Waste not: using diverse neural networks from hyperparameter search for improved malware detection

Pedro Marques
Centre for Software Reliability
City, University of London
London, United Kingdom
pedro.magalhaes-marques@city.ac.uk

Matilda Rhode
Airbus
Newport, United Kingdom
matilda.rhode@airbus.com

Ilir Gashi
Centre for Software Reliability
City, University of London
London, United Kingdom
ilir.gashi.1@city.ac.uk

***Abstract*—Many commercial anti-virus software already use some form of machine learning to help with detection. However, there has been little research on ways in which multiple algorithms can be combined to improve malware detection. This paper presents an analysis of a dataset of malware and benign software, analysed by diverse recurrent neural networks (RNNs). Our focus is on analysing the possible benefits and/or drawbacks in malware detection from using multiple algorithms in diverse configurations. We have analysed the sensitivity, specificity and accuracy of RNN combinations with up to 10 models per combination, using prediction results from a previous research. Our results show significant gains in malware detection when using combinations with 1-out-of-N adjudication schemes (an increase of 0.28), and likewise gains for specificity in N-out-of-N schemes (an increase of 0.14). We also look at the interplay between sensitivity and specificity when putting together systems that use a simple majority adjudication scheme (e.g. 3-out-of-5). Additionally, we highlight the major sources of diversity between the various RNN models used, and speculate on the benefits towards specific types of malware. To the best of our knowledge, similar results on the use of diverse machine learning algorithms for malware detection have not been presented in the past.**

***Index Terms*—malware detection; design diversity; diverse machine learning; model diversity; cybersecurity**

I. Introduction

The use of design diversity, especially using available off-the-shelf components, for improving security is well known in the security community [1]. The expanding availability of off-the-shelf software has allowed security administrators to develop complex and highly reliable defence mechanisms at a fraction of the cost of developing purposefully built tools[1]. This is especially true for network defence systems, where a large number of Intrusion Detection Systems (IDSs), anti-virus and malware detection software, etc., exist [2], [3].

For malware detection, static data analysis is a prominent technique, sometimes in the form of signature-based approaches which is incorporated in all modern anti-virus software solutions. However, these approaches can be easily circumvented through code obfuscation [4], and are additionally of limited use when attempting to detect new malware. A possible solution is a dynamic data analysis approach, in which the activity generated by a potentially malicious sample is analysed in real-time by a defence system, and malicious actions are detected from this analysis. Many solutions claim to do this through the use of machine learning (ML) algorithms (e.g. Lastline's Defender[2]).

The use of multiple ML models may further increase the security guarantees, although little research has been carried out in this field (recent examples include [5], [6]). Ensemble learning algorithms, such as Adaboost [7] and Random Forests [8], make use of multiple models but these algorithms prioritise efficiency and/or accuracy rather than diversity. The benefits gained from the use of diverse models can vary significantly based on the problems to which they are applied, and the diversity of the models themselves. For ML models, this diversity would come from the parameters used to build the models, as well as the way in which the models are trained.

In this research, we have analysed the potential diversity benefits of using multiple recurrent neural network (RNNs) models for the classification of malware and benign samples. We used a dataset from a previously published research by one of the authors of this paper [9], which contains predictions from a set of 37 RNN models, trained and tested on a dataset of over 4000 software samples. The models operate as a dynamic data analysis mechanism, analysing the activity generated by samples as they are run on a virtualised system. The authors of [9] analysed the performance of each model individually. In this paper we present analysis of combining the various models in combinations of up to 10 elements, in 1-out-of-N, N-out-of-N and simple majority schemes. A 1-out-of-N voting scheme, classifies a sample as malicious if any 1 out of N models in the combination classifies it as such. Conversely, a N-out-of-N voting scheme, classifies a sample as malicious only if all N out of N models in the combination classifies it as such. In a simple majority scheme, a sample is classified as malicious

[1]e.g. Security Information and Event Management tools such as Splunk, ELK, Security Onion etc.

[2]https://www.lastline.com/solutions/network-defender/

if the majority (N/2+1) out of N models in the combination classify it as such. We have also analysed the main sources of diversity between the various models, and how additional RNN models could be built in the future to leverage the most gains in terms of both sensitivity (correct detection of malware), specificity (correct classification of benignware) and overall accuracy (optimising for a balanced improvement in correct classification of malware and benignware). Additionally, we explore the benefits of this diversity when used to detect specific types of malware.

The main findings are as follows:

- An average increase in sensitivity from 0.55 (single models) to 0.83 (1-out-of-10 adjudication schemes), with a maximum sensitivity observed of 0.94 in 1-out-of-10 schemes compared to 0.90 in single models.
- An average increase in specificity from 0.82 (single models) to 0.96 (10-out-10 adjudication schemes), with a maximum specificity observed of 0.99 in 10-out-of-10 schemes compared to 0.93 in single models.
- Two characteristics have the biggest impact in creating diverse models: i) the choice of optimiser used for the models; and ii) how long each model is given to observe the sample execution before it generates a prediction on whether the sample is malicious or benign.
- We observe similar benefits from the use of diverse RNN models when analysing different malware types, with only small differences in the detection capabilities between malware types.

To the best of our knowledge similar analysis of diversity of ML algorithms for malware detection has not been published before.

The rest of this paper is organised as follows: in Section II we discuss related work in the fields of design diversity and malware detection with the use of machine learning models; in Section III we detail the manner in which the ML models in our research were built, trained and tested, as well as how the samples used were collected; Section IV gives a brief description of the dataset we have analysed; Section V contains the main body of our analysis results, pertaining to the use of diversity analysis of the dataset; and finally in Section VI we provide a discussion of our results.

II. Related work

The security community is well aware of diversity as potentially valuable [10], [11]. Discussion papers argue the general desirability of diversity among network elements, like communication media, network protocols, operating systems, etc., but only sparse research exists on how to choose diverse defences (some examples in [3], [11]–[13]). Potential benefits from design diversity for safety and reliability have been studied for many years. See for example work on a probabilistic model of diversity outlined in [14], [15], which were motivated by the work on N-version programming [16]. It has been discussed as a risk reduction strategy, particularly at the start of a project [17]. The authors of [17] also warn that different application areas require different measures to calculate the effectiveness of design diversity, and as such the publishing of results from one area might not be directly applicable in other areas. Littlewood et al. [10] further compound this point by discussing how measuring the performance of IDS's should be done based on categories of attacks, rather than using an average mixture of attack classes.

Design diversity with machine learning, also known as machine learning ensembles, have been researched for a while (e.g. [7], [8]), though the goals differ somewhat from the more traditional use of design diversity, by being focused on improving the final model accuracy rather than attempting to create systems which are as diverse from one another as possible. Two popular techniques are often mentioned, *bagging* and *boosting* [18]. With bagging techniques (e.g. random forests), different machine learning models are trained with different subsets of the same dataset, essentially creating data diversity. Models are given N-sized subsets of the data to train on, where N is the size of the original dataset. This means that samples can (and often will) be repeated for the same model, and virtually guarantees that different models will have a different data distribution to learn from. Boosting techniques expand on those found in bagging approaches by turning it into an iterative approach. In boosting techniques (e.g. AdaBoost), models are given a random subset of training data, and subsequently tested. After this, new models are created with another random subset of training data, although this time samples that were previously poorly classified are given extra weight, meaning that they should be further represented in the new set of models. This process continues until the models achieve a desired performance metric. Bagging and boosting methods are predicated on the assumption that many poorly performing models, known as weak learners, can create a strong learner when their predictions are combined. Ensembles of ML models can be made from any combination provided there is a rule for determining a final classification (such as majority voting). The diversity in predictions of the components of the ensemble enable the ensemble to achieve better accuracy than using a single constituent part.

Recent work with design diversity of machine learning models includes work done by Machida [6], where the authors propose an N-version architecture for machine learning models, and point out the usefulness of model diversity and input diversity in creating models that would be expected to exhibit failure diversity. Goodfellow et al. [19] have noted that variance in neural network (NN) predictions can be promoted by altering the hyperparamters of the model and/or the data used for training. The average prediction of these models can then be used as the final prediction. NN models have provided state of the art results in a number of domains including computer vision and natural language processing [20]. NNs have many hyperparamaters to tune prior to training and the best set of hyperparameters is found through experimentation (random, Bayseian or other method). As Hansen and Salamon pointed out in 1990 [21], hyperparameter tuning is computationally expensive and discarding all configurations but the single best-performer may be considered a waste of using that energy if other models can be used to improve model performance. Recently, Xu et al. [5] emphasised that diversity in the training data has the most significant effect in the diversity in failure behaviour of NN models.

When making predictions on malicious versus benign software samples, the features used by machine learning models are typically derived from either the code or the behaviour exhibited by these samples. Static data, derived directly from code is easily collected and analysed, though signature-based methods fail to detect obfuscated or entirely new malicious code. This is demonstrated by Saxe and Berlin [22] where their solution falls from a 95.2% true-positive rate to 67.7% when tested only on *new* samples not seen during training.

Methods that use dynamic data assume that malware must enact the behaviours necessary to achieve their aims. Typically, these approaches capture behaviours such as API calls to the operating kernel. Firdausi et al. [23] compare machine learning algorithms trained on API calls and achieve an accuracy of 96.8% using correlation-based feature selection and a J48 decision tree. Other examples ([24]–[26]) report similar levels of success when using dynamic data methods.

III. Methodology

As this analysis makes use of the hyperparameter tuning conducted for [9], we will first describe the methodology in which that work was carried out. The original work focused on the creation of a set of recurrent neural network (RNN) models for the classification of malware and benignware samples, through the use of dynamic data analysis. As opposed to static data analysis, models using dynamic approaches look at the activity generated by a sample as it runs in a system. One of the key insights needed for this is the ability for models to process time-series data (the data generated by a sample during the time it is ran), and two major machine learning models are useful for this, RNN and Hidden Markov Models (HMM). The justification in [9] for using RNN is that these are capable of processing continuous time series data, unlike HMM.

A total of 37 RNN models were created. Each model works by analysing the activity generated by a sample when run in a virtualised environment, using 10 different metrics as feature inputs: system CPU usage, user CPU usage, packets sent, packets received, bytes sent, bytes received, memory use, swap use, the total number of processes currently running and the maximum process ID assigned. A snapshot of the metrics is taken every second for 20 seconds whilst the sample executes, starting at 0 seconds, such that at 1 second, there will be two feature sets or a sequence length of 2. Whilst API calls to the operating system are the most popular behavioural features used in dynamic malware detection, they were not used for this study, as recent work has shown that they achieve comparable performance to the machine metrics used in this paper but are a less robust data source when tested on data from a different underlying distribution [27].

Previous work [9] has looked at the contribution of these features to malware classification by excluding each one from high-performance models and noting the accuracy depreciation. Each feature had an impact on the models tested and no two features are perfectly correlated. It may seem odd that CPU usage can distinguish malicious and benign behaviour given the amount of work done by background system processes which will also consume computational resources but since each sample has been executed in a virtual machine reset to the same snapshot between each sample. Furthermore it is not the CPU usage alone which enables distinction between malware and benignware but the combination of this data with other features.

In order to create RNN models that are different from one another, each model was built with a different set of hyperparameters. Table I describes these hyperparameters in more detail. When creating the models, a random search of the hyperparameter space was used as this allowed for easier parallelisation and implementation. Additionally, random space search has been found to be more efficient at finding good configurations when compared to grid search [28], which uses a discrete search space for hyperparameter values rather than a continuous one.

To be able to generate predictions for the entirety of the sample dataset, while still making use of this same dataset for training purposes, a 10-fold cross validation technique was used. The sample dataset was split into 10 equally sized partitions. Models were trained with 9 of these and tested on the remaining partition, a process which was repeated 10 times, each time changing which partition was tested. The predictions generated for the 10 testing partitions were then combined, for ease of performance comparison between different models. It is crucial to note that when generating a prediction for a sample, a model was never trained using that same sample. The predictions generated by the RNN models are expressed in values between 0 and 1, where a prediction of 0.5 or higher is interpreted as a malware classification, with anything under this value being considered a benignware classification.

The sample dataset contains equal parts benign and malicious samples. The malware and benignware were collected from a variety of different online sources, including VirusTotal, Softonic, PortableApps, SourceForce[3] and Windows OS. These were subsequently labelled as malicious or benign and with a malware type, if applicable, using the VirusTotal API. Each sample was run in a virtualised environment, using Cuckoo Sandbox, and machine activity metrics extracted using a custom auxiliary module reliant on the Python Psutil library. This is illustrated in Fig. 1.

The predictions generated by the models individually for each of the samples was published in [9]. For this work, we have expanded the analysis by looking at the possible gains and drawbacks of using multiple RNN models in more traditional adjudication schemes. A basic example of this would be to use two different models rather than one, in

[3]www.virustotal.com; www.softonic.com; www.portableapps.com; www.sourceforge.net

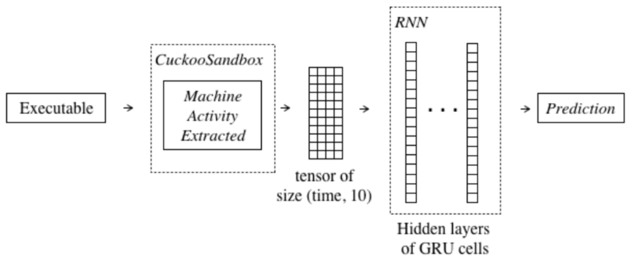

Executable
CuckooSandbox
Machine Activity Extracted
tensor of size (time, 10)
RNN
Hidden layers of GRU cells
Prediction

Fig. 1. High-level model overview

TABLE I
MODEL HYPERPARAMETERS

Attribute	Description
Model architecture	
Depth	Number of hidden layers in GRU-RNN
Hidden neurons	Number of neurons (GRU-cells) in each hidden layer
Bidirectional	Time series processed forwards as well as backwards
Model training	
Batch size	Number of training samples seen by the network between each weight update
Epochs	Number of times model is exposed to full dataset
Optimiser	Weight update rule
Learning rate	Multiplier on weight changes during training for SGD (0.001 learning rate used for Adam)
Dropout rate	Proportion of randomly zero-ed neurons during training to help with overfitting
L1 recurrent weight regulariser	L1 normalisation on recurrent weights
L2 recurrent weight regulariser	L2 normalisation on recurrent weights
R1 recurrent weight regulariser	R1 normalisation on recurrent weights
R2 recurrent weight regulariser	R2 normalisation on recurrent weights
Dataset	
Sequence length	Amount of data seen by each model for each behavioural trace

a 1-out-of-2 (1oo2) adjudication scheme. In this scenario, our overall system would generate a malware classification if either of our two internal models generated a malware classification. This configuration would generally lead to an increased sensitivity metric (the number of correct malware classifications), while decreasing the specificity (the number of correct benignware classifications) by potentially generating more false positives. In a similar manner, a 2-out-of-2 (2oo2) adjudication scheme would lead to an increased specificity and decreased sensitivity.

By looking at the raw predictions generated by the models for each individual sample, we are able to extrapolate what each possible adjudication scheme would predict as a whole, allowing us to calculate various performance metrics (such as accuracy, sensitivity, and specificity) for each possible combination of models, with each possible adjudication scheme. We have done this for combinations of up to 10 models[4] in three different types of adjudication schemes: i) 1ooN schemes, which improve sensitivity; ii) NooN schemes, which improve specificity; and iii) simple majority schemes (i.e. 2oo3, 3oo5, 4oo7, 5oo9) which represent a middle ground between sensitivity and specificity.

Additionally we have done this analysis based on the various hyperparameter values of each model, allowing us to better understand which parameters lead to an increased diversity between models, as well as based on malware type, making it possible to determine whether design diversity techniques have different effects for different type of malware.

IV. DESCRIPTION OF THE DATASETS

We have analysed a dataset from [9], containing the predictions generated by 37 RNN models against a sample dataset of 4,066 malware (1,925) and benign (2,141) samples, this is consistent with previous work in dynamic executable malware detection (e.g. [29]: 4,700; [30]: 1211, [31]: 2,200, [32]: 500) with notable outliers from large enterprises using millions of samples [24], [33]. The predictions generated range from 0 (benign) to 1 (malicious), where a value of 0.5 or above is considered a prediction of malware, as explained previously. The 1,925 malware used in training and testing of the RNN models are split into multiple types, based on the outputs given by the anti-virus tools available from VirusTotal, and are described in Table II. Trojans - malware disguising itself as benignware - were the most common type seen in the dataset. Ransomware - malware which destroys, encrypts or somehow makes user data inaccessible - are divided into ”trojanransomware”, ”virusransomware” and ”adwarerandomware”, the ”adware” ”virus” and ”trojan” descriptors refer to the injection mechanism, whereas ”ransomware” indicates the malicious behaviour. The reason for these divisions, rather than classifying them as trojans, viruses or adware is that ransomware is easily identifiable, due to the nature of its acts. Similarly, ”aptvirus” and ”aptbackdoor” are subtypes of viruses and backdoors which represent parts of advanced persistent threat (APT) campaigns. These malware typically gain access to a network and remain dormant until an opportune moment to enact malicious behaviour arises.

V. RESULTS

A. Individual models

We first look at how all of the models in our dataset performed on their own. In Fig. 2 we show the accuracy, sensitivity and specificity that all 37 models achieved during their testing, ordered by their overall accuracy. Most of the models achieve close to 0.7 accuracy, with a few outliers on

[4]We ran the analysis in a distributed computational environment at City, University of London, which utilised three VMs, each with 20 CPU cores, and with 64GB RAM in each VM. It took approximately 10 days to run the 1oo10 experiment, and, due to the combinatorial explosion, we estimated it would take more than a month to do 1oo11, and even longer for higher combination sizes. For this reason we did not continue with higher combinations.

TABLE II
MALWARE TYPE COUNTS

Type	Count
trojan	1056
virus	309
backdoor	114
adware	86
trojanransomware	72
bot	71
virusransomware	52
adwareransomware	50
application	36
worm	24
aptvirus	20
infostealertrojan	16
rootkit	11
aptbackdoor	7
unknown	1

either side of the graph. However, the same cannot be said for sensitivity and specificity, as these vary significantly between the models, with some being better at detecting malware, while others better at not raising false alarms for benignware. The three worst models (models 29, 24 and 6) all vary significantly from the rest of the models, by having their sensitivity close to 0.2, while model 1 (being the best in overall accuracy) is the only model to achieve a higher value for sensitivity then specificity.

When looking at these graphs it is important to keep in mind the various parameters with which the models were built. The three worst models from the previous figure (models 29, 24 and 6) are the only models in our dataset that have a value of 20 for sequence length (the amount of time models look at a sample's activity before generating a prediction), which is also the highest value of sequence length in the dataset. On the other hand, the best performing model (model 1) is the only model in the dataset with a sequence length of 2, which also happens to be the lowest value of sequence length in the dataset. If we order the information we have by the value of each model's sequence length, as we have done in Fig. 3, we can see that as the value of sequence length increases, there is a consistent decrease in that model's sensitivity, with an opposite but subtle increase in specificity.

Of all the hyperparameters used when building the models, sequence length is the only one that seems to have a demonstrable impact on the various metrics achieved by the individual models. For the reader's insight, we show in Table III all of the models in the dataset alongside their achieved accuracy, sensitivity and specificity.

It is also important to understand the difficulties that each sample in our dataset poses to the models. Fig. 4 represents the "difficulty" of each sample, by identifying the total number of models that failed to correctly identify the various malware and benignware samples (i.e. they labelled malware as benign and benign as malware). An important aspect to highlight in each of the lines is that, at times, they reach a total of 37 models, meaning that there are both malware and benignware samples that are always incorrectly classified by all of the models in our dataset. This effectively imposes certain upper bounds on the maximum amount of sensitivity and specificity we can ever achieve with just combinations of these same models, something we will touch on in the next section. For malware, there are a total of 101 samples that are never correctly identified as malware, which means that the upper bounds for sensitivity with other adjudication schemes is (1925 - 101) / 1925 = 0.94753. For benign samples, there are a total of 18 samples that are never correctly identified as benign, which means that the upper bound for specificity with other adjudication schemes is (2141 - 18) / 2141 = 0.99159.

TABLE III
ACCURACY, SENSITIVITY AND SPECIFICITY VALUES FOR SINGLE MODELS

Model	Accuracy	Sensitivity	Specificity
1	0.80792	0.89922	0.72583
8	0.79415	0.79481	0.79355
30	0.79292	0.76312	0.81971
32	0.75848	0.65403	0.85241
7	0.75283	0.72675	0.77627
33	0.72873	0.6426	0.80617
13	0.71323	0.54026	0.86875
27	0.712	0.55792	0.85054
20	0.70782	0.55065	0.84914
18	0.70659	0.54857	0.84867
34	0.70659	0.53455	0.86128
4	0.70512	0.51688	0.87436
22	0.70192	0.62182	0.77394
36	0.69651	0.53039	0.84587
28	0.69651	0.49818	0.87482
9	0.69577	0.51169	0.86128
35	0.69356	0.59377	0.78328
14	0.69356	0.53922	0.83232
11	0.69306	0.53714	0.83326
26	0.69306	0.4987	0.86782
23	0.69208	0.51636	0.85007
12	0.69011	0.48	0.87903
31	0.68962	0.46909	0.8879
2	0.68888	0.65714	0.71742
15	0.68839	0.5974	0.7702
25	0.68839	0.60052	0.7674
21	0.68519	0.53091	0.82391
16	0.68028	0.49351	0.8482
3	0.67831	0.53818	0.8043
19	0.67511	0.53403	0.80196
10	0.67216	0.5174	0.8113
0	0.66773	0.64104	0.69173
5	0.63207	0.49247	0.75759
17	0.62863	0.52	0.7263
6	0.59272	0.21558	0.93181
24	0.58534	0.21714	0.91639
29	0.58436	0.20831	0.92247

Of the 101 malware samples that were always misclassified, there doesn't appear to be a clear reason, other than their intrinsic difficulty. In Table IV we break down these 101 samples based on their malware type, in which trojan and virus are the most numerous. However, this is likely due to the distribution of overall malware types in the sample dataset, as this same order is found when looking at all of the malware samples in the dataset (as we previously demonstrated in Table II).

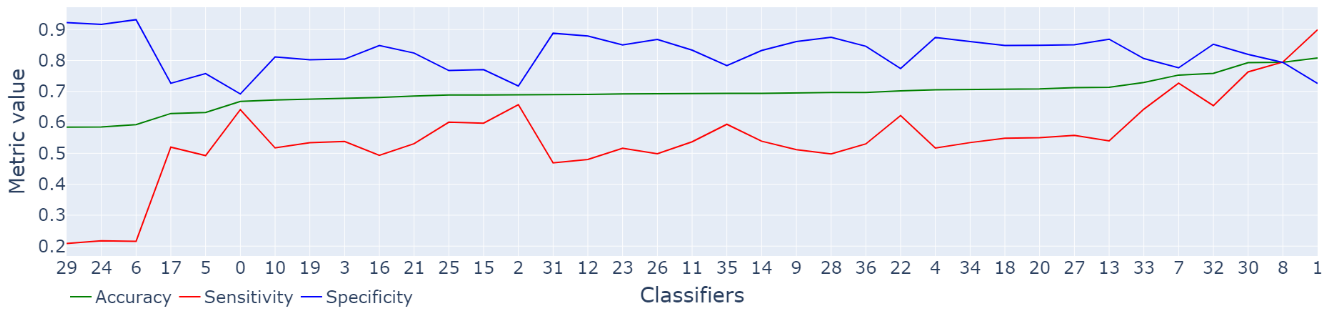


Fig. 2. Single model metrics, ordered by accuracy

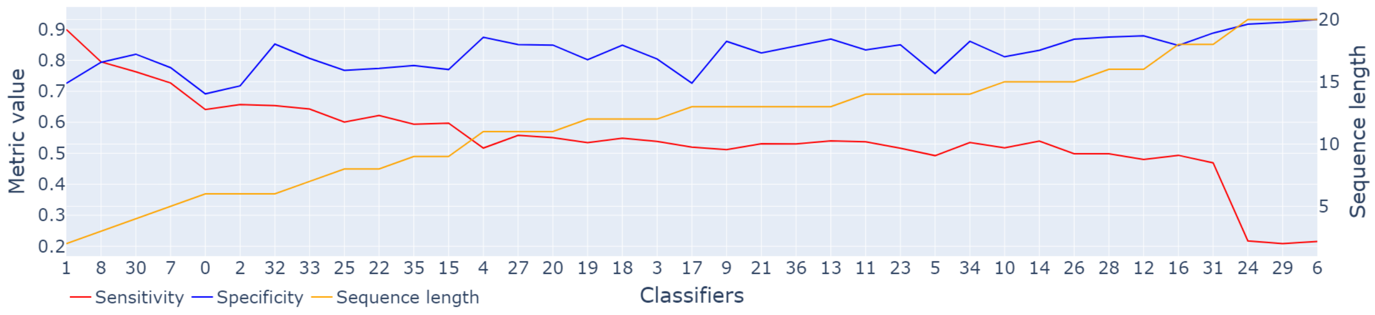


Fig. 3. Single model metrics, ordered by sequence length

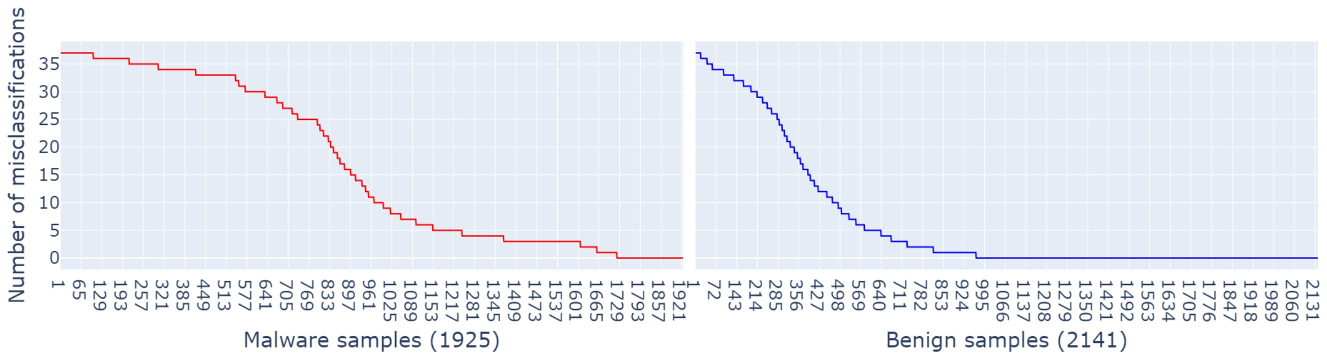


Fig. 4. Sample difficulty for malware and benign samples

B. *Overview of diversity*

Now that we have analysed the individual models, we can start to combine them in different adjudication schemes to try to understand what sort of benefits and drawbacks we can get, in terms of overall accuracy, as well as sensitivity and specificity. We have looked at 1-out-of-N schemes which excel at improving the sensitivity of the system (correctly identifying malware), N-out-of-N schemes which excel at improving the specificity of the system (correctly identifying benignware) and simple majority schemes, which are often a mix of the best of both worlds.

In Figs. 5 and 6 we show the sensitivity and specificity improvements with 1ooN and NooN adjudication schemes, respectively, up to and including combinations of 10 models. As expected, each adjudication strategy improves the respective metric of the overall system, with 1ooN improving sensitivity and NooN improving specificity. For 1ooN (Fig. 5), there is a gradual increase in sensitivity up until 1oo4 schemes, with two more noticeable jumps after this point, going from 1oo5 to 1oo6, and from 1oo7 to 1oo8, as noted by the jump in the median.

For NooN (Fig. 6) we see a similar picture for specificity,

TABLE IV
DESCRIPTION OF MALWARE NEVER IDENTIFIED

Type	Count
trojan	56
virus	19
backdoor	9
adwareransomware	4
bot	3
trojanransomware	3
adware	2
infostealertrojan	2
aptvirus	1
virusransomware	1
worm	1

TABLE V
MIN, MEAN AND MAX SENSITIVITY FOR 1OON ADJUDICATION SCHEMES

Adjudication	Min	Mean	Max	Upper bound
1oo1	0.20831	0.54836	0.89922	0.94753
1oo2	0.21922	0.64942	0.92364	0.94753
1oo3	0.25143	0.69698	0.9361	0.94753
1oo4	0.50442	0.72969	0.94078	0.94753
1oo5	0.54026	0.75519	0.9439	0.94753
1oo6	0.55117	0.7761	0.94597	0.94753
1oo7	0.56312	0.79373	0.94701	0.94753
1oo8	0.57299	0.8089	0.94753	0.94753
1oo9	0.5761	0.82213	0.94753	0.94753
1oo10	0.57922	0.83381	0.94753	0.94753

with more significant improvements to the metric going up to 4oo4 schemes, and subtly continuing thereafter.

In Tables V and VI we show the minimum, mean and maximum values for sensitivity and specificity for 1ooN and NooN adjudication schemes, respectively, along with the upper bounds for these metrics, as we calculated previously when looking at the sample difficulty. In accordance with the data in the previous boxplots, these metrics have a consistent growth as the number of models in each combination increases. For sensitivity, the maximum theoretical value is achieved with a combination of 1oo8. The maximum sensitivity is thus not increased for 1oo9 and 1oo10 combinations, but the minimum and mean sensitivity values continue to grow.

For specificity however, it's theoretical maximum value is not observed in the current analysis we have performed, with 10oo10 schemes only reaching a maximum of 0.98926 (differing from the upper bound by only 0.00233). The upper bound of specificity concerning this dataset, would only be achieved with a minimum adjudication of 15oo15.

We have also taken a look at majority voting schemes. As a mix between 1ooN and NooN schemes, majority voting tends to strike a balance between sensitivity and specificity, leading to an increase in accuracy. In Figs. 7 and 8, we show the improvements for all three of these metrics when looking at majority voting schemes. For all three metrics, the effects of majority voting seem to be restricted to narrowing the range of possible values to the original median found in 1oo1. This significantly reduces the lowest minimums found in some combinations, but at the same time it also limits the outliers that overachieve.

TABLE VI
MIN, MEAN AND MAX SPECIFICITY FOR NOON ADJUDICATION SCHEMES

Adjudication	Min	Mean	Max	Upper bound
1oo1	0.69173	0.82452	0.93181	0.99159
2oo2	0.72957	0.88045	0.96217	0.99159
3oo3	0.76833	0.90511	0.97431	0.99159
4oo4	0.79169	0.92058	0.97992	0.99159
5oo5	0.79682	0.93176	0.98319	0.99159
6oo6	0.81037	0.94041	0.98505	0.99159
7oo7	0.81831	0.94737	0.98645	0.99159
8oo8	0.83372	0.95312	0.98739	0.99159
9oo9	0.8412	0.95794	0.98832	0.99159
10oo10	0.8496	0.96203	0.98926	0.99159

C. Sources of diversity

In the previous section we showed the analysis of the observed diversity when building combinations of N models. The question then arises as to what is the source of the diversity between the various models. We analysed all of the hyperparameters (cf. Table I) to check for the effects they have on the observed diversity. Two of these hyperparameters appear to have the largest effect on the diversity between the models, namely the model's sequence length parameter, and the model's optimiser, both of which we will present in this section. For all other hyperparameters the differences they produce in the performance of combinations of different sizes are negligible or at best, random.

We start by looking at the effects of sequence length, which is the amount of time a model looks at a sample's activity before it generates a prediction. As we previously pointed out, when looking at individual models, an increase in the model's sequence length clearly accompanied a decrease in sensitivity for that model. In Figs. 9 and 10, we show the mean sensitivity and specificity for combinations in 1ooN and NooN schemes, respectively, based on the mean distance in sequence length of all the constituent models. The various model combinations are grouped together by looking at the individual sequence length value of the individual models in that combination, and calculating the mean difference between all of them. In essence, this means that combinations with a higher mean distance in sequence length have their constituent models more evenly spaced out in the range of all possible values of sequence length.

One thing that is clear from looking at Figs. 9 and 10 is that both sensitivity for 1ooN schemes, and specificity for NooN schemes increases as the mean distance of sequence length increases as well. This indicates that combinations with a wider range of sequence lengths are more "diverse". For 1ooN schemes, a more diverse combination leads to a higher number of positive (malware) samples being correctly identified, due to individual models alerting distinctly on different samples. For NooN schemes this same logic applies, but because all of the models need to generate an alert for one to be raised overall, this "diversity" between the individual models means that fewer false positives are generated.

One of the reasons that could explain these diversity gains

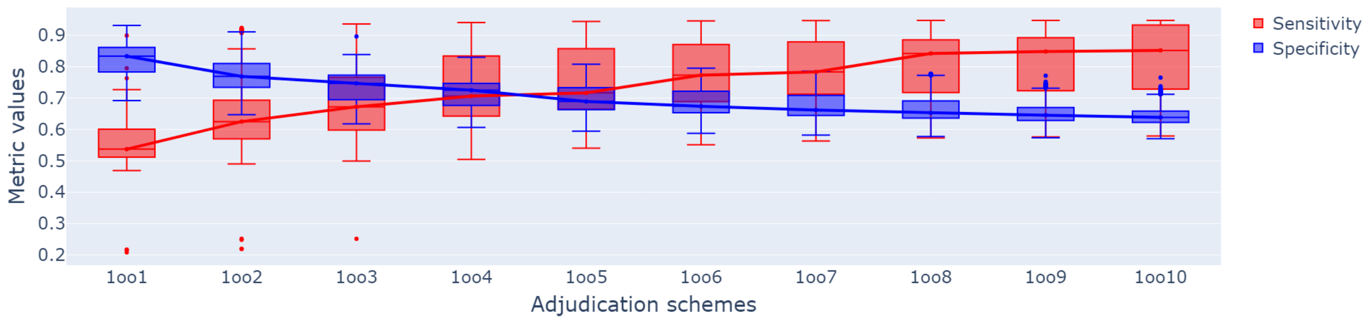


Fig. 5. Sensitivity and Specificity for 1ooN adjudication schemes

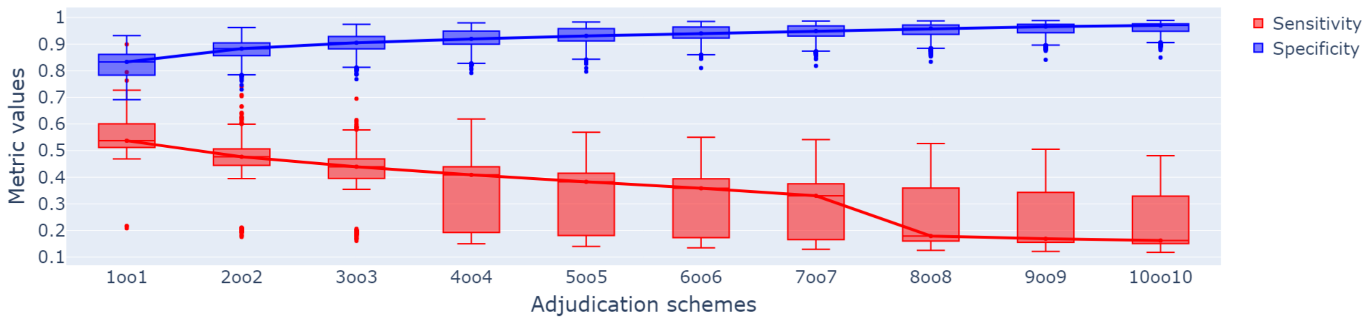


Fig. 6. Sensitivity and Specificity for NooN adjudication schemes

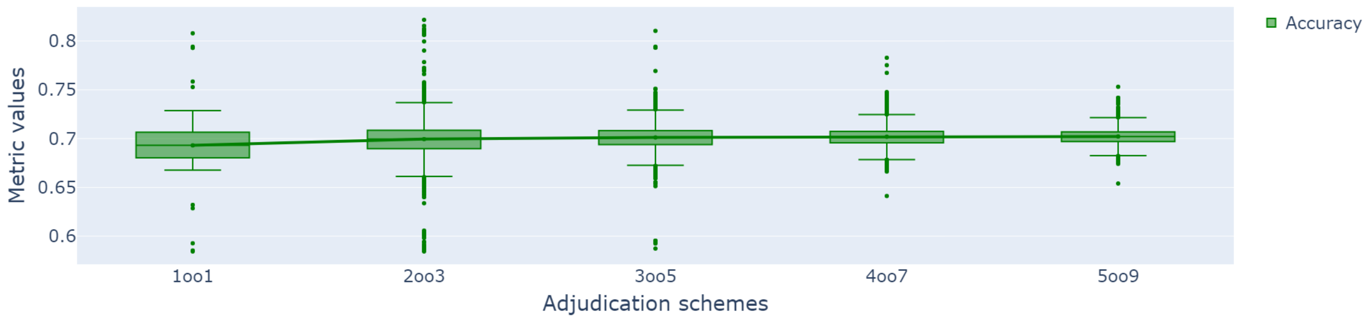


Fig. 7. Accuracy for majority voting adjudication schemes

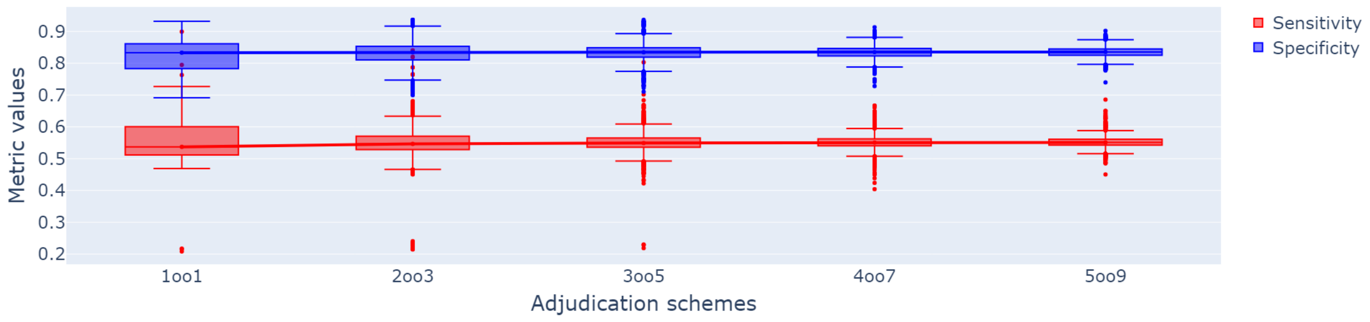


Fig. 8. Sensitivity and Specificity for majority voting adjudication schemes

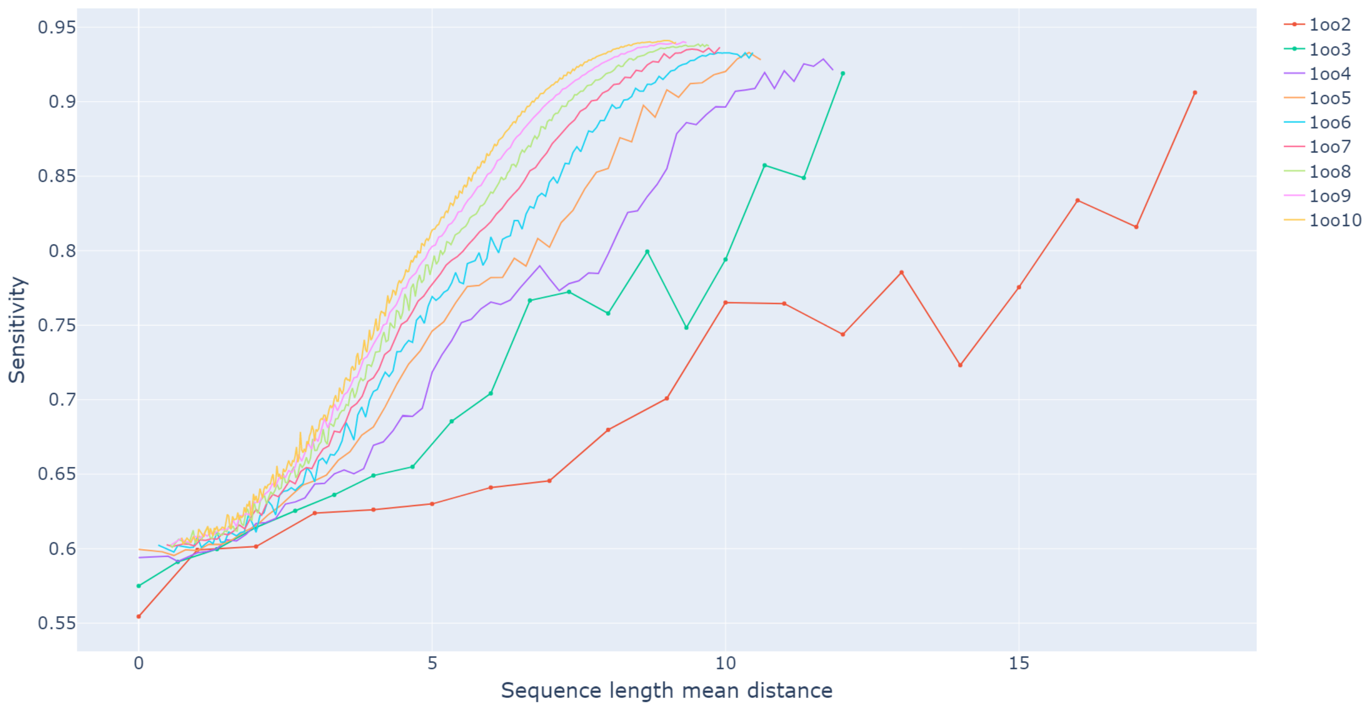


Fig. 9. Mean sensitivity based on the mean distance between sequence length values of models in a particular combination

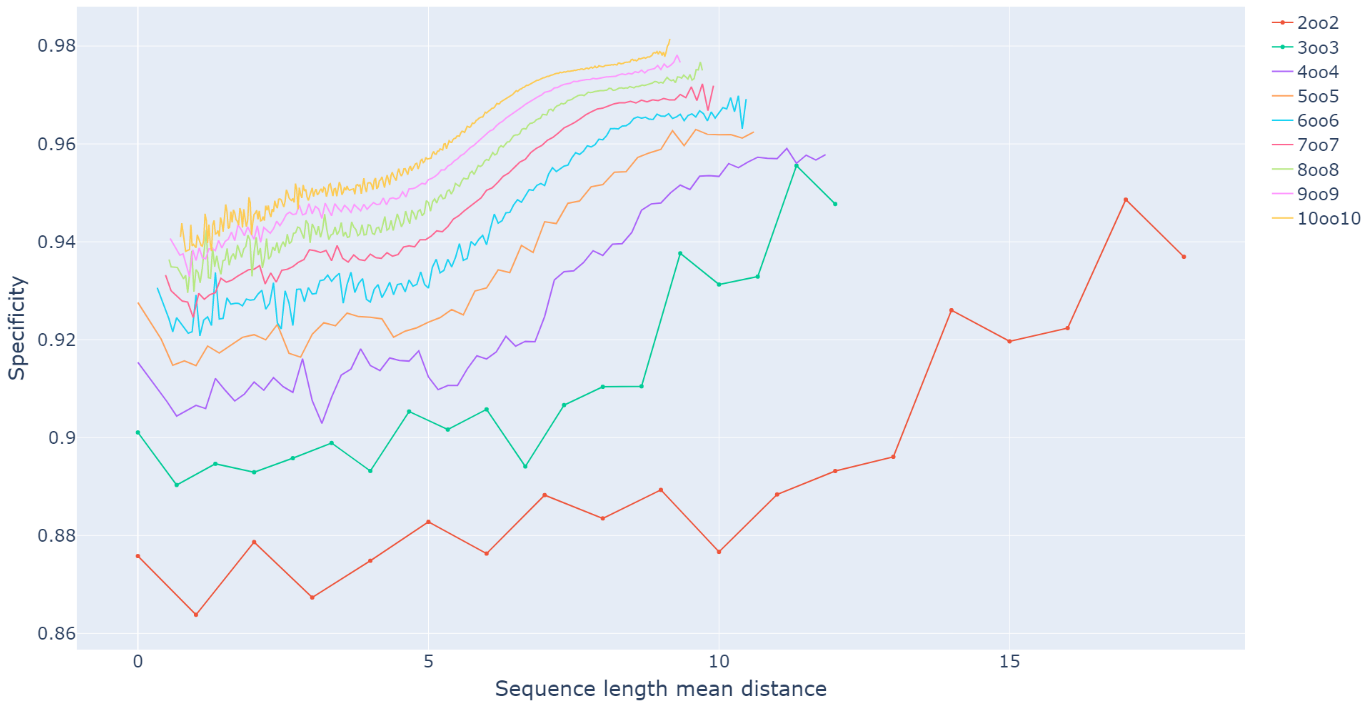


Fig. 10. Mean specificity based on the mean distance between sequence length values of models in a particular combination

coming from different sequence length values is the methods of attack of different malware samples. The patterns of attack of each sample will have a great effect on the activity observed by each model. For example, a malware might start generating suspicious activity as soon as it is run, such as creating a backdoor, but fall-off in activity afterwards, performing actions that may look more normal. In these cases, a model with a lower sequence length would have an advantage in its prediction, as it may only have seen the abnormal activity generated by the malware. The same logic could apply for slower acting malware, which could start off benign in order to mask its activity, and only generate more abnormal behaviour later on, meaning that a model with a higher sequence length would generally perform better. In situations where both types of malwares are to be expected, the use of different types of machine learning models would thus be a worthwhile advantage.

We now move on to the effects of the optimiser used for each model. Each model operates using one of two optimisers, either Adaptive Moment Estimation (Adam) [34] or Stochastic Gradient Descent (SGD) [35]. SGD uses gradient descent to minimise error in machine learning, computing using just a subset of the data, but requires the model builder to choose a learning rate: the rate at which model parameters are updated. Adam has risen to popularity as it adapts the learning rate and typically performs well using an initial default learning rate. However there is debate as to which yields are more accurate model [36]. In Fig. 11 we compare the mean sensitivity and specificity values for 1ooN and NooN schemes based on the various possible configurations of Adam and SGD models. We display this by plotting on the X-axis the number of Adam models present in the combinations, such that on the left most point of the graphs, where X = 0, there are only SGD models in the combination, and as we move right, we replace one of the SGD models with an Adam one.

The analysis shows that, for this dataset, the higher the number of Adam models in a combination, the better the performance of that combination. This is true for both sensitivity and specificity, in both 1ooN and NooN schemes, with the only exception being sensitivity in NooN schemes, where this only is true towards higher proportion of Adam vs. SGD model combinations, and subtly so. This aspect mostly highlights how Adam seems to be a better option when compared to SGD when it comes to applications of malware detection. This may be explained by the learning rates used by these algorithms. The learning rate determines the amount by which to update the parameters of the NN and may benefit from being large at the start of training, to allow big changes to the model parameters, and smaller later on, to make smaller adjustments once the model is performing reasonably well. Adam adapts the learning rate during training whilst SGD does not.

D. Effects of diversity on malware types

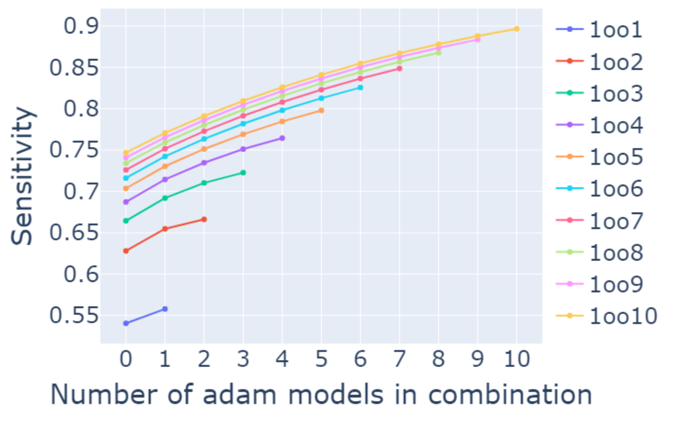
An interesting aspect to take a look at is the impacts of diversity on different types of malware samples. In Fig. 12, we show the improvements to sensitivity when going from 1oo1 to 1oo2 and 1oo3 schemes, broken down by malware sample types. Note that the data is ordered on the graph based on the improvement achieved when going from 1oo1 to 1oo3 schemes. For most malware types, going from 1oo1 to 1oo3 leads to close to 0.15 improvement in sensitivity. However, for some, like ”application”, this improvement can go as high as 0.22.

Additionally, we can further break down the mean sensitivity by the distance in sequence length properties of models, like we did previously in Fig. 9. In Fig. 13 we show the mean sensitivity achieved by combinations of models in 1oo3 schemes, based on the mean distance in sequence length for each of the malware types present in our dataset. As was the case when looking at the mean sensitivity of all possible combinations, a higher distance between the sequence length property of models in the same combination leads to an increase in sensitivity. Note that, for two types of malware, namely ”aptbackdoor” and ”rootkit”, higher distances of sequence length actually achieve a sensitivity of 1, meaning that all samples of these types are correctly detected. Keep in mind however, that his might just be due to the small number of samples of these two types (11 and 7 samples respectively).

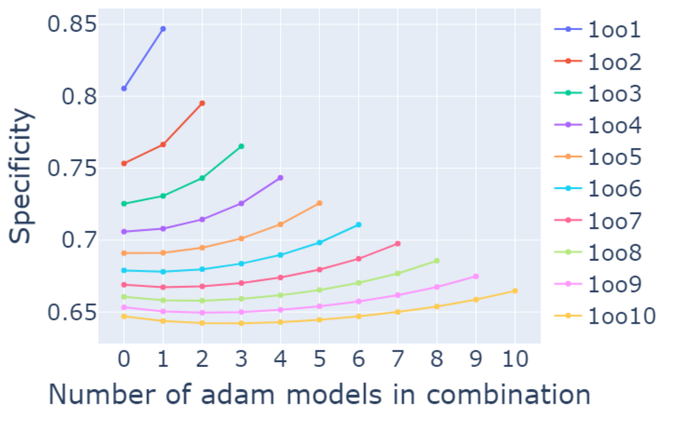
VI. Discussion, Conclusions and Further Work

In this research, we presented the analysis of the potential benefits of diversity from using multiple recurrent neural networks (RNNs) models for the classification of malware and benign samples. We used data from the authors of [9], which contained predictions generated by a set of models trained and tested against a dataset of over 4000 different samples, containing a mixture of both malware and benignware. The authors of [9] analysed the performance of each model individually. In this paper we presented the analysis of diverse adjudication schemes for these models.

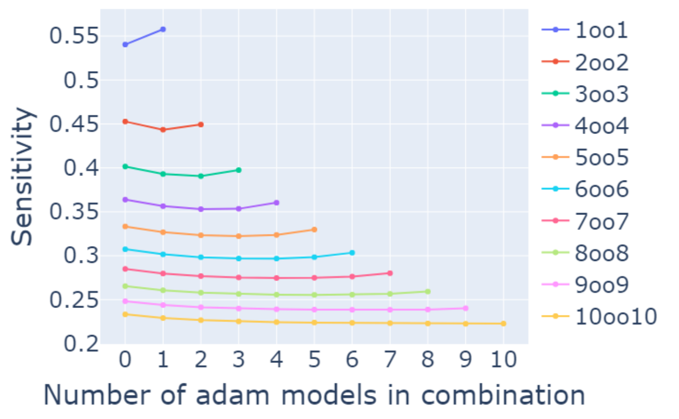
We have analysed possible configurations of schemes of up to 10 models, in 1ooN, NooN and simple majority adjudication schemes and found significant improvements to both sensitivity (correct classification of malware) and specificity (correct classification of benignware). In terms of sensitivity, 1oo10 schemes achieved a mean sensitivity of 0.83381, compared to the mean sensitivity achieved by single models of 0.54836 (an increase of 0.28545). The theoretical maximum value of 0.94753 was achieved in combinations as early as 1oo8 (the theoretical maximum value is not 1 for our dataset because there are a number of malware samples that are never correctly classified as such). For specificity, 10oo10 schemes improved the mean value, going from 0.82452 from individual models to 0.96203 (an increase of 0.13751). As we saw from the results present in section V.(b) improvements in sensitivity in 1ooN schemes do come with a high penalty to specificity, and vice-versa for NooN schemes, where improvements to specificity comes with a penalty for sensitivity. This prompted us to looked at the interplay between sensitivity and specificity in simple majority schemes (e.g. 3-out-of-5). For these cases, higher combinations of models do not strictly increase of decrease either performance metric. Instead the performances



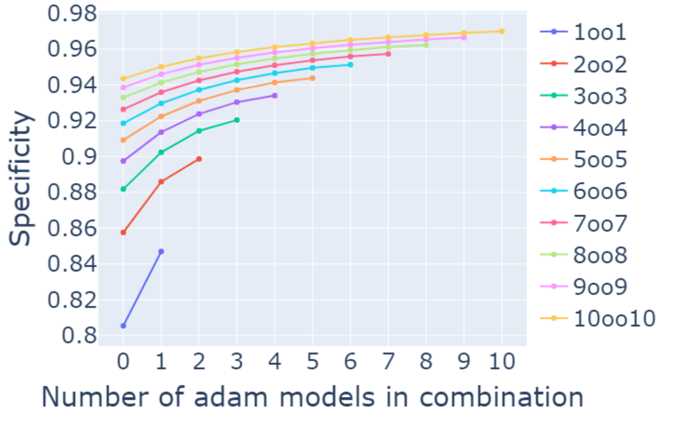
(a) Mean sensitivity based on optimiser combinations for 1ooN schemes



(b) Mean specificity based on optimiser combinations for 1ooN schemes



(c) Mean sensitivity based on optimiser combinations for NooN schemes



(d) Mean specificity based on optimiser combinations for NooN schemes

Fig. 11. Mean sensitivity and specificity for 1ooN and NooN schemes based on optimiser combination (x-axis indicates the number of models in that combination using an Adam optimiser)

of the various model combinations are brought closer together, eliminating the more extreme outliers in either direction, and in essence generating more predictable systems. This is the case for both sensitivity and specificity. We are currently looking into the use of other adjudication schemes that allow for more optimal interplay between sensitivity and specificity (see the last paragraph of this section).

Further to this, we have attempted to determine what causes diversity to emerge between different models and found that two hyperparameters lead to most of this diversity, the sequence length and optimiser properties of the models. For sequence length, the more varied the value the more diverse the models become, such that for combinations of N models, a wider range of sequence length values leads to an increase in sensitivity for 1ooN schemes, and an increase in specificity for NooN schemes. A similar situation happens with the optimiser used for each model, with models built with the ”adam” optimiser having a higher diversity amongst themselves.

Finally, we have taken a look at the benefits of the diversity between models to detect specific types of malware. We found that detecting most types of malware is improved with the use of multiple RNN models, with an average increase in sensitivity of 0.15 when going from single models to 1oo3 adjudication schemes, with the notable outliers of detecting ”rootkit” malware and ”application” malware, which were improved further, at 0.2 and 0.22 respectively. The diversity gains from sequence length and optimiser seem to have roughly the same impact across all types of malware.

An area in which we have ongoing work is in the use of a mechanism called ”optimal adjudication” ([37], [38]), which can produce optimal adjudication results for a given configuration, conditional on a loss value associated with the two types of failures possible (wrong classification of both malware and benignware). The use of optimal adjudication can potentially

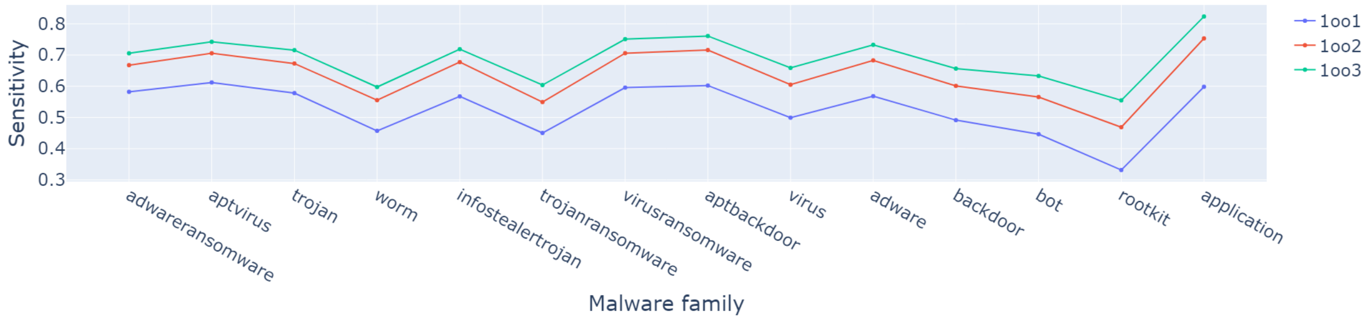


Fig. 12. Mean sensitivity by malware type (ordered by largest improvement from 1oo1 to 1oo3)

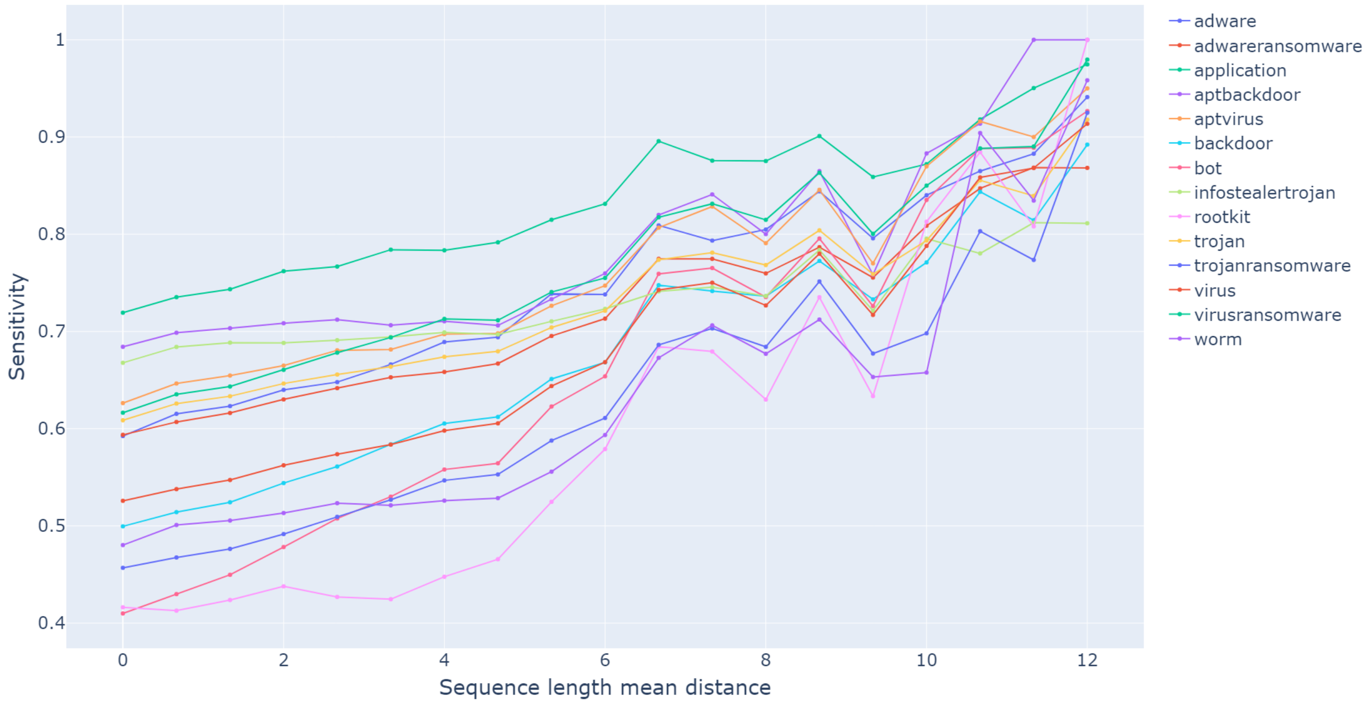


Fig. 13. Mean sensitivity by malware type based on sequence length mean distance (1oo3)

enhance our research in two major ways. Firstly, it is possible that the upper bounds for sensitivity and specificity may be achieved in smaller sized RNN combinations, as opposed to our current 1oo8 and 15oo15 requirements. Additionally, it may even be possible to reach perfect sensitivity or specificity using optimal adjudication, even though there are malware and benignware samples that are never correctly identified by any single model. We plan to investigate this for different malware types, as well as in new datasets.

REFERENCES

[1] S. O. Solutions, "Security onion," December, 2019. [Online]. Available: https://securityonion.net/

[2] H. Asad and I. Gashi, "Diversity in open source intrusion detection systems," in *Computer Safety, Reliability, and Security*, B. Gallina, A. Skavhaug, and F. Bitsch, Eds. Cham: Springer International Publishing, 2018, pp. 267–281.

[3] P. Bishop, R. Bloomfield, I. Gashi, and V. Stankovic, "Diversity for security: A study with off-the-shelf antivirus engines," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, Nov 2011, pp. 11–19.

[4] C. Collberg, "Code obfuscation: Why is this still a thing?" in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 173–174.

[5] H. Xu, Z. Chen, W. Wu, Z. Jin, S. Kuo, and M. Lyu, "Nv-dnn: Towards fault-tolerant dnn systems with n-version programming," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, June 2019, pp. 44–47.

[6] F. Machida, "N-version machine learning models for safety critical systems," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, June 2019, pp. 48–51.

[7] Y. Freund, R. E. Schapire *et al.*, "Experiments with a new boosting algorithm," in *icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[8] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] M. Rhode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Computers Security*, vol. 77, pp. 578 – 594, 2018.

[10] B. Littlewood and L. Strigini, "Redundancy and diversity in security," 09 2004, pp. 423–438.

[11] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Softw. Pract. Exper.*, vol. 44, no. 6, pp. 735–770, Jun. 2014.

[12] S. Singh, M. Cukier, and W. H. Sanders, "Probabilistic validation of an intrusion-tolerant replication system," in *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, June 2003, pp. 615–624.

[13] V. Gupta, V. Lam, H. V. Ramasamy, W. Sanders, and S. Singh, "Dependability and performance evaluation of intrusion-tolerant server architectures," vol. 2847, 09 2003, pp. 81–101.

[14] B. Littlewood and D. R. Miller, "Conceptual modeling of coincident failures in multiversion software," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1596–1614, Dec 1989.

[15] D. E. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12, pp. 1511–1517, Dec 1985.

[16] A. Avizienis and L. Chen, "On the implementation of n-version programming for software fault tolerance during program execution," 01 1997.

[17] P. Popov, A. Povyakalo, V. Stankovic, and L. Strigini, "Software diversity as a measure for reducing development risk," in *2014 Tenth European Dependable Computing Conference*, May 2014, pp. 106–117.

[18] R. Maclin and D. Opitz, "An empirical evaluation of bagging and boosting," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, ser. AAAI'97/IAAI'97. AAAI Press, 1997, p. 546–551.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[20] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. V. Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," 2018.

[21] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[22] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, Oct 2015, pp. 11–20.

[23] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Dec 2010, pp. 201–203.

[24] W. Huang and J. Stokes, "Mtnet: A multi-task neural network for dynamic malware classification," 07 2016, pp. 399–418.

[25] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in api calls with machine learning algorithms for malware detection," in *Proceedings of the 2Nd ACM Workshop on Security and Artificial Intelligence*, ser. AISec '09. New York, NY, USA: ACM, 2009, pp. 55–62.

[26] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *2010 5th International Conference on Malicious and Unwanted Software*, Oct 2010, pp. 23–30.

[27] M. Rhode, L. Tuson, P. Burnap, and K. Jones, "Lab to soc: Robust features for dynamic malware detection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Industry Track*. IEEE, 2019, pp. 13–16.

[28] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[29] B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert, "Adaptive semantics-aware malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 419–439.

[30] Z. Salehi, A. Sami, and M. Ghiasi, "Using feature generation from api calls for malware detection," *Computer Fraud & Security*, vol. 2014, no. 9, pp. 9–18, 2014.

[31] T. Kim, B. Kang, and E. G. Im, "Runtime detection framework for android malware," *Mobile Information Systems*, vol. 2018, 2018.

[32] R. Mosli, R. Li, B. Yuan, and Y. Pan, "Automated malware detection using artifacts in forensic memory images," in *2016 IEEE Symposium on Technologies for Homeland Security (HST)*. IEEE, 2016, pp. 1–6.

[33] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3422–3426.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[35] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[36] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," 2017.

[37] F. Di Giandomenico and L. Strigini, "Adjudicators for diverse-redundant components," in *Proceedings Ninth Symposium on Reliable Distributed Systems*, Oct 1990, pp. 114–123.

[38] D. M. Blough and G. F. Sullivan, "A comparison of voting strategies for fault-tolerant distributed systems," in *Proceedings Ninth Symposium on Reliable Distributed Systems*, Oct 1990, pp. 136–145.