CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Nuclear Sciences and Physical Engineering

Tool for big data statistical analysis

Nástroj pro statistické zkoumání velkých toků dat

Bachelor's Degree Project

Author:	**Aleksandra Vecherskaya**
Supervisor:	**doc. Ing. Miroslav Virius, CSc.**
Consultant:	**Bc. Ondřej Surý**
Language advisor:	**PaedDr. Eliška Rafajová**

Academic year: 2015/2016

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky Akademický rok: 2015/2016

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Posluchač:	Aleksandra Vecherskaya
Obor:	Aplikovaná informatika
Název práce:	Nástroj pro statistické zkoumání velkých toků dat / Tool for big data statistical analysis

Osnova:

1. Seznamte se s nástroji pro analýzu velkých toků dat.
2. Seznamte se s problematikou datových toků na příkladu DNS serverů.
3. Na základě získaných informací formulujte požadavky na nástroj pro automatickou streamingovou analýzu dat DNS serverů.
4. Požadavky analyzujte, navrhňete a implementejte aplikaci.
5. Vytvořte k ní uživatelskou příručku.

Doporučená literatura:

1. Josh Wills, Sandy Ryza, Sean Owen, and Uri Laserson: Advanced Analytics with Spark. O'Railey Media 2015.
2. Guojun Gan, Chaoqun Ma: Data Clustering: Theory, Algorithms, and Applications. SIAM 2007.
3. Peter Flach: Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press 2012.
4. Nick Pentreath: Machine Learning with Spark. Packt Publishing 2015.
5. Gary Miner, Robert Lisbet, John Elder IV: Handbook of Statistical Analysis and Data Mining Applications. Elsevier Inc. 2009.

Vedoucí bakalářské práce:	doc. Ing. Miroslav Virius, Csc.
Adresa pracoviště:	KSI FJFI ČVUT, Trojanova 13, Praha 2
Konzultant:	Bc. Ondřej Surý

Datum zadání bakalářské práce:	31.10.2015
Datum odevzdání bakalářské práce:	4.7.2016

V Praze dne 11. listopadu 2015

[illegible]

vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

[illegible]

děkan

Acknowledgment:

I would like to thank doc. Ing. Miroslav Virius, CSc. for his expert guidance and express my gratitude to PaedDr. Eliška Rafajová for her language assistance.

Author's declaration:

I declare that this Bachelor's Degree Project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, July 7, 2016 Aleksandra Vecherskaya

Název práce:

Nástroj pro statistické zkoumání velkých toků dat

Autor: Aleksandra Vecherskaya

Obor: Aplikovaná informatika

Druh práce: Bakalářská práce

Vedoucí práce: doc. Ing. Miroslav Virius, CSc., KSI FJFI ČVUT

Konzultant: Bc. Ondřej Surý, CZ.NIC z.s.p.o.

Abstrakt: Tato bakalářská práce představuje nástroj pro částečně řízenou analýzu DNS dat ze sít'ového provozu. Navrhovaná metoda zkoumá data na úrovni jednotlivých uživatelů a skládá se ze dvou částí: přípravy dat a modelování dat. Pro přípravu dat se zavádí nový způsob měření aktivity uživatelů, to jest normalizovaná entropie. Chování uživatelů v síti je reprezentováno časovou řadou hodnot entropie. Zmíněné časové řady jsou rozdělené do clusterů s využitím DTW jako měřítka podobnosti za účelem získání labelů pro řízené učení neuronové sítě. Modelovací část obsahuje obousměrnou LSTM neuronovou sít', která je trénovaná na behaviorálních řadách. Trénovaná neuronová sít' je schopná rozpoznávat vzorce aktivity uživatelů v provozu v sítě ve skutečném čase.

Klíčová slova: BLSTM, DNS, klasifikace sít'ových dat, neuronová sít'

Title:

Tool for big data statistical analysis

Author: Aleksandra Vecherskaya

Abstract: This thesis presents a tool for semi-supervised statistical analysis of the DNS network traffic data. The proposed method examines the traffic on the host level and consists of the data preparation part and the modelling part. For the data preparation we introduce a new way of measuring the host activity, i.e. normalised entropy. Host behaviour in the network is represented as temporal sequence of entropy values. The temporal sequences of DNS packets are clustered with the DTW as a similarity measure in order to obtain class labels for a supervised training of the neural network. The modelling part consists of the bidirectional LSTM neural network which is trained on behavioural sequences. After the training, the neural network is able to recognise patterns of the host activity in real-time mode.

Key words: BLSTM, DNS, network traffic classification, neural network

Contents

Introduction

This thesis is aimed to provide a software tool for statistical analysis of the DNS network traffic which falls within the big data category. First of all, big data itself is a challenging point because large amount of data is hard to analyse and even harder to model. The purpose of the analysis is to gain information about the nature of data. There is a lot of ambiguity in big data which is caused by the presence of noise and by its scalability. Noise distorts the true distribution model of data and the scalability means that datasets tend to contain more than one subpopulation which distribution models may distort each other. Thus, when analysing such data, the first step is to define the subpopulations which will be modelled and then to find the measure which suits this exact model. In case of DNS network traffic, there are multiple distinguishable classes of host activity which could be measured by the diversity.

Secondly, crucial part of any data mining tool is data preprocessing. The network traffic data are sequences of DNS packets. DNS packet itself is, in simple terms, a set of categorical features with large domains. This research uses normalised entropy over the sliding window to prepare raw data for further examination. Further, agglomerative clustering with the DTW as a similarity measure is used to prepare data for modelling.

The critical point of the modelling part is to define the type of distribution because for each distribution model there is an appropriate modelling tool. Different papers report different distributions of the network traffic [1], [4]. Moreover, the tool developed in this thesis is intended to be scalable with respect to different distributions. In order to satisfy such restrictions, a nonparametric method is used for building a model, namely bidirectional LSTM neural network. Input data are sequences of entropy values and class labels are clusters found during the clustering. The training method used is Adadelta as it enables fast tuning of neural network parameters. The trained model is then applied to a test set for obtaining a classification. For a given host it allows to determine its type of behaviour.

The structure of the thesis is as follows. The first chapter gives the overview of the problem and briefly discusses existing solutions. Second and third chapters are devoted to detailed explanation of the neural networks basic components and give the overview of existing neural network architectures. The BLSTM neural network used for modelling purposes in this thesis and the training algorithm are presented in the fourth chapter. The fifth chapter clarifies techniques used in the preprocessing module of the developed system. The main purpose of the sixth chapter is to describe the structure of modules in general. Besides that, it guides the reader through the process of data preparation and modelling. The seventh chapter contains a user guide. It gives detailed information on available methods and explains how to use them. The last chapter demonstrates the results of testing the developed tool on the real-life data.

Chapter 1

State of The Art

Conventional classification techniques such as port-based and Deep Packet Inspection (DPI) proved their unreliability. Accuracy of port-based methods is low beacuse nowadays the substantial part of applications use nonstandard or even random ports. The DPI related approaches are still in use, however recent advances in packet encryption and protocol obfuscation significantly narrowed the area of their application [18]. What is more, it requires enormous computational capacity to examine every packet in the network traffic flow. As a result, the focus shifted to the flow-level and behavioural analysis with the help of Machine Learning (ML) algorithms [18]. ML algorithms are able to overcome the aforementioned restrictions. For instance, they are lightweight in sense of computational complexity as they are intended to deal with so called big data.

The purpose of this thesis is to find a semi-supervised approach of network traffic behavioural classification and reidentification with the help of ML methods. This is indended to be a basis for an Intrusion Detection System (IDS). In terms of classification, behaviour is network traffic produced by a host. Such point of view is proven to be effective in identification different types of host activity. For example, Auld et al. [5] use the Bayesian Neural Network (BNN) as a classifier. They use manually classified semantically complete TCP connections as training data. Every TCP flow is represented by a set of features and a class label. They use pairs of hosts as a grouping criterion. Trained in such manner, BNN reached the 95%-99% accuracy in recognition TCP flows. In this thesis, the host address is used as a grouping criterion as well: raw packet sequence gathered during the day is divided into smaller activity intervals according to the host which generated it.

Al-Jarrah et al. in [3] apply Recurrent Neural Networks (RNN) to the classification task in a different way. The RNN works as a preprocessing unit where the output layer is represented by the Principal Component Analysis (PCA) unit. PCA checks the correlation of features in the input and reduces it to a set of uncorrelated ones. The system of Al-Jarrah et al. recognises behavioural patterns based on the number of uncorrelated features. Their system is able to distinguish between Port Scan and Host Sweep with 100% accuracy. However, the system was trained on a relatively small amount of training data (7 weeks) and does not give any insight on the major drawback of RNN which is a poor training performance on long sequences. Moreover, the proposed system was tested on DARPA Intrusion Detection Evaluation dataset which does not guarantee it will have the same accuracy rate when applied to contemporary attacks.

The research [15] by Gu et al. introduces the application of entropy in the area of traffic modelling and classification. They utilise the protocol information and destination port numbers for grouping raw packets. They first model the baseline entropy density of the observed network traffic trace using the Maximum Entropy estimation method and then track changes in the relative entropy of each packet group. As a result, even slight changes in the density model can be detected and classified. However, the proposed method suffers from the large amount of false positives because it detects any change in the density. It also does not provide any information about detected attacks and thus further examination is impossible.

This thesis is an attempt to develop an approach capable of classifying network traffic with the least possible amount of false positives. Detection of change in the model used by Gu et al. is not as sufficient as pattern matching proposed in this research. Recurrent neural networks proved to be a perfect tool for learning the sequential patterns. However, vanilla RNN used by Al-Jarrah in [3] are not effective at learning temporal behaviour of the network traffic dynamics. The extended version of Long Short-term Memory Neural Network (LSTM) capable of modelling complex temporal correlations is used in this thesis.

The purpose of IDS is not only to detect anomalous behaviour but also to track its origin. That is the reason behind dividing the dataset into groups according to hosts. The proposed method utilises the measure of entropy for summarising the traffic, but in a different manner than in [15]. The normalised entropy is calculated over the sliding window in order to express the feature distribution. In addition, during the preparation of training data the detailed patterns of network traffic dynamics are obtained. This is beneficial for further research in terms of IDS development.

Chapter 2

Basic Concepts of Artificial Neural Networks

Neural network is a powerful tool for solving a wide variety of problems in the fields of data analysis ranging from classification to prediction. Similarly to the human brain neural networks show remarkable effectiveness in mining knowledge in an unsupervised manner. The recent advances are especially noteworthy. For instance, the chess engine Giraffe [19] is able to learn how to play chess without any hard-coded rules provided or the deep neural network developed by DeepMind Technologies [20] is capable of evaluating Atari games rules from bare pixel positions. On the other hand, the mathematical apparatus of neural networks is clear and straightforward. This chapter gives the overview of the introductory concepts needed for building any neural network.

2.1 Neural network

Neural network is represented by a directed graph of units called neurons that are fully pairwise connected. The basic neural network, namely feedforward, is acyclic, while its extension, recurrent neural network, require cycles for sharing information between temporal states. The visualisations are presented in figure 2.1.

Networks are distinguished by three characteristics: interconnection pattern between layers, neuron activation function used for squashing information and the training algorithm used for learning feature representation. Configuration of connections between neurons depends on a problem being solved, while two other points will be briefly discussed in further sections.

2.2 Neuron

The concept of artificial neural network is clearly inspired by a human brain. The unit of a human brain is a neuron. It receives the input signal from the bunch of its dendrites, processes it inside the body and passes it further through the branching axon. The artificial neuron operates the information in the same manner. The graphical representation could be found in figure 2.2.

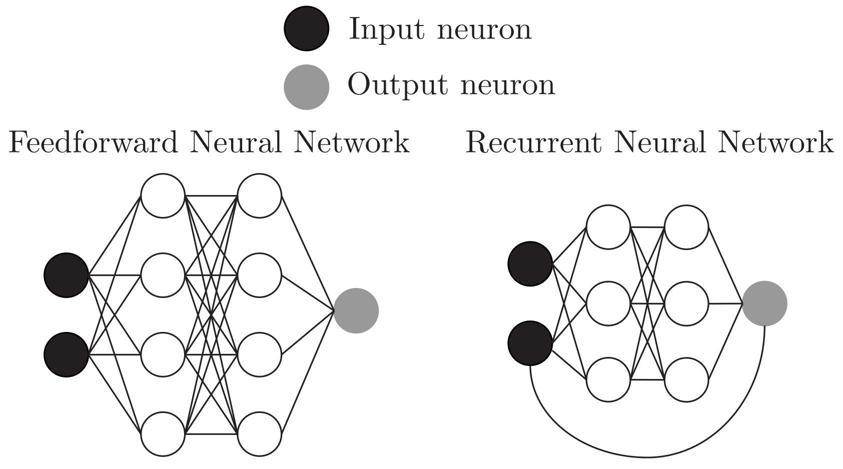

Figure 2.1: Types of Neural Networks

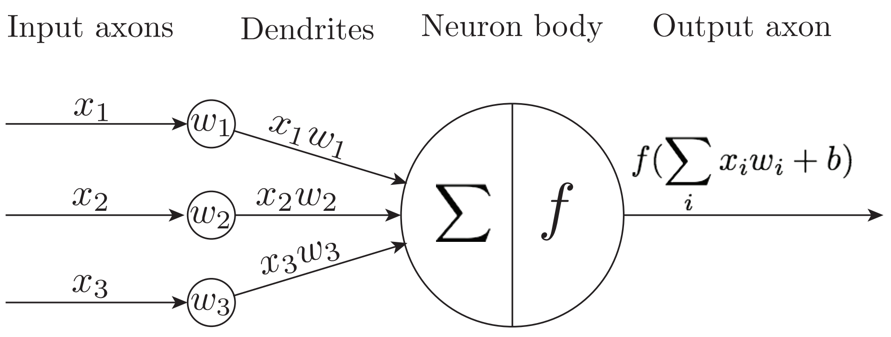

Figure 2.2: Artificial Neuron

The artificial neuron consists of following parts: multiple input dendrites which gather information from axons of other neurons, a body and a single output axon. The input is multiplied by the so called weight matrix which represents the way through the dendrite. The information from multiple dendrites is summed up in the body and then passed to the activation function. The crucial point is that the weight matrix is an adjustable parameter which acts similarly to a filter with respect to the incoming signal. The activation function is a univariate non-linear function. In terms of neurobiology the activation function could be interpreted as the action potential of the cell which functions in the following manner: when the output is near either of the boundary values of the function range, then the neuron is either transmitting the signal fully or partially. The resulting signal is on its way through the axon to the another neuron. So it is in the artificial neural network: the output of the activation function is multiplied by the weight parameters, summed up with another outputs of the same layer and passed to the next activation function.

2.3 Activation Function

Activation function makes neural network capable of modelling multivariate non-linear functions. All multivariate continuous functions can be represented by a finite number of superpositions and compositions of univariate non-linear functions [8]. There are several activation functions, e.g. hyperbolic tangent, sigmoid, ReLU. The characteristics of the hyperbolic tangent and sigmoid functions will be explored because these functions are used in this project.

First of all, the activation function is required to be continuously differentiable to enable obtaining the derivatives in the training phase. The other requirement is related to the function range. The hyperbolic tangent maps the input value into the range $[-1, 1]$, i.e. it is zero-centered. That property enables the neuron to handle skewed inputs without affecting the gradient dynamics. Sigmoid function range is [0, 1] and it matches the neuron action potential model perfectly, but it makes the sigmoid less reliable in dealing with the skewed inputs.

Another characteristic is also associated with the range of the function. It is more of a drawback which is relevant for both the hyperbolic tangent and the sigmoid. The output could be close to either of the boundary values, e.g. close to -1 or 1 in case of tangent. This will cause the gradient to be near zero or to be extremely high during the error propagation. Consequently, the chain of multiplied gradients will reach zero or to rise disproportionally in comparison to other gradient chains. Such network performs poorly during the training. The solution is to initialise weight parameters with the help of a special technique before training in order to avoid the saturation or “killing” the gradients, which would be discussed it further chapters.

Finally, the function derivative is generally required to be monotonic because it guarantees better convergence.

2.4 Output Layer

There are a lot of tasks neural network can perform. The appearance of the output layer is dependent on the task. Some neural networks may have a function in the output layer node, some may have not.

In this section the output layer function is examined on the example of the softmax classifier function. Softmax is used for classification as it has the possibility of distinguishing between multiple classes. Softmax is often explained in terms of the probability theory. Given a certain input x and a parameter set θ, softmax assigns to each class a normalised probability of being the class of x, that is:

$$P(x \in k|x, \theta) = \frac{e^{x_k}}{\sum_{i=1}^{K} e^{x_i}} \tag{2.1}$$

where $k \in \{1, 2 \ldots K\}$. The resulting classification is the class with the highest probability.

2.5 Loss Function

The training is performed in order to teach randomly initialised (which is, however, a misleading statement as there are initialisation techniques proved to affect the convergence speed) neural network to transform input data into meaningful outputs. The set of parameters θ fitting the data the best is to be found. Mathematically expressed, training is the process of determining the minimum of the loss function. The loss function generally defines the difference between the output of the network and the given value.

The cross-entropy loss function computes the difference between two probability distributions p and q. Here, p is considered to be the softmax classifier output and q to be the set of target probabilities. For the vector y of computed output probabilities and the target vector t of length K, the cross-entropy loss function takes the following form:

$$H(y, t) = -\sum_{i=1}^{K} y_i \log t_i \tag{2.2}$$

Chapter 3

Neural Networks

This chapter gives a brief introduction into recurrent neural networks in general and refers to important publications within the field. Besides, the LSTM neural network used for sequential analysis is discussed together with its bidirectional extension. The concepts from the previous chapter could be freely applied to the architectures explored in the following sections.

3.1 Recurrent Neural Networks

Recurrent neural network (RNN) is the type of neural network which has a recurrent unit in its architecture. The recurrent architecture of RNN makes it possible to use the information about the state of the system over a certain number of time steps. Therefore RNN outperforms dramatically traditional feed forward neural networks (FFNN) in sequential data modelling. RNNs are typically used for mining time-series, for example, in predicting financial data of speech recognition.

However, RNN possesses a major drawback known as the problem of vanishing and exploding gradients. To avoid the proliferation of gradients, the contextual information held by the network, is restricted to a certain number of time steps. Restricted amount of time steps is unable to reflect the temporal behaviour of data, thus leads to unsuccessful training. As the network is trained over a long time period using gradient based approaches, it tends to behave unpredictably. There are many research papers devoted to this handicap, for instance, the famous paper by Bengio et al. [6]. Nonetheless, the long short-term memory neural network introduced by Hochreiter and Schmidhuber in [16] still outperform vanilla RNN and remain the most efficient tool for modelling sequential data.

3.2 Long Short-Term Memory Neural Networks

Long short-term memory neural networks (LSTM) are said to retain the efficiency even when trained over an arbitrarily long time window. This is achieved by the following improvement in the recurrent architecture: a so called memory cell retains its state over a particular time period by means of the gating units which regulate the amount of context information in the cell. The recurrent unit in the cell is an internal state which allows to preserve learning error without causing gradients to vanish or explode. This is referred to as the CEC (Constant Error Carousel) technique. The architecture of LSTM neural networks will be explained in details in the chapter devoted to BLSTM neural networks.

3.3 Bidirectional Long Short-Term Memory Neural Network

Bidirectional LSTM was first described by Graves and Schmidhuber [13] and is generally used as a tool for speech processing. However, BLSTM are potentially useful for modelling any type of sequential data.

There are two ways of processing sequential data: overlapping windows and recurrent structure. Overlapping window technique suffer from its dependency on hyperparameters, namely window size and shift size [12]. LSTM neural networks are able to learn the sequential information independently of the input format. This means, that the long input sequence can be memorised by cutting it into shorter intervals and passing them one by one to the network. However, in case of vanilla LSTM networks, the resulting distribution model is forwardly biased because during training the output is always fully based only on the information from previous values in sequence. The bidirectional architecture of BLSTM neural network is aimed to handle this handicap as well. Because the system proposed in this thesis is based on BLSTM neural network, the following chapter is devoted to its detailed explanation.

Chapter 4

Bidirectional LSTM Neural Network

This chapter is devoted to the detailed explanation of the bidirectional architecture, the composition of the LSTM neural network and the information flow within the network. The accompanying information on training algorithms can also be found here.

4.1 Architecture

Bidirectional LSTM neural network consists of two LSTM blocks connected to a single output layer. One of the blocks is used for processing input sequences forwards while the other provides computation of those sequences in a backward direction. This approach ensures the completeness of sequential information before and after a particular point. LSTM blocks are composed of an arbitrary number of hidden layers. The larger the amount and the size of hidden layers, the less stable is the training process. Nonetheless, larger layer size enables modelling more difficult functions. The output layer is presented by an activation function. The figure 4.1 depicts the simple scheme of the BLST neural network.

4.1.1 LSTM Block Architecture

There are several neurons called gates and a memory cell in the block. Vanilla LSTM block contains input, forget and output gates. However, the block could be implemented in a number of different ways. Often the vanilla architechture is extended with peephole connections introduced by Gers and Schmidhuber in [10]. The number and the type of gates are task-

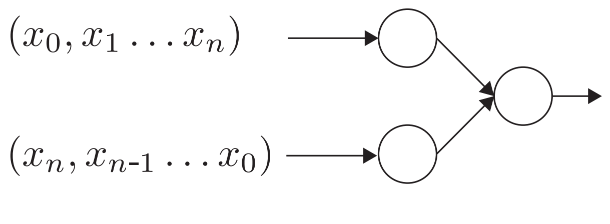

Figure 4.1: BLSTM Neural Network Scheme

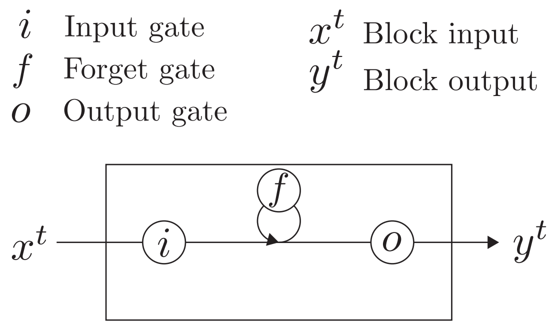

Figure 4.2: LSTM Block

dependent aspects. For instance, Gated Recurrent unit (GRU) block introduced by Cho et al. in [7] has input and forget gates coupled into a single update gate, while no peephole connections neither output gates are presented.

Input gate specifies the amount of information passing through into the block. The following unit in the flow is a memory cell which preserves information by means of self-recurrent connection with the forget gate determining the information penetrating further into the state of cell in the next time step. The described system of connected gates and a memory unit enables to keep gradients safe while retaining constant error flow, which is referred to as the Constant Error Carousel (CEC). Peephole connections are sets of weights which connect gates and the cell state. They allow the gates to get an access to the information about the state, which in turn reflects in this gate output. Finally, the output gate lets out the filtered state cell information. The block architecture is depicted in figure 4.2.

4.2 Forward Pass

This section gives the overview of the information flow inside the LSTM block. The graphical representation can be found in figure 4.3.

In the forward pass the input sequence is cut into t non-overlapping intervals referred to as states. The states are then passed one by one or in a vectorised form to the neural network. The goal of the neural network in the forward pass phase is to determine to which class belongs the input sequence. In a particular step t the block output y^{t-1} of a preceding step and the input vector x^t enter the current LSTM block. The block input z^t is then calculated according to the following equation, where $\odot$ is the operation of element-wise multiplication [14]:

$$z^t = \tanh(W_z x^t + R_z y^{t-1} + b_z) \tag{4.1}$$

The result after passing through the input gate:

$$i^t = \sigma(W_i x^t + R_i y^{t-1} + b_i) \tag{4.2}$$

The output for the forget gate f^t is:

$$f^t = \sigma(W_f x^t + R_f y^{t-1} + b_f) \tag{4.3}$$

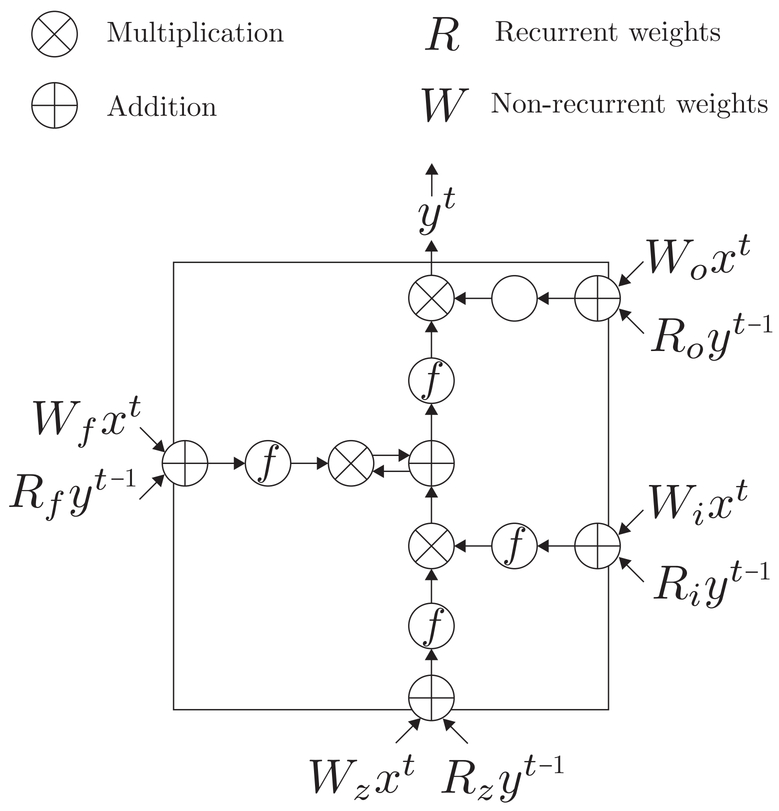

Figure 4.3: Information Flow in LSTM Block

The cell state c^t at any particular time step t is:

$$c^t = i^t \odot z^t + f^t \odot c^{t-1} \tag{4.4}$$

The output gate result is calculated as following:

$$o^t = \sigma(W_o x^t + R_o y^{t-1} + b_o) \tag{4.5}$$

Where for $j \in \{z, i, f, o\}$:

- W_j are weight matrices
- R_j are recurrent weights
- b_j are biases associated with gates z, i, f, o
- tanh and σ are activation functions: hyperbolic tangent and logistic sigmoid respectively

Finally, y^t, a block output, is:

$$y^t = o^t \odot \tanh(c^t) \tag{4.6}$$

4.3 Backward Pass

In the backwards pass phase the prediction obtained for a certain time step is first compared to the data observed in the system for the same time step. The function which computes the error based on the output of the network and the desired output is referred to as loss function. In order to make the predictions produced by the neural network fit the real data better, the adjustable parameters, namely weight matrices, are to be adjusted according to the attained error. This process is called backpropagation of error.

Moving towards the loss function minimum results in reducing the error. The gradient descent algorithm used in the backwards pass is based on the idea of deriving the loss function $L(\theta)$ with respect to the weight parameters θ in the neural network. The gradient purpose is to push the weights in the direction of the steepest descent of the loss function.

The backpropagation algorithm applied to recurrent neural networks (RNN) has a specific notation of backpropagation through time because of the possibility of such networks to be unfolded over an arbitrary number of time steps in order to propagate an error.

4.4 Weights Initialisation

In order to present a powerful LSTM structure with the ability to learn the desired information it is a good practice to extend the vanilla architecture by a pair of improvements. For example, Jozefowicz, Zaremby et al. [17] emphasise the importance of a proper bias initialisation for the forget gate. They recommend it to be set to approximately 1 to enable gradient flow and, consequently, to fasten learning of the long term dependencies. Neural network initialised with random weights is not guaranteed to converge even after many iteration steps. Commonly used heuristic is to set biases to 0 and draw weight values from uniform distribution in the range $(-n, n)$, where n is a size of a previous layer [11].

4.5 Training

Training is the crucial and the most challenging phase an artificial neural network development. In general, the training phase is broken into following stages. Given the input sequence, the predicted output is first compared to the target output data. This information is used for further optimisation of network parameters. The error function is computed and the error itself is propagated backwards through the network. Backpropagation of the observed error is realised by means of obtaining partial derivatives of the error function with respect to adjustable parameters and correcting parameters according to that information. The parameters, namely weights, are updated so that they cause the output to be closer to the real data and, therefore, to minimise the error function, which is referred to as gradient descent algorithm.

Training the neural network to recognise patterns of the host behaviour in the network traffic data is performed in a supervised manner. The process of extracting target values is described in the next chapter. The training is realised in the following way. The host activity represented by the numerical sequence is passed by smaller intervals of predetermined length to the network. After the whole sequence is processed, the neural network produces the output value which is then compared to the target value. The target value here is a class to which the sequence belongs. The loss function is calculated next and the weight parameters are updated. The same procedure is repeated for all classes. Datasets representing the classes have to be of an equal size. Trained neural network is able to classify the host behaviour.

4.5.1 Gradient Descent Algorithm

Gradient descent (also known as method of the steepest descent) is an optimisation algorithm which aims at finding a minima of a given loss function $L(\theta)$. Gradient descent achieves its goal by following the steepest descent direction given by the gradient of this function.

The algorithm receives a particular initial set of parameters θ as input and repeatedly adjusts them according to the update rule. The update rule is applied to the set of tunable parameters until the minima is reached or until the stopping condition is met. The stopping condition is met when the minima of the given function is found. Generally, the update rule could be expressed in the following form:

$$\Delta\theta_i = \alpha \nabla L(\theta_i) \tag{4.7}$$

$$\theta_{i+1} = \theta_i - \Delta\theta_i \tag{4.8}$$

where θ_i is a set of the parameters being updated at the iteration step i and α is a learning rate, i. e. the value which regulates the speed of computing the minimal solution. To highlight the importance and trickiness of setting the learning rate, it should be noticed that chosen improperly it may affect the training in such way that the network will oscilate near minima unable to converge or will diverge completely.

Depending on the amount of training data there are three different types of the algorithm. Batch version of the gradient descent computes the update based on the entire dataset, which is costly in terms of both memory and performance. Additionally, the accumulated update changes increases proportionally to the size of the dataset. Thus, there is a risk for the large datasets of overshooting the function minima.

Stochastic version of the algorithm partially solves the drawbacks of the batch version by performing the computation per each training example. However, the local gradients could be noisy or in contradiction with each other and, as a consequence, will produce fluctuations while converging to a local minima [22].

The mini-batch gradient descent algorithm deals with the weaknesses of both approaches. The averaged update change of the subset of examples is used iteratively for tuning parameters. However, the vanilla mini-batch algorithm realisation does not guarantee stable convergence. Besides, the algorithm still requires additional optimisation to handle learning rate in a way that maximises its efficiency as well as to manage non-convex functions with multiple suboptimal minima. There are plenty of optimisation approaches addressing the mentioned problems for mini-batch gradient descent algorithm, one of which, namely Adadelta, will be described in detail further. Nonetheless, the general adjustment approach does not undergo any dramatic change, the optimisation brings a bunch of corrections in the way the update delta is computed. For instance, the Momentum optimisation method introduces a velocity concept defined as follows:

$$\Delta\theta_i = \gamma \Delta\theta_{i-1} - \alpha \nabla L(\theta_i) \tag{4.9}$$

where γ is a constant determining how much the update delta of the previous iteration contributes to the one of the current iteration step.

4.5.2 Adadelta

Adadelta was firstly presented by M.D. Zeiler [23]. The update rule is defined as follows:

$$\Delta\theta_i = -\frac{RMS\,[\Delta\theta]_{i-1}}{RMS\,[g]_i} g_i \tag{4.10}$$

$$\theta_{i+1} = \theta_i + \Delta\theta_i \tag{4.11}$$

where g_i is a shortcut for derivation $\nabla L(\theta_i)$ with respect to parameter set θ_i and RMS stands for a Root Mean Square.

Adadelta method was developed to overcome the sensitivity of the algorithms of this kind to the hyperparameter selection. For example, the performance of the above-mentioned Momentum method is thoroughly dependent on the learning rate parameter. Choosing a learning rate parameter by hand is more of a state-of-art problem. Adadelta redefines the update rule so that the human factor is no longer involved in process because the learning rate is computed according to the history of past updates through the exponentially decaying average:

$$E[\Delta\theta^2]_i = \rho E[\Delta\theta^2]_{i-1} + (1-\rho)\Delta\theta_i^2 \tag{4.12}$$

where ρ is a decay constant regulating the effect of past history on the current value, which satisfies the condition $0 < \rho < 1$. After the offset constant ϵ is added, the square root is taken. Resulting value is nothing but a root mean square (RMS):

$$RMS\,[\Delta\theta]_i = \sqrt{E[\Delta\theta^2]_i + \epsilon} \tag{4.13}$$

The optimisation methods commonly use the same global learning rate for tuning all weight parameters, which results in poor performance. Adadelta solves this issue by means of the adaptive learning rate similar to that of Adagrad, where the concept was firstly presented [9]. Adaptiveness means that the different learning rates are applied to different weight parameters, i. e. the less frequent parameters receive the greater update and vice versa.

However, Adagrad learning rate is inclined to decrease monotonically as it accumulates squared past gradients for updating weight parameters. And this is another problem Adadelta is aimed to solve. The method of Adadelta uses a window of a fixed size to bound the number of accumulated gradients. Moreover, it is not necessary to store all gradient values as Adadelta accumulates them by means of exponentially decaying average. Thus, for the iteration step i, the average $E[\nabla L(\theta)^2]$ is defined as follows:

$$E[\nabla L(\theta)^2]_i = \rho E[\nabla L(\theta)^2]_{i-1} + (1-\rho)\nabla L(\theta)_i^2 \tag{4.14}$$

The RMS is computed for the gradient update as well:

$$RMS\,[\nabla L(\theta)]_i = \sqrt{E[\nabla L(\theta)^2]_i + \epsilon} \tag{4.15}$$

The approach is represented in algorithm 1.

Algorithm 1 Adadelta algorithm

```
1: procedure adjustWeights($gradients, learning\_rate, stability\_factor, rho$)
2:     $delta\_grad \leftarrow 0$
3:     $delta\_upd \leftarrow 0$
4:     for $grad \in gradients$ do
5:         $delta\_grad \leftarrow rho * delta\_grad + (1 - rho) * grad^2$
6:         $RMS_{upd} \leftarrow \sqrt{delta\_upd + stability\_factor}$
7:         $RMS_{grad} \leftarrow \sqrt{delta\_grad + stability\_factor}$
8:         $update = grad * \frac{RMS_{upd}}{RMS_{grad}}$
9:         $delta\_upd \leftarrow rho * delta\_upd + (1 - rho) * update^2$
```

4.5.3 Backpropagation Training Algorithm

The training is realised by the propagation phase followed by weight parameters update. In the first phase, the information in propagated forwards the neural network, the error is computed next and propagated backwards. After that the parameters update changes are obtained. During the next phase the simultaneous parameters adjustment is performed. The whole process described is referred to as the backpropagation training algorithm. The problem is that this approach is not fully applicable for training recurrent neural networks because it does not take into account their recurrent structure.

A special kind of error propagation is backpropagation through time (BPTT). BPTT enables a so called unfolding of the network for a certain amount of time steps. This makes possible to accumulate error over those time steps. The structure used for unfolding is the weight set which connects the memory cells in a recurrent manner. Thus, the unfolded recurrent neural network is an extended version of the traditional feedforward one. The final error is propagated backwards, and the weights are penalised multiple times for each contribution to it.

The BPTT accumulates the error over the whole length of the input sequence. This, however, results in a high cost of a parameter update [21], as well as this kind of the BPTT cannot be performed during the online training because the complete sequence is not provided. The solution is to restrict the amount of memorised states to a particular number. This is referred to as truncated BPTT. The input sequence is divided into subsequences of size s, the BPTT is performed under condition that the state of each memory cell is used for processing the error of the next one. The drawback of the suggested approach is that the memory beyond the truncating threshold is not fully captured by the model in comparison to the full BPTT.

Chapter 5

Data Preprocessing

Data preprocessing is an important part of any data mining system. Its purpose is to provide informative but consistent features for the mining process. There is a wide range of challenges related to the field of data preparation for its further usage. Real world data possess the following general characteristics which prevent the researcher from working directly with them:

- incompleteness, which means there is a lack of values of the observed feature,
- inconsistency, which indicates a logical collision presented in data,
- noisiness, which refers to a presence of redundant data and, as a result, distorted representation of the process originating the data.

The rest of the chapter gives the overview of the data preparation process.

5.1 DNS Attacks

The DNS (Domain Name System) is the system that is responsible for resolving a domain name to the corresponding IP address. Today the DNS is certainly one of the most important parts of the Internet. Therefore, extra attention should be paid to its security. New forms of hiding attacks still arise, e.g. Fast Flux and older forms become more advanced, e.g. Nitol.

Consider the following example. The aforementioned attack Nitol belongs to a family of continuously evolving DDoS attack (DDoS stands for Distributed Denial of Service). The botnet malware hijacks victim machines via TCP. As a result of the DDoS attack, the attacked server becomes inaccessible for users. DDoS reaches its aim by flooding a target server with packets, requests or queries. In simple terms, for each host the diversity of the queried domain name feature will be low. The same holds for the port feature because DDoS is realised via protocol that does not require handshaking, i. e. UDP (ICMP in case of a Smurf DDoS). Next, the Flags section typically has the same configuration for all request messages. This is how the DDoS attack can be tracked.

5.2 DNS Network Traffic Data

Collected data are parts of DNS request messages together with the client address and port which were aggregated online with the appropriate tool. The DNS protocol uses two kinds of messages: request and response. Each message contains a header and four sections: question, answer, authority and a section for additional information. The request message is formed after the domain name is resolved to the appropriate IP address. Explored features are the following:

- timestamp denoting the occurrence of request,
- host address,
- destination port,
- message identification number (ID),
- message flags,
- QName and,
- QType from the question section.

While timestamp and host address are self-explanatory, the rest of the features will be discussed further in the section. The message ID is located in the header section. It is a 16-bit identification integer generated by the program which performed the query. The response message holds the same ID, hence the ID is used for matching two messages. 16-bit flags section section is placed in the header. Its QR part (Query/Response Flag) holds information about the type of message itself: is it a query or a response and in case of it is a query, additionally, is it of recursive or non-recursive type (RD for Recursion Denied, RA for Recursion Available). Besides, it indicates whether the responding server is authoritative for the zone of the queried domain (AA for Authoritative Answer Flag) and the message truncation (Truncate Flag). 4-bit Opcode part is specified by the creator of the query and contains specific information about the query carried by message. Three bits left (Zero) are reserved bits. QName feature contains a domain name being queried. Finally, QType is another unsigned 16-bit value which contains the query type.

For further analysis the data were cut into pieces in range from 00:00:00 of the certain day to 23:59:59 of the same day. This time span was chosen in order to make data visualisation more clear to human eye. In addition, the 24-hour scope represents well the underlying dynamics of network traffic. The data in this scope are nonseasonal. An example can be found in the figure 5.1. The 24-hour scope itself forms a season that repeats over longer time range.

Another important concept is that the underlying distribution of the network traffic data is different in different networks. There is no universal distribution form. Moreover, the proposed classification system is assumed to be applicable to any network. Thus, the techniques for data preparation should be distribution-free.

Data were collected in the network for 1 month. Data are unlabelled and presumably consist of multiple examples of normal network dynamics and multiple cases of anomality. For instance, the normal behaviour is different at the weekdays and the weekends and has various meaningless outliers and noisy fluctuations which complicat the classification. There are numerous forms of the anomalous behaviour as well. However, the description of various types

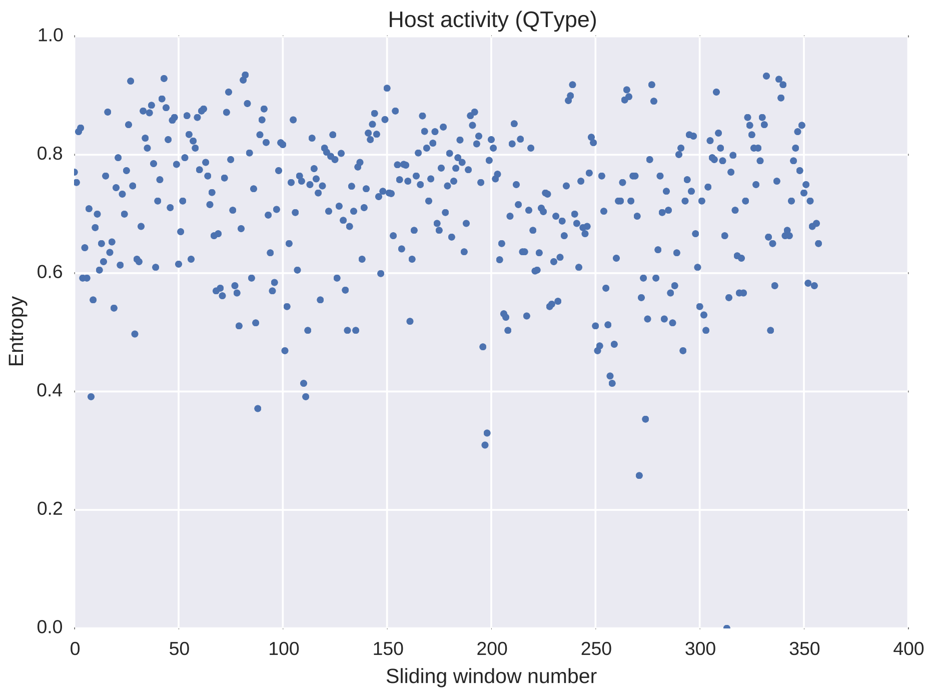

Figure 5.1: Example of ‘QType’ Entropy Distribution

of anomalous events are beyond the scope of this research. Thus, the goal of the preprocessing phase is to find general patterns of different network dynamics and use them as labels for preparing datasets sufficient for the training of the monitoring unit, i. e. BLSTM neural network.

During the preparation phase, original temporal sequences will be transformed from the categorical type to numerical, then smoothed and clustered in order to extract general patterns. However, smoothed data will not be used for further training of the neural network because the goal is to model the original data. Smoothing is performed in order to simplify the computation the similarity within the dataset.

5.3 Entropy

In data processing any variable is handled according to its data type. All variables from the list above except for timestamps are of categorical data type. Categorical data lack order and are nominal discrete values. In order to use numeric data mining algorithms the categorical data is commonly converted into numerical. When the feature domain is relatively small then the method of binarisation is applied. For instance, if some categorical feature has n different values, then after binarisation there are n binary features and exactly one of them takes on the value of 1 while the others are equal to 0. However, domains of features used in this paper are too large. For example, if there are 65535 ports, the port attribute binarisation would result in 65535 additional features, which is unacceptable. The same holds for unsigned 16-bit identification feature as it takes on values in the range of $[0, 2^{16}]$. The Flag section has less possible values because QR, AA, TC, RD, RA 1-bit values which could be either 0 or 1 and the Opcode section is also broken into four 1-bit parts values. Still the binarisation of even those two features is costly in terms of co called ‘curse of dimensionality’. High dimensionality makes analysed data sparse. The problem is that the amount of data needed to validate any mined knowledge grows exponentially with the growing sparsity.

The solution suggested in this paper is to use entropy as a summarising measure of network traffic dynamics. The reasoning behind this approach is presented further. First of all, using entropy eliminates the need to store and operate tremendous amount of information like, for example, in host based analysis method. Secondly, the entropy is defined as a measure of variability, which is also the definition of the network traffic dynamics. Normally, traffic diversity remains on intermediate level. Anomalous activity is close to either of extremes. For instance, DDoS attack is characterised by both dramatic decrease in the diversity of the destination addresses and simultaneous increase of the source addresses diversity. The equation (5.1) used for computing normalised entropy can be found in the next section.

5.3.1 Windowing

Since the monitoring system is supposed to work in the online mode, it is natural to introduce a segmentation approach in order to obtain meaningful statistics about the process generating the data on-the-fly. The sliding window technique is applied to the input time series. This technique has two adjustable parameters, i. e. the length l of the window itself and the shift size s. The shift parameter denotes the amount of time to be skipped until the next statistics computation. Experiments showed that for best sensitivity to the changes the window should be set to 10 minutes and the size of the shift to 4 minutes. Such overlapping window is able to capture the graduality of evolving nature time series.

The following attributes are derived from the data obtained in the interval of size l:

- m is the total number of items in the window
- k is the number of distinct values of feature
- n_i, $i \in \{1, 2 \ldots k\}$ denotes the number of the k-th value in the window
- $\log(k)$ is the normalisation term

The statistics are calculated for all input features described in the previous section. Next, the entropy of the window is calcucated according to the equation:

$$H = \sum_{i=1}^{k} \frac{\frac{n_i}{m} \log \frac{n_i}{m}}{\log(k)} \tag{5.1}$$

Minimum value of zero is attained when all of the items in the window are the same and the value maximum is taken on when the diversity is the highest possible. This reflects the network traffic dynamics very well.

5.4 Clustering

This section refers to the problem mentioned in the beginning of the chapter. The normal behaviour patterns presented in the collected data may vary. Moreover, there could potentially be groups of various anomalous patterns. They are to be detected and separated into groups of equal size in order to train the neural network efficiently. In case of the prevalence of any type of pattern, the performance of neural network may be highly biased.

For this purpose, the suggested approach is to perform clustering with DTW as the similarity measure between data samples. Data sample, as it was mentioned earlier, is a vector of a numerical values collected during the 24-hour scope. The approach and related techniques are described in the rest of the section.

5.4.1 Filtering

The data itself is the combination of the underlying function and added noise. Filtering the noise out helps to reveal the true pattern behind the data and, additionally, makes the clustering more efficient.

The filtering approach of choice is determined by the nature of the data. Any time series data are characterised by the following components: trend, cycle and season. The noisy fluctuations are generally referred to as an irregular component. The trend component is represented by the gradual shifting to higher or lower values. The seasonality accounts for repetition of some pattern in data and is commonly defined for fixed time periods, e.g. year. The cycle component describes the upward or downward shift in data that does not fall into any fixed period. The collected data presented in figure 5.1 has no significant trend, seasonal or cyclic component. It is partly caused by the scope of examination, i. e. 24 hours. The basic filtering technique, namely moving average, is sufficient to deal with the irregular component and to filter out the noise. However, there are many variations of moving average technique. EWMA (Exponentially Weightened Moving Average) is used in this research. EWMA smoothes given time series by calculating the average for a sample window of a specified size. For $t > 0$ exponentially decreasing weights are then applied to the previously smoothed samples in the following manner:

$$\widehat{y_t} = \alpha y_t + (1 - \alpha)\widehat{y_{t-1}} \tag{5.2}$$

where $Y = \{y_1, y_2 \dots y_n\}$ is the original time series divided into n overlapping windows, $\hat{Y}$ is the filtered version of Y, $0 < \alpha < 1$ is a weight constant and the starting condition is $\widehat{y_1} = y_1$.

5.4.2 Dynamic Time Warping

DTW (Dynamic Time Warping) is a advantageous method to measure the similarity between time series because that the two compared time series may vary in time or speed. DTW performs stretching or compressing of data along the temporal axis to find the best mapping between two series.

Let $Y = (y_1, y_2, \dots y_n)$ and $X = (x_1, x_2 \dots x_m)$ be time series, where x may not be equal to m. The distance matrix of size (m, n) is computed for all points of both series. Next, the DTW mapping matrix (m, n) is computed. The function of DTW takes the following form [2]:

$$DTW(i,j) = distance(x_i, y_j) + argmin \left\{ \begin{array}{c} DTW(i, j-1) \text{ repeat } x_i \\ DTW(i-1, j) \text{ repeat } y_j \\ DTW(i-1, j-1) \text{ repeat } neither \end{array} \right\} \tag{5.3}$$

The equation outputs the optimal distance between first i points of X and j points of Y. The *distance* function is chosen depending on the origin of the data. Generally, the Euclidean distance function is used:

$$distance(p, q) = \sqrt{(p-q)^2} \tag{5.4}$$

5.4.3 Agglomerative Clustering

The purpose of clustering is to divide the raw data set into groups. The agglomerative clustering is used in cases when the number of clusters is unknown. The agglomerative process starts with considering each point as a separate cluster. Clusters which are close to each other are merged into a single cluster. The procedure repeats until all points are located within one large cluster. The shortest distance between clusters for the single linkage algorithm is given by the closest elements from those cluster. The result of such clustering is generally depicted with the dendrogram as it enables to search for the best branching factor. The input for the clustering subroutine are sets of time series, where each sequence represents the distribution of a single feature, i. e. source ip address, destination port, query identification number, query name, query type or query flags.

Chapter 6

Module Design

Despite the progress made in the network security field, cyber threats are still an actual issue. New forms of threats are emerging, older forms are evolving. Such previously undefined hazards are to be described first, then it is possible to prevent networks against them. The purpose of this research is to develop a tool for the analysis of the network traffic data in order to obtain knowledge about the possible threats. The data collected in the network falls within the big data category which means that the traditional data processing methods are inapplicable to such data sets due to their complexity and vastness. The restrictions implied by such nature of the collected data are related to the part of the developed tool which is intended to perform real time analysis.

This research is focused on a host group behaviour classification and utilises the modelling power of the neural networks to achieve this goal. The proposed approach is not rule-based, because the definition of anomalous event is context-dependent. It is hard to denote some particular abnormal behaviour as an attack on the basis of existing vague definitions of attacks. What is considered as an attack in one case, could be an example of misbehaviour in the other. It could also be a singular coincident event. The method proposed in this thesis provides a tool for behavioural patterns recognition.

The proposed approach contains two phases: classification of hosts in a network and recognising the activity of a particular class. The classification is performed in an unsupervised manner. It enables the user who works with the developed system to obtain patterns of host behaviour in the network. In the second phase the user is required to prepare the datasets for the further training. After the training, the system is able to recognise the repetitive occurrence of behavioural patterns in the previously unseen network traffic. On one hand, it enables to track the presence of any particular type of behaviour and narrow the further research. On the other hand, it makes the re-identification of host groups possible.

The approach developed in this paper is implemented as a set of modules written in Python. The classification module is capable of finding patterns of malicious and normal host behaviour without any assumption about the distribution of traffic in a given network. The monitoring module is intended to deal with is the identification of hosts acting suspiciously in the network according to derived classes. The rest of the chapter discusses the functionality of modules and describes the approach in general. Descriptive diagrams can be found at the end of this chapter in figures 6.1, 6.2. Detailed information about methods is presented in the User Guide chapter.

6.1 Preprocessor Module

First of all, the preprocessor module provides instruments for the input preparation for its further usage. The raw data being explored are DNS packets in a textual format captured in a network during one month. Originally the data are in the format of a 31-day long sequence. All features are of categorical type and need to be converted into convenient numerical format. Then the data are cut into 24-hour sequences and categorical values for each feature are transformed to entropy with the overlapping sliding window. Next, the value sequences are grouped according to hosts. The hosts are arranged according to the order of their activity. The order is a number of host requests during a certain day. The user may specify the threshold of the host activity and then explore only the most active ones. The workflow is depicted in the figure 6.2.

After the input is prepared, it is smoothed in order to reduce significant fluctuations and misleading noise. Filtering extreme fluctuations out helps to make the procedure of similarity estimation faster and more precise. Smoothed sequences are passed to the method which computes the distance matrix with the DTW (Dynamic Time Warping) algorithm. DTW is a perfect tool for computing sequential similarity even if input sequences are not of equal length, have pattern shifts or mere distortions. Next, the agglomerative clustering is performed on the distance matrix and the hosts are grouped according to the obtained classification.

6.2 Monitoring Module

The monitoring module builds the model of the traffic dynamics with the help of BLSTM (Bidirectional Long Short-Term Memory) neural network. It consists of two LSTM blocks, one of which processes the input in its natural order and the other processes it in reversed, connected to the classifier. The architecture is showed in figure 6.1. The neural network training realised in the supervised manner: the user divides the data into training sets according to acquired behaviour patterns. The main benefit from using the recurrent neural network is that it can be applied to the data of unknown distribution. Furthermore, the distribution of traffic features is generally considered to be non-gaussian. This means there are no reliable parameters to estimate the distribution. Thus, the neural network, a well known tool for nonparametric regression, is the best choice for discovering the underlying model of data. Trained neural network is able to monitor the network traffic and classify the hosts according to their behaviour. Another reason for choosing the neural network is that it is vulnerable to the size of data set. The modelling with neural networks is an incremental procedure as well as further monitoring is, so it requires in-memory access only to its smaller parts. Neural network can also be trained in parallel.

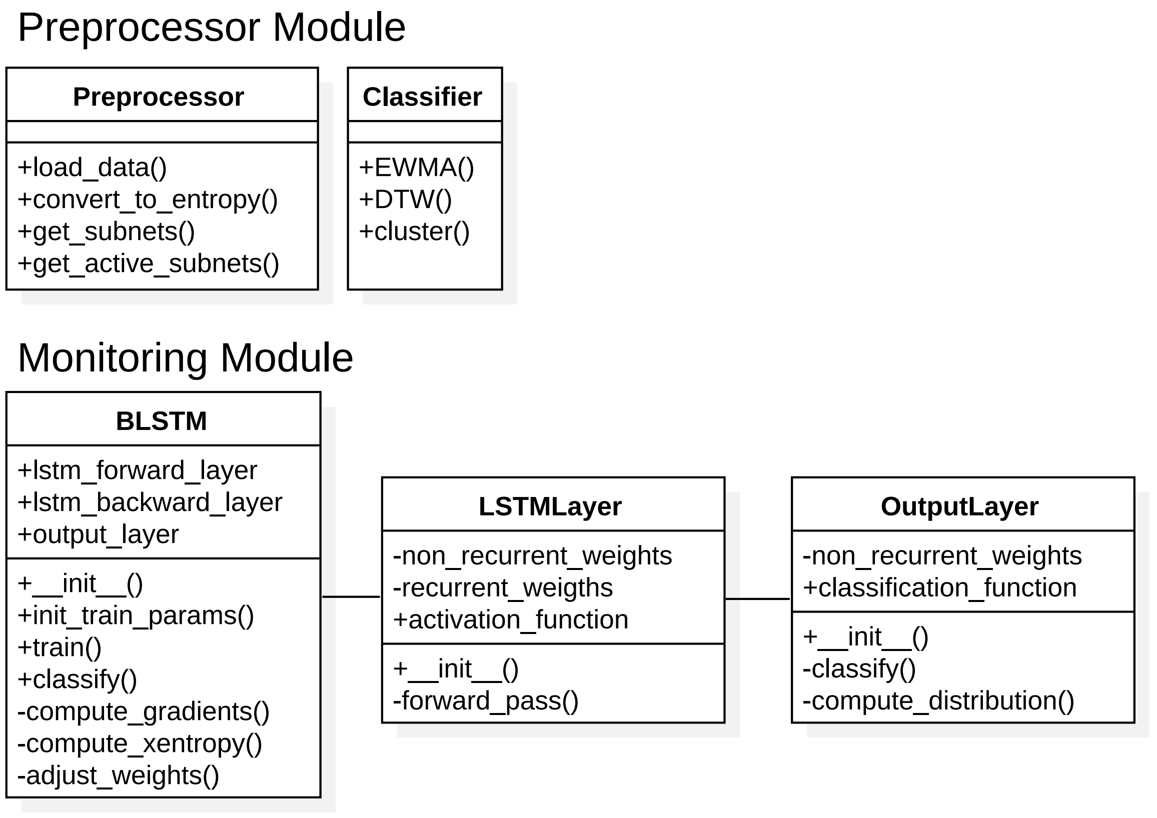

Figure 6.1: Class Diagram

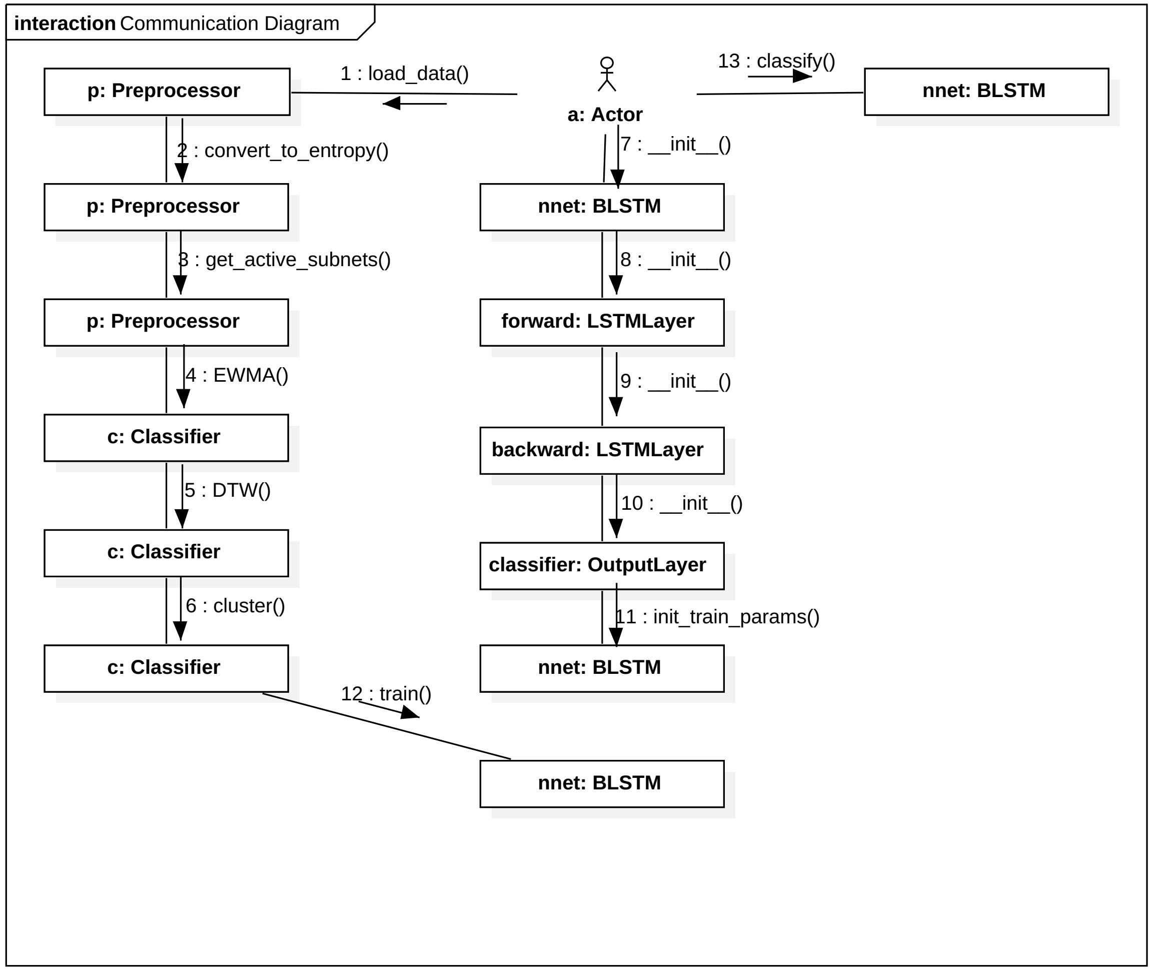

Figure 6.2: Preprocessor Module and Monitoring Module Communication Diagram

Chapter 7

User Guide

The chapter is written in a Python documentation-like style and gives the information on how to use each module methods, their input parameters and output values. In order to start using the modules place source files into the working directory, start Python kernel in and import them with the keyword ‘import’. Libraries required for using the module are listed further.

The language used for implementation is Python. Used libraries and modules are advanced array structures presented in numpy and pandas libraries, low-level tools for implementing neural networks available in library Theano and clustering algorithms available in library scipy.

The preprocessor and classifier parts are written and intended to be used in functional programming style which means that mutable data are avoided. The BLSTM neural network from the monitoring part requires not only the ability to mutate data stored within classes, i.e. iterative weights adjustment, but to manipulate expressions and statements as if they were plain data. This is a key feature of symbolic programming. Functions provided by the Theano library are an example of such programming style. It enables to represent functions as abstract rules of evaluation and treat them as a chain of interconnected units which enables to manipulate such chain of rules as it if it were the simple basic rule. For instance, a neuron accepts data, sums it up, applies an activation function and passes the result to neurons from the next layer. During the training, this chain of activation functions is transformed into the chain of derivatives in reversed order. In symbolic programming such nontrivial procedure is reduced to derivation of the reversed abstract rule.

7.1 Preprocessor Module

preprocessor.load_data(path)
Parses temporal data into pandas.DataFrame.

Parameters:

path: path to csv-file.

Returns:

$pandas.DataFrame$ of size $m * n$, where m is the number of samples in the dataset and n is the number of input features.

preprocessor.get_hosts(data)

Extracts all unique hostnames in the input data.

Parameters:

data: $pandas.DataFrame$ of size $m * n$, where m is the number of samples in the dataset and n is the number of input features.

Returns:

Array of strings, where each string contains hostname.

preprocessor.get_active_hosts(data)

Extracts hosts which sended a request more than the value of mean.

Parameters:

data: $pandas.DataFrame$ of size $m * n$, where m is the number of samples in the dataset and n is the number of input features.

Returns:

Array of strings, where each string contains hostname.

preprocessor.convert_to_entropy(data, window=10, shift=4)

Transforms categorical features of input data into the normalised entropy.

Parameters:

window: int

shift: int

data: $pandas.DataFrame$ of size $m * n$, where m is the number of samples in the dataset and n is the number of input features.

Returns:

numpy array of size $l * n$, where l is the new number of samples determined by the $window\_size$ parameter and n is the number of input features.

7.2 Classifier Module

classifier.EWMA(data, span=10)

Performs exponentially weightened smoothing of the input sequences.

Parameters:

data: numpy matrix of size $m * n$, where m is the number of samples and n is the number of input features.

span: int
Specifies the decay parameter accordig to the equation $\alpha = 2/(span + 1)$.

Returns:

numpy matrix of size $m * n$, where m is the number of samples and n is the number of input features.

classifier.DTW(data)
Performs pairwise comparison of sequences with the DTW.
Parameters:

data: numpy matrix of size $m * n$, where n corresponds to the number of sequences and m is the length of the single sequence

Returns:

condensed distance matrix in the form of *scipy.spatial.distance.pdist*.

classifier.cluster(distance)
Performs agglomerative clustering on the condensed distance matrix.
Parameters:

distance: Condensed distance matrix in the form of *scipy.spatial.distance.pdist*.

Returns:

matrix of class labels.

7.3 Monitoring Module

7.3.1 BLSTM Class

BLSTM.__init__(self, d_in, d_hidden, d_classes, classifier_name)
Creates an instance of the BLSTM class.
Parameters:

d_in: int
Input size.

d_hidden: int
Hidden layer size.

d_classes: int
Number of classes.

classifier_name: ‘softmax’ or ‘hard_softmax’

Returns:

An instance of the BLSTM class.

BLSTM.initTrainParams(self, learning_rate=0.1, stability_factor=1e-6, rho=0.9)
Initialises the relations between training parameters and their update functions according to Adadelta training algorithm.
Parameters:

learning_rate: float

stability_factor: float

rho: float

Returns:

Nothing. The function is applied to an existing instance of the BLSTM class. A symbolic graph representing the Adadelta learning algorithm is built as a result.

BLSTM.train(self, x_forwad, x_backward, target_label)
Runs the training.
Parameters:

x_forwad: numpy array
Input sequence or numpy matrix of input sequences for parallel computing.

x_backward: numpy array
Reversed input sequence or numpy matrix of reversed input sequences for parallel computing.

target_label: int
Class label or a numpy array of class labels for parallel computing.

Returns:

Nothing. Existing set of weights is modified as a result.

BLSTM.classify(self, x_forwad)
Classifies the input sequence.
Parameters:

x_forwad: numpy array
Input sequence or numpy matrix of input sequences for parallel computing.

Returns:

int or numpy array of int values, where each number represents a class label.

7.3.2 LSTMLayer Class

LSTMLayer.__init__(self, d_in, d_hidden, activation_function, x)
Creates an instance of the BLSTM class.
Parameters:

d_in: int
Input size.

d_hidden: int
Hidden layer size.

activation_function: ‘tanh’ or ‘sigmoid’

x: symbolic reference to the input sequence or a matrix of input sequences for parallel computing.

Returns:

An instance of the LSTMLayer class.

7.3.3 OutputLayer Class

OutputLayer.__init__(self, forward_y, backward_y, d_hidden, d_classes, activation_function)
Creates an instance of the BLSTM class.
Parameters:

forward_y: symbolic reference to the forward LSTM layer output

backward_y: symbolic reference to the backward LSTM layer output

d_hidden: int
Hidden layer size.

d_classes: int
Number of classes.

activation_function: ‘softmax’ or ‘hard_softmax’

Returns:

An instance of the OutputLayer class.

Chapter 8

Results

This chapter demontstrates the results of the method evaluation on the real-life dataset. The accuracy score is presented in tables 8.2 and 8.3 at the end of the chapter. Graphs presented in the chapter are made with the help of matplotlib and seaborn Python libraries.

8.1 Extracting Training Data

The following section shows how the processing module is intended to be used along with gained outputs and explanatory figures.

8.1.1 Raw Data Preparation

Firstly, the raw dataset was cut into 24-hour intervals. Next, each 24-hour interval was divided into sequences according to the client address. Selected features, namely identification number, destination port, query type, query name and flags, were converted from the categorical type to the numerical, i. e. normalised entropy, with the sliding window technique. Each resulting sequence describes the activity of a single host during the 24-hour scope.

Sequences were computed for the different window lengths and shift sizes. Following parameter sets reflect the process of hand-tuning: 10 minutes and 4 minutes, 4 minutes and 2 minutes, 1 minute and 30 seconds, 10 seconds and 2 seconds. The best granularity was achieved with the last parameters set. This configuration was able to reflect the short term abnormalities in traffic dynamics, e.g. scanning. However, such detailing is excessive in case when there is no unusual dynamics. This is because the small window cuts the original sequences into large number of small intervals which incorporate fewer packets. Smaller number of packets within an interval results in unnatural uniformity. This can be seen in figures 8.3, 8.4. Figures 8.1, 8.2, 8.3, 8.4 depict the entropy distribution of the ‘QType’ feature for different lengths of the sliding window and shift sizes within the same 24 hours for an anonymised host.

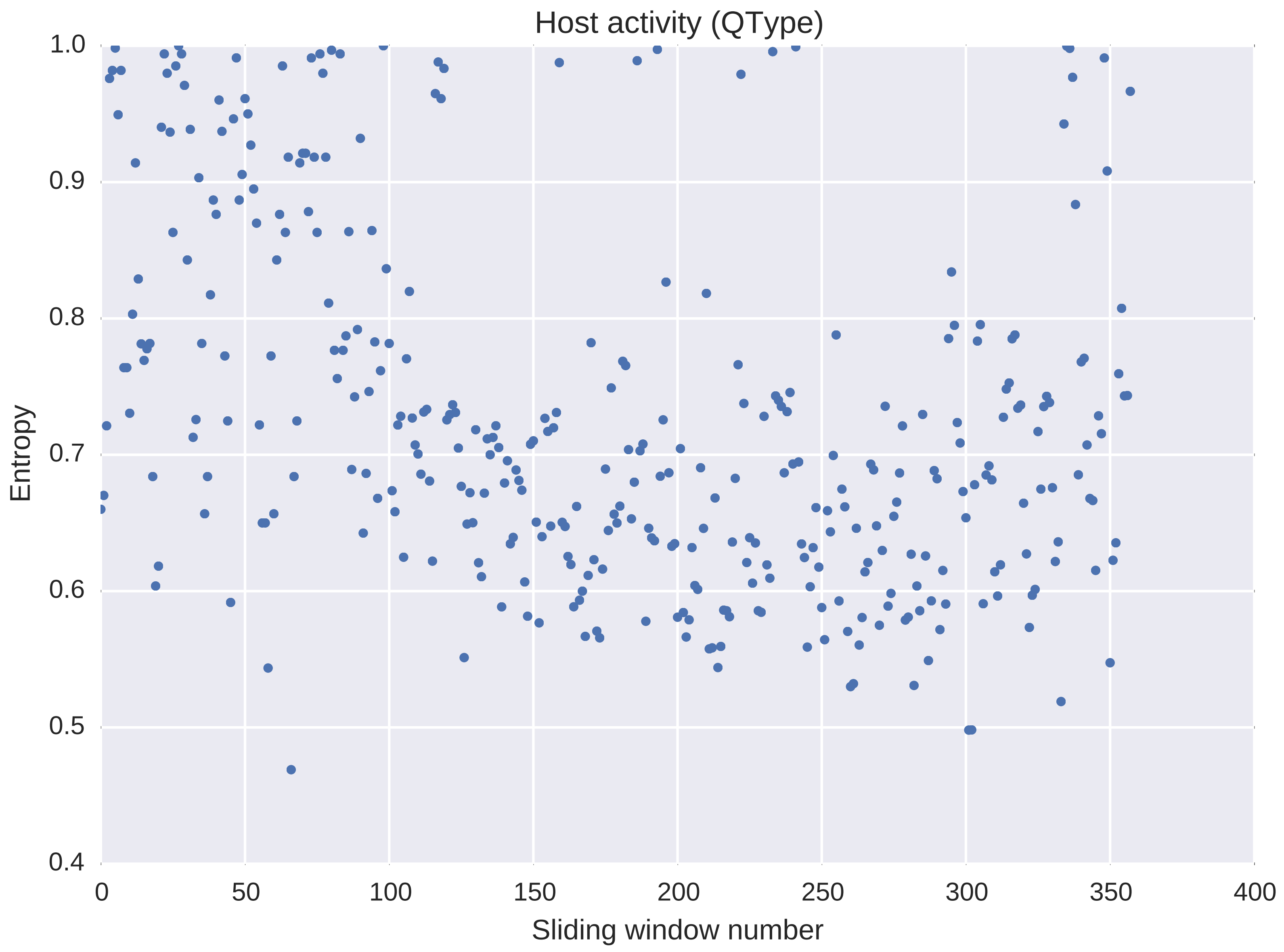

Figure 8.1: Entropy Distribution of ‘QType’ Feature (10 m, 4 m)

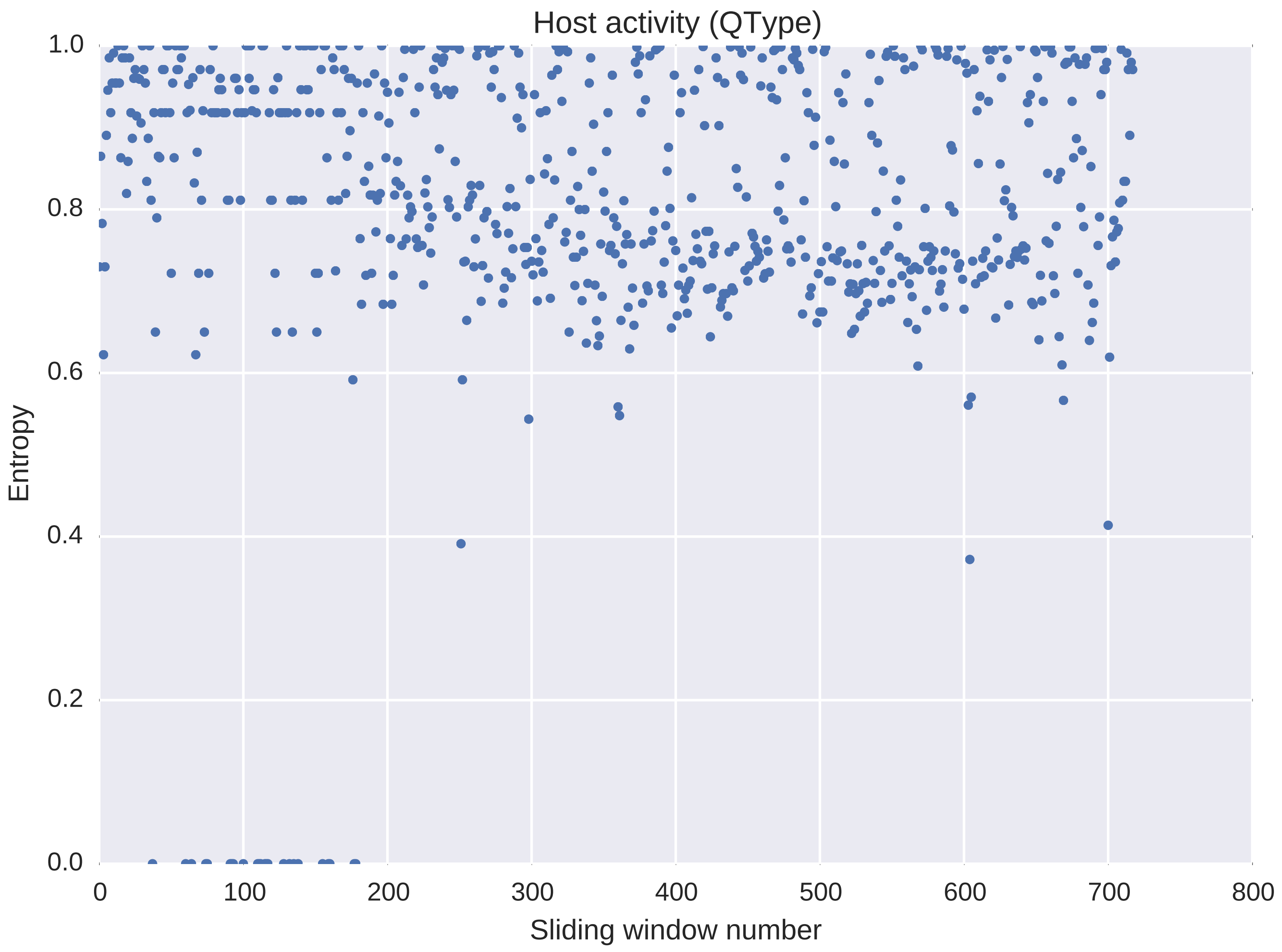

Figure 8.2: Entropy Distribution of ‘QType’ Feature (4 m, 2 m)

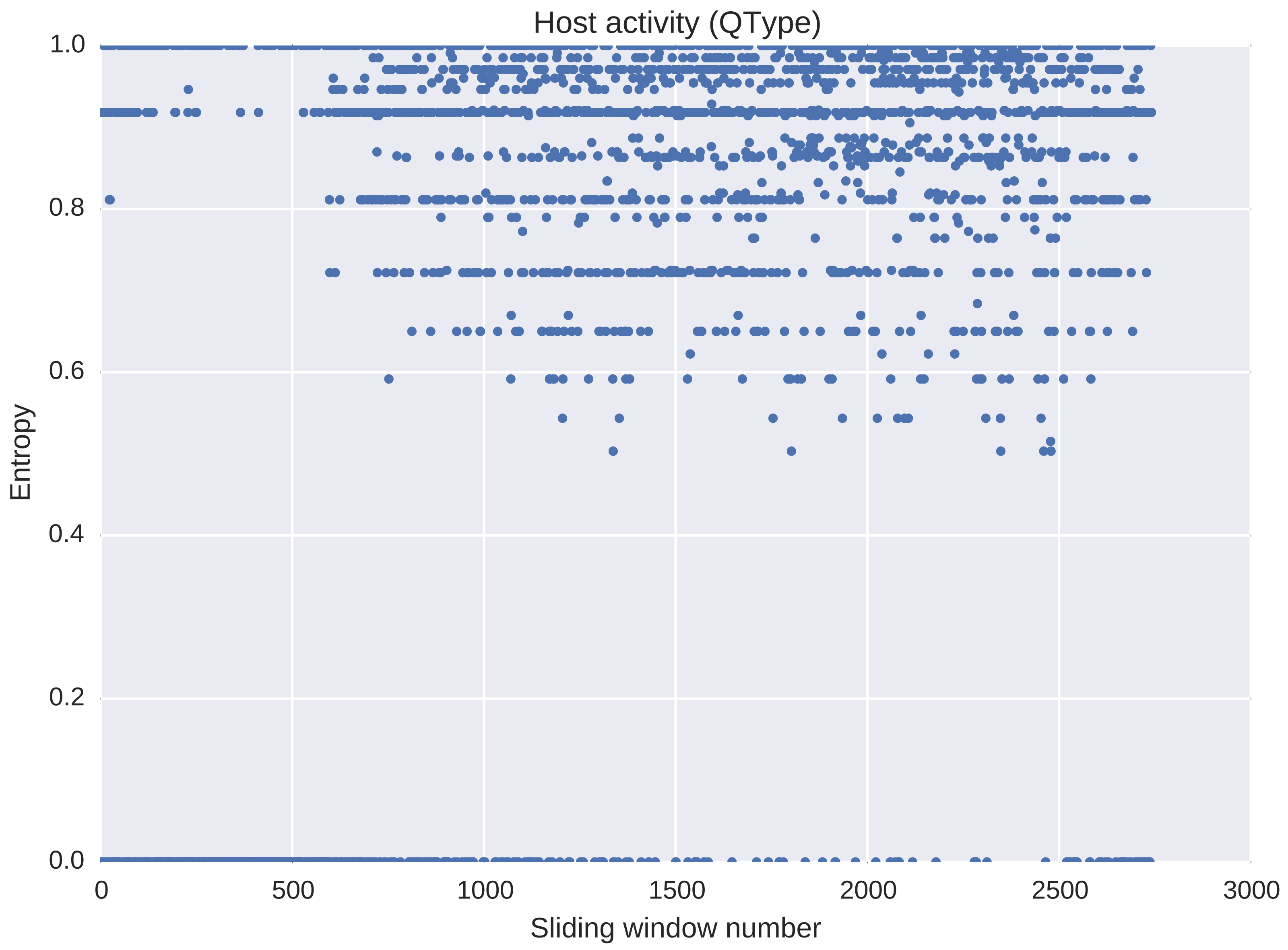

Figure 8.3: Entropy Distribution of ‘QType’ Feature (1 m, 30 s)

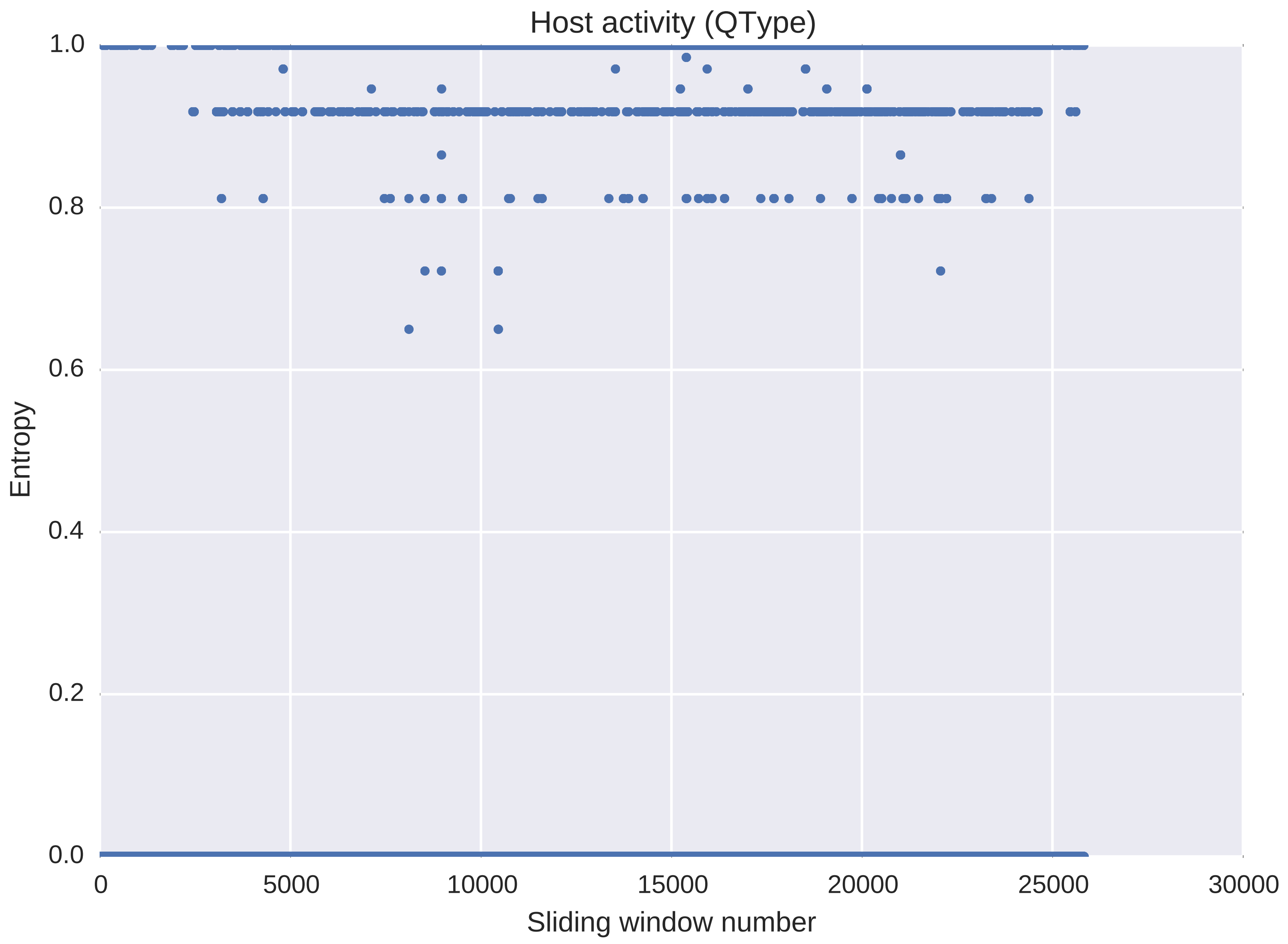

Figure 8.4: Entropy Distribution of ‘QType’ Feature (10 s, 2 s)

Parameters set 10 and 4 minutes reflects the dynamics adequately. The same parameters were used for the whole data set processing. Figures 8.5, 8.6 show the host activity 8.1 in the time-series format before and after exponential smoothing. Filtered sequences are used as input data for the clustering procedure. Filtering is used to increase the accuracy of the similarity computing routine by reducing noisy aberrations. Unsmoothed data are then used for training the neural network, which is described in the following section.

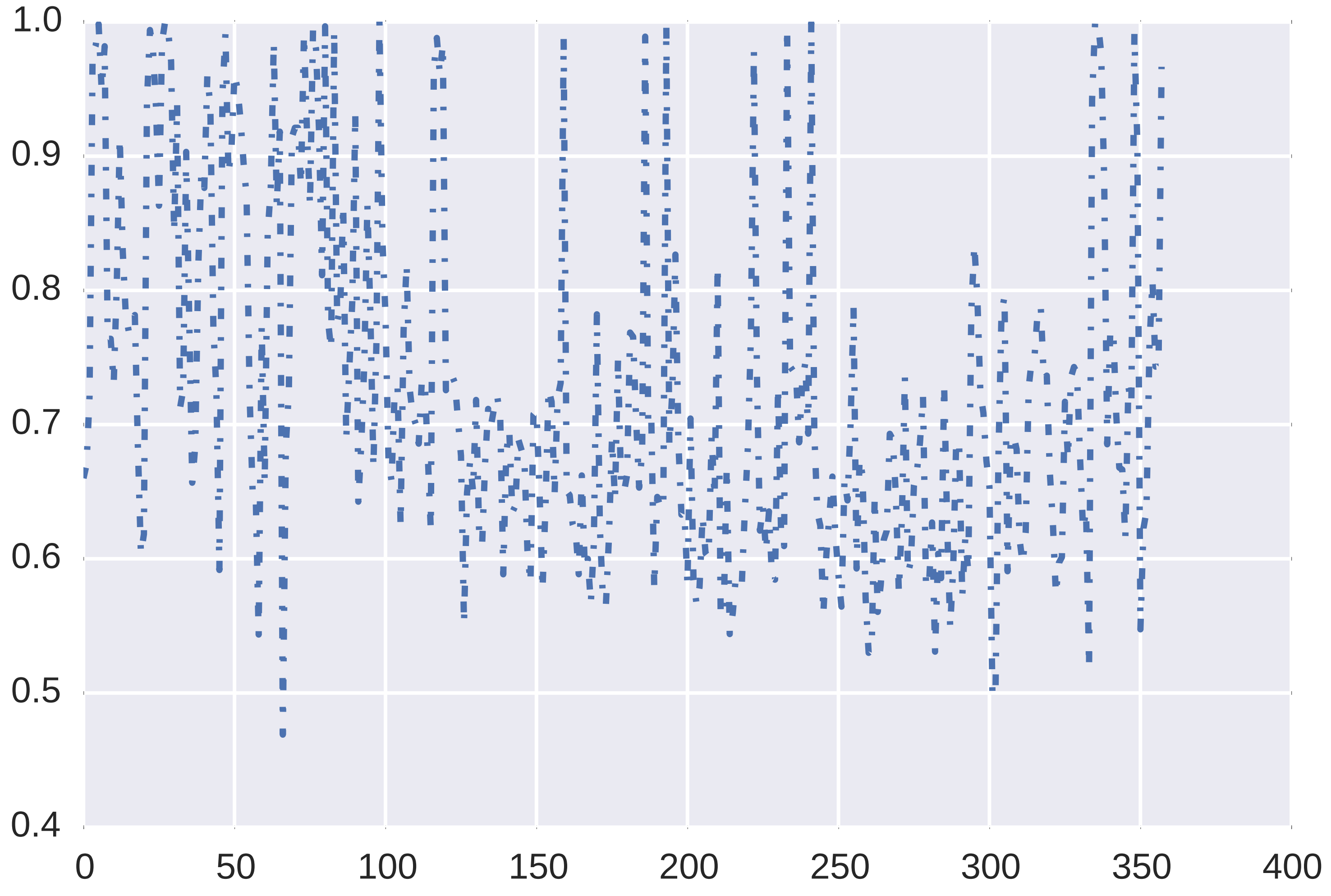

Figure 8.5: Entropy Distribution of ‘QType’ Feature (10 m, 4 m)

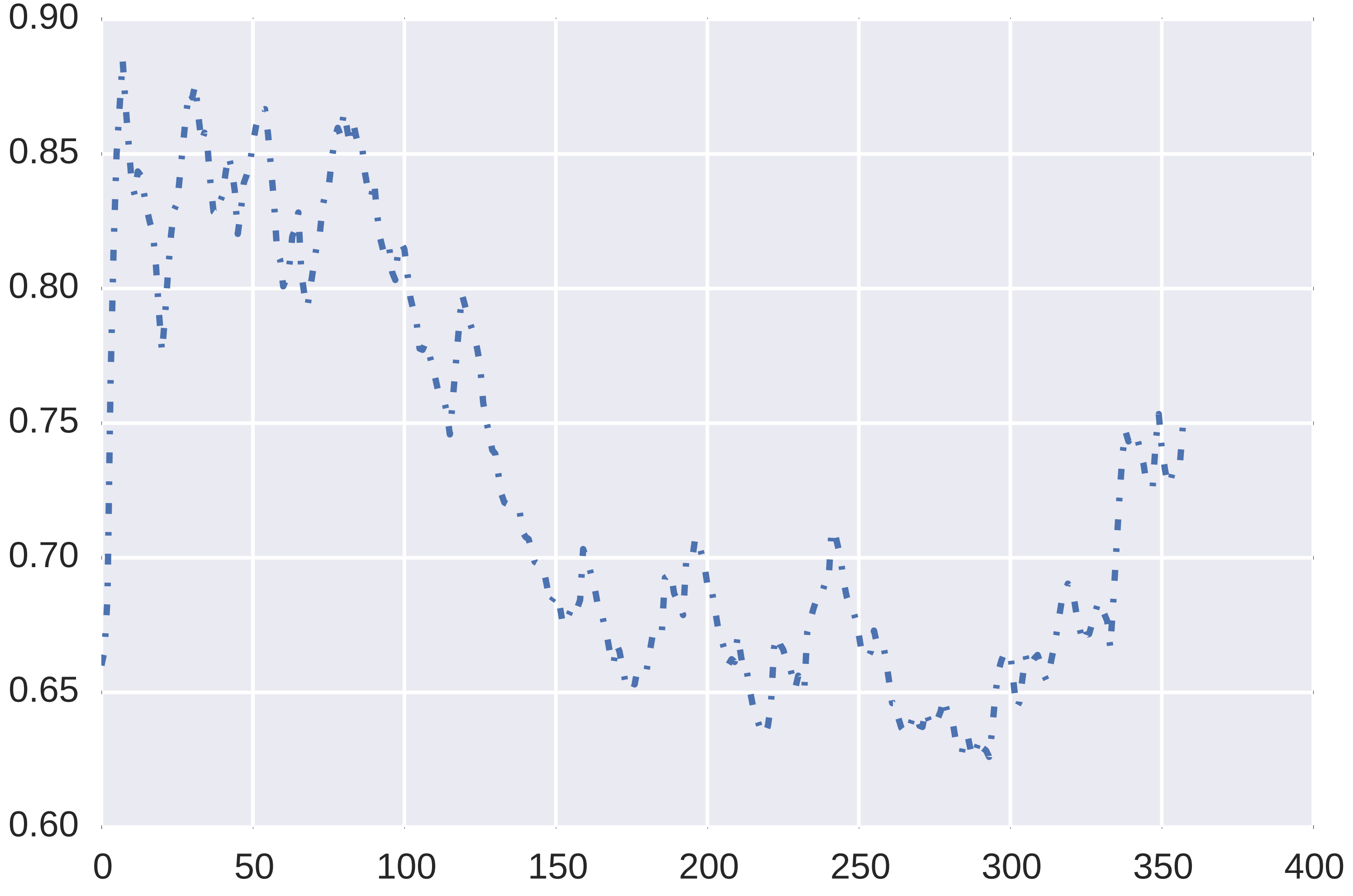

Figure 8.6: Smoothed Entropy Distribution of ‘QType’ Feature (10 m, 4 m)

8.1.2 Clustering

To decide on the number of patterns presented in the data, an agglomerative single-linkage clustering processes was performed with the use of DTW as a similarity metric. The results of clustering of the smoothed data can be found in figures 8.12, 8.13, 8.14, 8.15, 8.16. Each figure contains a dendrogram truncated at the 80th merge. The elbow method is used to determine the best number of clusters for each feature. Graphs are placed in figures 8.7, 8.8, 8.9, 8.10, 8.11. Blue line is the distance between clusters to be merged and the green line represents the sum of squared errors (SSE) between two consequent merges. The SSE in intended to be small for optimal number of clusters. However, SSE tends to decrease towards zero as the number of clusters increases. Thus the strongest elbow is a breaking point where SSE starts to decrease steadily.

Information on founded behavioural patterns is summarised in table 8.1.

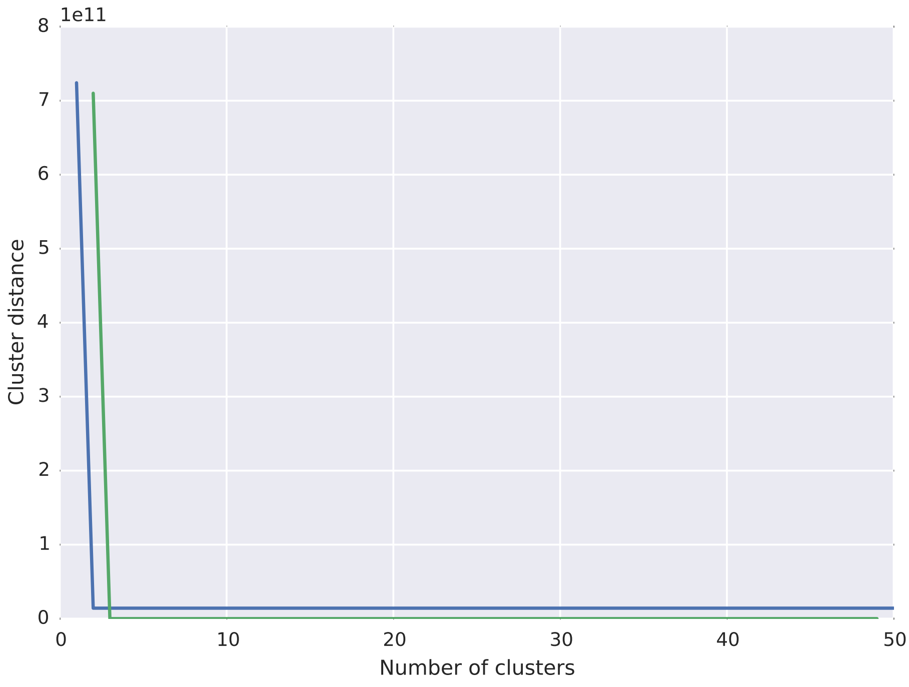

Figure 8.7: Elbow Method for ‘Flags’ Feature

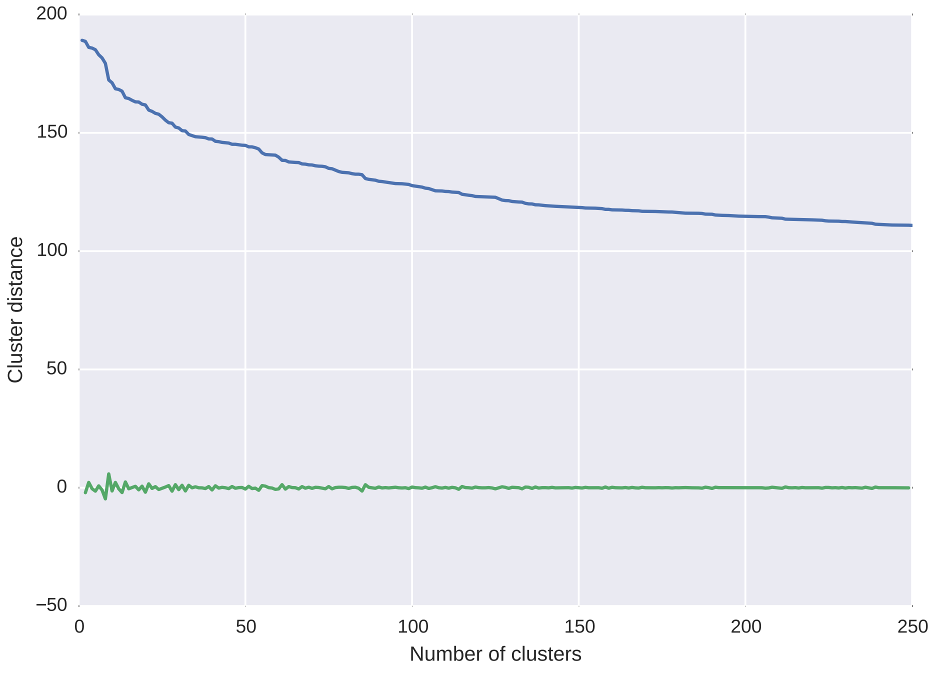

Figure 8.8: Elbow Method for ‘Id’ Feature

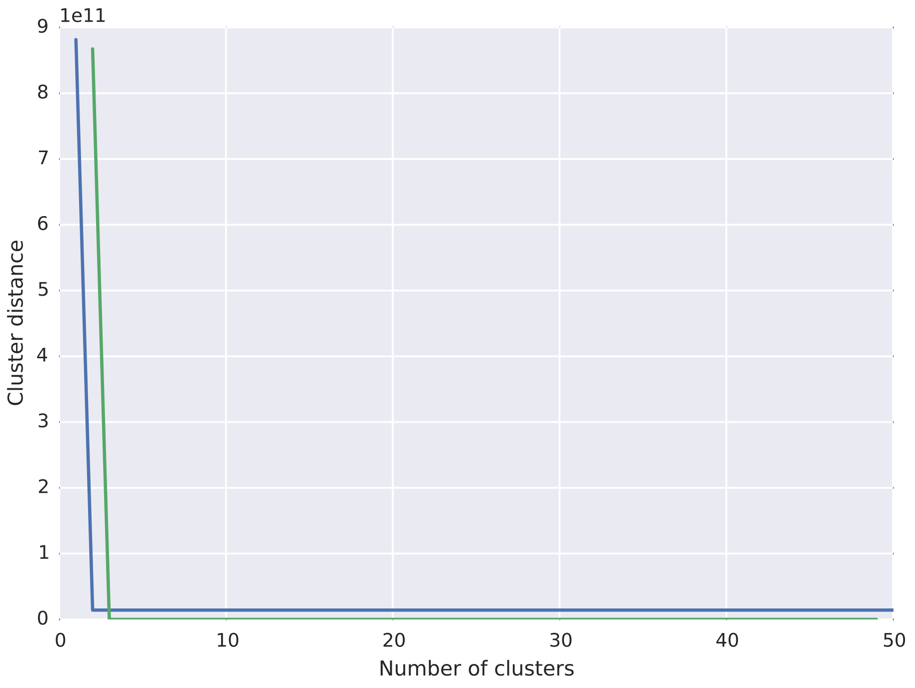

Figure 8.9: Elbow Method for ‘Port’ Feature

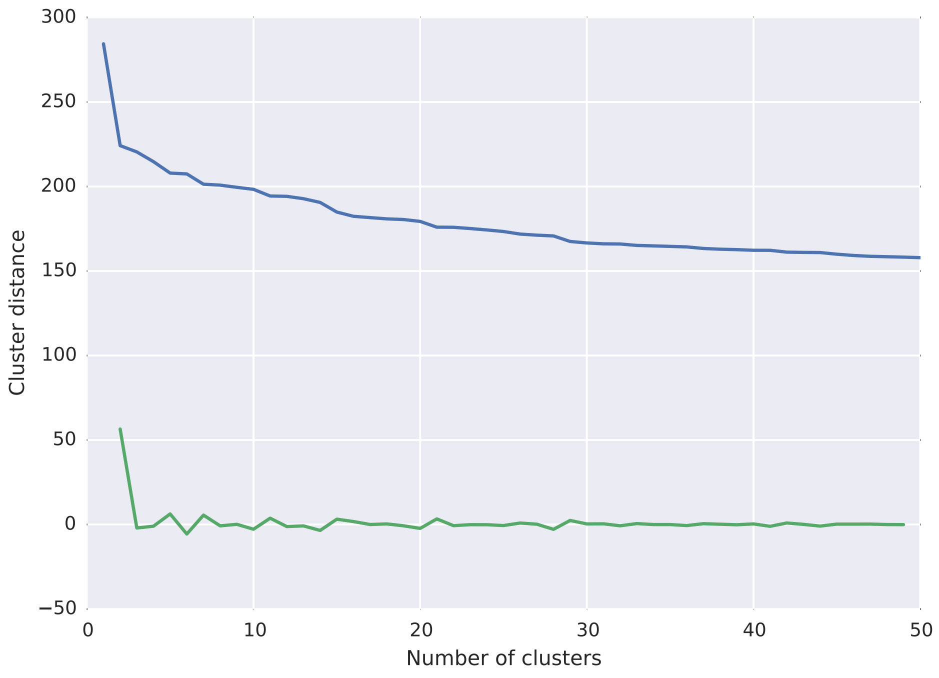

Figure 8.10: Elbow Method for ‘QName’ Feature

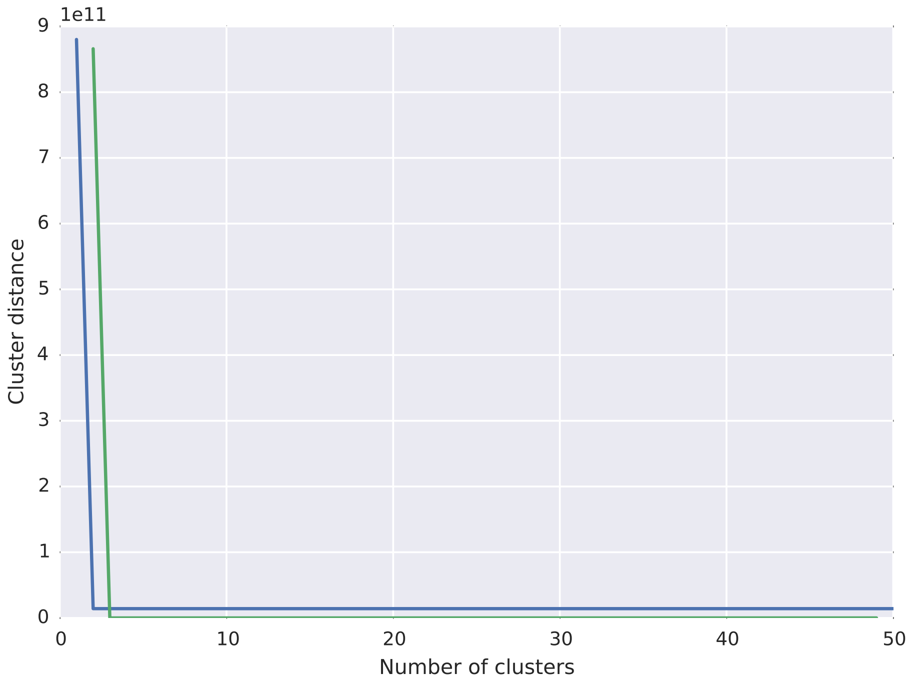

Figure 8.11: Elbow Method for ‘QType’ Feature

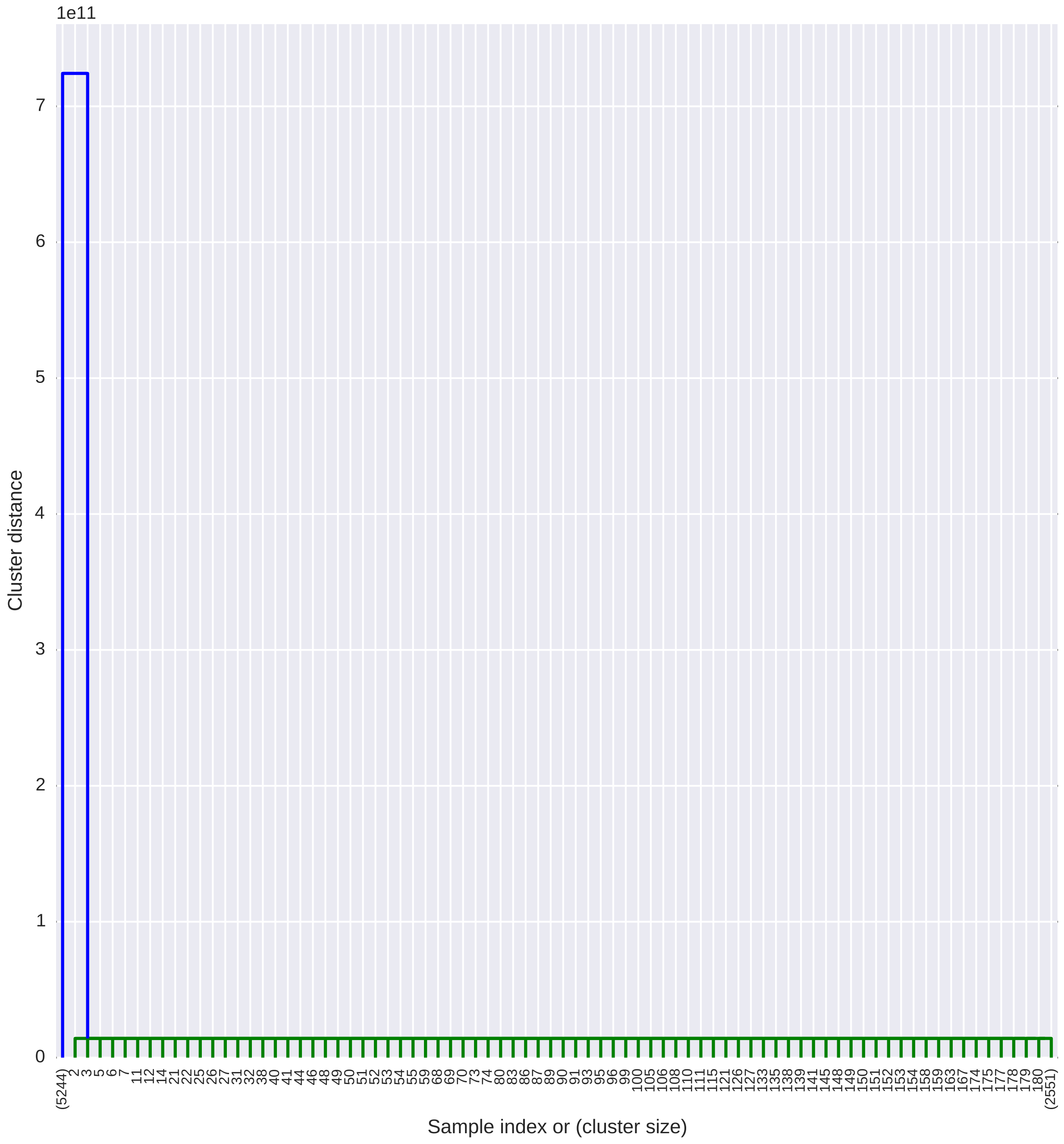

Figure 8.12: Single-linkage Clustering of ‘Flags’ Feature Dendrogram

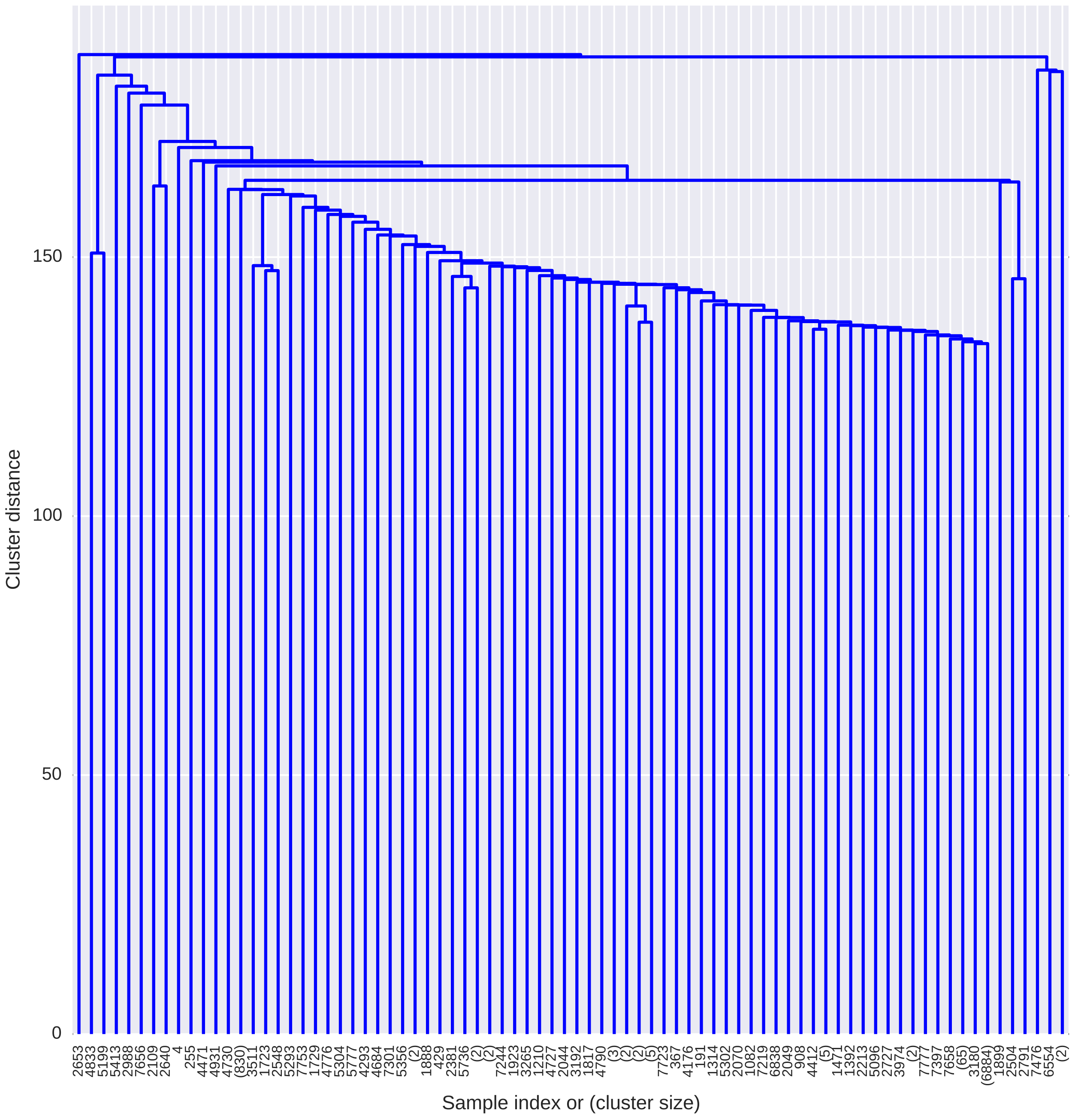

Figure 8.13: Single-linkage Clustering of ‘Id’ Feature Dendrogram

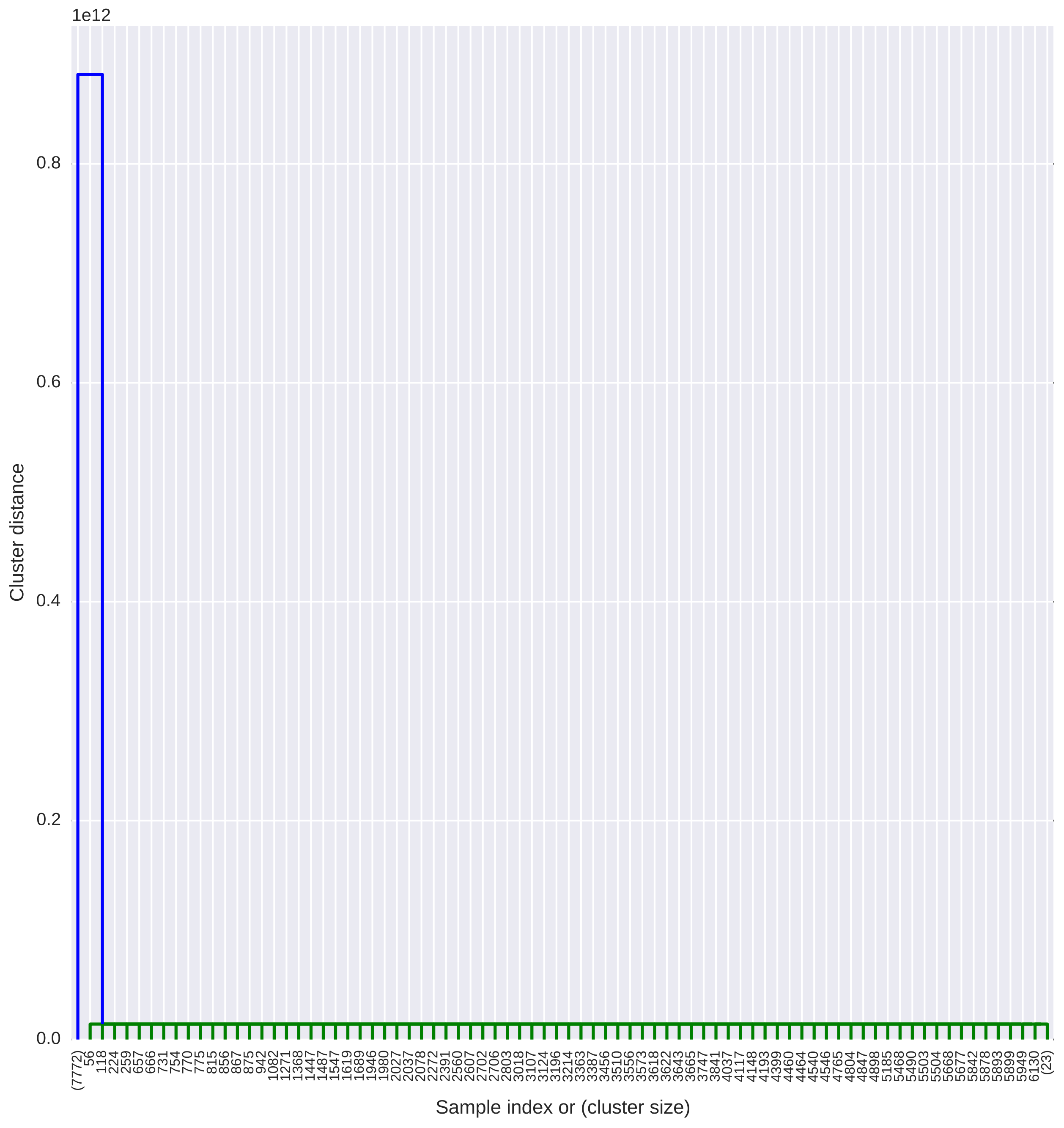

Figure 8.14: Single-linkage Clustering of ‘Port’ Feature Dendrogram

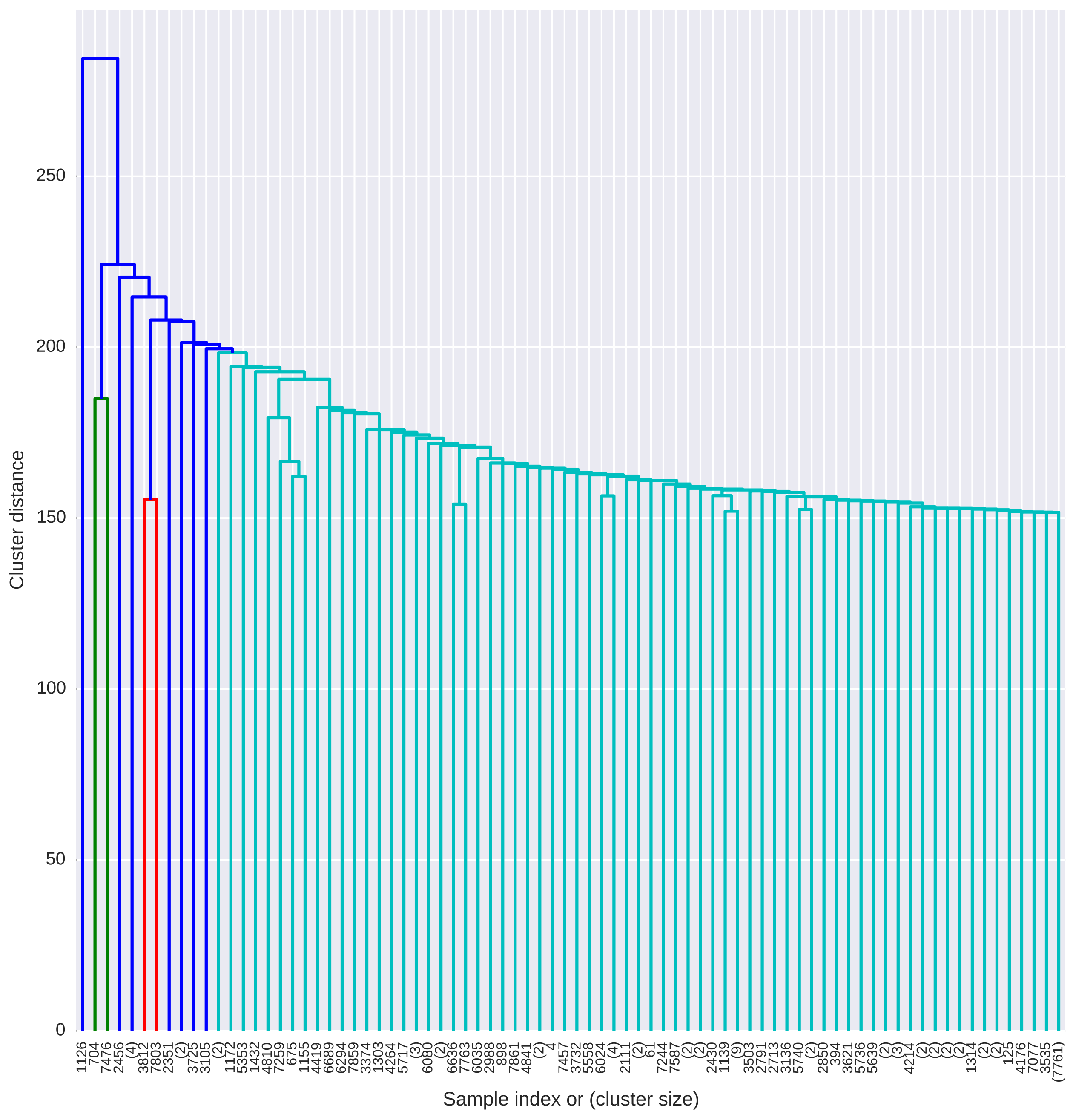

Figure 8.15: Single-linkage Clustering of 'QName' Feature Dendrogram

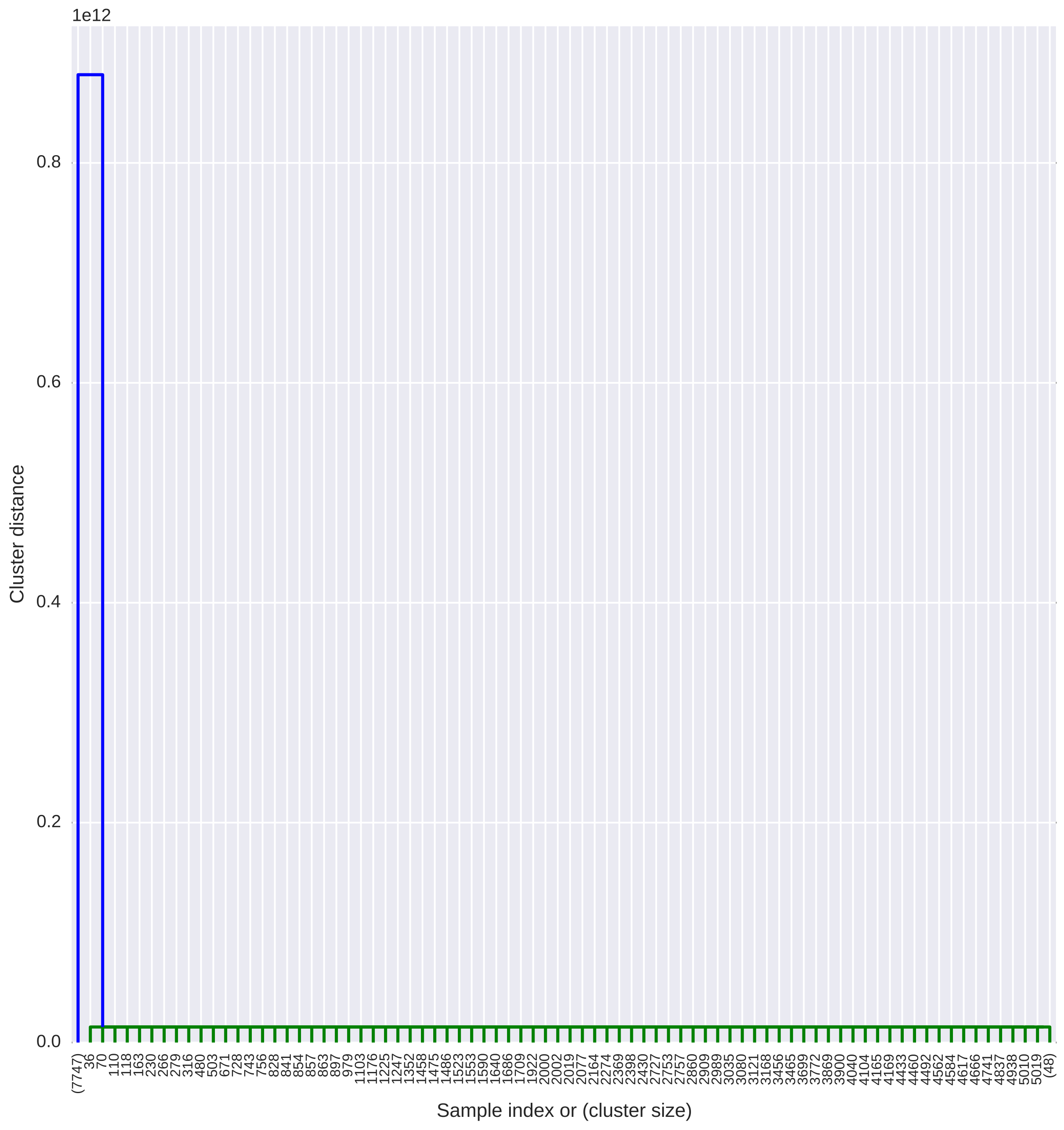

Figure 8.16: Single-linkage Clustering of ‘QType’ Feature Dendrogram

Feature	Number of clusters
Flags	2
Id	9
Port	2
QName	7
QType	2

Table 8.1: Clustering Results Table

8.2 Network Traffic Modelling

Clusters obtained in the previous step are used as labels for a preparation of datasets which are further used for training the neural network. Sets contain an equal number of samples in order to avoid biased classification. Each sample in the set is a sequence of entropy values in range $[0, 1]$ obtained as a result of the conversion of categorical features to numerical. Each sequence is cut into overlapping intervals and these intervals are passed to neural network one by one.

The training dataset consisted of smaller datasets of equal size. Each dataset represented one of the founded behavioural patterns and contained 500 labelled samples. Thus, there were 2 training datasets for ‘Flags’ feature, 9 datasets for ‘Id’ feature, 2 datasets for ‘Port’ feature, 7 datasets for ‘QName’ feature and 2 datasets for ‘QType’ feature. The whole training dataset had the size of 11000. The testing dataset contained 100 previously unseen samples of each feature. After training, the neural network was applied to testing data. The following tables represent the classification accuracy score.

Predicted as	Flags	Id	Port	QName	QType	Total predicted
Flags	84	6	2	2	1	95
Id	9	76	3	7	5	100
Port	2	3	88	3	2	98
QName	3	11	4	81	1	100
QType	2	4	3	7	91	107
Total number of samples	100	100	100	100	100	500

Table 8.2: Classification Statistics Table

Feature	Precision (%)	Recall (%)
Flags	88.42	84.00
Id	76.00	76.00
Port	89.79	88.00
QName	81.00	81.00
QType	85.04	91.00

Table 8.3: Classification Accuracy Score Table

Conclusion

The developed approach is capable of semi-supervised raw dataset mining. Original data are textual DNS packets of a monthly network traffic which were transformed to numerical sequences and separated into smaller datasets of an equal size by means of agglomerative clustering. Mined knowledge is further used for supervised training of the classifier. Bidirectional LSTM neural network was selected as a modelling and classification tool and proved to be a reliable tool for the statistical analysis of the network traffic data. Implemented modules were tested on real-life data and gained sufficiently high recognition rate.

This thesis resulted in a reliable modelling and classification system. This research is the basis for developing the behaviour-based Intrusion Detection System for network monitoring.

Bibliography

[1] Lada A. Adamic and Bernardo A. Huberman. Zipf's law and the Internet. *Glottometrics*, 3:143–150, 2002.

[2] C. Charu Aggarwal. *Similarity and Distances*, pages 63–91. Springer International Publishing, Cham, 2015.

[3] O. Al-Jarrah and A. Arafat. Network intrusion detection system using attack behavior classification. In *Information and Communication Systems (ICICS), 2014 5th International Conference on*, pages 1–6, April 2014.

[4] I Antoniou, V.V Ivanov, Valery V Ivanov, and P.V Zrelov. On the log-normal distribution of network traffic. *Physica D: Nonlinear Phenomena*, 167(1–2):72 – 85, 2002.

[5] T. Auld, A. W. Moore, and S. F. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, Jan 2007.

[6] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[8] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 5(4):455–455, 1992.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[10] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194 vol.3, 2000.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[12] Kratarth Goel and Raunaq Vohra. Learning temporal dependencies in data using a DBN-BLSTM. *CoRR*, abs/1412.6093, 2014.

[13] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, pages 5–6, 2005.

[14] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[15] Yu Gu, Andrew McCallum, and Don Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 32–32, Berkeley, CA, USA, 2005. USENIX Association.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, 1995.

[17] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350. JMLR Workshop and Conference Proceedings, 2015.

[18] N. Al Khater and R. E. Overill. Network traffic classification techniques and challenges. In *Digital Information Management (ICDIM), 2015 Tenth International Conference on*, pages 43–48, Oct 2015.

[19] Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. *CoRR*, abs/1509.01549, 2015.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[21] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[22] D. Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Netw.*, 16(10):1429–1451, December 2003.

[23] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.