Introduction

This project aims to identify a model which can accurately detect malware based on the input features.

Data Processing and feature selection

The original input data has 3394 features with a lot of missing data for some columns. It might take a long time to tune all these 3394 features and those missing values would likely not be predictive. Various features subsets were tested to determine the optimal number of features used for training. We used Random Forest to select the variables with importance larger than 0.01 (118 features selected). We also arbitrarily selected the first 1024 features as a lot of rows had missing values from the 1025th feature onwards. The 1025th feature was assigned as 1 if there were non-null values for more than 1024 features or 0 if there were less. We also selected the first 2049 features for model comparison. After selecting the features, those features with same value in all samples were removed. The results showed that the dataset with 1024 features outperformed the rest of the feature subsets for most models (Figure 1A). Missing values were filled with -1 since all features in the original dataset were positive. The data were then normalized by mean-centering and scaling to a standard deviation of 1. Train data was split into 80000 training set and 33636 validation set. Models were evaluated using validation AUC throughout the report.

Optimization strategy

Our strategy was to ensemble many small well-tuned multilayer perceptron (MLP) models for prediction. The best MLP architectures were first determined by using basic tuning parameters such as ReLU activation without batch normalization and learning rate decay. After the best MLP architectures were found, parameter tuning would be performed to optimize the model performance. These tuning steps would be performed on a low epoch number of 21 to save computation cost. Epoch number would then be increased gradually until performance plateaus to realize the optimal performance for the models. The models would then be ensembled by stacking with a support vector machine classifier to output the final predictions.

Architecture tuning

MLP with five different types of architecture were explored (Table 1 at end of the report).

a) Alternating layers with large and small number of nodes
b) Number of nodes gradually decreases over layers.
c) Number of nodes increases over layers before gradual decrease.
d) Multiple layers of 64 nodes or 32 nodes were used in the end.
e) Multiple layers of 512 nodes are deployed in the beginning.

There is no clear winner in the five categories and they yielded similar average AUC performance (Figure 1B). The top performers in each category were selected based on AUC and precision. Total 10 models (bold in Table 1) were selected for further tuning.

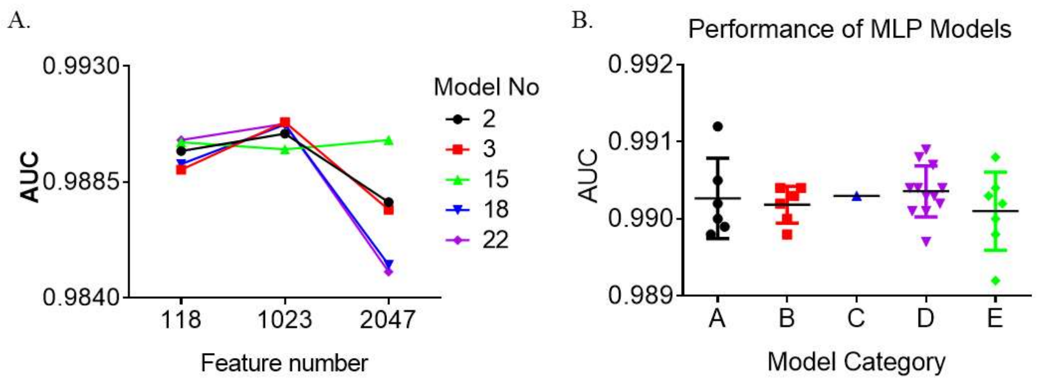

Figure 1: (A) Number of features used affected model performance. **(B)** Summary of performances of models in five different categories of MLPs shown in Table 1.

Activation function, normalization and learning rate

Due to mean-centering, the normalized inputs contain negative values which might be incompatible with ReLU activation. Sigmoid function with a working range that includes negative values were tested as the activation function (leaky ReLU could not be used in Keras 2.1.5). Batch normalization was also performed before the activation to keep input values small, ensuring that the values do not exceed the working range of the sigmoid function (± 5). Surprisingly, sigmoid activation together with batch normalization yielded lower AUC compared to ReLU alone for almost half of the models tested (Figure 2A). However, by using a learning rate schedule that reduced the learning rate to 10% every 2 epochs, sigmoid activation function with batch normalization outperformed ReLU for 9 models out of the 10 selected models (Figure 2A, 2B).

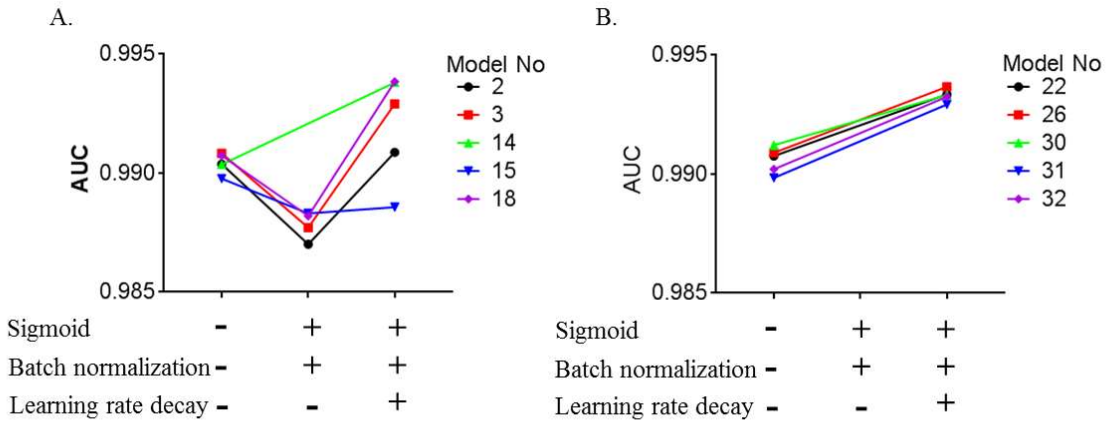

Figure 2: Effects of activation function, normalization and learning rate on model performances. **(A)** model 2, 3, 14, 15 and 18. **(B)** model 22, 26, 30, 31 and 32.

Regularization

We also tuned L2 regularization values for the 10 selected models. All models except model 15 and 18 had highest AUC when L2 = 0.00005. When L2 = 0.00001, there might be overfitting of training data and thus lower performance on validation data. When L2 = 0.0001, the penalty cost from the regularization is too high to fit training data properly thus yielding worse model performance (Figure 3A, 3B).

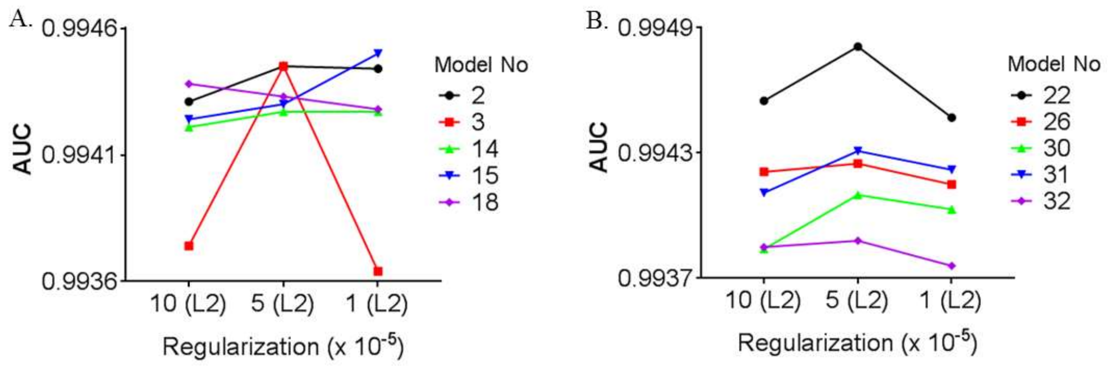

Figure 3: Effects of L2 regularization on model performance. **(A)** model 2, 3, 14, 15 and 18. **(B)** model 22, 26, 30, 31 and 32.

Epoch number

Epoch number 21, 51 and 101 and 151 were tested (Figure 4A, 4B). All models had a sharp increase in performance from 21 to 51 epochs. After that, performance of some models continues to increase from 51 epochs to 101 epochs. The performance of all models plateaued off after 101 epochs and further increase in epoch number didn't increase AUC. The models had reached their optimal performance and further training would not improve the model performance.

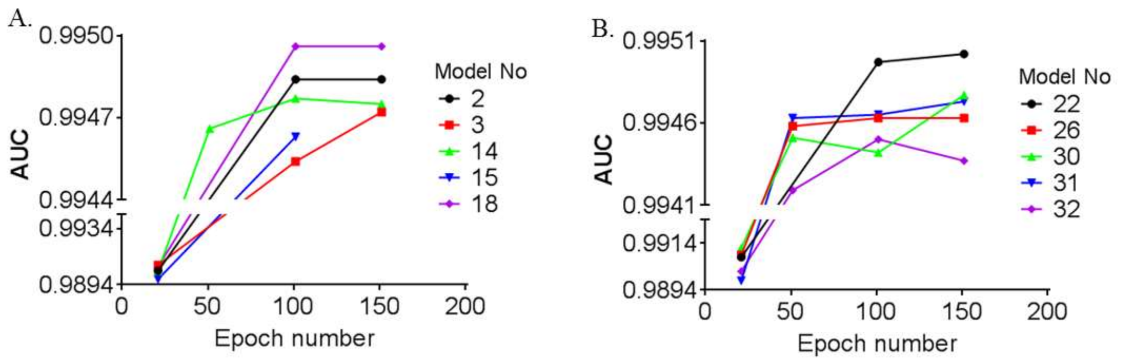

Figure 4: Increasing epoch numbers improves model performance. **(A)** model 2, 3, 14, 15 and 18. **(B)** model 22, 26, 30, 31 and 32.

Stacking

Stacking improved the model performance significantly. Support vector machine was used to stack selected models obtained after running 101 epochs and 151 epochs (Figure 5). Individual models after 101 epochs and 151 epochs have AUC between 0.994 to 0.995. Stacking models from training 101 epoch increased the AUC to 0.99604 and further to 0.99614 with 151 epochs. Stacking models from training 101 epochs and 151 epochs together further increased the AUC to 0.99621.

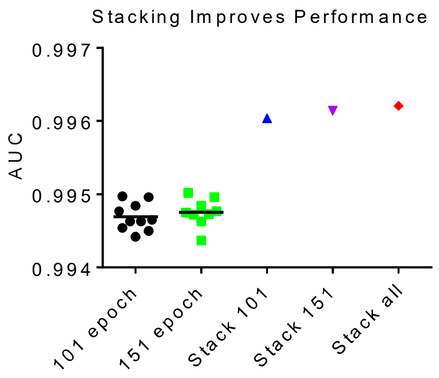

Figure 5: Stacking by SVM improves performance. Presented models for 101 epoch and 151 epoch are 2, 3, 14, 15, 18, 22, 26, 30, 31 and 32. Models stacked for 101 epoch are 2, 3, 14, 15, 22, 26, 30, 31 and 32. Models stacked for 151 epoch are 2, 3, 14, 18, 22, 26, 30, 31 and 32. Stack all stacks all models with 101 epoch and 151 epoch.

Future improvement

As malware is encoded in 8 bits or in binary coding, the features in the dataset might be better to be viewed as group of features to improve model performance. Using 1-D convolutional network with kernel size from 2 to 8 might be able to detect the high-level signatures hidden in the current features. A bi-directional RNN network could also be tested as it would consider the surrounding inputs when learning from the train data. Furthermore, variable input length like what is present in the data could be considered without resorting to truncation as was done in our method.

Category	**Model**	**Architecture**	**Precision**	**AUC**
A (Alternate)	10	1023, 512, 1023, 512, 256,128,64,32, 16, 8, 1	0.9750	0.9900
	28	512,64,512,64,512,64,1	0.9701	0.9905
	29	512,512,512,64,512,512,512,1	0.9688	0.9899
	30	**512,64,512,64,512,64,512,64,512,64,1**	**0.9707**	**0.9912**
	32	**512,64,64,512,64,64,512,64,64,512,64,64,512,64,64,1**	**0.9709**	**0.9902**
	31	**1023,64,1023,64,1023,64,1023,64,1023,64,1**	**0.9767**	**0.9898**
B (Gradual decrease)	1	512,256,128,64,32,16,8,1	0.9673	0.9902
	2	**1023, 512,256,128,64,32,16,8, 1**	**0.9683**	**0.9904**
	4	1023, 1023, 512,256,128,64,32,16, 1	0.9687	0.9903
	7	896,768,640,512,384,256,128,64,32,16,8,1	0.9690	0.9904
	9	1023, 512, 512, 512, 256,128,64,32, 16, 8, 1	0.9746	0.9900
	15	**1023, 1023, 256, 256, 128, 128, 64, 64, 16, 16, 1**	**0.9799**	**0.9898**
C (Increase then decrease)	8	1023,2048,4096,2048,1024,512,256,128,64,32,16,8,1	0.9760	0.9903
D (Multiple 64s)	**3**	**512,256,128,64,64,64,32,16,8,1**	**0.9718**	**0.9908**
	11	512,256,128,64,64,64,64,1	0.9753	0.9903
	19	512,256,64,64,64,64,64,1	0.9735	0.9903
	20	256,256, 64,64,64,64,64,1	0.9715	0.9901
	14	**512, 128,128, 128, 64, 64, 64,1**	**0.9721**	**0.9904**
	16	2046,1023,128,64,64,64,64,1	0.9670	0.9904
	18	**512,256,128,128,128,128,128,1**	**0.9780**	**0.9907**
	25	1023,1023,1023,256,256,64, 64,64,64,64,64,1	0.9739	0.9902
	5	512,256,128,128,128,64,64,64,32,32,32,16,8,1	0.9716	0.9901
	6	512,256,256,128,128,128,64,64,64,64,32,32,32,16,16, 8,1	0.9725	0.9897
	21	256,64, 64,64,64,64,64,64,64,64,64,64,1	0.9714	0.9904
	26	**1023, 64, 64, 64,64,64,64, 64,64, 1**	**0.9808**	**0.9909**
E (Multiple 512s)	13	512,512,512,1	0.9772	0.9904
	27	512,512,64,64,8,8,1	0.9684	0.9892
	22	**512,512,512,64,64,64,1**	**0.9709**	**0.9908**
	23	512,512,512,64,64,64,8,8,8,1	0.9705	0.9900
	12	512,512,512,512,512,512,512, 1	0.9694	0.9902
	17	1023,1023,1023,512,512,512,1	0.9731	0.9898
	24	4096,512,512,512,64,64,64,1	0.9738	0.9903

Table 1: Five categories of MLP tested. Selected models for stacking are shown in bold.