Deep Networks in Online Malware Detection

Jiří Tumpach[1], Marek Krčál[2], Martin Holeňa[3]

[1] Faculty of Mathematics and Physics, Charles University, Malostranské nám. 2, Prague, Czech Republic
[2] Rossum Czech Republic, Dobratická 523, Prague
[3] Institute of Computer Science, Czech Academy of Sciences, Pod vodárenskou věží 2, Prague, Czech Republic

Abstract: Deep learning is usually applied to static datasets. If used for classification based on data streams, it is not easy to take into account a non-stationarity. This paper presents work in progress on a new method for online deep classification learning in data streams with slow or moderate drift, highly relevant for the application domain of malware detection. The method uses a combination of multilayer perceptron and variational autoencoder to achieve constant memory consumption by encoding past data to a generative model. This can make online learning of neural networks more accessible for independent adaptive systems with limited memory. First results for real-world malware stream data are presented.

1 Introduction

Deep network architectures have many benefits. The most obvious one is the lack of need for comprehensive preparation of data. A large enough network probably finds relevant features automatically. So it is easier to pass data to training than to guess about the correct match in the triple problem-transformation-classifier.

However, deep network needs a lot of training data to perform in this way. Fortunately, many areas constantly generate large amounts of data.

Too much data may be a problem because parallel training for deep neural networks can be expensive. Some training examples may be unnecessary and contain only repeating relevant information with some random noise. In this case, they function as a weight for the relevant information.

Consider a situation where there is no expected change of the target function during its use (offline training). In this case, one can save similarity filtered latent features of the trained network. For example, latent features can be outputs of some middle layer. One application can be transfer learning where some trade-off between network performance and speed of training is already expected.

Online problems are specific because they are intended for situations, when some drift of information is expected. So training on all available data can be harmful. One easy solution is to train a model only on the most recent subset of training examples. This method reduces the need for parallel training, however, discarding a large proportion of data can cause quick overtraining, especially in case of slow drift.

This paper investigates faster retraining of neural networks on data with slow drift. Such a research is highly relevant for the application domain of malware detection because most of the malware is evolving, entailing a drift in data. The main idea is to have multiple pairs of generator-discriminator for each time interval. The current generator is trained with the last subset of training data (moving window) with the addition of generated samples based on the previous generator. Its job is to estimate the distribution of past data points and to use that distribution for generating new examples. A discriminator uses also labels generated by the previous discriminator if labels are not provided explicitly. The generative model stores some information about the importance of different training cases (weights) and acts as an implicit decay. For the generative model, we currently use variational autoencoders (VAEs) and intend to include also deep belief networks (DBN) soon. However, this idea can be generalized to any suitable classifier and generative model.

In Section 2, we present the state of the art in online malware detection. The used methods are described in Section 3. In Section 4, strategies for training and evaluation are proposed. In Section 5, our data and experiments on a real word malware dataset are presented.

2 Online Malware Detection

Malware is continuously evolving by exploiting new vulnerabilities and examining evading techniques [8]. Moreover, detection has to deal with significant data drift. It can make use of a signature database of previously detected malware. When the file is scanned, at first its is compared with the items in the database. So only modified and new malware needs to be detected giving high priority to generalization. Therefore, online detection methods, capable of keeping up with and adapt to such evolution, are desirable.

Malware detection techniques can be divided into static and dynamic methods [12]. The static methods focus on an analysis of program code while dynamic methods infer from program behaviour. They can log used resources and privileges, system or APIs calls or track sensitive data an inside application [8]. In connection with online learning, DroidOL [8] uses the analysis of inter-procedural control-flow graphs to achieve robustness against hiding attempts.

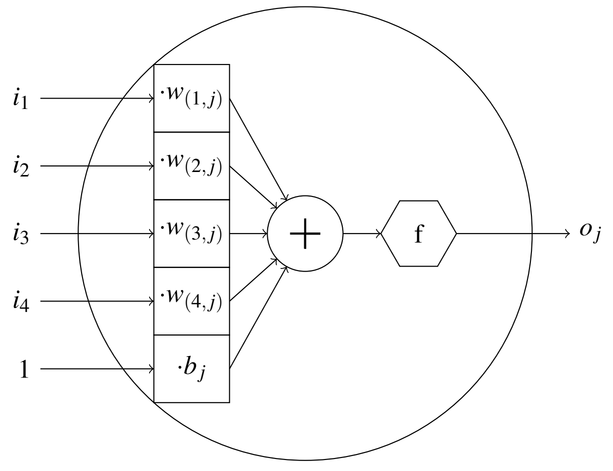


Figure 1: The neuron j, its inputs (i_i) are multiplied by corresponding weights ($w_{(i,j)}$) then summed together with a specific bias b_i. The resulting value is called activation. This value is mapped by the activation function $f(x)$ to the output o_j of the neuron j.

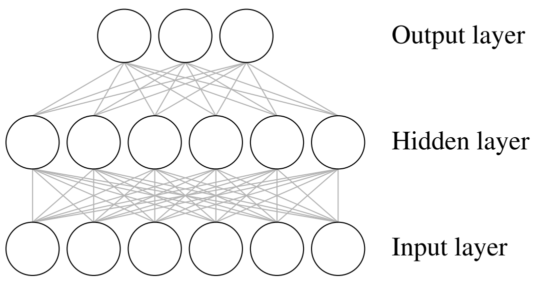
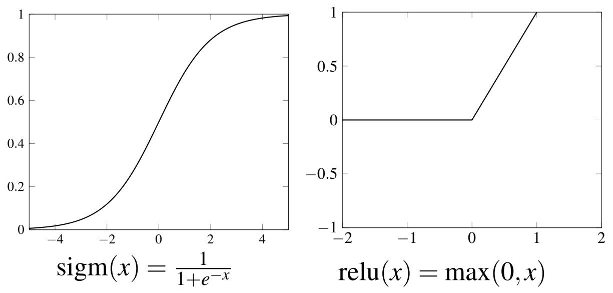


Figure 2: Multilayer perceptron with two layers.



(a) sigmoid

(b) relu

Figure 3: Important examples of activation functions.

It is trained with a fast online linear algorithm adapted to growing dimensionality. On real Android applications, DroidOL outperforms state-of-the-art malware detectors with 84.29% accuracy.

Another dynamic online method [11] reports using online learned Support Vector Machines with RBF kernel to detect malware from application behavior.

Users can have different sensitivity to give their data like location, contacts, or files to an author of a specific application. Antimalware programs then need to profile each user to not restrict them or overly bother. XDroid [12] tackles this problem by online hidden Markov model (HMM) learning.

3 Methodological Background

3.1 Multilayer Perceptron (MLP)

A multilayer perceptron is composed of neurons (Figure 1) arranged into layers (Figure 2) [3]. The first layer is called input layer, and its function is to receive values of the inputs. The last layer is called output layer and it has a similar structure as the remaining, aka hidden layers. Their neurons are connected to the output of each neuron in the previous layer. Figure 2 depicts a two layer MLP. It is a non-linear regression or discrimination model because its neurons use non-linear activation functions (Figure 3).

MLP is learned through minimizing some loss function usually by some kind of smooth optimization. The most simple, but still used kind of smooth optimization is gradient descent, in the area of neural networks also known as backpropagation, due to the flow of gradient computation. In high-dimensional spaces, its stochastic variant is commonly used, stochastic gradient descent. Exact second order methods like such as the Gauss-Newton method are usually inefficient [2, 17]. On the other hand, more successful methods are attempting to approximate second order behavior. One of the strategies is to have different learning rates (sizes of steps) for each learned variable (Adam, AdaGrad, RMSProp, SGD with Nestorov momentum, ...) [3]. Alternatively, some methods approximate second order derivatives from gradients history (Adam) [5].

One of the most popular loss functions used in regression problems is the Mean Square Error (MSE) loss [3] $\mathrm{L}_{\mathrm{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ where $\hat{y}_i$ is output of MLP given sample x_i from feature space with corresponding correct value y_i, N is the number of samples in one training cycle. In classification, to be able to learn probabilities of labels, one can employ cross-entropy loss. For classification into G classes, it is defined as $-\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{G} y_{il}\log(\hat{y}_{il})$ and the predicted probability $\hat{y}_{il}$ of the label l is given by the softmax activation function $\hat{y}_{il} = \exp(\hat{y}_{il})/\sum_{s=1}^{G}\exp\hat{y}_{is}$.

3.2 Autoencoders (AEs)

Autoencoders are neural networks capable of learning data representations called codings, usually with a much lower dimension than is the dimension of the input data [3]. They learn to copy the input to its output and are consisted of two parts: an encoder and a decoder, cf. the example in Figure 4. By restricting the flow of information, one can achieve interesting properties, for example denoising, detecting anomalies, generating unseen samples with a similar distribution as the training one and so on.

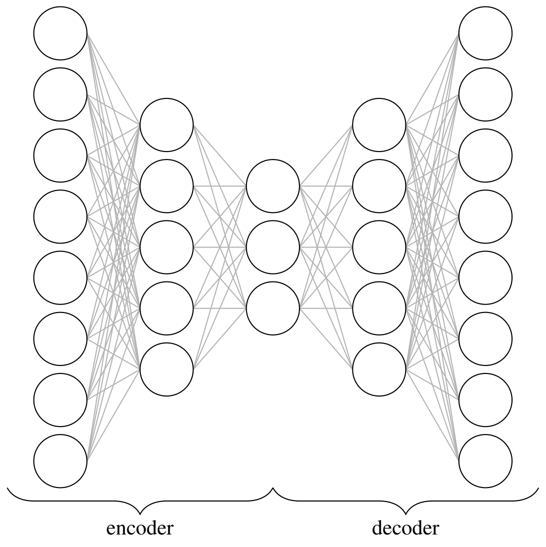


Figure 4: Autoencoder – the output of the encoder is the input to the decoder.

3.3 Variational Autoencoders (VAEs)

Codings in basic autoencoders can have nonstandard distributions [3]. This property makes it difficult to generate samples similar to the training dataset. VAEs solve this problem by employing the Kullback-Leibler (KL) divergence. KL divergence between two distributions p and q is defined as:

$$\begin{aligned} D_{\mathrm{KL}}(p||q) &= \mathrm{H}(p,q) - \mathrm{H}(p) \\ &= -\int_{-\infty}^{\infty} p(x)\ln q(x)dx + \int_{-\infty}^{\infty} p(x)\ln p(x)dx \\ &= \int_{-\infty}^{\infty} p(x)\ln\left(\frac{p(x)}{q(x)}\right)dx, \end{aligned}$$

where $H(p,q)$ is cross-entropy and $H(p)$ is entropy. The KL divergence is a measure of difference between two distributions. If $p(x)$ and $q(x)$ are the same, the divergence equals 0, otherwise it is positive value.

Because codings in AEs are deterministic, it is not possible to define KL divergence. The important idea in [6] is to map the codings to normal distributions, using a suitable neural network. The i-th coding now corresponds to one pair of output neurons of the network, and their activities represent a normal distribution for the i-th codding. So the first neuron defines the mean (μ_i) and the second one the standard deviation (σ_i) of that normal distribution. The normal distributions for different codings are mutually independent.

VAEs learn to minimize $\mathrm{L}_{\mathrm{VAE}}$ where $\mathrm{L}_{\mathrm{VAE}} = \mathrm{D}_{\mathrm{KL}}\left(\mathscr{N}(\mu,\sigma)||\mathscr{N}(\mathbf{0},\mathbb{1})\right) + \mathrm{L}_{\mathrm{MSE}}$. So they are learned to copy their inputs to the outputs, while maintaining approximately a normal distributions in the codings. In [6] has been proven that this divergence can be computed as

$$\mathrm{L}_{\mathrm{VAE}} = \mathrm{L}_{\mathrm{MSE}} - \frac{1}{2N}\sum_{i=1}^{N}\sum_{l=1}^{G}\left(1 + \log(\sigma_{il}^2) - \mu_{il}^2 - \sigma_{il}^2\right)$$

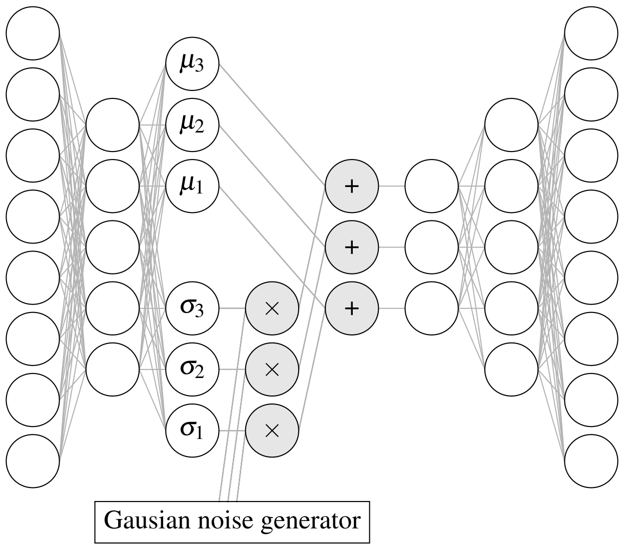


Figure 5: Variational Autoencoder. Gray nodes are operations, μ, σ nodes have linear activation fucntion.

.

In [3] has been proposed to speed up convergence in training by predicting logarithm of variance ($\log(\sigma_i^2) = v_i$) instead of standard deviation. Then $\mathrm{L}_{\mathrm{VAE}}$ will be:

$$\mathrm{L}_{\mathrm{VAE}} = \mathrm{L}_{\mathrm{MSE}} - \frac{1}{2N}\sum_{i=1}^{N}\sum_{l=1}^{G}\left(1 + v_{il} - \mu_{il}^2 - e^{v_{il}}\right)$$

The VAE encoder input is now $\vec{\mu} + \vec{\varepsilon}\cdot\vec{v}$, where $\vec{\varepsilon}$ is a vector of samples from standard normal random distribution. VAEs backpropagation is unchanged, all operations should be considered without any skipping.

If VAE is properly learned, sampling becomes easy. We can expect a normal distribution of its codings if we sample from a real learned distribution. The encoding part is then redundant and can be skipped. The result is only a random sampler which gives inputs to the decoder.

3.4 Support Vector Machine (SVM)

A support vector machine will be tested as an alternative to a multilayer perceptron for the starting classification of available data, due to a frequent use of SVMs in malware detection [7, 9, 10, 16].

A SVM is constructed with the objective of best generalization, i.e., maximal probability that the classifier ϕ classifies correctly with respect to the random variables X and Y producing the inputs and outputs, respectively,

$$\max P(\phi(X) = Y). \tag{1}$$

For our high-dimensional feature space $\mathscr{X} \subset \mathbb{R}^n$, it is sufficient to consider only a linear SVM, which classifies according to some hyperplane $H_w = \{x \in \mathbb{R}^n | x^\top w + b = 0\}$

with $w \in \mathbb{R}^n, b \in \mathbb{R}$,

$$(\forall x \in \mathscr{X})\ \phi(x) = \phi_w(x) = \begin{cases} 1 & \text{if } x^\top w + b < 0, \\ \text{-1} & \text{if } x^\top w + b \geq 0. \end{cases} \quad (2)$$

It can be shown [1, 14] that on quite weak conditions, searching for maximal generalization (1) is equivalent to searching for maximal margin between the representatives of both classes in the training data,

$$\max \frac{\rho}{\|w\|} \text{ with constraints } c_k x_k^\top w \geq \frac{\rho}{2}, k = 1, \ldots, p,$$
$$\text{where } \rho \text{ is the scaled margin and}$$
$$(x_k, c_k) \in \mathbb{R}^n \times \{-1, 1\} \text{ are the training samples,} \quad (3)$$

and that using the standard Lagrangian approach for inequality constraints, (3) can be transformed into the dual task

$$\max_{(\alpha,\rho)} -\frac{1}{4} \sum_{j,k=1}^{p} \alpha_j \alpha_k c_j c_k x_j^\top x_k + \frac{\rho}{2} \sum_{k=1}^{p} \alpha_k$$
$$\text{with constraints KKT, } \alpha_1, \ldots, \alpha_p \geq 0, \rho > 0,$$
$$\text{where } \alpha_1, \ldots, \alpha_p \text{ are Lagrange multipliers.} \quad (4)$$

The objective function in (4) is quadratic, thus it has a single global maximum, which can be found in a straightforward way. The abbreviation KKT in (4) stands for Karush-Kuhn-Tucker conditions

$$\alpha_k \left(\frac{\rho}{2} - c_k x_k^\top w\right) = 0, k = 1, \ldots, p. \quad (5)$$

Due to KKT, the classifier (2) in terms of the solution $\alpha_1^*, \ldots, \alpha_p^*, \rho^*$ of (4) turns to

$$(\forall x \in \mathscr{X})\ \phi_w(x) = \begin{cases} 1 & \text{if } \sum_{x_k \in \mathscr{S}} \alpha_k^* c_k x^\top x_k + \rho^* \geq 0, \\ \text{-1} & \text{if } \sum_{x_k \in \mathscr{S}} \alpha_k^* c_k x^\top x_k + \rho^* < 0, \end{cases} \quad (6)$$

where $\mathscr{S} = \{x_k | \alpha_k^* > 0\}$. The vectors in $\mathscr{S}$ lie in the support hyperplanes of the representatives of both classes in the training data. Therefore, they are called support vectors.

Because the size of input features is 540, and at least 40% of them are binary or look almost as constants, we decided to use a linear SVM. Moreover when polynomial kernel ($p = 2$) was used, the speed of convergence was too slow.

3.5 Linear Regression

To estimate the trend of a time series of model accuracies, we need to perform a linear regression [13] for C points in two dimensions $(x_0, y_0), (x_1, y_1), \ldots, (x_C, y_C)$. More precisely, the trend of the time series is described by the slope a of the line $\hat{y}_i = a x_i + b$ where

$$a = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} + \bar{x}^2} \qquad b = \bar{y} - a\bar{x}$$

with $\bar{t} = \frac{1}{C} \sum_{i=1}^{C} t_i$.

4 Proposed Strategy for Online Learning with VAEs

We propose an online learning strategy which focuses on more effective learning and a constant memory requirements of fetures. The strategy uses two deep learning architectures: MLP and VAE. While a MLP is trained to replicate labels, a VAE is used as a feature generator. Hence, a VAE can generate new unseen samples for a MLP representing the history. The pseudocode of the algorithm can be found in Algorithm 1 and a diagram of training data paths is depicted in Figure 6.

In the first week of training, the VAE is trained on current moving window, which act as a memory limit. The same applies for the MLP, but it also uses label information. Next weeks are different. The VAEs use also data sampled from previous weeks VAE, this provides something like a moving average. The problem is in choosing the right 1. time to update, 2. size of the generated data, 3. relative importance of generated data. All MLPs are also trained from VAEs generated data; because generated data lacks label information, the previous weeks MLP must be employed to add them.

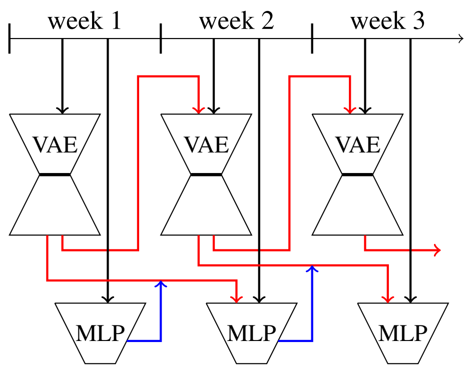


Figure 6: Training data paths for VAEs and MLPs for each week. Red indicates generated data, blue adds label classifications to features.

5 Experiments with Malware Detection Data

In this section, we describe several experiments with real-world data from the area of malware detection.

5.1 Data

We use real-word anonimized data, which feature malware and clean software in several categories, but we consider only two by merging some of them. The semantics of the individual features has not been made available by the company. The feature space is very complex, there are 540 features with various distributions. This makes particularly difficult to choose the correct data scaling. In Figure 7, several groups of features are differentiated:

Algorithm 1 Proposed online learning algorithm

```
Require:
    number $N$  ▷ $N$ is the number of inputs generated by the previous generator.
    number $M$  ▷ $M$ is the size of the considered most recent training data.
    function data_for_iteration(number)
                ▷ It gives access to stored data for some iteration with provided number.
    function labels_for_iteration(number)
                ▷ Same as previous function, but for labels.
Ensure:
    Provides discriminator updates for each client
                ▷ Discriminator can predict labels for new data.

 1: procedure CLIENT
 2:     discriminator ←function($x$){return default class}
 3:     while workstation runs do
 4:         if exists new version of discriminator then
 5:             discriminator ← update_discriminator()
 6:         end if
 7:         if new undecided file exists then
 8:             input ←get_features(x)
 9:             label ← discriminator(input)
10:             send_to_server(x)
11:             do task specific operation with file as label.
12:         end if
13:     end while
14: end procedure

15: procedure SERVER
16:     iteration ← 0
17:     while not last iteration do
18:         iteration ←iteration + 1
19:         data ←most_recent_data($M$)
20:         labels ←most_recent_labels($M$)
21:         if iteration $> 1$ then
22:             gen_data ← generator(N)
23:             gen_labels ← discriminator(gen_data)
24:             data ← [data; gen_data]
25:             labels ← [labels; gen_labels]
26:         end if
27:         generator ← learn_generator(data)
28:         discriminator ← learn_discriminator(data, labels)
29:         publish_discriminator(discriminator)
30:         while updating the discriminator is not needed do
31:             wait()
32:         end while
33:     end while
34: end procedure
```

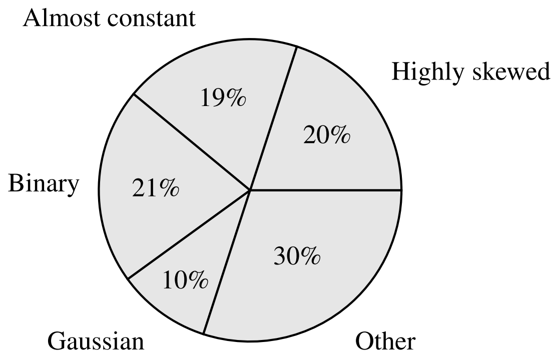


Figure 7: Distribution in the feature space.

- Binary feature
- Normally distributed feature: both absolute skewness and kurtosis is less than 2
- Highly skewed feature: skewness > 30
- Almost constant feature: more than 99.9 % values are identical
- Other unknown distributions

The data are initially divided by week. We decided to keep this natural division even though some of the weeks are mostly empty. We have used 375 weeks in our experiments, the number of files and proportion of malware files are for them depicted in Figure 8.

[p]

5.2 Performed Experiments and Their Results

To be able to decide if a neural network is a good model for this task, we compare it with a linear SVM. The number of recent training examples is chosen M = 150.000; it corresponds to about 5.5 average weeksand at least 309 MiB of RAM. In order to evaluate the full dataset, one must process 113 GiB of data, and train, sample and evaluate about 370 SVMs and VAEs.

Both the MLP and the SVM models are Bayesian optimized on first week following the first M of excluded data points by the GpyOpt library [15] using the maximum probability of improvement as acquisition function and mixed sequential and local penalization evaluation. The MLP model is using the Adam algorithm with early stopping after 10 unimproved evaluation of the validation data (25 % of the actual training data). The MLP uses only densely connected layers with cross entropy loss, the SVM uses squared hinge loss. The resulting hyperparameters are in Tables 1 and 2. Table 1 shows a noticeably larger network size together with a lot of regularization.

We have applied the Bayesian optimization also to the VAE, but the results were not conclusive. Layers prefer to be as large as possible because the L_{MSE} part of the loss can be reduced more with larger layers. Unfortunately, this does not reveal whether some increase in history size (M)

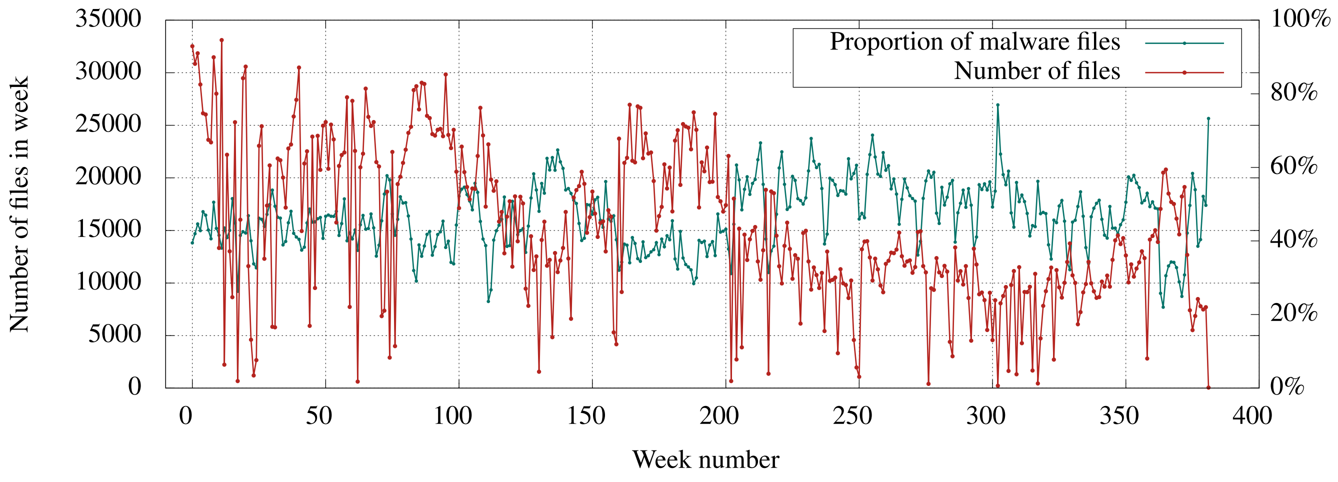


Figure 8: Number of analyzed files and the proportion of malware files in each week

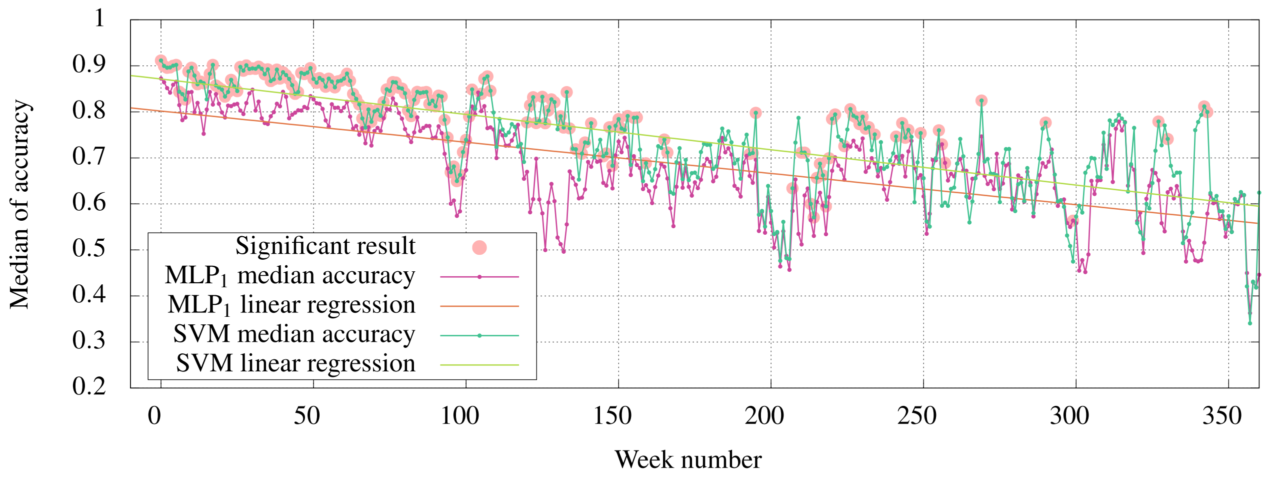


Figure 9: Comparison between two models trained on the data from the first week. The trend in the time series indicates that a data drift is present.

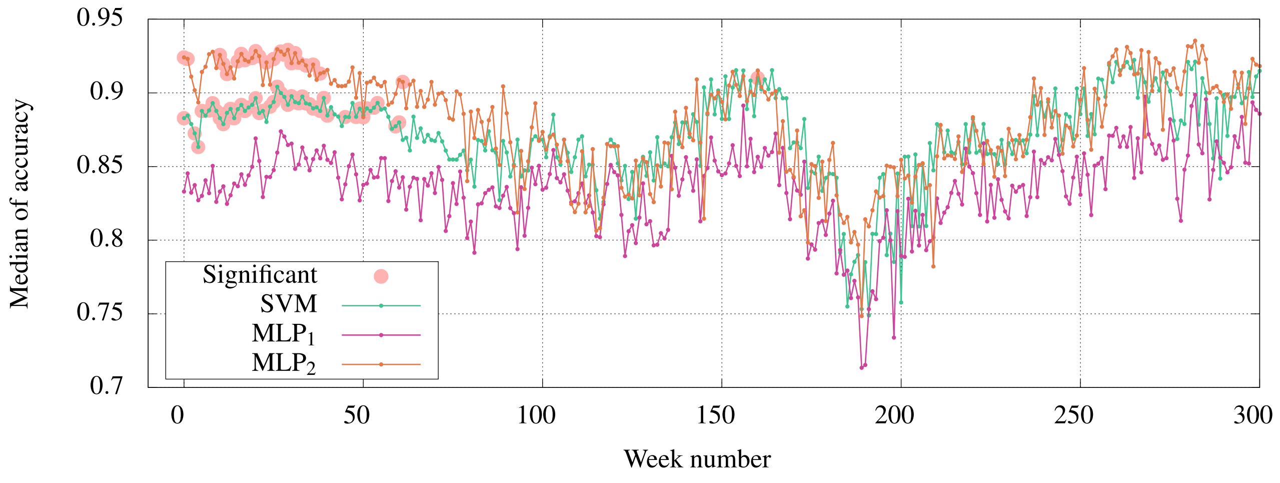


Figure 10: Comparison between SVMs and MLPs retrained for each week. There is no clear gradual increase in the difficulty of problem. MLP$_2$ seems to be the best of the compared models. A result is highlighted if it is significantly better than another worse result in the respective week.

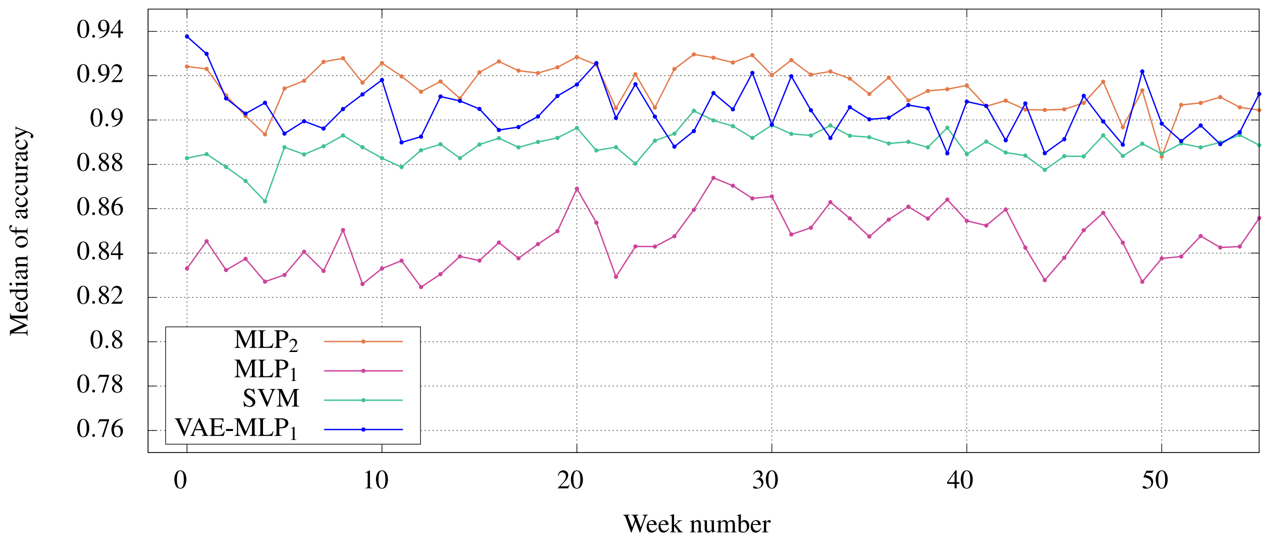


Figure 11: Results of our algorithm in first 55 weeks. The significantly worst MLP_1 gains significant performance advantage when combined with a VAE, to the point of basically matching MLP_2 and SVM. A summary of comparison results is given in Table 4.

Table 1: Results of MLP hyperparameter optimization.

Name	Selected value	Possibilities
Learning rate	0.00763	0.0001-0.01
Batch norm.	yes	yes/no
Dropout	0.22	0-0.7
Gaussian noise	0.795	0-1.0
Layers	354-322-316-305-2	up to 400-400-400-400-2
Activation	relu.	elu, selu, softplus, softsign, relu, tanh, sigmoid, LeakyReLU, PReLU, ELU
Minibatch size	730	10-1000
L1 regular.	0.01	0-0.1
L2 regular.	0.0998	0-0.1
Data scaling	Standard	Standard Robust MinMax

Table 2: Results of SVM hyperparameter optimization.

Name	Selected value	Possibilities
Penalty	64.44	0.001-80
Penalty type	l1	l1/l2
Data scaling	Standard	Standard Robust MinMax

helps more than the appropriate extension of a network. Layers also tend to have elu as the most suitable activation function together with batch normalization.

Altogether for baselines, we were using MLP_1, representing a small slightly regularized MLP, MLP_2 with optimal hyperparameters (Table 1), representing a large and highly regularized network, and a SVM.

In Figure 9, we see a data drift is indeed present and both models are similarly penalized in time. This observation is confirmed by Figure 10 where learning is done for each week and it does not seem that the difficulty of the problem is increasing. The models are not clearly overtrained because both achieved a rather high accuracy with a rather small training dataset. The results were statistically analyzed by the Wilcoxon ranksum test with Holm correction on the 5% family-wise significance level [4]. For models trained only once, the results showed that the SVM was better 88.3% of weeks while being significantly better 45.9% of them. MLP_1 was significantly better only in 0.8% of weeks. It is important to say that the MLP_1 in this test does not have optimal hyperparameters we, only want to see if its behaviour is evolving with time. The results of this comparison can be seen in Figure 9.

Subsequently, MLP_1, MLP_2 and SVM were trained repeatedly each week with a corresponding history of size M and then tested on the next week. The results are depicted in Figure 10, whereas a summary is in Table 3.

Our VAE-MLP algorithm is rather slow, due to inherent sequential training. For the VAE, we used the 540-200-200-10-200-200-540 fully connected architecture with elu activations and L_{MSE}. The network is updated with data from each week with $M = N = 150.000$. Figure 11 depicts an interesting property. The previously clearly inferior MLP_1 is improved by VAE to the point of matching

Table 3: Summary of baseline consideration, the MLP_1 is a small network with little regularization, MLP_2 is a large network with a lot of regularization and linear SVM is considered because it may have superior generalization properties.

		MLP1	MLP2	SVM
MLP1	is better than		6.1%	2.9%
MLP2		93.9%		61.6%
SVM		97.1%	38.4%	

		MLP1	MLP2	SVM
MLP1	is significantly better than		0.0%	0.0%
MLP2		18.1%		6.1%
SVM		12.8%	0.0%	

Table 4: Summary of the results of the first 50 weeks between baselines (MLP_2 and SVM) and MLP_1 with and without VAE.

		MLP_2	MLP_1	SVM	VAE_1
MLP_2	is better than		100.0%	100.0%	82.0%
MLP_1		0.0%		0.0%	0.0%
SVM		0.0%	100.0%		8.0%
VAE_1		18.0%	100.0%	92.0%	

		MLP_2	MLP_1	SVM	VAE_1
MLP_2	is significantly better than		100.0%	84.0%	4.0%
MLP_1		0.0%		0.0%	0.0%
SVM		0.0%	100.0%		0.0%
VAE_1		0.0%	100.0%	14.0%	

baselines performance. It clearly shows the potential of this algorithm, not only we do not optimize MLP and VAE together, but also we do not tune the vaues M and N. Table 4 further confirms the findings from Figure 11 as a nice summary.

If you are interested, you can try our model or help with development at the following links:

Bayesian hyperparameter optimization framework
`https://github.com/tumpji/Bayesian-optimizer.git`

Implementation of the proposed method
`https://github.com/tumpji/VAE-NN-Tensorflow.git`

Deep belief networks in Tensorflow
`https://github.com/tumpji/DBN-Tensorflow.git`

6 Conclusion

This paper presented work in progress on a new approach to online deep classification learning in data streams with slow or moderate drift. Such kind of learning is highly relevant for the application domain of malware detection. In the paper, the employed methods have been recalled and the principles of the proposed approach has been outlined. In ongoing experiments, the approach is currently being validated on a large set of real-world malware-detection data. This dataset contains Windows executable files from 375 weeks, in the amount up to 30.000 binary files from each week. Due to the large size of the dataset, only the baseline detection using a MLP alone has been tested up to now, and also compared to classification based on linear SVMs, frequently used in malware detection. The computational demands of testing the proposed new approach allowed to accomplish it so far for only 55 weeks. Results of the ongoing experiment will be available and presented at the workshop.

Acknowledgement

The research reported in this paper has been supported by the Czech Science Foundation (GAČR) grant 18-18080S. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

References

[1] P.J. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 43–54. MIT Press, Cambridge, 1999.

[2] W. L. Buntine and A. S. Weigend. Computing second derivatives in feed-forward networks: a review. *IEEE Transactions on Neural Networks*, 5(3):480–488, May 1994.

[3] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Boston, first edition edition, 2017.

[4] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[6] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, December 2013. arXiv: 1312.6114.

[7] M. Mursleen, A.S. Bist, and J. Kishore. A support vector machine water wave optimization algorithm based prediction model for metamorphic malware detection. *International Journal of Recent Technology and Engineering*, 7:1–8, 2019.

[8] A. Narayanan, L. Yang, L. Chen, and L. Jinliang. Adaptive and scalable Android malware detection through online learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2484–2491, July 2016.

[9] N. Nissim, R. Moskowitch, L. Rokach, and I. Elovici. Novel active learning methods for enhanced PC malware

detection in windows OS. *Expert Systems with Applications*, 41:5843–5857, 2014.

[10] H.H. Pajouh, A. Dehghantanha, R. Khayami, and K.K.R. Choo. Intelligent OS X malware threat detection with code inspection. *Journal of Computer Virology and Hacking Techniques*, 14:212–223, 2018.

[11] B. Rashidi, C. Fung, and E. Bertino. Android malicious application detection using support vector machine and active learning. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9, November 2017.

[12] Bahman Rashidi, Carol Fung, and Elisa Bertino. Android Resource Usage Risk Assessment using Hidden Markov Model and Online Learning. *Computers & Security*, 65, November 2016.

[13] Mathieu ROUAUD. *Probability, Statistics and Estimation: Propagation of Uncertainties in Experimental Measurement*. Mathieu ROUAUD, June 2017.

[14] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge, 2002.

[15] Machine Learning Group-University of Sheffield. GPyOpt.

[16] M. Stamp. *Introduction to Machine Learning with Applications in Information Security*. CRC Press, Boca Raton, 2018.

[17] William T. Vetterling, Brian P. Flannery, William H. Press, and Saul A. Teukolsky. *Numerical Recipes: The art of scientific computing*. Cambridge University Press, Cambridge, 3nd ed edition, 2007.