MtNet: A Multi-Task Neural Network for Dynamic Malware Classification

Wenyi Huang[1] and Jack W. Stokes[2]

[1] Pennsylvania State University
Information Sciences and Technology
University Park, PA 16802, USA
wzh112@ist.psu.edu
[2] Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA
jstokes@microsoft.com

Abstract. In this paper, we propose a new multi-task, deep learning architecture for malware classification for the binary (i.e. malware versus benign) malware classification task. All models are trained with data extracted from dynamic analysis of malicious and benign files. For the first time, we see improvements using multiple layers in a deep neural network architecture for malware classification. The system is trained on 4.5 million files and tested on a holdout test set of 2 million files which is the largest study to date. To achieve a binary classification error rate of 0.358%, the objective functions for the binary classification task and malware family classification task are combined in the multi-task architecture. In addition, we propose a standard (i.e. non multi-task) malware family classification architecture which also achieves a malware family classification error rate of 2.94%.

1 Introduction

PandaLabs recently reported that 27% of all of malware detected by their antivirus engine was first encountered in 2015 [16]. Malware authors continue to accelerate the automation of malware production using techniques such as polymorphism at an alarming rate. Clearly, automated detection employing highly accurate malware classifiers is the only option to combat this problem long term.

Recently, deep learning has led to significant improvements in diverse areas including object recognition in images [14] and speech recognition [8]. Broadly speaking, deep learning is a branch of machine learning which includes algorithms that learn a distributed feature representation of a training set using a neural network architecture composed of multiple non-linear hidden layers. For supervised deep learning algorithms where the training set includes labels, a deep learning classifier such as a deep neural network (DNN) can be trained to predict the label of unseen examples. DNNs are typically considered to be neural networks composed of two or more hidden layers while a neural network with a single hidden layer is known as a shallow neural network. Given the impressive vision and speech results, it is important that malware researchers explore different deep learning models to hopefully discover improved architectures for detecting malware.

Given the potential repercussions of installing malware on a corporate or personal computer, there have been many proposed solutions for automated malware classification [10]. Recently, researchers have been attempting to use deep learning models to improve malware classification. In 2013, Dahl et al. [7] first studied deep learning for malware classification in the context of dynamic analysis, and their best single neural network architecture has an error rate of 0.49%. Their architecture consists of a random projection layer to reduce the high dimensional (179 thousand) sparse binary input feature vector to a 4000 dimensional dense feature vector suitable for training a neural network. The authors found that adding a second and third hidden layer to the neural network did not improve the overall accuracy compared to a shallow architecture. Pascanu et al. [20] recently proposed a two component, dynamic analysis system for malware classification including a lower-level recurrent model, which learns a feature representation for API events, and a higher-level, potentially deep, classifier which uses the output of the recurrent model as features. The authors proposed eight different recurrent models, based on variants of either a recurrent neural network or an echo state network, and some of these models did learn a better representation for the input sequence compared to a bag of words model or a collection of trigrams. Similar to [7], the authors found that adding additional layers to the classifier again did not improve the overall accuracy presumably due to the small training set size of 65 thousand samples. Saxe and Berlin [21] proposed a static malware analysis classification system which consists of a two hidden layer DNN where the features are derived from the structure, including elements from the header, of a Windows portable execution (PE) file. However in this paper, the authors do not compare the results for their DNN with a shallow neural network or a DNN with more than two layers so we do not know if deep learning improves their classification rate.

While deep learning has achieved state-of-the-art classification results in speech recognition and visual object recognition, no one has been able to demonstrate any gains for deep learning applied to malware classification. In this paper, we propose *MtNet*, a new deep learning malware classification architecture which shows for the first time that deep learning offers a modest improvement compared to a shallow neural architecture. To achieve these results, *MtNet* includes several improvements over Dahl's architecture. Multi-task learning encourages the hidden layers to learn a more generalized representation at lower levels in the neural architecture. Our architecture also employees rectified linear unit (ReLU) activation functions and dropout for the hidden layers. ReLU activations and dropout were also used in [20] and [21], but the effects of these components were not analyzed. In our work, we study the contributions of these components and show that ReLU activation functions cut the number of epochs needed for training a binary malware classifier in half while dropout leads to significant reductions in the test error rate. When trained and tested on a dataset consisting of 6.5 million files these modifications allow *MtNet* to achieve a binary malware error rate of 0.358% and family error rate of 2.94% beating the previous best architectures by 26.17% and 19.21%, respectively. Contributions of our work include:

1. We propose and implement a novel multi-task neural network malware classification architecture. This architecture leads to modest gains for deep learning with a detection threshold of 0.5 where a file is predicted to be malware if the probability that file is malicious exceeds the probability that it is benign.
2. We conduct a deep learning study on an extremely large dataset trained with 4.5 million files and test the model with an additional 2 million files.
3. We demonstrate that dropout significantly reduces the error rate for both shallow and deep neural architectures.
4. We show that rectified linear activation functions allow a binary neural network model to be trained in half the number epochs compared to sigmoid activation functions which were used in previous work.

2 Deep Learning

To better understand deep learning, we next provide background on several key concepts. Figure 1 depicts a typical deep neural network architecture. A DNN usually consists of an input layer followed by several hidden layers and an output layer. The input layer consumes an input feature vector representing the object to be classified. The output layer is responsible for producing the class probability vector associated with the input vector. In total, the deep neural network predicts the class for the input vector.

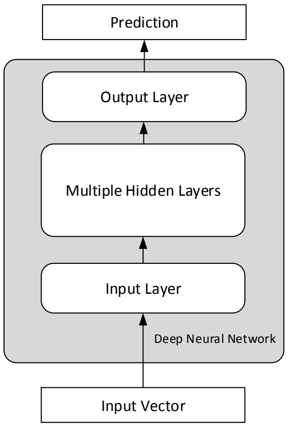

Fig. 1. A standard feed forward, deep learning architecture.

Hidden Units and Activation Functions: The basic component in a neural network is the hidden unit. A hidden unit takes an n-dimensional feature vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]$ from the input vector or the lower-level hidden units, and outputs a numerical output $y_j = f(\sum_{i=1}^{n} w_{ji}x_i + b_j)$ to the hidden units in higher layers or the output layer. For hidden unit j, y_j is the output, b_j is the bias term, while w_{ji} are the elements of a layer's weight matrix. The function $f(\cdot)$ is often referred to as the activation function which determines the hidden unit's output. The activation function introduces non-linearities to the neural network model. Otherwise, the network remains a linear transformation of its input signals.

Hidden Layers: A group of m hidden units forms a hidden layer which outputs a feature vector $\mathbf{y} = [y_1, y_2, \cdots, y_m]$. Each hidden layer takes the previous layer's output vector as the input feature vector and calculates a new feature vector for the layer above it:

$$\mathbf{y}_n = f\left(\mathbf{W_n}\mathbf{y}_{n-1} + \mathbf{b}_n\right) \tag{1}$$

where $\mathbf{y}_n$, $\mathbf{W_n}$, and $\mathbf{b}_n$ are the output feature vector, the weight matrix, and the bias of the nth layer. Proceeding from the input layer at the bottom of the DNN in Figure 1, each subsequent higher hidden layer learns a more complex and abstract feature representation which captures higher-level structure. The underlying idea of adding multiple layers is that these layers correspond to improved levels of abstraction or composition of the observed data.

Input and Output Layers: The lowest level of a deep neural network which receives the original feature vector is known as the input layer. The original feature vector is passed to the hidden layers from bottom to top and is transformed into a fixed-dimensional vector that the final layer can process. The final layer, which interacts with and presents the processed data, is called the output layer. The behaviour of the output layer depends on the problem we are solving. For example, in a classification task, the output layer transforms the last hidden layer's activation into a probability distribution that estimates the input sample's class. So far we have introduced the most basic components and concepts in deep learning. Next we consider deep learning's ability to improve the model's feature representation.

Feature Representation Learning: One of the promises of deep neural networks is that the model reduces the need for feature engineering. Instead, deep learning provides a way to automatically extract more complex, higher-level features derived from simple lower-level features. For example, in the case of object recognition of transportation vehicles in images, the lowest-level input layer consumes the raw pixel information from an image. The first hidden layer usually learns a set of edge-like features. Then, the second layer learns to combine the lower-level features from the first hidden layer to produce a slightly richer set of features. In our image recognition example, features extracted at higher levels might represent different types of components from the vehicles such as a door, wing, tire or handle bars. Finally, the output layer fine tunes the final classification based on the object labels allowing the system to distinguish between a car, an airplane, a motorcycle, and so on.

3 *MtNet* System

Figure 2 depicts the high-level overview for training the *MtNet* system and evaluating unknown files with the trained model. The top row provides the steps required for identifying the selected features and training the *MtNet* model, while the bottom row indicates the process for evaluating an unknown file given a set of selected features and the trained *MtNet* model. For training, raw data is extracted from labeled files during dynamic analysis by a modified version of a production anti-malware engine. Unlike in-depth emulation executed on a fully capable virtual machine (VM) such as Anubis [4], the anti-malware engine used in this study only provides lightweight emulation of the operating system and tries to coax the file into execution. Since anti-malware engines are designed to quickly scan unknown files for viruses, many more files can be evaluated with this method than using full VMs. Once the raw data has been collected from the labeled files for the training set, feature selection training is performed to produce the final sparse binary features (*File Extracted Features*) required for training *MtNet*. Next, the *MtNet* model is trained for two tasks including binary classification which predicts whether an unknown file is malicious or benign and 100-class family classification which predicts if the file belongs to one of 98 important families, a generic malware class, or the benign class. In our data, analysts provide labels for tens of thousands of individual malware families. However, they selected 98 families for the family classifier based on their severity and prevalence of infection. Files in the long tail belonging to the remaining families are assigned to the generic "Malware" class. All legitimate files belong to the "Benign" class. After training, the *Selected Features* are then used to restrict the features extracted by emulating unknown files and these *File Extracted Features* can then be evaluated by the trained *MtNet* model. The *MtNet* binary prediction score is used to automatically classify the unknown file as either malicious or benign. Likewise the family classifier attempts to assign a specific family label to the unknown file. We next consider some of these steps in more detail.

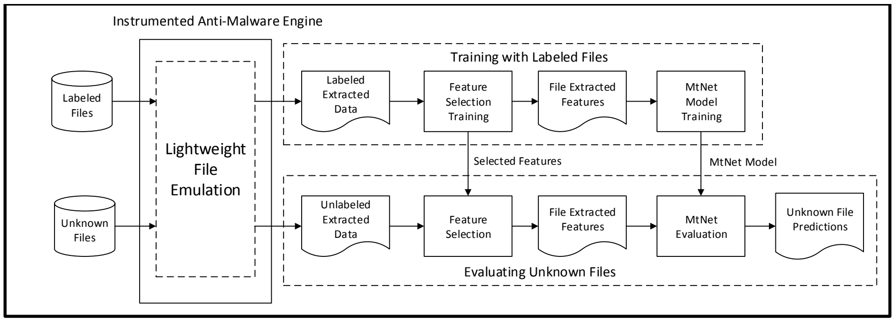

Fig. 2. High-level overview of *MtNet* training and unknown file evaluation.

Dataset: We were provided a large corpus of labeled, raw data by analysts from the Microsoft Corporation which was extracted from 6.5 million files. We believe this is the largest dataset used in a published malware classification study. Among this data collection, 2.85 million examples were extracted from malicious files and 3.65 million from benign files. The set of malicious files contained 1.3 million belonging to the 98 malware families and 1.55 million from the generic malware class. We randomly selected 4.5 million examples for training and 2.0 million for a hold out test set. All of the samples were scanned with a single combination of the anti-malware engine software and signature set. This dataset allows us to measure the performance of our system without introducing noise from varying anti-malware engine and signature set updates. Malicious files are labeled by professional analysts and anti-malware engine detections. The benign file collection is used in a production environment to prevent false positives by the anti-malware engine and was obtained either directly from legitimate companies or downloaded from verified web sites.

Features: Much research in the area of malware classification has focussed on improved feature generation. The underlying strategy is that malware experts handcraft potentially complex features using domain knowledge which hopefully leads to better overall classification performance. Deep learning takes the opposite approach and instead tries to learn the distributed feature representation from the raw input data. Just as in object recognition which learns from the raw pixels, we use low-level features extracted from dynamic analysis of the file as input for training.

For each executable file which is emulated by the anti-malware engine, two sets of raw information are extracted: a sequence of application programming interface (API) call events plus their parameters and a sequence of null-terminated objects recovered from system memory during emulation. A large percentage of malicious files are packed. During the unpacking process, null-terminated objects are often written to system memory by the malware. We find that the majority of the null-terminated objects are indeed unpacked strings but a few correspond to individual code fragments.

For the API and parameter stream, we use a many-to-one mapping to represent the API events. In the Windows software environment, there are multiple APIs which can be used to achieve the same objective. For example, three different ways to create a file include calling the CreateFile() method from user mode, the ZwCreateFile() method from kernel mode, or the fopen() call from C. All three of these create file API calls are mapped to a single higher-level CreateFile event. In total, there are 114 such high-level API events in our data.

Three sets of sparse binary features are derived from the two data sources. A sparse binary feature is set if the feature is present in the data; we do not use feature counts in $MtNet$ to prevent missed detections due to attackers polymorphically varying the number of critical features. The presence or absence of the null-terminated objects are used directly as one of the feature sets. Two additional feature sets are derived from the API and parameter stream. The first feature set is derived from each unique combination of high-level API event and one individual input parameter setting. As a result, several sparse binary features are generated from each API call. The second feature set consists of trigrams of API events. An API trigram event feature is generated by the unique combination of three consecutive API events. A trigram feature provides a small amount of context for each central API call.

Feature Selection: The combined feature set consisting of null-terminated tokens, API event plus parameter value, and API trigrams contains millions of potential features. In order to reduce the input space so that it can be classified by a deep neural network, we perform feature selection using mutual information [17] to generate features that best characterize each class. The output of the feature selection process is a ranked set of 50,000 features which is input to the *MtNet* system. The 50,000 features are initially selected during training. Later, these features are applied when evaluating an unknown file.

4 Multi-Task Neural Malware Classification

Figure 3 depicts the architecture of the proposed deep, multi-task malware classification model. We seek to use the features described in the previous section to identify whether unknown files are malicious or benign. We also want to classify the malicious files into different malware families with 100 classes.

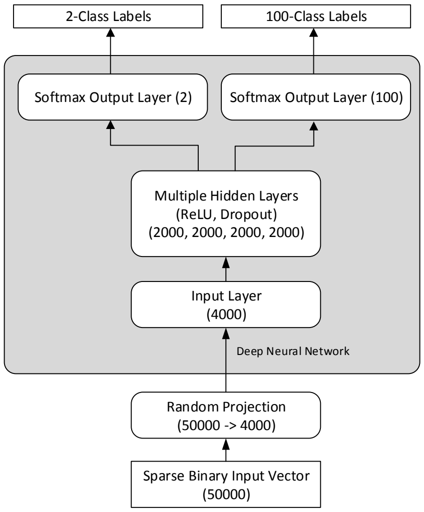

Fig. 3. Proposed deep model for multi-task learning.

4.1 Random Projections

After feature selection, the dimension of the input feature vector is reduced to 50,000. However, training a neural network with such a large input dimension is still computationally prohibitive. To overcome this problem, this original input feature vector must be projected to a lower dimensional subspace which then serves as the input vector to the neural network. Dahl et al. [7] experimented with principal component analysis (PCA) but were only able to project the original data down to 500 dimensions due to its $\mathcal{O}(N^3)$ computational requirements. Therefore to further reduce the data size to a suitable dimension for the neural network's input layer, we use the random projection technique [15] which is also used in [7]. The core idea of random projections, which has been shown to work well in practice [15], is that a sparse matrix that is randomly initialized can be used to project the original input feature vector to the reduced dimension subspace. The sparse random projection matrix R is initialized with 1 and -1 as

$$Pr(R_{i,j} = 1) = Pr(R_{i,j} = -1) = \frac{1}{2\sqrt{d}} \tag{2}$$

where d is the size of the original input feature vector. For $MtNet$, the dense, projected feature space of the random projection is reduced to 4,000 as in [7]. With $d = 50{,}000$ in our model, R is highly sparse and includes 0.22% of its values set to 1 and another 0.22% of its values set to -1. The remaining 99.56% of the values in the sparse, random project matrix have an implied value of 0.

4.2 Deep Neural Network

We next train a deep feed-forward neural network from the projected features for malware classification. The network architecture is identical to that described in Section 2 with the following details.

Normalized Input: Before inputting the feature vectors to the deep neural network, we first normalize the input vector so that every dimension has zero mean and unit variance. The normalized input makes the network training converge faster.

ReLU: The sigmoid activation function, used in [7], and the tanh activation function typically exhibit the vanishing gradient problem which makes the deep neural networks hard to train [9]. To overcome this problem, we use the rectified linear unit (ReLU) activation function for each layer. The ReLU function is defined as:

$$f(\gamma) = max(0, \gamma) \tag{3}$$

for any input value γ. It not only solves the vanishing gradient problem but also accelerates the convergence of stochastic gradient descent compared to the sigmoid and tanh activation functions.

Dropout: Dropout [24] is a regularization technique proposed for training deep neural networks. The core idea is that when updating a hidden layer, the algorithm randomly chooses not to update (i.e. "dropout") a subset of the hidden units. The intuition for dropout is that when randomly zeroing out hidden units in a layer, the network is forced to learn several independent representations of the patterns with identical input and output. In our model, we use dropout for all hidden layers of the neural network.

Loss function: The deep neural network learns different feature representations at each layer. The output layer implemented with the softmax function is used to output the categorical probability distribution. In our case for binary classification, the output is two dimensional representing malware and benign, while for the family classification task, the output size is 100 representing the different malware families, the generic malware class, and the benign files. To fine tune the deep model, we use the cross entropy loss function to quantify the quality of the neural network's classification results. The cross entropy loss is defined as

$$L_C(\theta(\mathbf{x})) = -\sum_{c \in C} g_c(\mathbf{x}) \log \theta_c(\mathbf{x}) \tag{4}$$

where $\mathbf{x}$ is the input feature vector, c is the class, C is the collection of classes to predict, $\theta(\mathbf{x})$ is the probability distribution output by the deep neural network, and g is the ground truth distribution.

Multi-Task Learning: In order to improve the generalization of the deep model, we train both the 2-class classification output and the 100-class classification output together with the same neural network. The multi-task model shares the same feature learning in the hidden layers, while the two top-level output softmax layers project these learned features into 2- or 100-dimensional vectors to calculate the probability distribution for each task. We define the multi-task loss function to be a weighted sum of each of the individual loss functions,

$$L_M(\theta(\mathbf{x})) = \alpha_1 L_2(\theta(\mathbf{x})) + \alpha_2 L_{100}(\theta(\mathbf{x})) \tag{5}$$

where the multi-task weights are α_1 and α_2, and $\alpha_1 + \alpha_2 = 1.0$. The two tasks are trained simultaneously with mini-batch stochastic gradient descent and back-propagation, and the gradients at each layer are updated with respect to the weight of each task.

5 Experimental Results

In this section, we evaluate the performance of our multi-task *MtNet* model, along with several baseline models, and seek to answer several questions about malware classification with deep learning including the following. *Does adding additional hidden layers in a deep neural network improve binary and family classification? Do larger datasets allow deep learning to help improve malware classification accuracy? How do the various deep learning components affect the classification accuracy? Can we improve detection rates at extremely low false positive rates?*

We implemented all models in this section, including the baseline system proposed in [7], using the computational neural toolkit (CNTK)[1]. The sparse, binary feature vectors for each file are extracted as described in Section 4. For all neural network models, we fix the input layer size to 4,000 and the hidden layer size to 2,000 for all layers. We choose the input layer size to match [7], whereas the hidden layer size is chosen by hyper-parameter tuning. The mini-batch size for stochastic gradient descent (SGD) is set to 300 samples, and the initial learning rate for mini-batch SGD is initialized to 0.01. The momentum of the gradient update is set to 0.9 to avoid getting trapped in a local minimum. We dynamically adjust the learning rate during training. If the loss does not drop after the current epoch, we reload the previous epoch's model, halve the current learning rate, and retrain the model for this epoch. After each epoch, the entire dataset is shuffled so that the data samples in each mini-batch are randomly selected. We train each model until convergence but no more than 200 epochs. Each model is trained and tested on a single NVIDIA Tesla K40 GPU. To evaluate the *MtNet* model, we report the test error rate which is defined as the ratio of misclassification in the entire test dataset. During test, an unknown file is predicted to belong to each class represented in the softmax layer. For binary classification, a file is predicted to malicious if $P(c = malware|\mathbf{x}) \geq P(c = benign|\mathbf{x})$ which corresponds to a detection threshold of 0.5 in Tables 1 and 3. In addition we also plot the receiver operating characteristic (ROC) curves of different models.

5.1 Comparison of the Baseline and Single-Task Baseline Models

Before investigating the performance of the multi-task *MtNet* model in the next section, we first evaluate the test error rates for a hold out test set on two baseline architectures for both binary and malware family classification. Tables 1 and 2, respectively, summarize the results of our best single-task deep models compared with the baseline method proposed in [7]. For reference, the second column presents the test error rates in [7] for up to three hidden layers originally evaluated using their implementation and dataset. The third column presents the results from our re-implementation of their architecture in CNTK and trained and tested with our new dataset. The number of epochs required for training to converge is listed in column 4. It should be noted that our CNTK implementation of Dahl's previously proposed models is independent and provides confirmation of their earlier results. In the final two columns, we present the results for the single task baseline versions of the *MtNet* model depicted in Figure 3 trained and tested with our dataset. For example, the single-task baseline model for binary classification, whose results are found in Table 1, only includes the top left softmax output layer. Similarly the single-task malware family classification model, whose results are listed in Table 2, only uses the righthand softmax output layer. Both of these baseline models employ rectified linear units and dropout.

Comparing the results for Dahl's model in [7] with our implementation of their model for binary classification in Table 1, we see that the best performing baseline model in our implementation uses three hidden layers compared to one in the original study. Several factors changed between these two experiments. The training and test set sizes were essentially doubled, the number of features and families both decreased, and the underlying implementation was completely changed. In addition, only family-based models were trained in [7], and the binary classification results were computed based on whether or not the predicted family was malicious or benign. In this study, the 2-class models were trained with the true binary labels. It is interesting that the lowest test error rates for the two implementations are essentially identical (i.e. 0.49% for their implementation and 0.4845% for ours). In addition to the hidden layer size, this single-task version of *MtNet* differs from [7] in two aspects: the sigmoid activation function is replaced with the rectified linear activation function and dropout is included. For both binary and family classification, our single-task models significantly improve the baseline classification results by 23.98% and 19.21%, respectively. These results indicate that switching to the ReLU activation function and adding dropout help the deep model to learn a better feature representation of the file for classification. In both tables, we also show the number of epochs needed to reach convergence. We found that although adding dropout to the hidden layers generally increases the number of required training epochs, ReLU accelerates the convergence of the mini-batch stochastic gradient descent process for binary classification. Compared to sigmoid activation functions, rectified linear activation functions significantly reduce the number of iterations required for training a binary classifier.

	Baseline Model (Original Results [7])	Baseline Model (Our Data)		Single Task Model (Our Data)	
Layers	Test Error(%)	Test Error(%)	Epoch	Test Error(%)	Epoch
1	0.49	0.5906	190	0.3711	64
2	0.50	0.4882	186	0.3702	82
3	0.51	0.4845	200	0.3686	77
4		0.4934	200	**0.3683**	81

Table 1. Comparison of two implementations of the baseline model versus our best single-task baseline model on 2-class binary classification.

	Baseline Model (Original Results [7])	Baseline Model (Our Data)		Single Task Model (Our Data)	
Layers	Test Error(%)	Test Error(%)	Epoch	Test Error(%)	Epoch
1	9.53	3.633	152	**2.935**	124
2	9.55	3.652	70	2.983	130
3	9.74	3.715	96	2.982	122
4		3.795	96	2.970	146

Table 2. Comparison of two implementations of the baseline model versus our best single-task baseline model on 100-class family classification.

5.2 Multi-Task Results

Table 3 compares the test error rates for the multi-task models with their single-task counterparts. Using hyper-parameter tuning, we set the weights for the binary classification task to $\alpha_1 = 0.8$ and for the family classification task to $\alpha_2 = 0.2$. We observe that for binary classification, classifiers trained with the multi-task models consistently improve the error rate. However, the multi-task, family classification models perform worse than the single-task variants. Compared to the baseline results shown in Table 1 and Table 2, we observe that both classifiers obtain significant improvements. While the family test error rate remains at 2.935% with a 19.21% improvement compared to the baseline result, the multi-task binary test error rate drops further to 0.3577% with a relative improvement of 26.17%.

	2-Class Test Error(%)		100-Class Test Error(%)	
Layers	Multi-Task	Single-Task	Multi-Task	Single-Task
1	0.3657	0.3711	2.935	**2.935**
2	**0.3577**	0.3702	3.025	2.983
3	0.3618	0.3686	3.026	2.982
4	0.3655	0.3683	3.070	2.970

Table 3. Test error rates for multi-task training vs. single-task training on 2-class and 100-class classification.

In Figure 4, we compare the ROC curves at very low false positive rates with $\alpha_1 = 0.8$ and $\alpha_2 = 0.2$. Although we do see some improvement in Table 3 for binary classification by adding additional layers, the 1- and 2-layer networks offer comparable performance for very low false positive rates.

In Figure 5, we compare the ROC curves for binary classification for the single-task model with two different *MtNet* models using $\alpha_1 = 0.8$ and $\alpha_1 = 0.9$. All models have a single hidden layer. This figure indicates that the multi-task *MtNet* model outperforms the single-task model at very low false positive rates; including the family classification task helps regularize the neural network model to learn better feature abstractions for binary classification.

5.3 Model Parameter Contributions

We perform hyper-parameter tuning on two additional parameters in *MtNet*, the dropout rate and the multi-task mixing weight, and measure their contribution to the *MtNet* model test error.

Dropout Rate: Figures 6 and 7, respectively, show the test error rates for binary and family classification with different dropout settings. It is clear that dropout is the main contributor to the improvement in classification accuracy in both cases. The best dropout setting for binary classification is 0.25, where

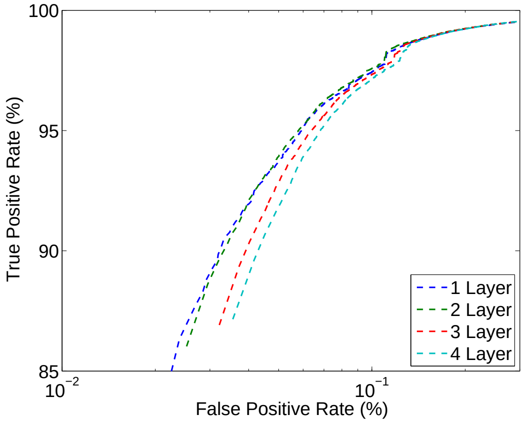

Fig. 4. ROC curves for the best performing multi-task *MtNet* at very low false positive rates.

MtNet is able to learn a better feature representation with more hidden layers. Although the 0.25 dropout rate also improves the family classification test error rate significantly, adding more hidden layers fails to learn better feature representations for this task.

Multi-Task Weight: We next vary the multi-task weight corresponding to binary classification task α_1, in (5), and measure its impact on *MtNet's* binary classification error rate in Figure 8 and family classification error rate in Figure 9. From Figure 8, we observe that as α_1 increases, the test error decreases until $\alpha_1 = 0.8$ for all models. Whereas in Figure 9, we observe that the error rate of the family classification models generally increases as α_1 increases. Note that setting $\alpha_1 = 1$, in the multi-task model, is equivalent to the single-task binary classification model, and setting $\alpha_1 = 0$ corresponds to the single-task family classification model. These two figures show that multi-task learning favors the task with the larger weight. In summary when $\alpha_1 = 0.8$, multi-task modelling significantly improves the binary classification result with the help of the family class labels.

5.4 Dataset Size and Deep Learning

Based on the published results, we believe this is the largest malware classification experiment run to date. We essentially doubled the number of training and test samples compared to [7]. However compared to the results reported in [7], our CNTK implementation of the baseline model shows similar test error rates. In addition, although we found modest gains by increasing the number of layers in *MtNet* in the case of 2-class binary classification, we did not find significant improvements using deep learning compared to other domains such as object and speech recognition. As a result, we do not believe that adding even more sam-

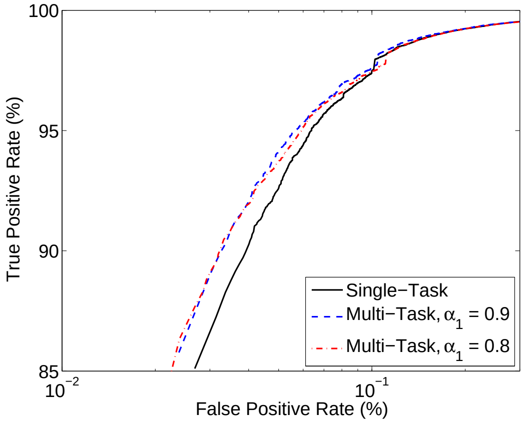

Fig. 5. ROC curves for binary classification by multi-task *MtNet* model versus single-task model for different binary classification task weights α_1.

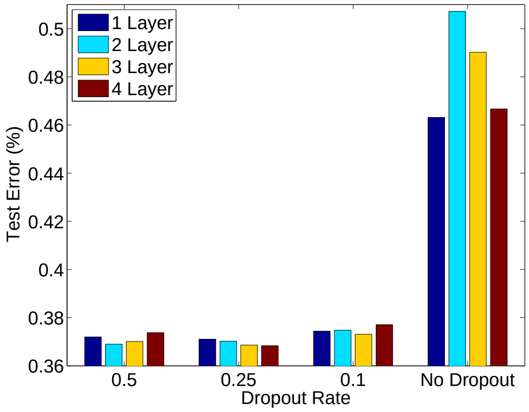

Fig. 6. *MtNet* test error rates for binary classification with different dropout rates of 0.5, 0.25, 0.1 and without dropout.

ples to our training set will enable deep learning to offer significant performance increases for the dynamic analysis features investigated in this study.

5.5 Training and Testing Efficiency

An important aspect of training large-scale neural network architectures is the training and testing efficiency. Table 4 presents the time required to train and test the large-scale *MtNet* multi-task, deep neural networks for up to four layers. These times are listed in (hours:minutes). The reason that the training times are similar for the 3 and 4 layer networks is because the 3 layer network trained for

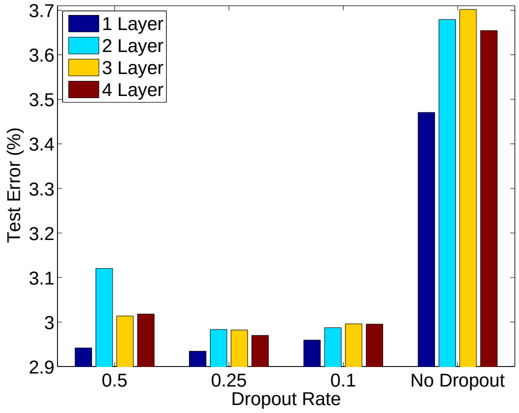

Fig. 7. *MtNet* test error rates for family classification with different dropout rates of 0.5, 0.25, 0.1 and without dropout.

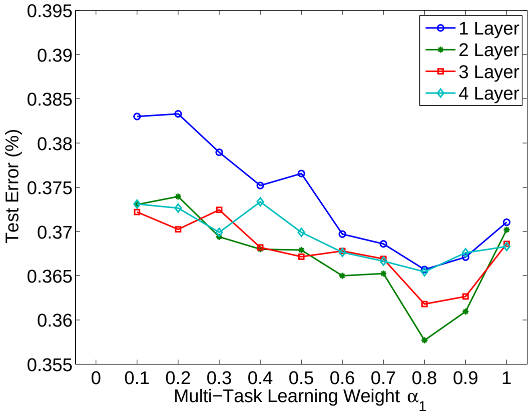

Fig. 8. *MtNet* test error rates for 2-class classification with different values of the multi-task learning weight α_1.

181 epochs before the early stopping criterion halted training while the 4 layer network only required 144 epochs. The most time consuming aspect of training and testing the system is the extraction of the data which required approximately 2 weeks on a single computer.

6 Discussions

We now discuss several aspects of our proposed *MtNet* multi-task, neural classification system and then consider how attackers may attempt to evade its detection.

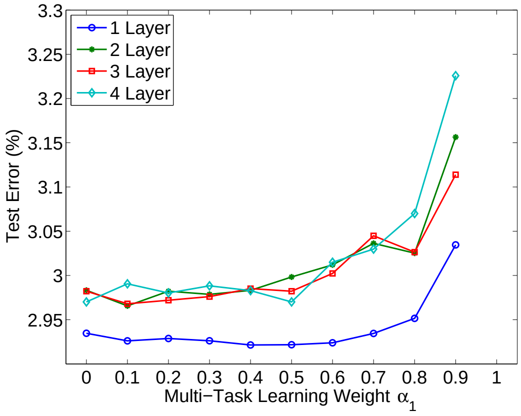

Fig. 9. *MtNet* test error rates for 100-class classification with different values of the multi-task learning weight α_1.

Layers	Training Time	Test Time
1	06:58	01:34
2	12:34	02:09
3	18:08	02:41
4	18:12	02:32

Table 4. Training and test times required in (hours:minutes) for evaluating up to four layers in the *MtNet* DNN malware classifier.

Achieving improvements using deep learning for malware classification is extremely challenging. The primary issue is that the classification accuracy for a neural network architecture with millions of files is already so good that it is difficult for additional layers to offer significant performance increases. For example, the best accuracy from the previous large-scale malware classification study [7] for a single neural network with one hidden layer is 99.51%. Figure 4 indicates that the ROC curves for this dataset are beginning to approach the ideal classifier. Although we do find some gains in Tables 1 through 3, there is not much room for significant improvement in the binary classification results by including additional hidden layers. In contrast, the detection error rates for object recognition and speech recognition were much higher prior to the significant improvements using deep learning. However, this study confirms that using other types of algorithmic techniques from the deep learning literature, such as dropout and rectified linear unit activation functions, can further improve the test error rate of a neural network malware classifier.

Even though it is somewhat disappointing that we cannot obtain significant improvements in malware classification using deep learning, this result has a major benefit. Shallow networks can evaluate unknown files more quickly because the computational complexity for each hidden layer is $\mathcal{O}(H^2)$ where H is the size of the hidden layer. As a result, we can scan more files with a shallow neural network than with a DNN.

All of the samples were analyzed at the same time with the same version of the anti-malware engine. Thus we expect the performance to be worse when analyzing new samples in a production setting where the anti-malware engine and its signatures are updated frequently.

As with many other malware detection systems, $MtNet$ is susceptible to attacks and can be evaded. $MtNet$ relies on dynamic analysis of a PE file. As such, the well known anti-emulation attack where the malware detects that it is being emulated and halts any malicious activity [3] will prevent $MtNet$ from detecting the malicious file. In addition, $MtNet$ is also vulnerable to the recently reported attack for deep neural networks proposed by Papernot, et al. [19]. In this attack, the authors construct adversarial samples and demonstrate that all ten digits in the MNIST database can be altered in such a way as to confuse a DNN classifier thereby producing any other digit. This attack is based upon computing the forward derivative of the DNN evaluated at the proposed initial input sample. Given this attack, the *MtNet* classifier should not be run on the client computer where the parameters of the DNN can be recovered by reverse engineering. However assuming a secure machine learning infrastructure with no intrusions, $MtNet$ can still be run on the backend to evaluate unknown files.

7 Related Work

Previous research most closely related to the *MtNet* system broadly falls into two main areas, deep learning and automated malware classification.

Neural networks have been explored for over three decades. Deep learning has recently become popular in many areas such as computer vision [14] and speech recognition [8]. Training deep models was not practical until the recent growth of computational power and large datasets. Newly proposed techniques such as dropout [24], and rectified linear units [18] solved several problems such as overfitting and the vanishing gradient problem. The multi-task learning approach [6] has recently gained popularity among deep learning models. It usually leads to a better primary task model when training simultaneously with other related tasks. Multi-task learning has been adapted in several applications such as text recognition [11] and speech recognition [23].

Given the problems associated with stolen credentials and data exfiltration, malware classification has been an active research area since 1994. Idika and Mathur [10] present a good overview of malware classification. Kephart et al. [12] were the first to use neural networks for malware classification. Later important malware classification studies include the works by Schultz et al. [22] and Kolter et al. [13] . Random projections were first proposed for malware classification by Atkinson [2].

A few researchers have started to explore deep learning architectures for malware classification. Dahl et al. [7] proposed a simple feed-forward neural network with random projections [15] to learn from a selected feature set extracted from the executable files. Dahl's shallow neural architecture is the current best performing malware classification model in terms of binary and family classification accuracy, but the deep models fail to improve the classification accuracy in their study. Our proposed model is closely related to Dahl's architecture [7]. We utilize multi-task learning and recent deep learning techniques which allow our deep model to outperform their model. Benchea and Gavrilut [5] combine a Restricted Boltmann Machine (RBM) with a One-Sided Perceptron for detecting malware. Their study is quite large consisting of over 1.2 million files although only 31,507 are malicious. An RBM is an unsupervised method for learning a stochastic neural network. It learns one set of weights from an input layer to a single hidden layer. Dahl et al. [7] found that pre-training their neural network classifier with an RBM slightly degraded the performance. Recurrent neural networks and echo state networks have been used to analyze executable files to identify malware [20]. However, recurrent models are computationally expensive when trained with many files and long sequences. Finally, a static analysis-based DNN was proposed by Saxe and Berlin [21].

8 Conclusions

In this paper, we propose and implement several different binary and family malware classifier architectures. The best binary classifier employs multi-task learning for the binary and family malware classification tasks. In particular, multi-task learning improves the classification results for extremely low false positive rates under 0.07%. The best performing two-class, binary classification architecture in Table 3 uses two hidden layers and multi-task learning while a shallow, multi-task network performs best for family classification. These results are achieved using rectified linear unit activation functions and dropout. Including dropout is the key to the majority of the accuracy improvement compared to Dahl's architecture, and rectified linear units reduce the number of epochs required for training by almost half. Given these results, we believe that training neural network architectures with millions of files offers the best overall performance for malware classification.

Acknowledgements: The authors would like to thank Mady Marinescu with helping in the data collection. We also thank our shepherd Juan Tapiador and the anonymous reviewers for their very valuable feedback.

References

1. Agarwal, A., Akchurin, E., Basoglu, C., Chen, G., Cyphers, S., Droppo, J., Eversole, A., Guenter, B., Hillebrand, M., Hoens, R., Huang, X., Huang, Z., Ivanov, V., Kamenev, A., Kranen, P., Kuchaiev, O., Manousek, W., May, A., Mitra, B., Nano, O., Navarro, G., Orlov, A., Padmilac, M., Parthasarathi, H., Peng, B., Reznichenko, A., Seide, F., Seltzer, M.L., Slaney, M., Stolcke, A., Wang, Y., Wang, H., Yao, K., Yu, D., Zhang, Y., Zweig, G.: An introduction to computational networks and

the computational network toolkit. Tech. Rep. MSR-TR-2014-112 (August 2014), https://github.com/Microsoft/CNTK

2. Atkison, T.: Applying randomized projection to aid prediction algorithms in detecting high-dimensional rogue application. In: Proceedings of the Annual Southeast Regional Conference (ACMSE) (2009)
3. Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., Vigna, G.: Efficient detection of split personalities in malware. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2010)
4. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A tool for analyzing malware. In: Proc. of 15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR) (2006)
5. Benchea, R., Gavrilut, D.T.: Combining restricted boltzmann machine and one side perceptron for malware detection. In: International Conference on Conceptual Structures (ICCS). pp. 93–103 (2014)
6. Caruana, R.: Multitask learning. Machine learning 28(1), 41–75 (1997)
7. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3422–3426. IEEE (2013)
8. Hinton, G., Deng, L., Yu, D., rahman Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., Dahl, G., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition. In: IEEE Signal Processing Magazine. vol. 29, pp. 82–97 (2012)
9. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kolen, J.F., Kremer, S.C. (eds.) A field guide to dynamical recurrent neural networks. IEEE Press. Wiley-IEEE Press (2001)
10. Idika, N., Mathur, A.P.: A survey of malware detection techniques. Tech. rep., Purdue Univ. (February 2007), http://www.eecs.umich.edu/techreports/cse/2007/CSE-TR-530-07.pdf
11. Jaderberg, M., Vedaldi, A., Zisserman, A.: Deep features for text spotting. In: Computer Vision–ECCV 2014, pp. 512–528. Springer (2014)
12. Kephart, J.O.: A biologically inspired immune system for computers. In: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems. pp. 130–139. MIT Press (1994)
13. Kolter, J., Maloof, M.: Learning to detect and classify malicious executables in the wild. In: Journal of Machine Learning Research (JMLR). pp. 2721–2744 (2006)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
15. Li, P., Hastie, T.J., Church, K.W.: Very sparse random projections. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ICDM). pp. 287–296 (2006)
16. Lopez, M.: 27% of all recorded malware appeared in 2015. http://www.pandasecurity.com/mediacenter/press-releases/all-recorded-malware-appeared-in-2015/ (2016)
17. Manning, C.D., Raghavan, P., Schutze, H.: An Introduction to Information Retrieval. Cambridge University Press (2009)
18. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 807–814 (2010)

19. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swamix, A.: The limitations of deep learning in adversarial systems. IEEE European Symposium on Security and Privacy (2016)
20. Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A.: Malware classification with recurrent networks. In: Proceeding of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1916–1920. IEEE (2015)
21. Saxe, J., Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. arXiv preprint arXiv:1508.03096v2 (2015)
22. Schultz, M., Eskin, E., Zadok, E., Stolfo, S.: Data mining methods of detection of new malicious executables. In: Proc. of the 2001 IEEE Symposium on Security and Privacy (SP). pp. 38–49. IEEE Press, New York (2001)
23. Seltzer, M.L., Droppo, J.: Multi-task learning in deep neural networks for improved phoneme recognition. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE (2013)
24. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15(1), 1929–1958 (Jan 2014), http://dl.acm.org/citation.cfm?id=2627435.2670313