Technical Report

UCAM-CL-TR-912
ISSN 1476-2986

Number 912

Network traffic classification via neural networks

Ang Kun Joo Michael, Emma Valla,
Natinael Solomon Neggatu, Andrew W. Moore

September 2017

ISSN 1476-2986

Network Traffic Classification via Neural Networks

Michael K.J. Ang Emma Valla Natinael S. Neggatu
Andrew W. Moore

Abstract

The importance of network traffic classification has grown over the last decade. Coupled with advances in software and theory, the range of classification techniques has also increased. Network operators can predict demands in future traffic to high accuracy and better identify anomalous behavior. Multiple machine learning tools have been developed in this field and each have had varying degrees of success. In this paper we use supervised machine learning within a neural network to develop a model capable of achieving high classification accuracy and maintaining low system throughput. We compare our model to previous work on Bayesian neural networks and other standard classification techniques in the context of real-time classification. The spatial and temporal stabilities of the different models will also be compared. Finally, we investigate the relationship between the convergence times of each model and the size of training dataset. Emphasis will be placed on experimental design and methodology to adequately justify and contextualize our analysis, as well as clarify the limitations of our results. Challenges in the field and areas for further work will also be discussed.

1 Introduction

Accurate identification of network traffic is an essential step to improving a multitude of network services: accounting, security monitoring, traffic forecasting and Quality of Service. However, high classification accuracies often necessitate the collection of large amounts of either data or metadata. The principal objective of this paper is the development of a lightweight model with high classification accuracy capable of real-time operation. We also investigate the spatial and temporal stability of this model and compare it to other established classification schemes. Our model is built using a perceptron network, often referred to as a neural network, implemented on the freely available Tensorflow software. Tensorflow uses optimized kernels to prune the computation graph prior to running the neural network, vastly improving computational time. A key advantage the neural network offers over traditional machine learning schemes is the introduction of non-linearity which can better capture complexities within class structures.

1.1 Background

Machine learning techniques fall into five broad categories: Classification, Clustering, Association, Numeric Prediction and Path Optimization [26, 27]. Outside of these five categories, there exist other techniques assisting in dimensionality reduction of data and anomaly detection, but these are most often used in support of the five principal objectives.

In the context of classifying network traffic flows, we are primarily interested in classification tools, although sometimes clustering or association techniques can also be adapted to perform classification. Work done by Callado et al. [3] provides a good overview of all the classification techniques currently available while Nguyen, Armitage [19] focuses specifically on those employed using machine learning.

Classification procedures are either protocol-based or statistical in nature. The former relies on heuristics derived empirically or from knowledge about known port numbers and signature protocols to assign classes. Examples of such applications include Snort [23], Bro [20] , GTVS [4] and the L-7 filter [13] which mainly use IP and port information in the TCP/UDP header and match signature strings in the payload packet to identify the flow traffic. The principal constraints of such a model are that some applications might not have a clear mapping to standard port numbers or signatures. Manual verification is often required to deal with fringe cases [4]. Advances in packet payload encryption also complicate signature matching. Overall, achieving high degrees of accuracy requires a high system throughput and is difficult to apply in real-time [13].

Machine learning uses statistical classification tools to build classification models based on pre-classified training data (ground-truth). These models can either give individual classes(C4.5 algorithm [21]) or generate probabilistic class distributions(Naive Bayes [18]) for each test datum. Unlike protocol-based methods, machine learning relies on the payloads meta-data (average segment size, graphlet of connections between the server and destination ports, etc.) to perform flow classification. Hence, when compared to protocol-based Deep Packet Inspection(DPI) methods there is always a trade-off between classification accuracy against computation time and ease of implementation. Due to their statistical nature, machine learning models also suffer from overfitting, where achieving high classification accuracy(99%-100%) often results in in low spatial and temporal stability, and the model performs poorly on data collected from other networks or from several years later.

Examples of single-class algorithms include the C4.5/J48 algorithm [21], Support Vector Machines [6] and the k-Nearest Neighbors algorithm [12]. BLINC [11] by Karagiannis et al. looks at the transport-layer interactions between hosts and users and other heuristics to classify traffic without packet payload inspection.

Algorithms generating class distributions(frequentist or Bayesian) include the Venn Probability Machine [8], k-means clustering algorithm [2], conformal predictors [8] and probability density function vectors [7]. Association algorithms(K2) can also be modified to generate class distributions [5].

$$P(c_j|y) = \frac{P(c_j)f(y|c_j)}{\sum_{c_j} P(c_j)f(y|c_j)} \tag{1}$$

Bayesian methods use Bayes rule(1) as a starting point to generate a posterior class distribution based on a prior class distribution and the observed attributes. Examples include the classic Naive Bayes method [18], Linear Discriminate Analysis, Quadratic Discriminate Analysis [25], Naive Bayes trees and Probabilistic Graphing Models [24].

One family of probabilistic algorithms of key interest to us is the neural network, or perceptron. This includes variations such as deep neural networks (multi-layered perceptron) and Bayesian neural networks [1]. Convolutional neural networks have been employed in the field of image processing, but have not yet been used to classify network traffic flow owing to the difficulty of finding attributes that are sufficiently related upon which a meaningful convolution can be generated. The neural network we use trains on class distributions to generate probability distributions on test data. However, the class with the highest probability is always chosen for classification, resulting in deterministic results for any given initial weight distribution.

1.2 Related Work

The definitive work on Bayesian neural networks in the field of network traffic classification is the 2007 paper by Auld et al. [1]. They found that an optimized Bayesian neural network could achieve up to 99.3% classification accuracy, dropping to 95.3% when tested on data from different sources. We will perform a similar analysis in this report, identifying optimal settings for our lightweight model and comparing its accuracy against other classification methods. Unlike Auld, we will construct the neural network in a frequentist setting and evaluate whether the initial choice of Bayesian/frequentist can significantly impact the output model.

The dataset we have chosen is freely available online and has been used in a number of analyses, most notably the 2005 work by Moore and Zuev [18] using the Weka software suite [27]. They concluded that the raw Naive Bayes model was not sophisticated enough to capture the relationship between the flow characteristics and classes. An additional kernel density estimation was required to improve accuracy to 93.5%. An optional Fast Correlation-Based Filter could also be applied to shrink the attribute space from 248 characteristics to only 11.

Efforts have also been made to find a lightweight classification model built using only Netflow records, 12 features from the flow records extracted using the Netflow software package [10]. Their model was designed to operate in real-time, with significantly reduced computing overhead. However, they only achieved 88.3% classification accuracy once again using the Naive Bayes classification with a kernel density estimation. This once again suggests that Naive Bayes methods are insufficiently robust and do not always perform well when restricted to fewer flow characteristics.

Using the same dataset as Moore and Zuev in 2005 [18], Li et al. [14] compared the spatial and temporal stability of the C4.5 decision tree and the Naive Bayes methods. They discovered that the C4.5 consistently outperformed the Naive Bayes, although spatial changes in the site would still result in a loss of between 4-5% of classification accuracy. This effect was more pronounced when split into classes, with low precision and/or recall values in the BULK, MULTIMEDIA, SERVICES and INTERACTIVE classes. The temporal loss affected the DATABASE, INTERACTIVE, SERVICES and BULK classes.

2 Experimental Setup

2.1 Data Collection

Throughout this paper we used data collected by the high-performance network monitor described in [15]. This dataset has also been used in [1, 14, 18]. The data was captured

Classification	Example Application
BULK	ftp-control, ftp-pasv, ftp-data
DATABASE	postgres, sqlnet oracle, ingres
INTERACTIVE	ssh klogin, rlogin, telnet
MAIL	imap, pop2/3, smtp
SERVICES	X11, dns ident, ldap, ntp
WWW	www
P2P	KaZaA, BitTorrent, GnuTella
ATTACK	Internet worm and virus attacks
GAMES	Microsoft Direct Play
MULTIMEDIA	Windows Media Player, Real

Table 1: Network traffic flow classes

using its loss-limited, full-payload capture to disk facility with time stamps of resolution better than 35 nanoseconds.

We examine data for several different periods of time(non-overlapping samples of similar duration spaced randomly throughout a 24-hour period) from the hosts of several Biology-related facilities, collectively referred to as a *Genome Campus*. There are three institutions on site and about 1000 researchers, administrators and technical staff(users). The campus is connected to the internet by a full-duplex Gigabit Ethernet link. Bi-directional traffic on this connection was monitored for each traffic set.

2.2 Classification and Discriminators

From our network traffic we split the data packets into flows consisting of one or more packets between a pair of hosts. We consider exclusively TCP/IP traffic flows, defined by a tuple consisting of the IP address of the pair of hosts and the TCP port numbers used by the server and the host. In this paper we limit ourselves to use only semantically complete TCP flows. These are flows for which a complete connection setup and tear-down were observed. This simplification allows us to focus on the classification process by having an identical set of attributes for all flows. Although the results do not necessarily generalize to UDP or incomplete TCP flows, the methodologies can easily be reapplied in those environments using different sets of discriminating attributes.

We identified ten common groups of network-based applications [Table 1] and listed a few common examples from each group belonging to the dataset. This non-exhaustive list is a reflection of the common applications at the time the data was collected. A key point to note is that the BULK class is classified into FTP-CTRL, FTP-PASV and FTP-DATA in the *Genome Campus* data. We are thus dealing with 12 classes instead of the original 10 as used in [1, 10, 14, 18].

Each flow also has an associated set of features, sometimes referred to as discriminators or attributes. We provide a small sample of our chosen discriminators in Table 2 and a full description can be found in [16].

Features
Flow metrics (total bytes, inter-arrival times, total packets)
Metrics of SACK packets, SYN packets, FIN packets, URG packets
Segment and window advertisement sizes
Missed or truncated data
Payload sizes (mean, variance, 1st and 3rd quartiles, min, mx)
Round-Trip Time(RTT) statistics
FFT of packet IATs (arctan of top ten frequencies)

Table 2: Examples of discriminators

2.3 Ground-Truth Data Classification

To implement supervised machine learning, we require our training data to be correctly pre-classified. This classification is also referred to as the ground truth.

Our ground-truth classification is a manual process that consists of identifying classes by inspecting the data content of flows. The membership of each flow is then decided based on both the contents and knowledge about the systems exchanged in the flow. This process of combining host knowledge, data signature matching and content verification allows the original data to be classified by hand with high accuracy. Further details of the process can be found in [17] and the method is developed and automated in [4].

3 Model Optimization

3.1 Neural Network Architecture

Our classification method of choice is the neural network, specifically the multilayered perceptron. It is important to note that previous work has been done, in a Bayesian setting, in [1] to classify network data collected from the *Genome Campus* using a multilayer perceptron. We will perform a similar analysis in a frequentist setting and compare the performances of the optimized models.

Figure 1 shows the network architecture of a multilayer perceptron network with one hidden layer. The first layer contains the inputs, the 248 discriminators x_i described in [16], with an additional bias node of constant value ($x_0 = 1$). The final output layer consists of twelve nodes corresponding to the different classes of membership. Between the input and output layers, there can be any number of hidden layers each containing any number of nodes. Each node is connected to all nodes in the layer above it via weighted connections w_{ij}. The neural network can then be characterized by $\mathbf{w} = \{w_{ij}\}$. As information is fed in from the input layer, the values of the nodes in the hidden layer are calculated as $y_j = g(\sum_i w_{ij} x_i)$, where g is a predefined function known as the activation function. This transformation is iterated from layer to layer until we reach the final layer where a *Softmax* function is applied. The probabilities associated with each class are calculated by $p_j = \dfrac{exp(y_j)}{\sum_k exp(y_k)}$. Test data can then be classified at this stage by assigning the class with the highest probability. Otherwise, the system optimizes the probability distributions via a system of backpropagation. The gradients of the weights

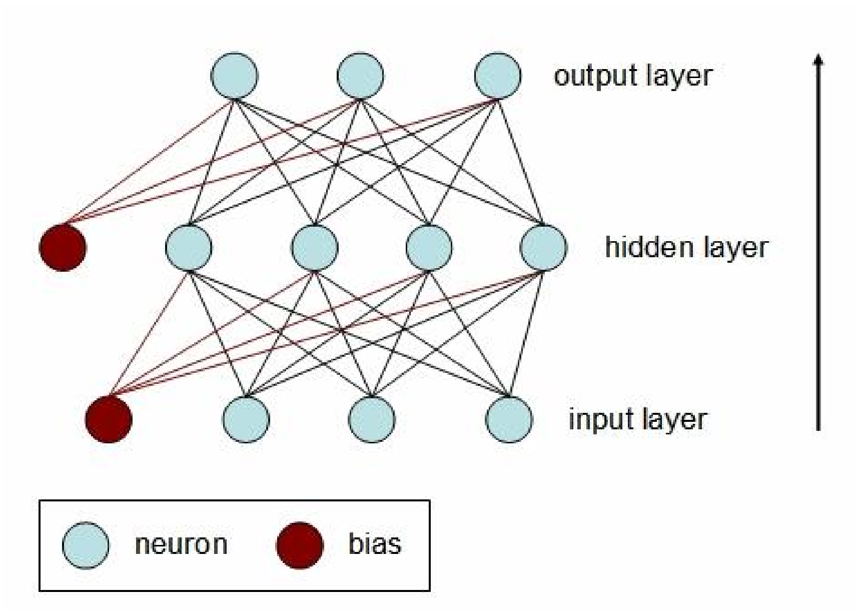


Figure 1: Perceptron network with one hidden layer

are calculated and the system modifies the weights to more closely reflect the ground truth. Further details on the specifics of this method can be found in [27].

Construction of a neural network involves several critical decisions:

- **Number of hidden layers**
- **Number of nodes in each hidden layer**
- **Choice of activation function**
- **Number of iterations of backpropagation applied**
- **Number of input attributes**

In this paper we attempt to maximize classification accuracy by finding an optimal combination of the above parameters.

3.2 Optimization Methodology

A guaranteed way to capture the optimal configuration of the neural network is to test every combination of attributes, iterations, nodes and functions. In practice, this is often infeasible due to the sheer number of possible configurations. We must therefore find an effective way to shrink the parameter space without severely compromising classification accuracy.

We carry out the optimization process by first making an informed choice of initial parameters to load our neural network with. Under these settings, we start with the full set of input attributes and reduce them to a number small enough to be collected in real-time. This is to ensure the classification process will not be bogged down by large amounts of data pre-processing. Next, we keep the number of iterations constant and optimize the number of nodes and activation function. Finally, we perform a stability analysis on the number of training iterations. Note that this step-by-step optimization method is not guaranteed to converge to the global maximum (or even a local maximum). However, provided we choose our initial parameters well, it provides a systematic and computationally efficient way of generating a good configuration. As demonstrated later in this paper, very high (>99%) classification accuracies can still be achieved.

In work done by Auld et al. [1], the authors reached a high classification accuracy using only a single hidden layer. Indeed, the Universal Approximation Theorem [9] states that any multilayered feed-forward network can always be approximated(to arbitrary accuracy) by a single hidden layer with a finite number of nodes. Auld plots the error rate against the number of nodes and notices the error rate stabilizes starting from 7 nodes. Additional nodes increase the computation time without appreciable changes in accuracy. It is not unreasonable to expect behaviour similar to this in the frequentist setting. Auld eventually settled on the choice of 10 nodes but we will be using 12. This can also be justified by the presence of 12 classes in the problem, since the number of nodes provides the dimension of the hidden space, the non-linear projection of the input space via the map created by the weights and the activation function. A dimension greater or equal to the number of classes could provide the necessary complexity to accurately classify all classes of traffic.

Auld [1] achieved high levels of accuracy using a tanh activation function, which we also use in our initial parametrization. Further on in the report, we test the performance of the tanh function against other common activation functions.

3.2.1 Handling Nominal Attributes

One problem faced by neural networks is the proper handling of nominal attributes. In our case, this refers to the server and client port numbers. Port numbers are nominal attributes in the sense that they are not numeric. Port numbers that are close together in integer value can have almost no correlation. There is no reason port 80 might be more closely related to port 20 rather than port 1280. We thus need to find some mapping of port numbers into attributes that make sense to the neural network.

Auld et al. [1] chose not to work with port numbers so as to provide a wider application of the classification technique. Our neural network will not have any such restriction. Port numbers have been shown in [18] to have high predictive power on their own and can be obtained from flow data with little to no overhead. Our objectives are classification speed and accuracy, thus we cannot choose to ignore such an important attribute.

We propose two techniques used for managing nominal attributes. The first is a modified application of Auld's method of normalizing numeric attributes, histogram equalization. It transforms the port numbers into the range $[0, 1]$ and smooths out large discontinuities. This technique is most commonly used to improve image contrast, but the drawback here is that the data is still constrained to one dimension. If we had multiple unique ports each performing a unique function, a linear weight function might not be comprehensive enough to capture the intricacies of the relationship.

The second technique is known as one-hot embedding. We rank the port numbers by their frequency in the training data, then select the most popular port numbers that cover most(>95%) of the flows. New attributes are created for those ports and each attribute is assigned the value 1 for a matching port number and 0 otherwise. Port numbers which do not appear on the list are assigned a value of 0 for all the new attributes.

This technique has two distinct advantages. It incorporates multidimensionality otherwise absent in the histogram equalization. By assigning attributes to only the most popular ports, the model also learns how to classify data for uncommon ports using other attributes. This provides sufficient training for the model should it encounter flows from a new port. In the histogram equalization, we might perhaps opt to assign the values of the new ports to 0 by default, but this results in a system with no experience training on such data. In the one-hot embedding, the reliability of the port number attribute is intricately tied to its observed frequency and the system is given sufficient experience training on 'uncommon' port numbers.

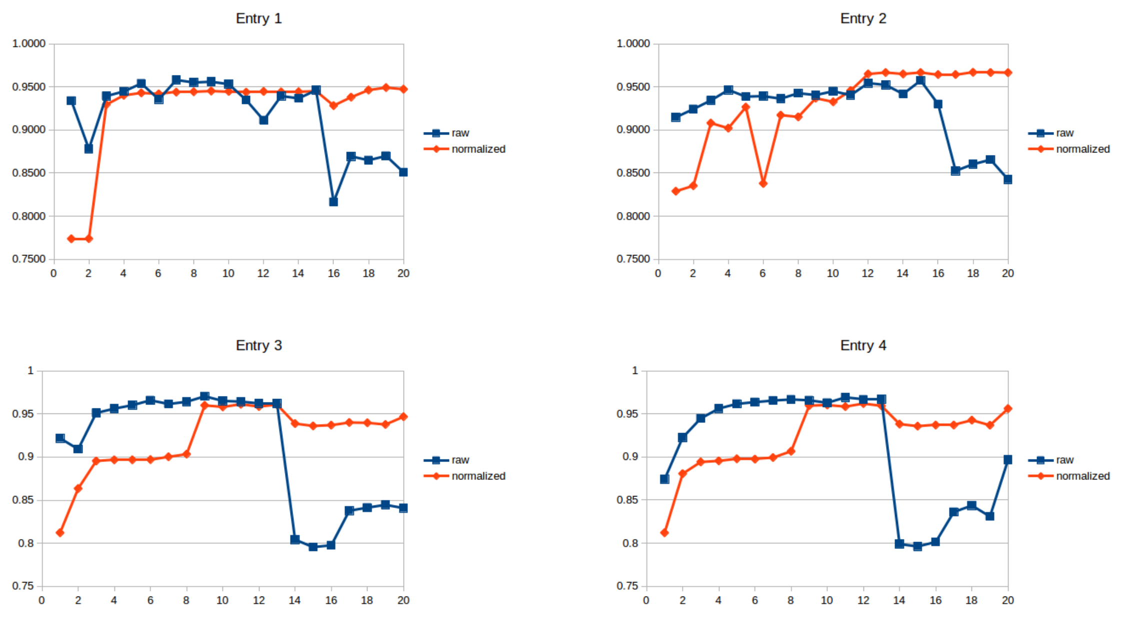
3.3 Attribute Space Reduction

3.3.1 Symmetric Uncertainty

We explore the possibility of reducing the discriminator space via two methods. The first is via Symmetric Uncertainty(SU), similar to the method employed by Moore and Zuev in [18]. Here we will be using the modified histogram equalization method, because its management of nominal attributes (as a single-dimensional attribute) more closely mimics the work in [18].

After processing all the data, we start with the data collected from a single period and rank the 248 attributes based on their ability to discriminate between the 12 classes. Using only the top 20 attributes, we performed stratified ten-fold cross-validation (training the network on $\frac{9}{10}$ of the data and testing on the remaining $\frac{1}{10}$). Details on symmetric uncertainty ranking and stratified cross-validation can be found in [27]. We repeated the experiment with the top 19,18...1 attributes before repeating the entire procedure with 9 other datasets collected from different periods.

In Figure 6 we compare the accuracy of the model using both the raw data and normalised data. The normalisation process involves linearly rescaling the numeric attributes and histogram equalization for the port numbers. This transforms all the data to lie in



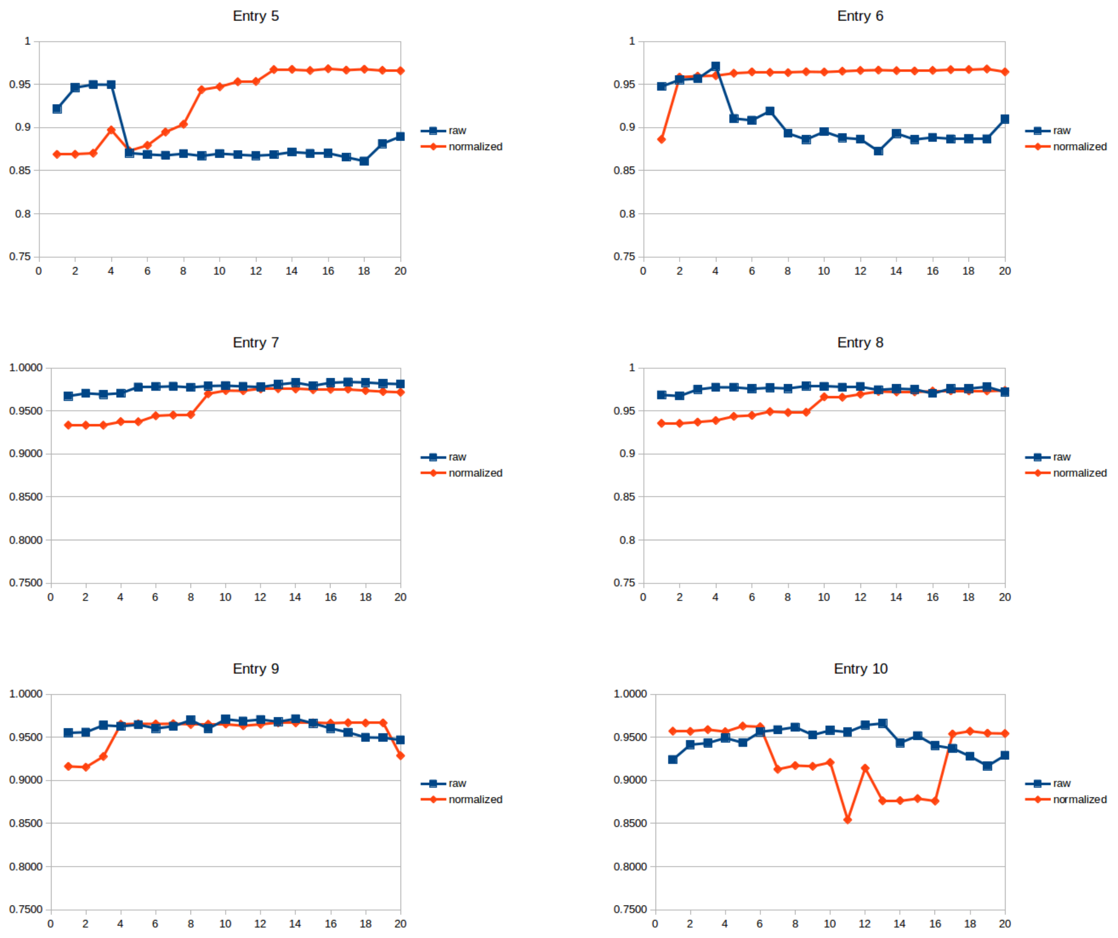


Figure 6: Accuracies against number of discriminators used, raw and normalised data, 10 datasets

the interval $[0, 1]$ without compromising the ratio of distances between data points.

In theory, our rescaling will not affect the predictive power of numeric attributes, as the smaller scale can always be compensated by larger weights. However, the histogram equalization of nominal attributes removes the bias created by large port numbers and this is evident in the above graphs. The performance of the normalised data is at least as good as the raw data as the number of attributes increases. Although we lack the multidimensionality offered by one-hot embedding, histogram equalization remains a viable transformation for nominal attributes in the context of neural networks. The normalised data has an optimal accuracy in the range of $94.9\% - 97.6\%$ but this only occurs when the right number of attributes are chosen. The presence of redundant attributes can reduce accuracy to as low as 80%.

To create a reduced list of discriminators we separated discriminators into 'effective' and 'ineffective'. Discriminators were effective if they appeared at peak accuracy(but before the plateau) and ineffective otherwise. A discriminator had to be effective in at least half the training sets to be added to the reduced list.

As expected, there is significant overlap between these attributes and the 11 discriminiators identified in [18], since both techniques employ Symmetric Uncertainty to sieve out weaker attributes. To further reduce the attribute space, we could also restrict our-

List of Attributes
Port number *server*
Number of bytes in initial window *client→server*
Total number of data packets *client→server*
Number of data packets with PUSH bit in TCP header *client→server*
Number of data packets with PUSH bit in TCP header *server→client*
Total number of Round-Trip Time(RTT) samples *client→server*
Number of segments cumulatively acknowledged *client→server*
Number of bytes in initial window *server→client*
Minimum segment size *client→server*
Median of total bytes in each IP packet *client→server*

Table 3: Reduced list of attributes as obtained by Symmetric Uncertainty

selves to discriminators effective in at least 70% of training sets, corresponding to only the first 5 attributes on the list. However, this is ill-advised as the neural network's performance tends to decay when working with fewer attributes, as evidenced by the above data. Using the 10 identified key attributes, we estimate a network accuracy of between 94% – 97% when testing on new data.

3.3.2 Reduction by Weights

A technique which is more organic to the neural network's inherent structure is a reduction by weights. We run the neural network for a large number of iterations and terminate when the weights stabilize. We then examine the weights of the connections emnating from each discriminator and eliminate the discriminators providing smaller contributions to the hidden layer.

Before the technique can be implemented, we need to normalize the data. Unlike the Symmetric Uncertainty method, this normalization is mandatory as we will be pruning attributes based on their connection weights. The attributes will thus have to be on a similar scale to make an equal comparison. The numeric attributes are rescaled as in the SU reduction and the port numbers are transformed using one-hot embedding on the 13 most popular ports.

We trained the neural network on data collected from 9 periods and tested it on data from a single period. After approximately 5000 iterations the accuracy plateaued at approximately 99.0% and the weights stabilized [Figure 7]. We then ranked the discriminators by the largest magnitude of their 12 weighted connections.

Considering port number as a single discriminator, we constructed a lightweight model using just the top 20 attributes and tested its efficiency against the full model, both run for 5000 steps. Finally, we repeated the experiment using the top 15, 10, 5, 1 attributes.

No. of attributes	Top 1	Top 5	Top 10	Top 15	Top 20	All
Accuracy(%)	98.63	98.74	99.11	98.30	98.78	98.70

Table 4: Total flow accuracy for different numbers of attributes

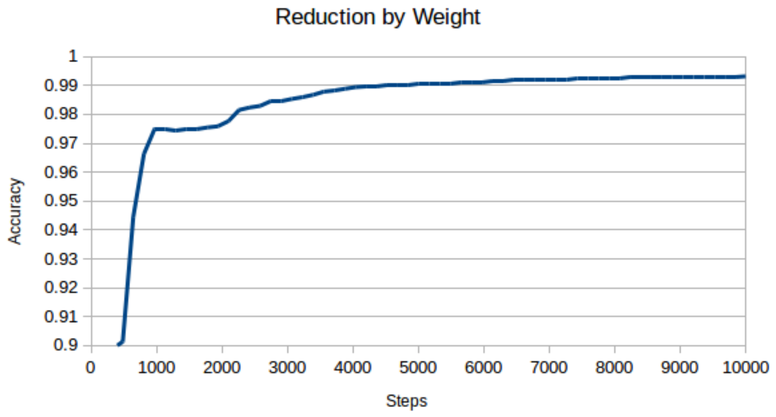


Figure 7: Accuracy of network trained on full attribute set (one-hot embedding for port number)

Accuracy. This is the total number of correctly classified flows divided by the total number of classified instances.

Precision. A per-class measure, this is the total number of flows assigned to and genuinely belonging to some particular class divided by the total number of flows assigned to that class.

Recall. This per-class measure counts the number of correctly classified flows of a particular class divided by the true number of flows belonging to that class.

Precision(%)	Top 1	Top 5	Top 10	Top 15	Top 20	All
WWW	99.75	99.75	99.75	99.75	99.74	99.68
MAIL	99.95	99.95	99.95	99.19	99.88	99.81
FTP-CTRL	100	100	99.33	100	100	98.68
FTP-PASV	N/A	26.28	25.47	14.39	20.40	18.93
FTP-DATA	82.13	89.94	96.52	93.62	94.29	97.33
P2P	97.16	98.04	98.23	96.53	96.32	84.15
DATABASE	100	99.58	100	98.76	98.76	98.31
ATTACK	100	100	100	97.14	100	89.47
MULTIMEDIA	98.78	100	98.53	100	98.72	100
SERVICES	100	100	100	100	100	100
INTERACTIVE	N/A	N/A	N/A	N/A	N/A	N/A
GAMES	N/A	N/A	N/A	N/A	N/A	N/A

Table 5: Per-class precision for different numbers of attributes

There are a few caveats to the experimental design which provide context for interpreting our results. Our goal is to create a model running on only a few attributes, capable of in real-time classification whilst retaining a suitably high accuracy. In practice, however, it is difficult to know which attributes to select. Choosing 20 attributes at random will likely result in a low accuracy and training on all combinations of 20 attributes is computationally infeasible.

Our devised ranking scheme gets around this complication. Because all attribute input

Recall(%)	Top 1	Top 5	Top 10	Top 15	Top 20	All
WWW	99.65	99.65	99.65	99.65	99.67	99.67
MAIL	99.98	99.98	99.98	99.98	99.98	99.98
FTP-CTRL	100	100	100	100	100	100
FTP-PASV	0	95.35	95.35	95.35	95.35	90.70
FTP-DATA	100	98.94	98.94	98.94	98.94	99.47
P2P	50.44	73.75	81.71	41	53.98	61.06
DATABASE	100	100	100	100	100	97.9
ATTACK	55.74	55.74	55.74	55.74	55.74	55.74
MULTIMEDIA	93.1	1.15	77.01	4.6	88.51	37.93
SERVICES	99.03	99.03	99.03	99.03	99.03	99.03
INTERACTIVE	0	0	0	0	0	0
GAMES	N/A	N/A	N/A	N/A	N/A	N/A

Table 6: Per-class recall for different numbers of attributes

lie in the range $[0, 1]$ after rescaling, if we assume each attribute to be sufficiently normally distributed, all input data will follow a similar scale. Thus, we can expect attributes with branches of small weights to have less impact on the hidden layer. The ranking scheme then sorts attributes by the weight of their most significant branch.

Whilst this scheme is not guaranteed to identify the optimal combination of attributes, it is computationally inexpensive and scales quickly to any number of attributes. Although the scheme stems from the model's inherent structure, it cannot be claimed to be mathematically rigorous - the impact of each attribute will depend on the combination of inputs as much as the weights themselves. However, the attributes chosen by this scheme perform exceptionally well in the neural network. Much like the SU method in Moore and Zuev's 2005 work [18], the results justify the selection. All our models produced using this selection method achieve at least 98% accuracy [Table 7].

The next hurdle comes from deciding how to compare different models. One option is to run each model until their accuracies stabilize, but as we can see in Figure 7, this is dependent on our definition of 'stabilized'. Although the accuracy tapers at 5000 steps, an cutoff at 10000 steps generates a marginal 0.3% increase in accuracy at 99%. To make the accuracies in Table 4 somewhat comparable, we trained all models for exactly 5000 steps. It is a reasoned assumption that lighter models running on fewer attributes will have also stabilized by that point, since typically greater degrees of freedom of the model require more steps to convergence.

As seen in Table 4, the overall accuracy does not diminish significantly using fewer attributes. Using only port number, it appears we can still achieve >98% accuracy, but this statistic is misleading. Around 87% of our traffic flows are web browser queries(WWW) and a further 7.5% are from emails(MAIL). The two classes alone constitute 95% of all internet traffic, and network traffic on most servers will follow a similar uneven distribution. Accuracy alone is an insufficient indicator of model performance. Models with high classification accuracy can still perform very poorly on the more infrequent classes. We need to employ a class-based metric for a more robust comparison.

In Table 5, the precision values are similar in all models for the classes WWW, MAIL, FTP-CTRL, DATABASE, MULTIMEDIA and SERVICES while the P2P and ATTACK classes have higher precision under the lighter models. This noise reduction from the smaller models provides good justification for reducing the number of attributes.

In Table 6, we assess each model's ability to identify flows belonging to specific classes. Using only port number, the neural network cannot properly identify FTP-PASV and P2P flows. When increased to 5 attributes, the MULTIMEDIA recall suffers. The 10-attribute model is the optimal compromise between model reduction and classification recall. Conveniently, it also has the highest classification accuracy at 99.11% [Table 4].

List of Attributes
Port number *server*
Minimum segment size *client→server*
First quartile of number of control bytes in each packet *client→server*
Maximum number of bytes in IP packets *server→client*
Maximum number of bytes in Ethernet package *server→client*
Maximum segment size *server→client*
Mean segment size *server→client*
Median number of control bytes in each packet *bidirectional*
Number of bytes sent in initial window *client→server*
Minimum segment size *server→client*

Table 7: Top 10 attributes as determined by connection weights

The top 10 attributes as identified by our ranking scheme are presented below in Table 7. The number of bytes in each packet, with or without headers, plays a significant role in class determination. There is no overlap with the 20 key attributes identified by the Bayesian neural network [1], which are mostly unrelated. The difference is explained by the absence of port number, the attribute determined by SU ranking to have the highest discriminating power. With port number, our model can afford to run on fewer attributes and its presence changes the type of metadata required for optimization. Auld et al. constructed their Bayesian model to operate in the absence of port information, but we did not be apply this constraint, given the port number's high utility and ease of extraction from the flow data.

3.4 Network Parameter Configuration

3.4.1 Node Number and Activation Function

We proceed to optimize the number of nodes in the hidden layer and the choice of activation function simultaneously. Figure 8 displays the classification accuracy of the neural network trained for 5000 steps on the 10 attributes identified in Section 3.3.2 using increasing numbers of nodes in the hidden layer. The performances of the three most commonly used activation functions, tanh, sigmoid and ReLU are compared.

The sigmoid activation function consistently performs worse than the tanh and ReLU functions, which have negligible performance differences. Their classification accuracy plateaus after approximately 10 nodes, providing justification for our initial network configuration. Repeated trials suggest that statistical variations outweigh any implied increases in accuracy beyond 10 nodes. To best preserve the integrity of the results ob-

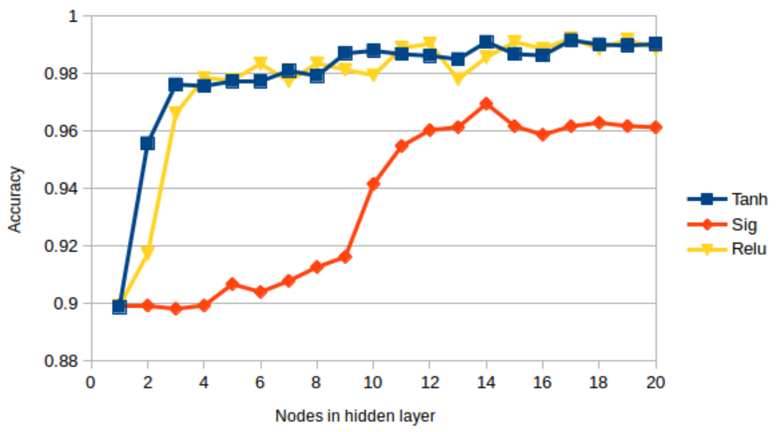


Figure 8: Accuracy of networks with different architectures

tained in Section 3.3.2, our lightweight network will stick to our initial configuration of 12 hidden nodes and the tanh function.

3.4.2 Training Duration

It is often beneficial to train neural networks for longer periods of time despite the computational cost. However, training for too many steps carries the risk of overfitting data. Figure 9 displays the classification accuracy of the optimized lightweight network trained for different durations. When testing on data collected on the same day as in Figure 7, our lightweight model has the same long-term performance after 10,000 steps, achieving a 20-fold reduction in attribute space. Counter-intuitively, our model requires more iterations to convergence, although fewer attributes are used in the classification. Nonetheless, it still manages to reach a peak accuracy comparable to the full model, implying that most of the discriminative power lies on only a few key attributes. This reduction in attribute space cuts down the preparation time of test data, allowing our lightweight model to perform classification at greater speed.

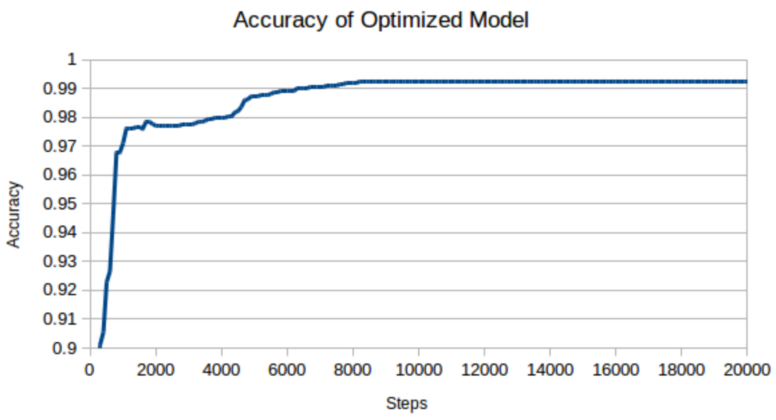


Figure 9: Same-day optimized performance

4 Spatial and Temporal Stability

4.1 Temporal Stability Analysis

4.1.1 Overfitting

To investigate temporal stability, we trained our lightweight model on ten datasets collected on the same day and tested it on data collected from the same server 12 months later. Figure 10 shows the accuracy of the model when trained for different numbers of steps. At around 5,000 steps the accuracy starts to decrease significantly. Comparing to Section 3.4.2, a model trained on data collected from a specific day becomes overfitted quicker when testing on temporally separated data than on same-day data.

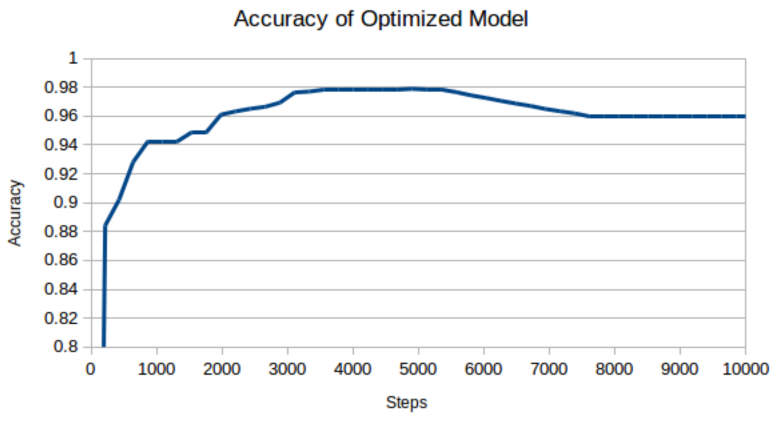


Figure 10: Performance of model on data collected after 12 months

4.1.2 Comparisons to Alternative Methods

Classification	NN		NB+K		C4.5/J48		BNN [1]	
	True	Delay	True	Delay	True	Delay	True	Delay
Accuracy(%)	99.0	96.7	98.3	94.7	99.8	98.8	99.3	95.3

Table 8: Temporal stability (1 year difference) of different classification schemes

Table 8 compares the accuracies of the four classification schemes highlighted in this report. The true classification accuracies were obtained by 10-fold cross validation training and testing on ten datasets collected from the same day. The delayed accuracies were obtained by training on the same ten datasets but testing on data collected from the same server 12 months later. For the Naive Bayes with Kernel Density Estimation and the C4.5 algorithm, we used the 11 attributes identified by Moore and Zuev in [18] to perform lightweight classification. Accuracies for the Bayesian neural network were taken directly from the 2007 work by Auld et al. [1] owing to difficulties in obtaining the proprietary MemSys software package.

As expected, all four schemes have reduced accuracy when testing on temporally displaced data. The neural network out-performs the Naive Bayes method under both true and delayed accuracies. The Bayesian neural network has a marginally higher (0.3%) true accuracy, but much lower temporal stability (4% loss compared to 2.3%). However, it is important to note that all the accuracies are probabilistic in nature, so the values in Table 8 should be interpreted as indicative trends of model performance rather than fixed percentages. Further work can be done with additional datasets to construct confidence intervals containing the accuracy values.

Remarkably, the C4.5 algorithm is the best method of the four, sporting the highest true and delayed classification accuracy. We know from [14] that the C4.5 has poor spatial and temporal stability. Combined with our results, this suggests that a difference of 1 year does not correspond to a large temporal shift in the attribute structure of network data. However, making this assumption would fail to explain the neural network's poor performance. One possible explanation is that the relationship between the flow's attributes and its class is simpler than we previously envisioned. The neural networks would thus create non-existent network architecture and overfit the training data. This is an area for further work, using larger datasets to test this hypothesis and to study the problem of overfitting.

Classification	NN		NB+K		C4.5/J48	
	True	Delay	True	Delay	True	Delay
Accuracy(%)	99.0	89.6	98.3	87.8	99.8	85.8

Table 9: Temporal stability (3 years difference) of different classification schemes

In Table 9 we see another temporal stability test, this time on data collected 3 years in the future. The experimental procedures were the same as in the previous stability test, but statistics for the Bayesian neural network were unavailable. As we would expect from literature, the accuracy of all methods decrease by 9-15%, since minor structural changes to each class will inevitably develop over time. Our neural network is the most robust of all three models, having the highest classification accuracy and the smallest accuracy drop after 3 years. This opens up potential applications of neural networks in servers with limited data availability. As in Table 8, further work can be done to create confidence intervals using additional data.

4.2 Spatial Stability

To test spatial stability, we split the entire dataset collected on the same day from the *Genome Campus* into four distinct sites.

- Site A : Data archiving research facility (25,959 flows)
- Site B : Data modelling research facility (168,412 flows)
- Site C : *Genome campus* site administrator (6,967 flows)
- Site D : Experimental research facility (176,184 flows)

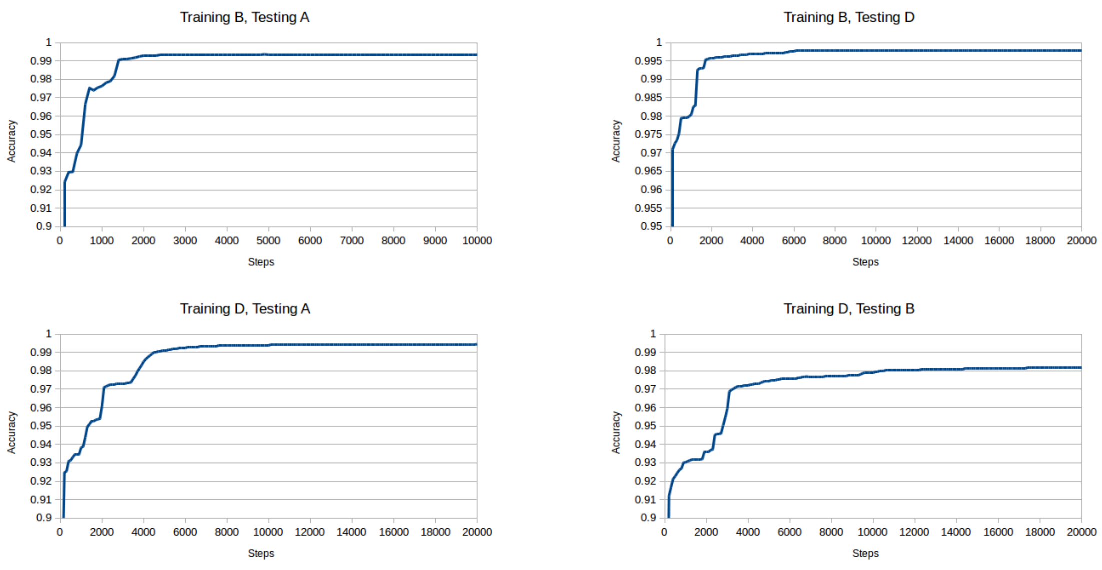


Figure 11: Spatial Stability of Neural Network

4.2.1 Training Set Size

Figure 11 shows the accuracy of the model when training and testing on data from different sites. Site A and Site C were not used for training as neural networks are known to have slight performance issues when the training dataset is small [22].

We compare the model's convergence rate to the control setup trained on 350,000 flows (Figure 9). We see that the models trained on the smaller datasets are more volatile when trained on fewer iterations. The intuition behind this behavior is that additional training data allows the network to correct itself more quickly, since small errors in the weight distribution are magnified in aggregate loss.

The presence of such behavior suggests that network operators must be careful not to undertrain their models when limited by small training sets. However, training too much also carries the risk of overfitting as evidenced in Section 4.1.1. We recommend the following procedure applicable to general neural networks to achieve a good balance between the two.

When presented with limited training data, first split the data into stratified sets. The number of sets is dependent on the amount of available data, but 10 is a common choice in literature [27]. Similar to cross-validation, pick a set to test on and the remainder become training data. Plot the accuracy against training duration and determine when the model converges. This duration will be a close approximation to the convergence time training on the full dataset. If the model is used to classify temporally separated data, the training should stop once the accuracy plateaus to minimize the overfitting risk.

4.2.2 Comparisons to Alternative Methods

Tables 10-12 show the spatial stabilities of the neural network, Naive Bayes and C4.5 methods. The control accuracies were taken from Table 8, where the data from all sites were consolidated to train a model testing on other consolidated data from the same day. The accuracies suggest that Sites A and D are more closely related to each other than Site

Accuracy% (NN)		Testing		
		Site A	Site B	Site D
Training	Site A	-	-	-
	Site B	99.3	-	99.7
	Site D	99.4	98.2	-
Control Accuracy		99.0		

Table 10: Spatial stability of Neural Network

Accuracy% (NB+K)		Testing		
		Site A	Site B	Site D
Training	Site A	-	97.1	99.3
	Site B	97.8	-	99.0
	Site D	98.0	96.4	-
Control Accuracy		98.3		

Table 11: Spatial stability of Naive Bayes

B. Compared to their control accuracies, the C4.5 and Naive Bayes perform slightly worse on average ($\sim 1\%$) when training and testing on different sites. Remarkably, the C4.5 and Naive Bayes methods still perform extremely well when training on Site A, which suggests high uniformity between data from different sites. This fact, when combined with the relatively small accuracy discrepancies, leads us to conclude that spatial differences have minimal impact on attribute structures within each class.

Due to limited availability of data, we were unable to test the effects of a larger spatial perturbation (e.g. data collected from servers in different countries). This is an area for further investigation, as well as testing on larger datasets to determine if the losses in C4.5 and Naive Bayes are significant or to within statistical variation.

5 Rate of Convergence

5.1 Experimental Procedure

To construct models on the fly, it is of interest to know how the time taken for the neural network to converge scales with the amount of training data. But before we can design experiments to determine this, we must first decide on an adequate metric for convergence.

Ideally, this would be when the gradients of the neural network disappear and the weights have completely stabilized. In practice, this could take unrealistic amounts of time and we often pick a stopping point by hand, based on the model's performance. The challenge then lies in deciding when models have converged sufficiently in a uniform but accurate manner. Fortunately, there two quantifiable metrics which lend themselves to easy application.

The loss function is a measure of deviation between the neural network's predictions on the training data and the ground truth. One option is to regard the model as having converged after its loss drops below a pre-defined threshold. This is difficult to implement in practice because it is extremely unlikely for any model to replicate its training data to

Accuracy% (C4.5/J48)		Testing		
		Site A	Site B	Site D
Training	Site A	-	98.9	99.8
	Site B	99.7	-	99.9
	Site D	99.7	99.0	-
Control Accuracy			99.8	

Table 12: Spatial stability of C4.5

100% accuracy, even if it achieves a high classification accuracy. The same lightweight model trained on different datasets generates different classification models, each with its own unique loss threshold. Both the height of this threshold and the time taken for the model to converge to it is highly dependent on the flows in the training dataset.

A more easily implementable metric is the model's classification accuracy on an independent test set of fixed size, collected from the same site on the same day. While it is difficult to interpret when a sufficiently low loss translates to a stable model, classification accuracy can give a reasonable approximation to stability. Since the model's refinements become smaller over time, it will reach a state where loss decreases despite there being no change in classification on the test set. In this section we define a model to have converged if its classification accuracy does not change in 3,000 iterations.

A minor caveat is that this does not guarantee the classifications assigned to test data will not change in future iterations. However, it serves as a good indication of model convergence and more importantly, a quantifiable metric. It is important to use a sufficiently large test dataset to ensure adequate representation of flows from each class. In this section, our test dataset contains approximately 25,000 flows.

To test the effect the amount of training data has on convergence time, we first designated one of the ten *Genome Campus* datasets as the test dataset. The remaining nine datasets were pooled to a large training dataset containing approximately 350,000 flows. The full training dataset was further stratified into smaller sets containing approximately 180,000, 60,000, 35,000, 12,000, 6,000 and 4,000 flows. The models were trained on the different datasets and terminated once convergence was reached. Each dataset size was tested ten times and each test was repeated ten times to account for differences in weight initialization.

5.2 Analysis of Convergence Time

Figure 12 displays the results of the experiments outlined in the previous section. In our computations we used a NVIDIA GeForce GTX 1080 Ti GPU running on a machine with 64GB RAM and a 6-core i7-4930K processor. The convergence time scales proportionally with the logarithm of the training set size. Although the time for each individual iteration scales roughly linearly with the size of training set, the presence of additional training data allows the network to converge with fewer iterations, as mentioned in Section 4.2.1. These two contributing factors provide justification for the observed logarithmic relationship.

There is significant statistical variation in the convergence time for each fixed size. This can be attributed to different weight initalizations of the neural network, different training data, slight variations in processing power assigned by the GPU to each trial,

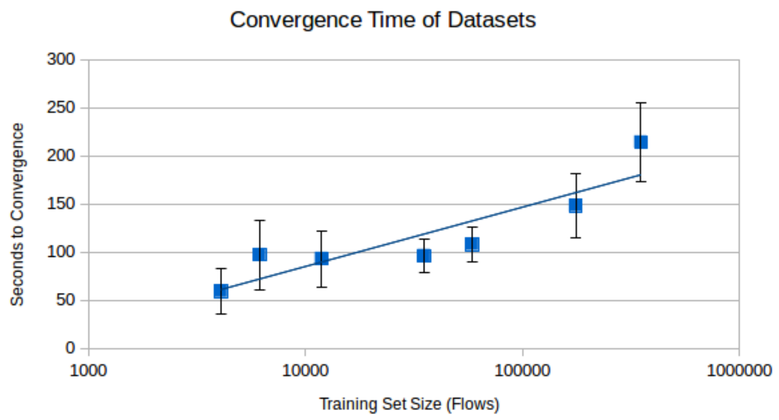


Figure 12: Convergence times of neural network with training sets of different sizes

the dependence of the test set in our definition of convergence, the number of attributes and the network structure. As such, it is difficult to draw further quantitative inference from our data beyond the observed logarithmic relationship. There is potential for further work using more data, but the aforementioned problems make it unrealistic for any precise analysis to have a wide range of application. The convergence times are too dependent on each network's configurations and data.

The $\mathcal{O}(\log N)$ relationship allows operators to approximate convergence times of their neural networks. Occasionally, networks will stabilize to some test classification for long periods of time before it changing, as seen in Figure 9. Knowing an approximate convergence time can serve to better detect these false positives.

6 Conclusions

In this paper we have demonstrated the construction of a lightweight neural network capable of real-time network traffic classification. In the process, we have also provided greater insight into methodologies used by different classification schemes. We discussed potential procedures for both data processesing and optimization which are generalizable to other supervised machine learning methods. We also outlined a fast method of identifying key attributes in the neural network based on the connection weights.

We showed that there were fundamental differences between the perceptron network and the Naive Bayes methods which manifest in the key attributes identified by both systems. The neural network achieved only $94\% - 97\%$ when running on the attributes identified by Symmetric Uncertainty rankings (Table 3). This increased to 99.0% when using the attributes identified by their weights (Table 7). Similar to work in [1], there were only two attributes, out of a possible ten, which overlapped with both the neural network and the Naive Bayes method.

Using previous work done on Bayesian neural networks as a starting point, we found an optimized configuration of the neural network capable of achieving >99% accuracy. Our lightweight model used one-hot embedding on the 13 most popular server ports, nine other attributes identified by weight, a single hidden layer with 12 nodes and a tanh activation function. Unfortunately, the number of steps the model takes to reach convergence is dependent on its training set. There is also the risk of overfitting data when training too long. In Section 4.2.1 we outline a method to determine the optimal training duration. As a general guide, our results suggest 10,000 steps is sufficient when training on at least 150,000 flows.

For the neural network, Naive Bayes and C4.5 systems, small spatial differences are observed to have a negligible impact on classification accuracy. Further work must be done to test if this holds true for larger spatial differences. With a temporal separation, all systems, including the Bayesian neural network, suffer a drop in classification accuracy. Although the C4.5 is the most robust in the short term, but the neural network offers the best long-term temporal stability. Depending on the availability of training data, this creates some niche uses in research for the neural network. However, for most network operators who have access to sufficient training data collected within the past year, the C4.5 will be the naturally preferred model. It appears that deep systems such as the neural network are best reserved for more complex tasks, such as image recognition and artificial intelligence. The upshot is that classifying internet traffic is not as difficult as one might imagine and simple methods such as decision trees can very effectively capture the essence of this system.

Despite significant variation in the data, we also discovered an approximately linear relationship between the convergence times of the neural networks and the logarithm of the training set size. Unfortunately, for the same amount of training data, the neural network still takes significantly more time to converge than the C4.5 method to compute. The C4.5 is decidedly a better classification tool for network traffic. However, the techniques employed in this paper have provided a framework for understanding the optimization process of neural networks. They can easily be reapplied in the context of classifying different types of data.

Throughout our report, we elaborated in detail on our methods and their limitations. Our accuracies and percentages were generated using limited online datasets. While this ensures high reproducibility, further work can be done using additional data to create confidence intervals around these figures. Testing the discrepancy between our chosen method and other means of optimizing the neural network are also potential ways to generate new insights. Our analysis here has treated all classes and flows equally it would not be optimized to test for specific uncommon classes, like in the case where failing to detect an ATTACK flow is extremely costly. Neural networks built specifically for identifying specific classes is an area for further research.

6.1 Acknowledgements

We are grateful to Filip Ayazi for his technical support and useful comments. We also give special thanks to Stephen Gull for his insight into Bayesian neural networks.

References

[1] Tom Auld, Andrew W Moore, and Stephen F Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on neural networks*, 18(1):223–239, 2007.

[2] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[3] Arthur Callado, Carlos Kamienski, Géza Szabó, Balázs Péter Gero, Judith Kelner, Stênio Fernandes, and Djamel Sadok. A survey on internet traffic identification. *IEEE communications surveys & tutorials*, 11(3), 2009.

[4] Marco Canini, Wei Li, Andrew W Moore, and Raffaele Bolla. Gtvs: Boosting the collection of application traffic ground truth. *TMA*, 9:54–63, 2009.

[5] Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[6] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[7] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007.

[8] Mikhail Dashevskiy and Zhiyuan Luo. Predictions with confidence in applications. In *MLDM*, pages 775–786. Springer, 2009.

[9] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[10] Hongbo Jiang, Andrew W Moore, Zihui Ge, Shudong Jin, and Jia Wang. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 299–304. ACM, 2007.

[11] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[12] Daniel T Larose. K-nearest neighbor algorithm. *Discovering Knowledge in Data: An Introduction to Data Mining*, pages 90–106, 2005.

[13] Wei Li, Kaysar Abdin, Robert Dann, and Andrew Moore. Approaching real-time network traffic classification. Technical report, 2013.

[14] Wei Li, Marco Canini, Andrew W Moore, and Raffaele Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*, 53(6):790–809, 2009.

[15] Andrew Moore, James Hall, Christian Kreibich, Euan Harris, and Ian Pratt. Architecture of a network monitor. In *Passive & Active Measurement Workshop*, volume 2003, 2003.

[16] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, 2013.

[17] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *PAM*, volume 5, pages 41–54. Springer, 2005.

[18] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM, 2005.

[19] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[20] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[21] J Ross Quinlan. *C4. 5: programs for machine learning.* Elsevier, 2014.

[22] Pero Radonja and S Stankovic. Neural network models based on small data sets. In *Neural Network Applications in Electrical Engineering, 2002. NEUREL'02. 2002 6th Seminar on*, pages 101–106. IEEE, 2002.

[23] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[24] Charalampos Rotsos, Jurgen Van Gael, Andrew W Moore, and Zoubin Ghahramani. Probabilistic graphical models for semi-supervised traffic classification. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 752–757. ACM, 2010.

[25] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148. ACM, 2004.

[26] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[27] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.