317**	0.9796	**0.9929**
CNN arbitrary length	**0.8451**	0.9257	**0.9803**	0.9924

It is clear, that the developed model based on character-level CNN model outperforms the logistic regression solution by a considerable margin. The usage of variable length inputs is debatable.

[5] `https://github.com/VowpalWabbit/vowpal_wabbit/`

Conclusion

The theoretical part of this thesis sums up most of the elementary knowledge needed to efficiently understand the concepts used in deep learning. Special attention is put to the description of deep character-level convolutional networks for natural language processing.

One of the goals of this thesis was to research current approaches used for the URL classification task. This has traditionally been done using classical machine learning models on top of character n-grams or word-level features. However, these are outperformed by modern classification systems inspired by natural language processing. Current state-of-the-art approaches such as eXpose or URLNet utilize convolutional neural networks on top learned dense embeddings.

Based on the survey of the related work, several different architectures varying in depth, structure and different input features were implemented and evaluated on a real life dataset. Experimental results confirm, that the use of character-level convolutional networks is a viable alternative to the classical machine learning algorithms.

Simple convolutional networks with character-level inputs were observed to achieve great results even when compared to models like URLNet which utilize both character and word-level features. This indicates, that the word-level input is not necessary for effective URL classification as the character-level convolutional filters are able to learn the important features themself.

A LSTM recurrent neural network was implemented and tested as an alternative to the convolutional neural networks. Compared to the CNN-based models, the training was very difficult and results inferior. Concluded experiments suggest, that the the performance of the URL classification system does not necessarily improve with increased the depth of the network. Instead, the classifier benefits from powerful feature extraction of a shallower CNN models.

Best performing model was described in depth in the implementation chapter. Since this model outperformed the baseline solution, it is currently getting deployed in a real production environment. The model will be trained on dataset several magnitudes larger than utilized in this thesis. Since the neural networks are notoriously known to scale well when enough training data is available, final model is expected to perform even better.

Preprocessing

When dealing with numerical features, especially with inputs of different scale or range, it can be beneficial to standardize or normalize the inputs. However, these techniques do not apply to models dealing with sequences of characters. A common preprocessing practice in NLP, lemmatization, can be used to reduce each word to its basic form. This leads to lower dimension of the input vocabulary.

Since URLs are often sequences of arbitrary characters, it does not make a great sense to utilize lemmatization. One of the features of character-level neural networks is that they do not require any prior preprocessing. Except from removing the protocol and domain, lowercasing of the input was the only preprocessing step applied to the URLs. No improvements were observed with uppercase letters preserved.

Challenges Encountered

Although not described as a part of this thesis, the first task that had to be solved was the creation of the dataset itself. Working with Big Data can be a time consuming and error prone task. Since accuracy is not a representative score when working with imbalanced datasets, a challenge, not obvious at first, was to select a representative metric. Also, a custom training dashboard was developed in order to monitor the training progress.

Since the train, test and validation datasets were split per domain, a large dataset had to be utilized in order to provide representative results. To train the models in an effective manner, a virtual machine with several modern GPUs was used in the cloud environment. Due to the size of the dataset and memory requirements the models could not be easily trained on a consumer grade workstation.

A lot of effort was put to train the neural network with inputs of varying sizes. All of the approaches in related work utilized inputs of fixed length. The solution to this task was to implement a custom batch generator and pad each batch to the length of the longest sample. By sorting the dataset prior to prediction, faster predictions can be achieved thanks to the fact, that the most of the URLs are relatively short.

Benefits and Limitations of Character-Level Models

When training classical machine learning models with textual inputs, the texts need to be first tokenized using a bag-of-words model or its TF-IDF variant. To determine the size of the input feature space, all of the training data usually needs to be preprocessed prior to training. When working with large datasets, the number of features can become extremely high. This preprocessing can be time and memory consuming too. This issue can be tackled by using the so-called hashing trick, however it usually comes with a decline in performance.

Main feature of the character-level neural networks is a small-to-none need for text preprocessing. There is no need to tokenize the dataset prior to training since we are dealing only with sequences of individual characters and fixed alphabet.

A common limitation to all neural network models is the limited explainability of the inference mechanism. Development of neural networks can be demanding, as the hyperparameter space is extremely high and finding an ideal solution is often a long haul.

Effective training of neural networks is generally a time consuming task and is highly dependent on fast GPUs. This is not a common equipment of many of the companies since there is usually no usage for GPUs other than for machine learning.

Proposed Improvements

Keras library currently does not support zero masking[6] of zero-padded sequences in its convolutional layers. The model therefore learns embeddings even for the padding characters. This is not desirable as this noise negatively affects the performance of the model. This is most likely the main limiting factor of the model with variable input lengths.

The proposed models could possibly be improved by using an embedding such as Word2Vec [15], GloVe [16] or fastText [56]. Since these methods are unsupervised, there is no shortage for training data.

[6] `https://github.com/keras-team/keras/issues/411`

Bibliography

[1] Sahoo, D.; Liu, C.; et al. Malicious URL detection using machine learning: a survey. *arXiv preprint arXiv:1701.07179*, 2017.

[2] Kan, M.-Y.; Thi, H. O. N. Fast webpage classification using URL features. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, 2005, pp. 325–326.

[3] Shih, L. K.; Karger, D. R. Using urls and table layout for web classification tasks. In *Proceedings of the 13th international conference on World Wide Web*, ACM, 2004, pp. 193–202.

[4] Francois, C. *Deep learning with Python*. Manning Publications Company, 2017.

[5] Steinberg, R. 6 areas where artificial neural networks outperform humans. [online], accessed: 2019-04-08. Available from: `https://venturebeat.com/2017/12/08/6-areas-where-artificial-neural-networks-outperform-humans/`

[6] Savov, V. The OpenAI Dota 2 bots just defeated a team of former pros. [online], accessed: 2019-04-08. Available from: `https://www.theverge.com/2018/8/6/17655086/dota2-openai-bots-professional-gaming-ai`

[7] Russell, J. Google's AlphaGo AI wins three-match series against the world's best Go player. [online], accessed: 2019-04-17. Available from: `https://techcrunch.com/2017/05/24/alphago-beats-planets-best-human-go-player-ke-jie/amp/`

[8] Simeone, O. A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, volume 4, no. 4, 2018: pp. 648–664.

[9] Le, H.; Pham, Q.; et al. URLnet: Learning a URL representation with deep learning for malicious URL detection. *arXiv preprint arXiv:1802.03162*, 2018.

[10] Bird, S.; Klein, E.; et al. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[11] Goodfellow, I.; Bengio, Y.; et al. *Deep learning*. MIT press, 2016.

[12] Khurana, D.; Koli, A.; et al. Natural language processing: State of the art, current trends and challenges. *arXiv preprint arXiv:1708.05148*, 2017.

[13] Goldberg, Y. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, volume 57, 2016: pp. 345–420.

[14] Martin, J. H.; Jurafsky, D. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall Upper Saddle River, 2009.

[15] Mikolov, T.; Chen, K.; et al. Efficient Estimation of Word Representations in Vector Space. 2013, `1301.3781`.

[16] Pennington, J.; Socher, R.; et al. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[17] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, volume 65, no. 6, 1958: p. 386.

[18] Raschka, S. Single-Layer Neural Networks and Gradient Descent. [online], accessed: 2019-2-28. Available from: `https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html`

[19] Karel Klouda, Juan Pablo Maldonado Lopez, Daniel Vašata. Lecture notes in Data Mining. Accessed: 2019-2-28.

[20] Rumelhart, D. E.; Hinton, G. E.; et al. Learning representations by back-propagating errors. *Cognitive modeling*, volume 5, no. 3, 1988: p. 1.

[21] Hayou, S.; Doucet, A.; et al. On the Impact of the Activation Function on Deep Neural Networks Training. *arXiv preprint arXiv:1902.06853*, 2019.

[22] Ramachandran, P.; Zoph, B.; et al. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[23] Clevert, D.-A.; Unterthiner, T.; et al. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[24] Kukačka, J.; Golkov, V.; et al. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[25] Hinton, G. E.; Srivastava, N.; et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[26] Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[27] Collis, J. Glossary of Deep Learning: Batch Normalisation. [online], accessed: 2019-04-24. Available from: `https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82`

[28] Doukkali, F. Batch normalization in Neural Networks. [online], accessed: 2019-04-24. Available from: `https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c`

[29] O'Shea, K.; Nash, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[30] Springenberg, J. T.; Dosovitskiy, A.; et al. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[31] Kim, Y. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[32] Jacovi, A.; Shalom, O. S.; et al. Understanding Convolutional Neural Networks for Text Classification. *arXiv preprint arXiv:1809.08037*, 2018.

[33] Olah, C. Understanding LSTM Networks. [online], 2015, accessed: 2019-04-24. Available from: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

[34] Ketkar, N.; et al. *Deep Learning with Python*. Springer, 2017.

[35] Patterson, J.; Gibson, A. *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

[36] Gers, F. A.; Schmidhuber, J.; et al. Learning to forget: Continual prediction with LSTM. 1999.

[37] Huang, Z.; Zweig, G.; et al. Accelerating recurrent neural network training via two stage classes and parallelization. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, IEEE, 2013, pp. 326–331.

[38] Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation*, volume 9, no. 8, 1997: pp. 1735–1780.

[39] Bahnsen, A. C.; Bohorquez, E. C.; et al. Classifying phishing URLs using recurrent neural networks. In *2017 APWG Symposium on Electronic Crime Research (eCrime)*, IEEE, 2017, pp. 1–8.

[40] Yan, S. Understanding LSTM and its diagrams. [online], 2016, accessed: 2019-04-24. Available from: `https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714/`

[41] Hugo Ferreira's blog - Confusion matrix and other metrics in machine learning. [online], accessed: 2019-03-26. Available from: `https://medium.com/hugo-ferreiras-blog/confusion-matrix-and-other-metrics-in-machine-learning-894688cb1c0a`

[42] SKLearn documentation - sklearn metrics fbeta-score. [online], accessed: 2019-03-24. Available from: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html`

[43] Özgür, A.; Özgür, L.; et al. Text categorization with class-based and corpus-based keyword selection. In *International Symposium on Computer and Information Sciences*, Springer, 2005, pp. 606–615.

[44] Yang, Y.; Liu, X.; et al. A re-examination of text categorization methods. In *Sigir*, volume 99, 1999, p. 99.

[45] Fawcett, T. An introduction to ROC analysis. *Pattern recognition letters*, volume 27, no. 8, 2006: pp. 861–874.

[46] Monkey Learn Inc. Text Classification - A Comprehensive Guide to Classifying Text with Machine Learning. [online], accessed: 2019-04-01. Available from: `https://monkeylearn.com/text-classification/`

[47] Saxe, J.; Berlin, K. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv preprint arXiv:1702.08568*, 2017.

[48] Eriksson, T. *Automatic web page categorization using text classification methods*. Master's thesis, CSC School of Computer Science and Communication, Stockholm, Sweden, 9 2013.

[49] Ma, J.; Saul, L. K.; et al. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2009, pp. 1245–1254.

[50] Hou, Y.-T.; Chang, Y.; et al. Malicious web content detection by machine learning. *Expert Systems with Applications*, volume 37, no. 1, 2010: pp. 55–60.

[51] Wang, D.; Navathe, S. B.; et al. Click traffic analysis of short url spam on twitter. In *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, IEEE, 2013, pp. 250–259.

[52] Zhang, X.; Zhao, J.; et al. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 2015, pp. 649–657.

[53] Conneau, A.; Schwenk, H.; et al. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[54] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[55] Yan, L.; Dodier, R. H.; et al. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 848–855.

[56] Bojanowski, P.; Grave, E.; et al. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, volume 5, 2017: pp. 135–146, ISSN 2307-387X.

Appendix A

Acronyms

URL	Unique resource locator
ML	Machine learning
NLP	Natural language processing
BOW	Bag of words
OOV	Out of vocabulary (token)
MLP	Multi layer perceptron
NN	Neural network
CNN	Convolutional neural network
FC	Fully connected (layer)
TanH	Hyperbolic tangent
ReLu	Rectified linear unit
ELU	Exponential linear unit
RNN	Recurrent neural network
LSTM	Long-short term memory
RF	Random forest
ROC	Receiver operating characteristics
AUC	Area under curve

Appendix B

Contents of Enclosed CD

- `readme.txt` ..... file with CD contents description
- `src` ..... directory containing the source code
  - `requirements.txt` ..... file containing python dependencies
  - `jupyter` ..... directory containing Jupyter Notebooks
  - `python` ..... directory containing python functions and classes
  - `thesis` ..... directory of LaTeX source codes of the thesis
- `text` ..... thesis text directory
  - `thesis.pdf` ..... thesis text in PDF format