FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Semi-supervised learning of deep neural networks
Student:	Bc. Jan Koza
Supervisor:	prof. Ing. RNDr. Martin Holeňa, CSc.
Study Programme:	Informatics
Study Branch:	System Programming
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2020/21

Instructions

1. Get familiar with main approaches to semi-supervised learning, particularly in context of deep nueral networks.
2. Get familiar with the methodology of experimental comparison of different semi-supervised methods and with the methodology of statistical assessment of such experiments.
3. Implement at least two existing methods of semi-supervised learning for deep nueral networks, and evaluate your implementation on benchmark data commonly used in deep learning.
4. Based on the results of the evaluation of the implemented methods, attempt to propose at least one modification, hybridization or other extension of them.
5. Compare all variants of the considered methods on the benchmark data used in 3.
6. Finally, compare all variants of the considered methods also on at least one set of real-world data provided by the supervisor.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 2, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Semi-Supervised Learning of Deep Neural Networks

Bc. Jan Koza

Department of Theoretical Computer Science
Supervisor: prof. Ing. RNDr. Martin Holeňa, CSc.

January 9, 2020

Acknowledgments

I would like to express my thanks to my supervisor Martin Holeňa for his precious time, support, and help he gave me. I am also grateful for the patience and kindness of my girlfriend, Markéta. Additionally, access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum is appreciated.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on January 9, 2020 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Koza, Jan. *Semi-Supervised Learning of Deep Neural Networks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Semi-supervizované učení se vyznačuje tím, že využívá i dodatečné informace z neoštítkované části trénovacích dat. Tato práce porovnává dva algoritmy semi-supervizovaného učení hlubokých neuronových sítí na reálných malwarových datech. Jedním z nich je metoda Pseudo-labeling. Ta využívá neoštítkované vzorky klasifikované s vysokou jistotou, jako by tak byly skutečně označené. Druhý přístup je založen na zachování konzistence výsledků neuronové sítě za různých okolností. Byl implementován jeden takový algoritmus, Π-model, který porovnává výstupy sítě pro různě pozměněná vstupní data. Pro srovnání jsou také uvedeny výsledky plně supervizovaného učení, které používá pouze oštítkované vzorky. Přesnost predikce těchto metod je vyhodnocena v závislosti na poměru velikosti oštítkované části trénovacích dat.

Klíčová slova Strojové učení, Semi-supervizované učení, Hluboké neuronové síte, Detekce malwaru, Pseudo-labeling, Π-model

Abstract

Semi-supervised learning is characterized by using the additional information from the unlabeled data. In this thesis, we compare two semi-supervised algorithms for deep neural networks on a large real-world malware dataset. Specifically, we evaluate the performance of a rather straightforward method called Pseudo-labeling, which uses unlabeled samples, classified with high confidence, as if they were the actual labels. The second approach is based on an idea to increase the consistency of the network's prediction under altered circumstances. We implemented such an algorithm called Π-model, which compares outputs with different data augmentation and different dropout setting. As a baseline, we also provide results of the same deep network, trained in the fully supervised mode using only the labeled data. We analyze the prediction accuracy of the algorithms in relation to the size of the labeled part of the training dataset.

Keywords Machine learning, Semi-supervised learning, Deep neural networks, Malware detection, Pseudo-labeling, Π-model

Contents

List of Figures

List of Tables

Introduction

One of the application domains that pay the most attention to the progress of and new developments in machine learning is malware detection. Vendors of antivirus software cannot keep up with the increasing number of malicious programs and their increasingly sophisticated obfuscation and polymorphism without using more and more advanced machine learning methods, most importantly, methods for anomaly detection, classification and pattern recognition.

The most successful machine learning methods for classification and pattern recognition definitely include artificial neural networks (ANN), especially deep networks. However, they have a high number of degrees of freedom, thus requiring a large amount of labeled training data, whereas most of the data for malware detection is unlabeled because its labeling requires expensive involvement of human experts. One possible way how to tackle the lack of training data is semi-supervised learning. In a narrow sense, this means supervised learning that simultaneously to labels also uses some information from additional unlabeled data, in a broad sense any combination of supervised learning and unlabeled data, e.g., unsupervised learning followed by supervised learning. In the context of malware detection, however, research into semi-supervised ANN learning is only emerging [1, 2]. The results reported in this thesis are a small contribution to it.

In the first chapter, we briefly outline the theory of machine learning and its types and describe artificial neural networks. The next chapter focuses on related work. Its first part is about semi-supervised learning, especially of neural networks. The second part is about the use of neural networks in malware and network intrusion detection. The third chapter deals with the details of our implementation of Pseudo-labeling and Π-model. The core of the work is in the fourth chapter, where we present our experiments and their results. We test our implementations on a simple dataset, and then we thoroughly evaluate them on real-world malware data.

Chapter 1

Overview of Theory

Artificial intelligence (AI) is a field of computer science that focuses on solving a wide range of complex problems using machines. The progress of AI goes hand in hand with the development of computers. With the rise of computing power, more problems became solvable using computer programs, including the area of artificial intelligence. Especially in the last decade, AI gains massive attention.

The research in artificial intelligence can be divided into several areas including reasoning, planning, natural language processing, robotics, computer vision, and machine learning. For the following brief introduction to machine learning methods related to this thesis, we used the sources [3, 4, 5, 6, 7].

1.1 Machine Learninng

In a broad sense, machine learning concerns algorithms, that are able to learn. In other words, to extract relevant information or knowledge out of input data and find a generalization that can be used later. Machine learning algorithms are often based on the idea of improving their performance gradually during the learning process. That is achieved by iterative optimization of a particular objective function that specifies the goals of learning. There are several main types of machine learning algorithms, according to how they handle the input data, we describe them below.

1.1.1 Supervised Learning

In *supervised learning*, the algorithm learns from input data with assigned labels for every sample. The training data consist of an input and a corresponding desired output. The task is to create a mapping function, aka model, that for an arbitrary input, produces the correct output. The challenge of model construction is to make it capable to generalize to the new unseen data, instead of just memorizing the training dataset. The most common problems

in supervised learning are classification and regression. In classification, the task is to assign the input sample to one of the given categories. While the outcome of the regression model is usually a real number or a vector. We will focus the most on the classification in this thesis because we will be solving a classification problem in the application part. There is a large number of approaches to supervised machine learning, such as Naive Bayes, k-NN, decision trees, Support Vector Machines, or neural networks.

1.1.2 Unsupervised Learning

Unsupervised learning means that the model learns to gain knowledge about the structure and patterns in the input data without knowing the corresponding output. In unsupervised learning, some similarity measure among the data samples is often used or modeling their probability density. This type of learning includes problems like clustering, dimensionality reduction, or anomaly detection. Notable methods used to solve these tasks are k-means, hierarchical clustering, and particular kinds of neural networks, such as autoencoders, Deep Belief Networks, Generative Adversarial Networks, or Self-Organizing Maps.

1.1.3 Semi-supervised Learning

As the name suggests, *semi-supervised learning* is a combination of supervised and unsupervised learning. During the learning, the model uses data, where only part of them have assigned the output. So the training dataset is divided into two parts, usually denoted as a labeled and unlabeled set. The model learns from the labeled set in a supervised way but also uses additional information from unlabeled data to further improve its performance. Like in unsupervised learning, a similarity measure can be utilized in semi-supervised learning. The size of the unlabeled dataset is often much larger than of the labeled one. Nowadays, having access to a large amount of data is quite common, but obtaining the correct ground truth labels for them may be difficult, costly, or time-consuming. That is due to the fact that getting labels usually involves human effort or an experiment. Therefore semi-supervised learning may help to overcome this issue. However, according to [6], the unlabeled data must hold certain assumptions, to be able to improve the model's accuracy.

Smoothness Assumption

If two points x_1, x_2 are close, then so should be the corresponding outputs y_1, y_2. This assumption should be met in the fully supervised learning. In semi-supervised learning, this assumption also depends on the density of the data. So for the semi-supervised learning, if two points x_1, x_2 in a high-density region are close, then so should be the corresponding outputs y_1, y_2.

Cluster Assumption

The cluster assumption states that if points are in the same cluster, they are likely to be of the same class. The idea behind this assumption is intuitive and puts together the principles of supervised and unsupervised learning. When the data form clusters, it is reasonable to assume that the samples in a particular cluster belong to the same class. On the other hand, it does not mean that data from one class have to form a single cluster. Actually, the early methods of semi-supervised learning were based on clustering, followed by assigning the clusters with classes of labeled data. This assumption is related to the requirement of the low-density separation, which says that the decision boundary should lie in a low-density region.

Manifold Assumption

This assumption states that the high-dimensional data lie roughly on a low-dimensional manifold. When the dimensionality of the input data is high, then it is often that a problem with measuring distances in a high-dimensional space arises. The problem is that the differences in pairwise distances of samples from different classes, tend to be less pronounced, and therefore less useful. However, if the data lie in a manifold, then distances can be measured inside the low-dimensional space containing the parameters of the manifold.

1.1.4 Active Learning

Active learning builds on the same principle as semi-supervised learning of using both labeled and unlabeled data together for the training. The difference is that active learning extends this idea by letting the model itself decide which of the unlabeled data samples should be selected for labeling. The algorithm is able to behave interactively and query the user to provide outputs for selected data. Active learning can be used in combination with incremental or online learning to improve the model's accuracy gradually. The most straightforward approach used in active learning is uncertainty sampling. When the model is probabilistic, it can evaluate how certain it is about a particular prediction. Then it selects those samples with the least confident predictions.

1.2 Artificial Neural Networks

Artificial neural networks (ANN) belong to the most successful machine learning methods for classification and pattern recognition. The development of artificial neural networks was inspired by biological neural networks in the brains of animals. An artificial network as well consists of nodes and their connections, like neurons and synapses in a brain. The node in an ANN is called an artificial neuron and represents a mathematical function. An artificial neuron has one or more weighted inputs. It sums them and adds a bias

and passes it to a non-linear activation function to produce the output. The output of a single neuron $f(\boldsymbol{x})$ for an input vector $\boldsymbol{x}$ of length n can be written as the value of a function in the following way:

$$f(\boldsymbol{x}) = \varphi\left(\sum_{i=0}^{n} w_i x_i + b\right),$$

where φ is the activation function, w_i is the weight of a x_i, the i-th component of x, and b is a bias. Because the neurons in a network are connected together to a graph structure, it is convenient to visualize them as nodes shown in Figure 1.1.

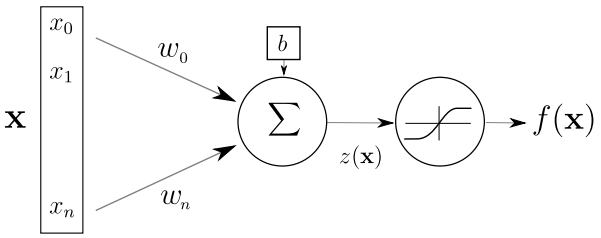


Figure 1.1: A diagram of an artificial neuron with a sigmoidal activation function. Taken from the lecture slides [8].

It is essential to choose a non-linear activation function, to make the network able to learn other than linear features. There are many functions that can be used as activation. Usually, some sigmoidal function is used or nowadays a *rectified linear unit* (ReLU) or some of its variants [9]. An example of a sigmoid function that is commonly used is the logistic function:

$$S(x) = \frac{1}{1 + e^{-x}}.$$

The simple function ReLU is commonly used as a default in many frameworks for neural computation. Its definition follows:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

As mentioned above, an artificial neural network is created by connecting neurons together to form a directed graph. A node can have its input set to accept signals from outside, therefore it is called an input neuron. Similarly, output neurons directly produce the output of the whole network. The graph of a network can contain cycles, and such a network is then called recurrent. A network without cycles is named feed-forward, and its structure can be often grouped into layers according to the distance from the input nodes. We

will focus only on feed-forward networks in this thesis, which are commonly used for classification and regression problems. When each neuron in a layer is directly connected to neurons in the following layer, the network is called a *multilayer perceptron* (MLP). Layers other than input and output are referred to as hidden layers. A diagram of a multilayer perceptron with two hidden layers is given in Figure 1.2.

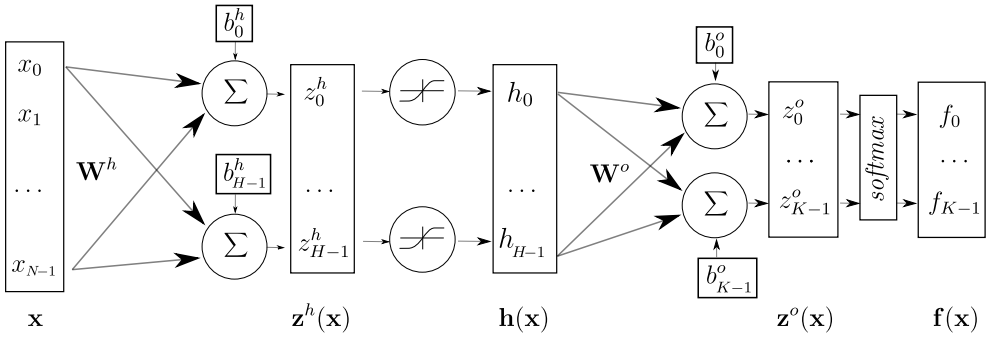


Figure 1.2: An example of a feed-forward neural network with two hidden layers. Taken from the lecture slides [8].

When an ANN is used as a model for classification into more than two classes, the output layer has a size equal to the number of target classes. Classes are represented as a one-hot encoding, which sets the value of the output corresponding to a particular class to 1 and all the others to 0. Generally, the neurons' output can be any real number, therefore to normalize the output of the last layer between 0 and 1, the softmax function is used. It maps the network's output to a probability distribution of target classes. The softmax function for an output x_i is defined by the formula:

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}},$$

where n is the dimension of the input data.

1.2.1 Learning of Neural Networks

So far, we only described the inner structure of neural networks. What makes an ANN useful is its ability to learn general patterns from the samples of input data. The output of a neural network is determined by the input data, the hyperparameters, and the trainable parameters. In regular neural learning, the hyperparameters, such as the number of neurons, the topology, activation functions, etc. are set before the training. What is iteratively learned are the input weights and biases of a particular neuron. These are the only trainable

parameters of a standard MLP. The learning process is an optimization of those parameters to produce as correct results as possible. To measure how good the prediction of a network is, a loss function is used. The loss function usually measures the difference between the actual target label for the input sample and the network's prediction. *Mean squared error* (MSE) is a simple example of such a measure, its formula for true labels $\boldsymbol{y}$ and network's predictions $\hat{\boldsymbol{y}}$ follows:

$$\mathrm{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where n is the number of classes. But a more commonly used loss function for classification tasks is the *cross entropy*, which yields better results according to [10]. The formula of discrete cross entropy is:

$$H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i.$$

Then learning of a neural network is a minimalization of the loss function. Input data are presented to the network and the mean of corresponding loss functions is evaluated. The loss function is calculated for every training sample, and the mean of losses is the overall error of the network in the current state. The error function than can be written as $E(\boldsymbol{w})$, where $\boldsymbol{w}$ is the vector of all weights of the network. The network's output also depends on the values of biases, but for simplification, they can be included in the weights $\boldsymbol{w}$, in such a way that, a bias is the weight of a neuron whose activation is always 1. In order to minimize the error function, we need to find its gradient $\nabla E(\boldsymbol{w})$. The gradient is a vector of partial derivatives of a function. Then we can use an optimization algorithm called *gradient descent*. Gradient descent iteratively changes the values of the vector of weights $\boldsymbol{w}$ in the opposite direction of the gradient. The change can be written as:

$$\Delta \boldsymbol{w} = -\alpha \nabla E(\boldsymbol{w})$$

where α is the learning rate and determines the size of the change. Therefore the change of each weight component w_i is:

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}.$$

However, changing the weights only after computing the error of the network for the whole dataset is computationally inefficient. Therefore, the dataset is usually split into smaller parts called mini-batches. Then the error of the network is computed for each batch, and the weights are updated. The size of the batch is an additional hyperparameter for the learning algorithm. This way, the update frequency is higher, and the learning usually converges

faster. However, this approach is slightly different from the standard gradient descent, because it does not compute the exact gradient but only its approximation. Therefore, because of the randomness in the batch selection, this learning method is referred to as mini-batch *stochastic gradient descent* (SGD). An effective way of computing the gradient of large networks with many connections and weights is the *backpropagation* algorithm. It is computed by applying the chain rule iteratively, from the last layer to the input later.

There are also more complex optimization algorithms that use techniques like momentum to avoid ending in local minima. An example of such an algorithm is the *Adam* optimizer [11], which is implemented in modern frameworks and popular lately.

Chapter 2

Related Work

Semi-supervised learning has been intensively studied at least since the early 2000s, as it is documented in the literature survey [12]. The first semi-supervised approaches were based on a generative statistical model, such as Gaussian mixture models, where unlabeled data can help to produce more accurate mixture components.

A different, rather straightforward approach is *self-training*, an example of which was used here [13] on the language processing domain. It is a simple example of using predictions of a model itself during the training. The model is first trained only on the labeled data samples. Then the trained model is used to label the unlabeled dataset. Typically, only the most confident predictions are selected as targets. These predictions are then combined with the initially labeled part of the data and used for later training together. This method can slightly improve the robustness of the model. However, it could also damage the performance because it amplifies the errors of the original supervised model.

Another method is to train multiple different models simultaneously with different views of the data. Then the unlabeled samples are used for training of the models if their predictions meet certain mutual conditions. This group of algorithms is called *multi-view* training, and a simple example of such a method is *Co-training* [14]. Co-training requires an assumption about the input dataset to be met. The labeled data L need to be split into two conditionally independent subsets L_1 and L_2, and each of them should be sufficient to train a good model. First, two models m_1 and m_2 are trained using the initially split datasets L_1 and L_2. Then all the unlabeled data are classified using both models m_1 and m_2. The prediction of a particular sample is used as a target for the other model, but only if the confidence according to the former model is high and confidence according to the latter model is low.

Similar approaches are *self-ensembling* methods that combine different variants or instances of the same model. Self-ensembling methods often use predictions of the same model under different configurations to improve it.

Recent examples of such methods are Ladder networks [15], Π-model [16], Temporal Ensembling [16], Mean Teacher [17], and Virtual Adversarial Training [18]. In the following section, we will describe those and additional self-training method Pseudo-Labeling [19] in more detail. We will focus only on algorithms for classification problems using neural networks.

2.1 Semi-supervised Learning of Neural Networks

According to the overview paper [20], the following four approaches are most important for semi-supervised learning of neural networks, especially deep networks. The authors of the paper implemented these methods and compared them using image classification benchmark datasets.

2.1.1 Pseudo-Labeling

A very simple approach from the family of self-training methods is *Pseudo-labeling* [19]. The term pseudo-label denotes ANN prediction of the correct class for unlabeled data, provided the network has a sufficient confidence in such a prediction, i.e., if for an unlabeled input x,

$$\arg\max_{c\in C} f_c(x) \geq \vartheta \sum_{c\in C} f_c, \tag{2.1}$$

where C denotes the set of classes, $f_c(x)$ the activity of the output neuron corresponding to the class $c \in C$ for the input x, and $\vartheta \in (0, 1)$ is a given threshold. The threshold helps to prevent the network from learning incorrect labels, which would result in an increase in error rate. Similarly to labels, also pseudo-labels can be represented with $\{0, 1\}$-valued vectors with dimension equal to the number of classes, with exactly one component equal 1. Also the loss of network predictions with respect to pseudo-labels can be evaluated with the same loss function as their loss with respect to labels. In [19], cross entropy has been used to this end. The losses for all labels and pseudo-labels are then summarized to an overall loss function being minimized during semi-supervised learning. However, the losses for labels and pseudo-labels have different weights, or equivalently, the losses of pseudo-labels are multiplied by some trade-off coefficient $\alpha(t) > 0$. This notation refers to the fact that the trade-off coefficient depends on the training epoch t. The formula for the overall loss function of a mini-batch of data, consisting of n labeled and n' unlabeled data is

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{c\in C} L\left(y_i^m, f_i^m\right) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{c\in C} L\left(y_i'^m, f_i'^m\right), \tag{2.2}$$

where f_c^m is prediction of m's sample in labeled data, y_c^m its label, $f_c'^m$ prediction for unlabeled data and $y_c'^m$ is the pseudo-label of that.

The Pseudo-labeling algorithm can improve the generalization performance using unlabeled data according to the cluster assumption listed in the previous chapter, which states that the decision boundary should lie in low-density regions to improve generalization performance [6]. The unsupervised loss component of the Pseudo-labeling method forces the predicted class probabilities to be near 1-of-K code, thus having one component set to 1 and all others to 0. In other words, it makes the network to be more confident with its predictions. This results in minimizing the conditional entropy of the network predictions:

$$H\left(y|x'\right) = -\frac{1}{n'}\sum_{m=1}^{n'}\sum_{c\in C} P\left(y_c^m = 1|x'^m\right)\log P\left(y_c^m = 1|x'^m\right). \tag{2.3}$$

where y_c^m is the unknown label of the mth sample, and x'^m is the corresponding input vector. The entropy measures the overlap of classes, therefore lowering the entropy shifts the decision boundary to a region with lower-density of samples. It means that Pseudo-labeling can be considered as a form of Entropy Regularization [21].

2.1.2 Increasing the Consistency

Another two methods evaluated in the overview paper are *Π*-model and *Temporal Ensembling*. These self-ensembling methods are similar, and both are based on an idea of increasing the consistency of the network's predictions. This approach encourages the network's output to be consistent when the same input sample is presented but under different circumstances. This can mean evaluating the predictions for the same input between two instances of a neural network differing through a random perturbation. Such a perturbation is typically introduced through random noise or through dropout. The overall loss function minimized during semi-supervised learning is then the superposition of the loss of supervised learning and a loss reflecting the inconsistency of the considered ANN instances.

This approach was first applied in [15] to *Ladder Networks*, which are basically chained denoising autoencoders. In a Ladder Network, the consistency is controlled in each layer, between the output of the previous layer and the application of a denoising function to a noisy output from the next layer.

In [16], two similar kinds of neural networks using this approach to semi-supervised ANN learning were proposed that can be viewed as simplifications of Ladder Networks. The consistency is in them controlled only at the network output, like the accuracy of supervised learning. The first kind, called Π-model, evaluates both randomly differing ANN instances on each minibatch of data. The second kind, called Temporal Ensembling, evaluates only one of them and then uses its predictions in the inconsistency loss. As a compensation, predictions from multiple previous network evaluations are aggregated

into an ensemble prediction. Both methods use MSE to calculate the unsupervised inconsistency loss. Due to targets changing only once per epoch, Temporal Ensembling becomes unwieldy when learning large datasets. The difference between both methods in terms of how they compute the loss is shown in Figure 2.1 taken from the original paper.

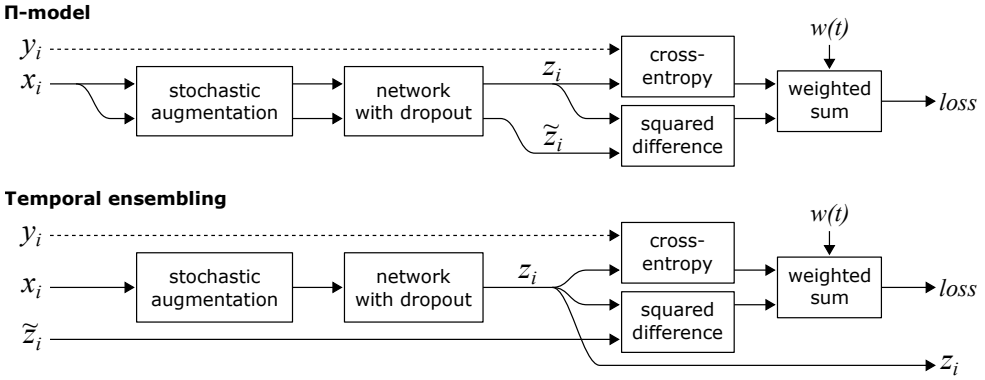


Figure 2.1: Two diagrams comparing the flow of data used to compute loss function in both the Π-model (top) and the Temporal Ensembling (bottom). Labels y_i are available only for the labeled inputs, and the associated cross-entropy loss component is evaluated only for those. Taken from the paper [16] by Laine and Aila.

2.1.3 Mean Teacher

To overcome the problem, that the targets change only once in an epoch, an approach called *Mean Teacher* has been proposed in [17]. Instead of aggregating predictions, it aggregates network's weights, more precisely, averages them. Mean Teacher maintains an exponential moving average (EMA) of the model weights. Averaging model weights over training steps produces a more accurate model than using only the final weights. This way, the method aggregates information after every step, not just after every epoch. Moreover, because the weights are averaged in all layers, not only the output softmax layer, the model learns faster from the unlabeled data. Theoretically, this should lead to better generalization, and the approach has indeed achieved smaller test errors on several known benchmarks.

2.1.4 Virtual Adversarial Training

In [18], the most sophisticated among the four considered approaches has been proposed, called *Virtual Adversarial Training*. The name comes from the loss function proposed by Goodfellow et al. to train networks against adversarial

inputs [22], and known as *adversarial loss*:

$$L_{\text{adv}}(x, \theta) = D[q(\cdot|x), p(\cdot|x + r_{\text{adv}}; \theta)] \tag{2.4}$$

$$\text{where } r_{\text{adv}} = \arg\max_{\|r\| \leq \varepsilon} D[q(\cdot|x), p(\cdot|x + r; \theta)], \tag{2.5}$$

where $q(\cdot|x)$ represents our knowledge of the true conditional distribution of labels given a particular input x, whereas $p(\cdot|x; \theta)$ represents the corresponding distribution implied by the neural network for particular values of their parameters θ, $\varepsilon > 0$ and D is some non-negative function on pairs of probability distribution, such as cross entropy, which was used in [18]. And the term "virtual" refers to the fact that in supervised learning, this loss needs to be minimized on unlabeled inputs instead on adversarial ones. In [18], our knowledge of the true conditional distribution of labels was represented by $p(\cdot|x; \hat{\theta})$, where $\hat{\theta}$ denotes the estimate of θ at the current stage of learning. This leads to a specific case of (2.4), a loss function that is called *local distributional smoothness (LDS)*:

$$LDS(x, \theta) = D[p(\cdot|x; \hat{\theta}), p(\cdot|x + r_{\text{vadv}}; \theta)] \tag{2.6}$$

$$\text{where } r_{\text{vadv}} = \arg\max_{\|r\| \leq \varepsilon} D[p(\cdot|x; \hat{\theta}), p(\cdot|x + r; \theta)]. \tag{2.7}$$

LDS is then averaged over all labeled and unlabeled data and combined with the supervised learning, which is in [18] controlled by maximum likelihood. Consequently, the final objective of semisupervised learning in this approach is the minimization

$$\min_{\theta}(-L_{\text{lik}}(\{(x, y)|x \in \mathcal{L}_\theta\}) + \frac{\alpha}{|\mathcal{L}| + |\mathcal{U}|} \sum_{x \in \mathcal{L} \cup \mathcal{U}} LDS(x, \theta)), \tag{2.8}$$

where $|\mathcal{L}|$ and $|\mathcal{U}|$ are the cardinalities of the sets $\mathcal{L}$ of labeled and $\mathcal{U}$ of unlabeled data, and $\alpha > 0$ is a trade-off coefficient. Moreover, the authors of [18] propose to enhance virtual adversarial training through adding, to the sum of functions minimized in (2.8), the entropy of the conditional distribution $p(\cdot|x; \theta)$ implied by the network, again averaged over all labeled and unlabeled data.

We have implemented the first two of those approaches, the second in both variants Π-model and Temporal Ensembling. Some details of our implementation are given in the next chapter.

2.2 Neural Networks in Malware and Network Intrusion Detection

As malware detection is strongly interconnected with and closely related to network intrusion detection, using ANN will be reviewed here in both areas. Probably the first proposal to use neural networks in them was in 1990 by Lunt [23] and was implemented two years later [24] in a network trained on inputs from audit log files.

The authors of [25] employed user commands as input, but rather than trying to learn benign and malicious command sequences, they were detecting anomalies in frequency histograms of user commands calculated for each user.

The paper by Cannady [26] summarised ANN advantages and disadvantages for misuse detection. As the two main advantages, the flexibility with respect to incomplete, distorted and noisy data, and the generalization ability are viewed, whereas as the main disadvantage, the ANN black-box nature.

In the late 1990s and early 2000s, self-organizing maps were quite popular in this context [27, 28, 29]. In particular, Depren et al. [28] used a hierarchical model where misuse detection based on *Self-Organizing Maps* (SOMs) was coupled with random forest-based rule system to provide not only high precision, but also some sort of explanation.

Much research has been devoted to comparing different kinds of ANN, or more generally, different classifiers including one or more kinds of ANN, on real-world malware detection or intrusion detection data. Probably the most popular among such data is an extensive intrusion detection dataset that was used at the 1999 KDD Cup [30]. Zhang et al. [31] compared five different kinds of ANN. Mukkamala et al. [32] compared a multilayer perceptron (MLP) with *Support Vector Machines*.

Among more recent ANN applications to malware and network intrusion detection, [33] should be mentioned for using synthetically generated attack samples to train an MLP, as well as [34] for a malware detection with recurrent networks. Expectedly, the kinds of ANN applied to these two areas during the last decade are most often deep networks [35, 36, 37]. In [38], deep learning was used together with spectral clustering to improve the detection rate of low frequency network attacks. ability to process raw inputs and learn their own features. Saxe et al. [39] employed a convolutional neural network (CNN) to extract features that were subsequently used as the input for an MLP detecting malicious activities. CNNs seem to be particularly suitable to learn spatial features of network traffic [40, 41]. In [40], a CNN was in addition combined with a long short term memory learning temporal features from multiple network packets.

To our best knowledge, there were so far only two particular ANN applications to malware or network intrusion detection that included semi-supervised learning in the narrow sense. In [1], various settings of semi-supervised lad-

der networks (see Section 2.1) were compared on the above mentioned intrusion detection dataset [30]. In [2] (cf. also the thesis [42]), *skipgram* networks [43] extended with semi-supervised learning based on pseudo-labels (see Section 2.1) were used for Android malware detection. Skipgrams are neural networks embedding large sets of structured non-numeric data into low-dimensional vector spaces. Whereas in [43], skipgrams were proposed for the embedding of text (*word2vec*), the input set in [2] is the set of rooted subgraphs around every node of three dependency graphs representing the API dependencies, permission dependencies, and information source and sink dependencies of the considered Android application. However, skipgrams were not used directly for malware detection in [2], only for representation learning of the structured input, whereas the malware detection itself was performed by a support vector machine. So far, no semi-supervised neural networks have been used directly for malware detection, and also none have been used with unstructured inputs simply listing values of the evaluated features, which are encountered much more frequently than dependency matrices.

Chapter 3

Design and Implementation

Most of the implementations of semi-supervised deep learning algorithms we found, including those in the overview paper [20], were designed for the image data. Their performance was tested on common image classification benchmark datasets like MNIST [44], The Street View House Numbers (SVHN) [45] or CIFAR-10 [46]. Authors usually used a convolutional neural network (CNN) as the underlying machine learning model. CNNs are types of feed-forward neural networks that are able to learn filters to detect features in images. Their advantage is that they can find hierarchical patterns in image data represented as a two-dimensional array.

We intended to use our implementation with more general data. Therefore we used a multilayer perceptron (MLP) as our underlying network, which is a universal model. On top of this model, we implemented three semi-supervised learning algorithms: Pseudo-labeling, Π-model and Temporal Ensembling. We used Python frameworks Tensorflow and Keras and functions from Scikit-learn library. Most parts of these algorithms we used share the same implementation. Fundamentally, they only differ in the way they compute the unsupervised component of the loss function. Firstly, both methods use the same MLP architecture with ReLU as the activation function in the hidden layers and utilize the same optimizing algorithm Adam [11] with the initial learning rate set to 0.001, $\beta_1 = 0.99$, and $\beta_2 = 0.999$. As was shown above, the optimized loss function is defined as a weighted sum of supervised and unsupervised loss:

$$L = L_S + w(t)L_U. \tag{3.1}$$

The weight $w(t)$ depends on the ratio between the number of labeled and unlabeled data, and the current epoch. We ramp up the value of the weight

using a Gaussian curve:

$$w(t) = w_{\max} \frac{|\mathcal{L}|}{|\mathcal{L}| + |\mathcal{U}|} e^{\left(-5(1-t)^2\right)}, \tag{3.2}$$

where $t = \max(\frac{e}{r_u}, 1)$, e is number of the current epoch,r_u is the length of the ramp up period and w_{max} is a parameter specifying the maximum weight.

Increasing the weight of the unsupervised loss during the training is necessary as the network needs to learn to classify the supervised data first. Eventually, it can learn to incorporate the unlabeled information as well. Similarly, at the later phase of the training, the learning rate and the β_1 parameter of the Adam optimizer are decreased to improve the exploitation:

$$lr_e = w_d lr_{e-1} \quad \text{and} \quad \beta_1 = 0.4 w_d + 0.5, \tag{3.3}$$

where $w_d = e^{\left(-12.5t^2\right)}$, $t = \max(\frac{e}{r_d}, 1)$ and r_d is the length of the ramp down period. We also included a type of elitism to select the resulting model with the lowest total loss per epoch calculated with the maximal weight for the unsupervised component instead of a weight in the current epoch.

3.1 Pseudo-Labeling

The unsupervised loss in the Pseudo-labeling algorithm is calculated using cross-entropy between network's predictions and pseudo-labels, but only for predictions with confidence above a specified threshold ϑ (cf. 2.1). We compute the vector of pseudo-labels y' for every data sample x and the corresponding network output $f(x)$ using the following formula for i-th component:

$$y'_i = \begin{cases} 1 & \text{if } i = \operatorname{argmax}_{i'} f_{i'}(x) \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Then the resulting formula based on cross entropy for the unsupervised loss component L_U of a particular data sample x is:

$$L_U(x) = -\sum_{i=1}^{|C|} y'_i \log(y_i), \tag{3.5}$$

where $|C|$ is the number of classes.

3.2 Increasing the Consistency

We also implemented two variants of the consistency preserving, self-ensembling algorithms: The Π-model and the Temporal Ensembling. Both approaches use mean squared error (MSE) to compute unsupervised loss. What is different is

the target for which is MSE evaluated. Π-model compares two predictions of the same state of the network using different inputs and different dropped out neurons. To augment the data for the second prediction, we multiplied the input feature vector with a noise sampled from normal distribution $\mathcal{N}(1, \sigma^2)$. We chose to multiply the data with the noise instead of adding it because it is invariant to the differing variances of the individual features.

The second variant, Temporal Ensembling, compares the prediction of the network in the current epoch with the predictions obtained in the previous epoch. The dropout and data augmentation can be used as well. So the unsupervised loss L_U for this approach is calculated as follows:

$$L_U(x) = \sum_{i=1}^{|C|}(y_i - \tilde{y}_i)^2, \tag{3.6}$$

where y is the current output of the network in the training step and $\tilde{y}$ is the output of the network in a different state or for augmented input.

Chapter 4

Experiments

Firstly, we tried our implementations of two semi-supervised methods mentioned above and a fully supervised baseline on a two-dimensional example. We chose simple generated moon-shaped data, which are often used for visualization of classification or clustering algorithms.

4.1 Experiment With Moon-Shaped Data

The data consist of two classes that are linearly inseparable but do not overlap so that the classification can be performed with no error. The advantage is that we can easily visualize the classification decision border in two dimensions and examine the behavior of the algorithm. An example of generated moon-shaped data is visualized in Figure 4.1. There are together 1500 points, one half is red and the other blue. The points are randomly projected on a half-circle with additional Gaussian noise with a standard deviation equal to 0.08. To generate the data, we used the function make_moons from the Scikit-learn Python library. Then we normalized the points between 0 and 1. For every method in this experiment, we used the same MLP architecture with two hidden layers, the first having 64 neurons and the second 32 neurons.

In Figure 4.2, we present two different arrangements of labeled and unlabeled data, each solved by the fully supervised learning, Pseudo-labeling, and Π-model. In the first experiment, we tested the ability of the algorithm to learn from a small amount of data, there are two moon-shaped clusters, each having 1000 samples, where only 16 of each are labeled. We let each network to train for 300 epochs. Even though the supervised learning had available samples distributed over the whole cluster, it was not able to learn the correct shape using only 32 samples. The Pseudo-labeling algorithm could not improve the results using the unlabeled data. However, the results of the Π-model are notably better as it managed to capture the moon shape quite well.

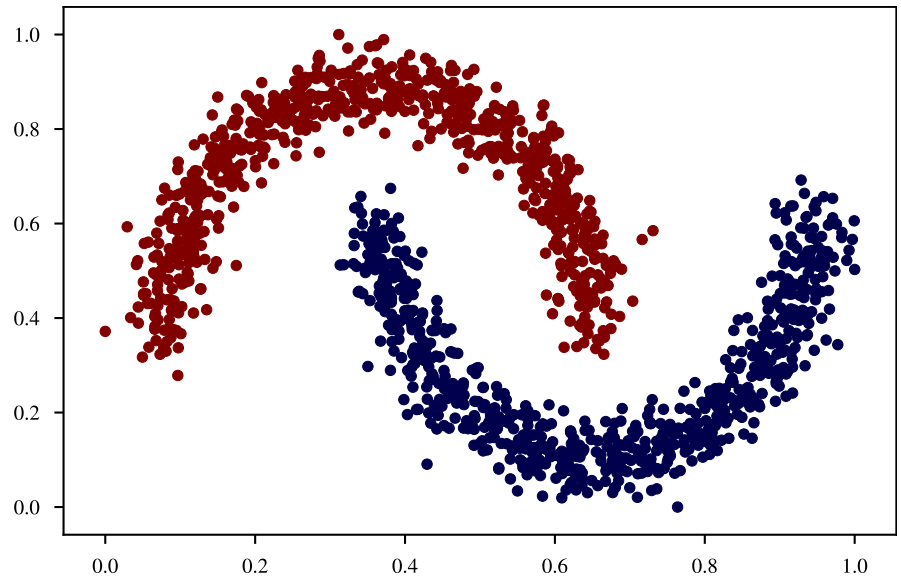
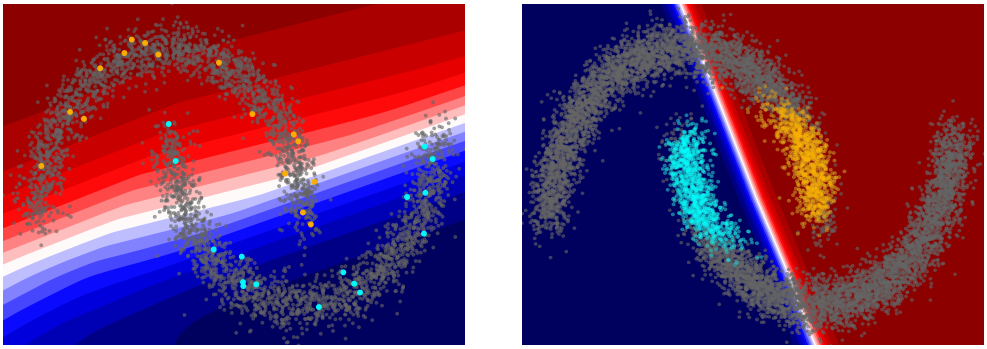


Figure 4.1: Visualization of a simple two moons dataset. There are 1500 two-dimensional points that lie on two half-circles with additiona noise.

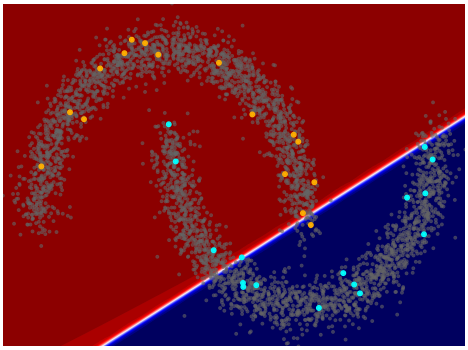
In the second experiment, we tried if the algorithms can deal with a drift in the training data. This time we used clusters with 10000 samples and labeled only 1000 points that lie near the center, for each class. We trained the networks for 100 epochs as having it run longer did not improve the results of either of the methods. The supervised algorithm could only use the labeled data that are linearly separable. So it learned to classify the labeled data with zero error, and we present it only as a baseline for comparison. Pseudo-labeling again failed to use the information contained in the unlabeled data, and its accuracy was similar to the fully supervised learning. Also in this task, the Π-model was able to use the smoothness of the data and performed the best of three methods. To quantify the results, we summarized the prediction accuracies tested on the whole clusters in Table 4.1.

Completing these experiments, we observed that the results of the Pseudo-labeling correspond to the idea behind the algorithm. It makes the network's decision more confident as it uses the interim predictions as if they were the true labels. Also, the decision border did not seem to converge to a stable finale state throughout the learning. It kept shifting closer to one or the other class, roughly in the range where the confidence of the supervised learning was low. We managed to get decent results using the Π-model, and it proved to be able to capture the smooth distribution of data. However, the algorithm was susceptible to inappropriate setting of hyperparameters. It often happened that one class became dominant during the training, and the Π-model could not recover from that.

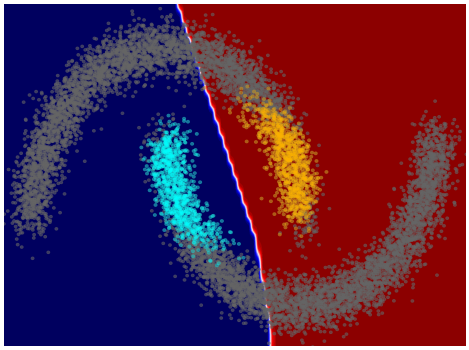


(a) supervised

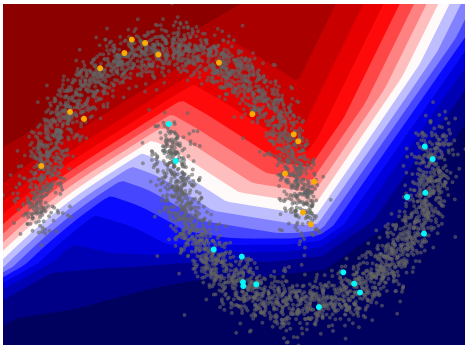
(b) supervised



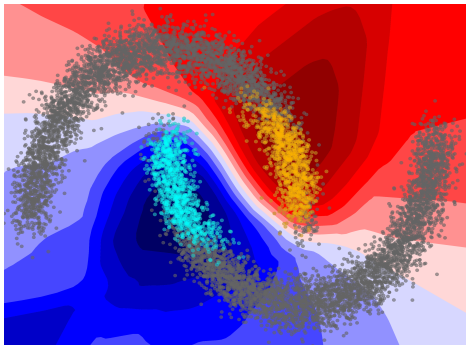
(c) pseudo



(d) pseudo



(e) pi-model



(f) pi-model

Figure 4.2: Comparison of the decision border of three algorithms on simple moon-shaped data. The decision border is visualized as a transition from blue to red. The saturation expresses the classification confidence of the network. The labeled data are shown as cyan or orange circles, while unlabeled are drawn in gray. On the left side, we randomly labeled only 16 samples out of 2000 from each class. On the right side, we labeled 1000 samples close to the center out of 5000 from each class.

Table 4.1: A summary of test accuracies on simple moon-shaped data. The table compares Pseudo-labeling, Π-model, and fully supervised learning on a test data covering the whole moon cluster. There are results of two experiments. In the first one, only 16 points out of 1000 were uniformly selected and labeled for both classes. In the second, we labeled 1000 points in the center out of 10000 samples for both classes.

Method	Test case	
	16 pts uniform	1000 pts in center
Supervised	89.1 %	46.2 %
Pseudo-label	85.4 %	42.9 %
Π-model	95.7 %	76.0 %

4.2 Experiments With a Real-World Malware Dataset

4.2.1 Data

We tested our implementation using a real-world malware detection dataset provided by Avast. The data concern Windows Portable Executable files, which were collected during 380 weeks. It consists of 540 real-valued features derived directly from the binary files using static analysis. Because of theír confidentiality, the data were anonymized. This means that we do not know the meaning of each feature, so we can not tell which properties of the binary files are more important than others. Each file is labeled with one of the five classes:

- Malware
- Adware
- Infected
- Potentially unwanted program
- Clean

There were some attributes with zero or very low variance among the whole set. Therefore we used principal component analysis (PCA) to reduce the dimensionality of the feature space and speed up the training. First, we min-max normalized the data between 0 and 1, and then we projected them to the subspace spanned by the 128 main components while keeping more than 99 % of the explained variance.

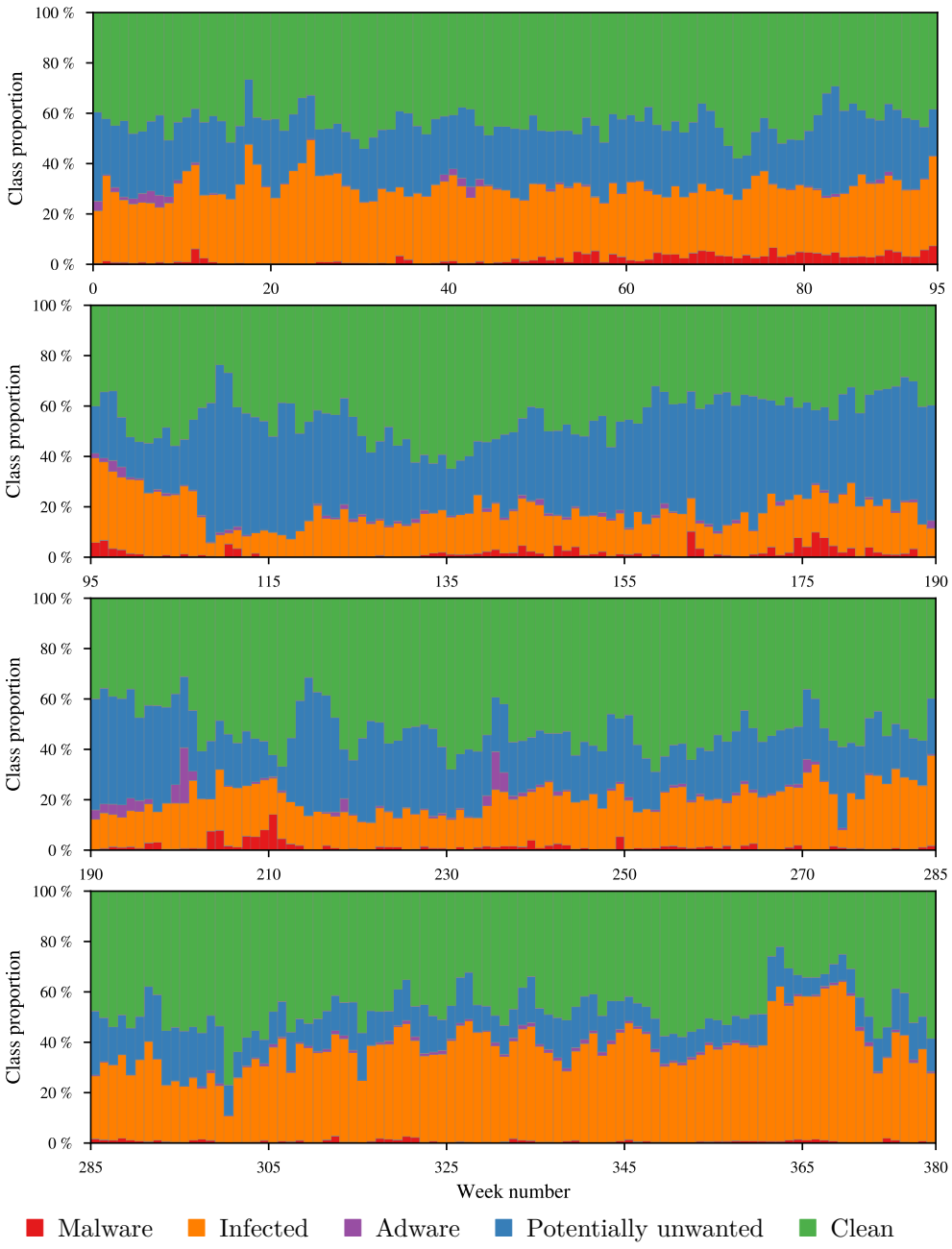


Figure 4.3: Visualization of the drift in the proportion of classes in time. The plots show the ratio between individual classes of binary files for each of the 380 weeks. Each type of file is represented by a different color.

Over time, the structure and the properties of the malware change. Therefore, the features of the dataset change too. This drift has a notable impact on the performance of machine learning models working with such data. The models tend to lose the classification accuracy gradually as the malware evolves. But the evaluation of these models is also affected by the drift in the proportion of classes of captured files. We analyzed the distribution of malware types and present it in Figure 4.3. There can be seen major shifts in the occurrence of some malware classes in the range of tens of weeks. Also, some classes, such as adware, are underrepresented.

4.2.2 Hyperparameters

At first, we analyzed the hyperparameters of each algorithm and optimized those that we expected to have the greatest impact, based on the results during early tests of our implementation. We chose the data from five weeks between the 50th and the 55th week. We performed stratified random sampling and selected 10,000 training and 5000 testing records. We kept only 5 % of the labeled from the training set, and the rest remained unlabeled. Using this data, we evaluated the classification accuracy for various sets of hyperparameters.

For the Pseudo-labeling algorithm, we optimized the threshold ϑ and the maximal weight w_{max} for the unsupervised loss component. For the consistency preserving algorithms, we optimized the standard deviation σ of the noise used in data augmentation and again the parameter w_{max}. Furthermore, we repeated the search of parameters for all six combinations of variants of the algorithm, which were: Π-model or Temporal Ensembling and whether to use dropout, augmentation or both. We took the parameters from the following sets:

$$\begin{aligned} w_{max} &\in \{0.1, 1, 2, 5, 10, 15, 20, 30, 50\}, \\ \sigma &\in \{0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5\}, \\ \vartheta &\in \{0.5, 0.7, 0.8, 0.9, 0.95, 0.98, 0.99\}. \end{aligned}$$

However, because of the high time requirements, we did not perform the full factorial search through all possible combinations. Instead, we optimized only one parameter at time, keeping others on default values which were:

$$\begin{aligned} w_{max} &= 30, \\ \sigma &= 0.1, \\ \vartheta &= 0.9. \end{aligned}$$

The rest of the hyperparameters we used as stated in the original papers or we modified them slightly according to our observations because the domain of our dataset is entirely different. The results of hyperparameter optimization are visualized using bar charts in Figure 4.4.

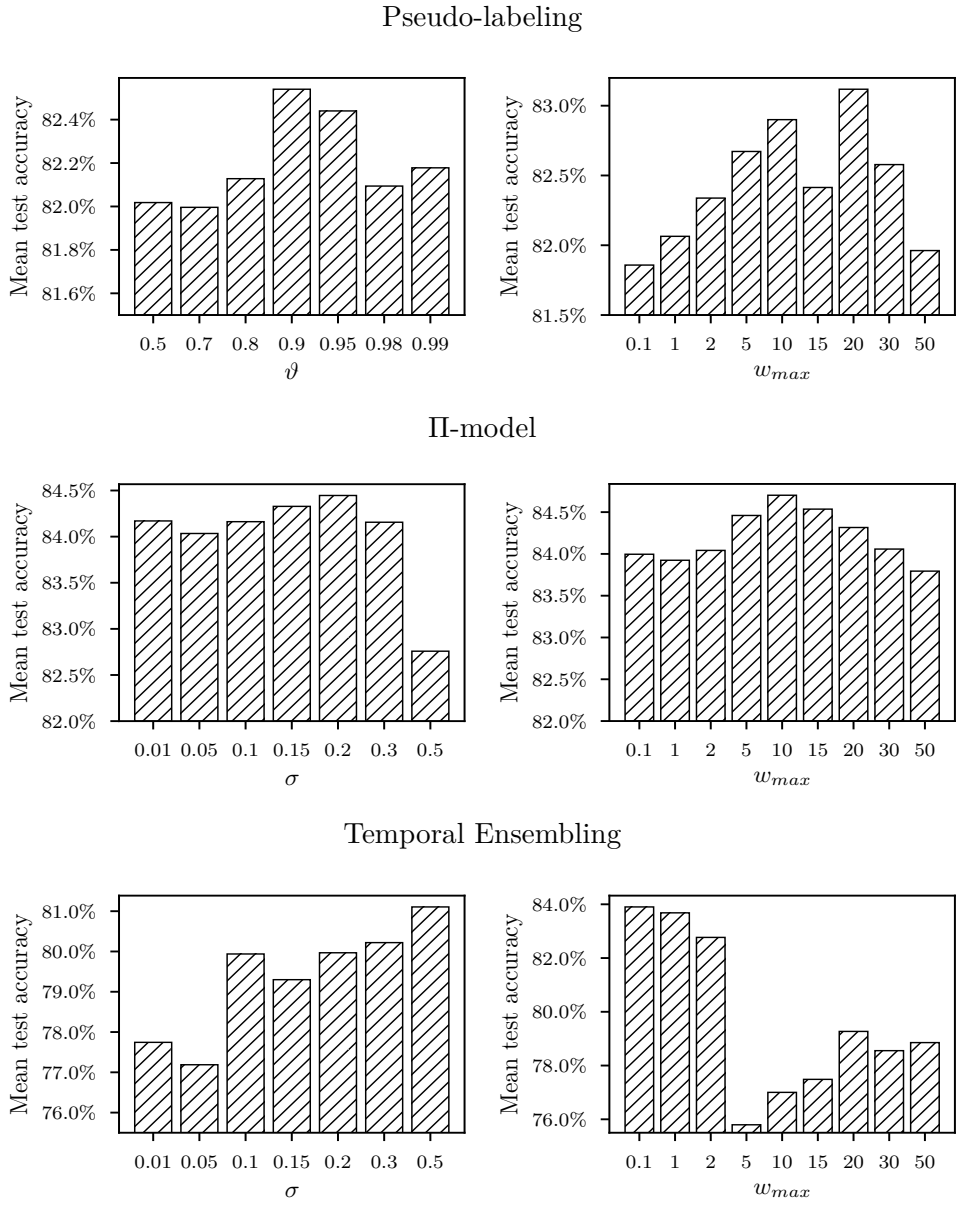


Figure 4.4: Results of the hyperparameter optimization. There are two bar charts for each implemented method, showing the mean test accuracy with different values of hyperparameters.

Each bar chart represents the average test accuracy from 20 independent training runs. The Pseudo-labeling algorithm performed the best with $\vartheta = 0.9$ and $w_{max} = 20$. However, the differences in the results were small, especially for the threshold ϑ. Then we tested two variants of the consistency increasing algorithm. We present only results, in which both augmentation and dropout were enabled because they reached the best accuracy. The better of the two was Π-model. The best noise deviation σ seemed to be 0.2 and w_{max} 10. The results concerning the maximal weight of unsupervised loss w_{max} in the Temporal Ensembling suggest that this semi-supervised algorithm had a rather negative effect on the accuracy. The network with the lowest values of the weight, therefore with a little semi-supervised influence, performed much better than with higher values. Interestingly, higher values of the σ for noise worked better with Temporal Ensembling. Based on these results, we continued only with Pseudo-labeling and Π-model in later experiments. The final values of the chosen hyperparameters used in experiments follow in Table 4.2. In every experiment, we used the same MLP architecture with five layers and the topology 128-96-64-32-5.

Table 4.2: Final setting of training parameters and model hyperparameters.

Common	
Number of training epochs	100
Training batch size	100
Weight ramp-up period r_u	70
Optimizer ramp-down period r_d	20
Initial learning rate	0.001
Pseudo-Labeling	
Pseudo-labeling threshold ϑ	0.9
Maximal weight w_{max}	10
Consistency preserving	
Consistency preserving variant	Π-model
Use dropout	Yes
Use data augmentation	Yes
Maximal weight w_{max}	20
Standard deviation σ of the noise	0.2

4.2.3 Experimental Design

Then we measured the performance of the Pseudo-labeling, Π-model, and the purely supervised baseline for different proportions of labeled data. We varied the ratio $r = |\mathcal{L}| : (|\mathcal{L}| + |\mathcal{U}|)$ in the range

$$r \in \{0.5\%, 1\%, 2\%, 5\%, 10\%, 25\%, 50\%, 75\%\}.$$

As the training union of labeled and unlabeled data, we took 10,000 stratified samples from 5 consequent weeks and split them in the considered ratios. Then we trained 20 separate instances of the network and calculated the average accuracy on a stratified test set of size 5000 for them. We repeated this experiment for four distinct groups of weeks: 1-5, 51-55, 101-105, and 151-155. We also evaluated the performance of trained networks on the data from all of the following weeks. This is particularly interesting from the point of view of the considered application domain. Because the structure of malware changes over time, the prediction accuracy of the newer data tends to get worse. That means that if semi-supervised learning could overcome this problem, it could be beneficial. Therefore, we also tried to take the data from newer periods than the labeled weeks as the unlabeled training set. So we trained the network using labeled data together with unlabeled data from several weeks later. Unfortunately, we did not manage to outperform the standard fully supervised learning this way using any of the implemented methods. We present the results of these experiments in the following section.

4.3 Results

Using the hyperparameters setting presented in the previous section, we measured the average test accuracy of 20 training runs of our three implementations in relation to the proportion of the labeled data in the training data set. The results can be found in Table 4.3. We can see that the performance of the fully supervised learning depends on the number of labeled data as it is the only learning source for the network. The results of the semi-supervised algorithms Pseudo-labeling and Π-model are more interesting. Both algorithms bring a slight increase in the accuracy of low ratios of the labels. The most noticeable improvement is when there are only around 1 or 2 % of labels. When the ratio gets above 10 %, the accuracy gain is negligible, and for the higher values, the semi-supervised effect is even negative. Also, it seems that Π-model outperforms Pseudo-labeling, as its accuracy is higher in most of the measurements.

Table 4.3: Comparison of the Π-model, Pseudo-labeling and the supervised baseline, relative to the ratio of labeled data. The table depicts the percentage of the average testing accuracy on four different periods. The S columns contain the results of the supervised baseline, the Δ P and Δ Π columns show the difference using Pseudo-labeling and Π-model respectively.

Ratio	Weeks: 1–5			Weeks: 51–55		
	S	Δ P	Δ Π	S	Δ P	Δ Π
0.5 %	67.9	+0.4	+3.1	63.9	+2.8	+3.5
1 %	71.0	+1.7	+4.5	67.1	+5.0	+6.4
2 %	76.8	+1.3	+2.5	73.9	+3.4	+5.7
5 %	82.4	−0.1	+1.1	82.2	+0.4	+2.4
10 %	85.1	+0.0	+1.1	86.1	−0.4	+0.8
25 %	88.3	−0.4	+0.3	89.2	−0.5	+0.1
50 %	89.9	−0.4	−0.1	90.6	−0.7	+0.0
75 %	90.4	−0.1	−0.1	91.2	−0.3	−0.3
100 %	90.9			91.4		

Ratio	Weeks: 101–105			Weeks: 151–155		
	S	Δ P	Δ Π	S	Δ P	Δ Π
0.5 %	56.8	+5.2	+6.8	67.6	+0.3	+1.9
1 %	61.8	+6.9	+9.0	70.3	+5.7	+6.4
2 %	69.7	+5.9	+6.2	76.6	+1.9	+2.0
5 %	77.8	+3.7	+3.3	80.4	+0.2	+0.8
10 %	83.2	+1.0	+1.1	81.7	+0.3	+0.6
25 %	87.4	+0.7	+0.3	83.1	+0.3	+0.3
50 %	89.8	−0.3	−0.2	84.2	−0.1	−0.2
75 %	90.7	−0.1	−0.4	84.4	+0.3	−0.2
100 %	91.3			84.8		

4.3.1 Statistical Tests

To verify our observations, we performed multiple comparisons statistical tests for the classification accuracies of three implemented methods. We made more measurements using another testing data to see whether the methods are significantly different. We took the testing data from all of the weeks later than the training set. Then we evaluated the accuracy for each week and for all considered ratios of labeled and unlabeled data. For each combination of ratio, testing week, and algorithm, we obtained twenty different results using twenty separately trained instances of the networks.

Firstly, we applied the Friedman test [47] to find out whether all the methods achieve the same accuracy. The Friedman test is a nonparametric test that compares at least three matched groups of measurements. It uses the ranking of results of different methods on the same testing data. In our case, it orders the accuracies of three methods from low to high and assigns the rank to them. Then it sums all the ranks of each method separately and compares the sums. If the sums are very different, the resulting p-value of the test will be small. We present the results of the Friedman test in Table 4.4. All of the p-values were small enough that we could reject the null hypothesis of the equal accuracy of all three methods in all considered cases. Because we performed multiple comparisons at once, we corrected the p-values with the Holm method [48]. After that, we still could conclude that all three methods are not equal at the 5 % level of family-wise significance.

Table 4.4: Results of the Friedman statistical tests. There are p-values for every tested combination of ratio of labeled sample and training week. All values are very low so that the null hypothesis of the method equality can be rejected in every case with the Holm correction at a 5 % level of family-wise significance.

Ratio	Training weeks			
	1–5	51–55	101–105	151–155
0.5 %	1.87×10^{-40}	3.64×10^{-21}	1.01×10^{-09}	7.90×10^{-31}
1 %	2.72×10^{-03}	4.76×10^{-40}	2.90×10^{-25}	0.00×1
2 %	7.24×10^{-27}	1.75×10^{-20}	1.49×10^{-12}	4.98×10^{-13}
5 %	2.45×10^{-15}	2.57×10^{-11}	0.00×1	7.18×10^{-25}
10 %	1.46×10^{-06}	0.00×1	1.51×10^{-20}	0.00×1
25 %	0.00×1	0.00×1	1.49×10^{-12}	8.41×10^{-45}
50 %	1.69×10^{-25}	6.24×10^{-38}	0.00×1	0.00×1
75 %	3.03×10^{-26}	1.98×10^{-08}	3.40×10^{-25}	0.00×1

Table 4.5: Results of the post hoc statistical tests. The number is the p-value of the test, and the asterisk marks whether the null hypothesis was rejected.

Supervised and Pseudo-labeling

Ratio	Training weeks			
	1–5	51–55	101–105	151–155
0.5 %	1.68×1	1.41×10^{-05} *	2.01×1	1.59×1
1 %	2.03×1	7.97×10^{-06} *	5.10×10^{-01}	0.00×1 *
2 %	3.82×10^{-12} *	1.90×10^{-08} *	1.96×10^{-02} *	7.48×10^{-09} *
5 %	3.90×10^{-14} *	1.18×10^{-10} *	3.00×10^{-25} *	1.02×10^{-04} *
10 %	2.23×10^{-01}	3.52×10^{-18} *	1.20×10^{-20} *	4.20×10^{-45} *
25 %	6.84×10^{-17} *	0.00×1 *	1.14×10^{-12} *	4.20×10^{-45} *
50 %	1.02×10^{-14} *	4.80×10^{-24} *	2.53×10^{-02} *	0.00×1 *
75 %	3.64×10^{-20} *	1.96×10^{-02} *	2.37×10^{-01}	0.00×1 *

Supervised and Π-model

Ratio	Training weeks			
	1–5	51–55	101–105	151–155
0.5 %	1.84×10^{-33} *	2.90×10^{-05} *	5.90×10^{-08} *	3.72×10^{-22} *
1 %	1.91×10^{-02} *	3.00×10^{-15} *	1.00×10^{-22} *	0.00×1 *
2 %	9.76×10^{-03} *	1.44×10^{-02} *	1.41×10^{-12} *	7.76×10^{-11} *
5 %	5.66×10^{-01}	4.42×10^{-01}	7.85×10^{-44} *	4.11×10^{-08} *
10 %	5.54×10^{-02}	3.45×10^{-12} *	4.00×10^{-07} *	2.42×10^{-05} *
25 %	6.23×10^{-17} *	3.74×10^{-15} *	2.72×10^{-03} *	4.72×10^{-08} *
50 %	3.72×10^{-01}	5.66×10^{-01}	4.45×10^{-40} *	1.23×10^{-04} *
75 %	5.91×10^{-20} *	3.70×10^{-08} *	5.56×10^{-14} *	1.27×10^{-24} *

Pseudo-labeling and Π-model

Ratio	Training weeks			
	1–5	51–55	101–105	151–155
0.5 %	7.24×10^{-28} *	1.64×10^{-21} *	2.19×10^{-06} *	8.24×10^{-25} *
1 %	1.87×10^{-01}	6.25×10^{-40} *	4.79×10^{-15} *	1.79×10^{-06} *
2 %	3.08×10^{-26} *	3.64×10^{-20} *	4.07×10^{-04} *	2.03×1
5 %	1.74×10^{-08} *	1.93×10^{-05} *	1.43×10^{-02} *	4.49×10^{-25} *
10 %	3.90×10^{-06} *	0.00×1 *	1.44×10^{-03} *	1.71×10^{-19} *
25 %	0.00×1 *	8.13×10^{-28} *	3.93×10^{-03} *	4.84×10^{-15} *
50 %	2.91×10^{-23} *	5.32×10^{-33} *	0.00×1 *	2.21×10^{-24} *
75 %	1.11×1	6.44×10^{-02}	2.91×10^{-23} *	9.82×10^{-08} *

Subsequently, we compared all three pairs of algorithms separately using a post hoc pairwise test. Supervised to Pseudo-labeling, Supervised to Π-model, and Pseudo-labeling to Π-model. The results of it can be found in Table 4.5. We again present the p-values of the test. The tests, where we could reject the null hypothesis of the equal accuracy of both compared methods at the 5 % level of family-wise significance with Holm correction, are marked with an asterisk. A significant difference between the compared methods was found for 80 among the 96 compared pairs corresponding to the 32 combinations of training weeks and ratios.

In the cases where the pairwise test identified methods to be different, we compared the means of accuracies to decide which method performed better. We summarized the results in Table 4.6. If we consider only tests with ratio up to 5 %, where the difference was significant, then the Pseudo-labeling was significantly better than supervised learning in 3 cases and the Π-model in 11 cases. Pseudo-labeling was significantly better than Π-model in only 3 out of 14 significant comparisons. In tests with a higher ratio of labeled samples, the results are less conclusive.

Table 4.6: Multiple comparisons test of three methods for different ratios of labeled data, tested on the data from all of the following weeks till the end. Each cell contains a triplet of symbols representing the results of three post hoc pairwise tests. The order of the comparisons is: supervised to Pseudo-labeling, supervised to Π-model, and Pseudo-labeling to Π-model. The dash means that the comparison was statistically insignificant and three letters S, P, and Π marks whether supervised, Pseudo-labeling, or Π-model were significantly better than the other algorithm in the test.

Ratio	Training weeks			
	1–5	51–55	101–105	151–155
0.5 %	–, Π, Π	S, Π, Π	–, Π, Π	–, Π, Π
1 %	–, Π, –	S, Π, Π	–, Π, Π	P, Π, Π
2 %	S, S, Π	S, S, Π	P, Π, P	S, S, –
5 %	S, –, Π	S, –, P	P, Π, P	S, Π, Π
10 %	–, –, Π	S, Π, Π	S, S, P	S, S, Π
25 %	P, Π, Π	S, S, Π	P, S, P	S, S, Π
50 %	P, –, Π	S, –, Π	P, S, P	S, S, Π
75 %	P, S, –	P, S, –	–, S, P	S, S, Π

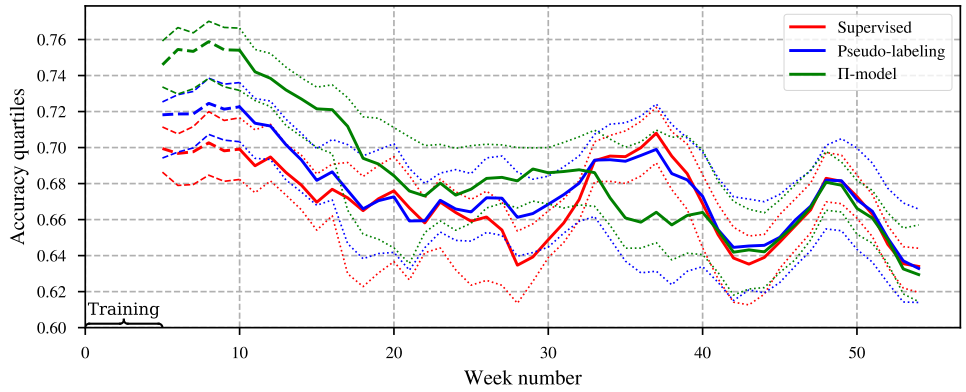


Figure 4.5: The progression of the classification accuracy evaluated on weeks between the 5th and the 55th.

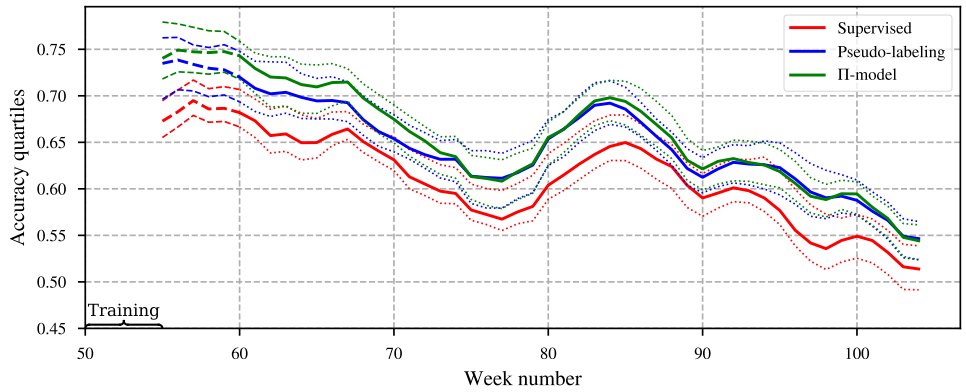


Figure 4.6: The progression of the classification accuracy evaluated on weeks between the 55th and the 105th using Pseudo-labeling, Π-model, and fully supervised learning, trained using a dataset with 1 % of labels. For each plot, there are three quartiles visualized; the median is drawn with a solid line, while the first and the third quartiles are dotted. The curves correspond to the moving average with the window size of five weeks. The first five dashed weeks are means of all previous weeks. The first five weeks at the beginning of each plot were used for the training.

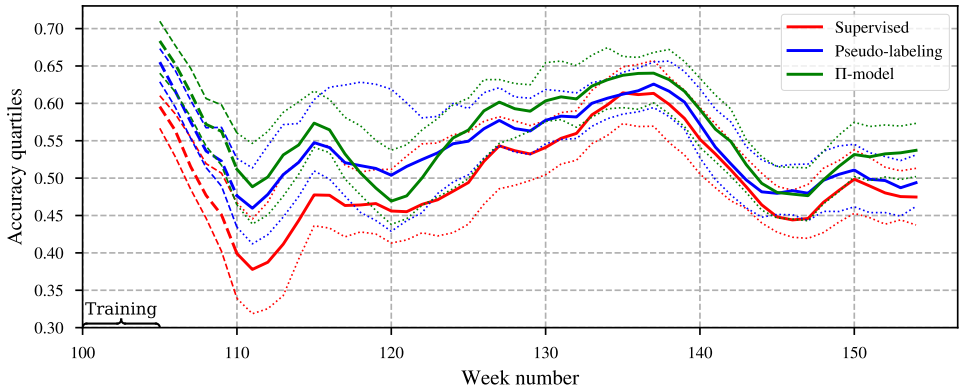


Figure 4.7: The progression of the classification accuracy evaluated on weeks between the 105th and the 155th.

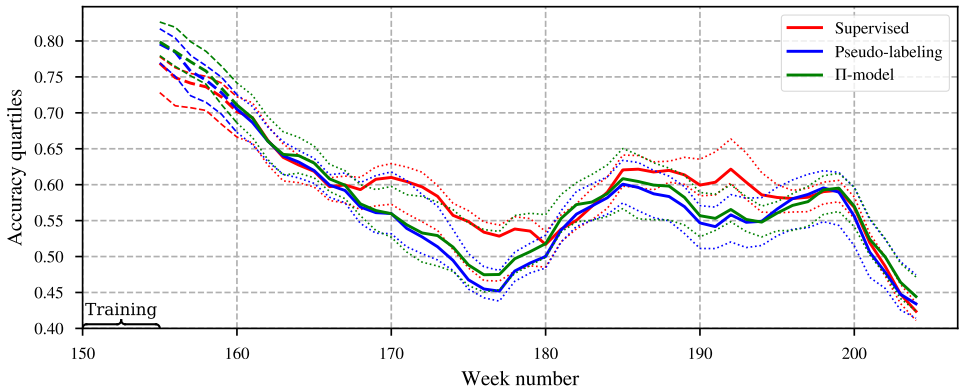


Figure 4.8: The progression of the classification accuracy evaluated on weeks between the 155th and the 205th using Pseudo-labeling, Π-model, and fully supervised learning, trained using a dataset with 1 % of labels. For each plot, there are three quartiles visualized; the median is drawn with a solid line, while the first and the third quartiles are dotted. The curves correspond to the moving average with the window size of five weeks. The first five dashed weeks are means of all previous weeks. The first five weeks at the beginning of each plot were used for the training.

We also visualized the progress of the classification accuracy over time in plots, examples of which are in Figures 4.5 to 4.8. To capture the variance of the results, we plotted three quartiles. Because the accuracies oscillated greatly through the individual weeks, we used a moving average to smooth the curves. We can see that both semi-supervised algorithms slightly improved the accuracy of the network on the roughly first 30 weeks. The Pseudo-labeling is around 1 or 2 % better than supervised learning, while Π-model gets another 1 or 2 % above the Pseudo-labeling. After around 40 weeks, the results of all three methods are very similar. As the properties of the data shift over time, the overall results on the data beyond 50 weeks got considerably worse and fluctuated more for all methods.

4.3.2 Semisupervised Learing With Data Drift

Then we tried to use semi-supervised learning of neural networks to overcome the problem with the drift of data over time. We tried to simulate a real use case of machine learning in malware detection. We fixed the labeled part of the training to a certain period of time, like in previous experiments. Nevertheless, we wanted to use also the information in the unlabeled newer weeks. To create a more realistically sized dataset, we increased the number of training weeks as well as the number of samples. We took 50,000 samples from a ten-week period. To choose suitable weeks for the experiment, we first trained our fully supervised baseline several times on fives and tens consequent weeks, starting at different points in time. Then we evaluated the classification accuracy in all of the following weeks to see the drift in data.

We selected data between the 275th week and the 285th week because the test accuracy oscillated less, and there was a somewhat steady decrease in the accuracy in the first twenty weeks. Then we used the data in later weeks as unlabeled for semi-supervised learning. For each following week, we trained the network again, taking all samples in one week as the unlabeled set. Because training the network for almost a hundred times was rather demanding on computational time, we decided to examine only one semi-supervised algorithm. We used the Π-model because it performed better so far, and the Pseudo-labeling did not seem to work well with the data drift in the simple experiment with moon data. The classification accuracy was evaluated on the same samples as in the unlabeled training part.

The results of this experiment are visualized in Figure 4.9. Unfortunately, the semi-supervised Π-model did not manage to use the unlabeled data to improve the classification at all. The accuracy is worse in all but three weeks and is lower by 5.5 % on average. Moreover, there are three major drops in the accuracy visible on the plot.

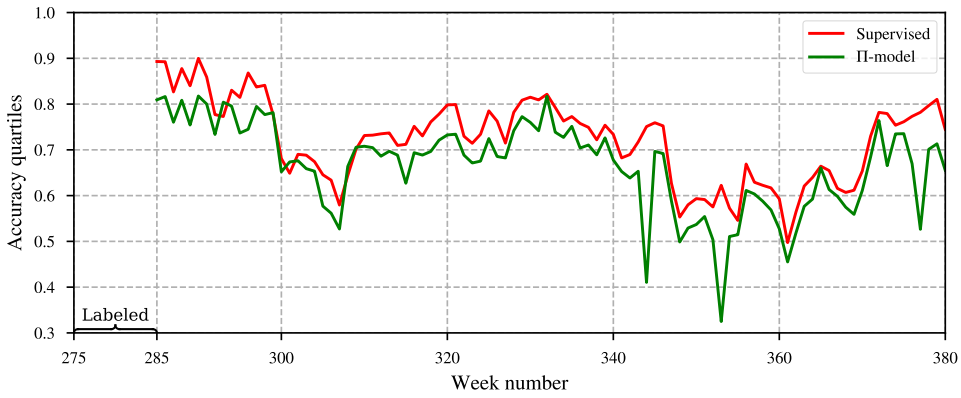


Figure 4.9: The use of Π-model with unlabeled data from later weeks. The plot compares the test accuracies of a fully supervised baseline (red) with the Π-model (green). The data from ten weeks between the 275th and the 285th week were labeled. Supervised learning used only those, while Π-model used samples from the particular tested week as unlabeled.

4.4 Discussion and Future Work

We compared two semi-supervised algorithms Pseudo-labeling and Π-model. These algorithms were both designed to be used for the classification of image data with convolutional neural networks. We applied them to a completely different domain of data extracted from malware files. Nonetheless, we will try to compare our results with these studies. While the original papers of algorithms present better results, we will try to relate our observations with the overview paper [20] as it provides a more realistic evaluation. The authors of the paper from the Google Brain team tested these algorithms on two datasets. On CIFAR with 4000 labeled images and SVHN with 1000 labeled images. They reported that the Pseudo-labeling increased the accuracy on CIFAR by 2.5 % and by 5.2 % on SVHN. The improvement with the Π-model was 3.9 % and 5.6 %, respectively. When we tested these methods using the malware dataset, we noted that it performed the best when around 1 % of data were labeled, which corresponds to only 100 samples out of 10,000. In this setting, our implementation of Pseudo-labeling brought an accuracy increase between 1.7 % and 6.9 %, depending on the test case. The increase by Π-model ranged from 4.5 % to 9 %. When we evaluated the accuracies on the data from later weeks with a drift in the features and class proportions, the improvements were slightly less noticeable. However, we should note that the overall classification accuracy was lower in our experiments than in the original articles, which leaves more space for the improvement with the use of semi-supervised learning.

Even though the experiments are not directly comparable because of the difference in the domain, we assume that we have shown that these algorithms can be used with non-image data too. Though, it seems that these algorithms are useful only when a limited number of labeled samples are available. That might be useful for some applications where the ground truth outputs are difficult or expensive to obtain. However, that is not necessarily the case with malware data as the antivirus companies usually have access to a large amount of training data. What we think would be more beneficial is if semi-supervised learning could help to capture the evolution of malicious files. That is what we tried to accomplish in the last experiment in the previous section. However, the effect of the semi-supervised Π-model was almost only negative. There are several possible reasons for that.

The first explanation could be that we just did not find the appropriate setting of learning parameters and hyperparameters of the Π-model and thoroughly tuned parameters would outperform the semi-supervised baseline. But we do not think that this is the case because we manually tried a lot of different settings and still did not manage to get better results with Π-model. A different reason might be that drift in the data is that strong, that it breaks the assumptions stated in the first chapter. In other words, it would mean that the newly recorded files and their corresponding calculated features are that different, that they produce a completely different output of the network. Therefore Π-model would not be able to find a correct connection between the old and the new data. By looking at Figure 4.9, we can see that there is a quite large and steady decline in the accuracy in the first 20 test weeks. It could be caused by a gradual shift in the properties of data, but also by a constant increase in the proportion of new completely different samples. Another reason could be that the Π-model algorithm is only useful when the labeled set is much smaller than the unlabeled. However, in the last experiment, the labeled set was usually around two times larger than the unlabeled set. This seems likely when we consider the results of our previous experiment with different ratios of labeled data.

But we believe that it does not mean that semi-supervised learning could not help with this problem in general. If the assumptions for semi-supervised learning are at least partially met, there should be a way how to use the unlabeled data to mitigate the negative effect of data drift. Therefore, for follow-up research, we propose to focus on solving this challenge. One approach could be to utilize some kind of similarity measure on the input data and emphasize the unlabeled samples, which are more similar to the labeled set. Then the algorithm could try to gradually move to more different samples. We tried to experiment with this and analyzed distances among samples from the same class in different time periods. We tried several commonly used similarity measures like Euclidean, Manhattan, Cosine, Chebyshev, and Mahalanobis. The Cosine distance seemed to be best to separate the different classes, but the overall results were not conclusive enough to be used straight away. The

data are not simple enough to be separable using just distance measures. We tried this only on the raw input vectors. Hence the next step could be to incorporate the similarity measure in other parts od the machine learning model. For example, to use it on the activations of neurons in hidden layers, where the network learned to extract more complex features. Also, datasets similar to the malware one we used, might be good candidates for online learning, where the model is incrementally improved using new data samples. It could also be used with active learning, which would help to choose the best samples for labeling.

Conclusion

In this thesis, we presented an application of semi-supervised learning of deep neural networks, mainly to malware data. At the beginning, we outlined machine learning and its types. We also described artificial neural networks and how they learn from training data. The next chapter was dedicated to related studies. We outlined a few early semi-supervised algorithms and described four approaches of deep semi-supervised learning in detail. Then we described our implementations of two of the algorithms, Pseudo-labeling and Π-model. In the last chapter, we presented our experiments with these two methods.

We tested them using a simple two-dimensional dataset and visualized the learned decision border. Then we compared them with the fully supervised baseline. We evaluated the classification accuracy on a real-world malware dataset divided to 380 weeks by the time of the first recording of the respective binary file. Despite having been developed for the classification of image data, the results showed that both methods could improve the performance of a neural network on malware data. However, implemented algorithms seem to be beneficial only when the proportion of labeled data is low, ideally around 1 %. We have also found that these semi-supervised methods can increase the accuracy on data newer than the training set, for which drift in structure is likely to occur, but only to a certain extent. Furthermore, we tried to overcome the problem with the data drift and used the newer samples as the unlabeled set for the Π-model. However, its effect was mostly negative. Based on our experiments, the relatively more complex algorithm, Π-model has got slightly better results than Pseudo-labeling in most cases.

Bibliography

1. Nadeem, M.; Marshall, O.; Singh, S.; Fang, X.; Yuan, X. Semi-supervised Deep Neural Network for Network Intrusion Detection. In: *Conference on Cybersecurity Education, Research and Practice*. 2016, pp. 0–11.
2. Narayanan, A.; Soh, C.; Chen, L.; Liu, Y.; Wang, L. Apk2vec: Semi-supervised Multi-view Representation Learning for Profiling Android Applications. In: *IEEE International Conference on Data Mining*. 2018, pp. 357–366.
3. Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.
4. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*. The MIT Press, 2016.
5. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Press, 2009.
6. Chapelle, O.; Schlkopf, B.; Zien, A. *Semi-Supervised Learning*. 1st. The MIT Press, 2010.
7. Anthony, M. *Discrete Mathematics of Neural Networks: Selected Topics*. Society for Industrial and Applied Mathematics, 2001.
8. Grisel, O.; Ollion, C. *Master Datascience Paris Saclay — Deep Learning course slides* [online] [visited on 2019-12-24]. Available from: `https://m2dsupsdlclass.github.io/lectures-labs/`.
9. Ramachandran, P.; Zoph, B.; Le, Q. V. Searching for Activation Functions. *CoRR*. 2017, vol. abs/1710.05941.
10. Simard, P. Y.; Steinkraus, D.; Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings*. 2003, pp. 958–963.

11. Kingma, D. P.; Ba, J. *Adam: A Method for Stochastic Optimization*. 2014.

12. Zhu, X. *Semi-Supervised Learning Literature Survey*. 2005. Technical report. Computer Sciences, University of Wisconsin-Madison.

13. McClosky, D.; Charniak, E.; Johnson, M. Effective Self-Training for Parsing. In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. 2006, pp. 152–159.

14. Blum, A.; Mitchell, T. Combining Labeled and Unlabeled Data with Co-training. In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*. ACM, 1998, pp. 92–100. COLT' 98.

15. Rasmus, A.; Valpola, H.; Honkala, M.; Berglund, M.; Raiko, T. Semi-supervised Learning with Ladder Networks. In: *NIPS*. 2015, pp. 1–9.

16. Laine, S.; Aila, T. Temporal Ensembling for Semi-supervised Learning. In: *ICLR*. 2017, pp. 1–13.

17. Tarvainen, A.; Valpols, H. Mean Teachers Are Better Role Models: Weight-Averaged Consistency Targets Improve Semi-supervised Deep Learning Results. In: *NIPS*. 2017, pp. 1–16.

18. Miyato, T.; Maeda, S.; Koyarna, M.; Ishii, S. Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2019, vol. 41, pp. 1979–1993.

19. Lee, D. Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. In: *WREPL: ICML Workshop Challenges in Representation Learning*. 2013, pp. 1–6.

20. Oliver, A.; Odena, A.; Raffel, C.; Cubuk, E.; Goodfellow, I. Realistic Evaluation of Deep Semi-Supervised Learning Algorithms. In: *NIPS*. 2018, pp. 1–19.

21. Grandvalet, Y.; Bengio, Y. Semi-supervised Learning by Entropy Minimization. In: *Proceedings of the 17th International Conference on Neural Information Processing Systems*. MIT Press, 2004, pp. 529–536. NIPS'04.

22. Goodfellow, I.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. In: *ICLR*. 2015, pp. 1–11.

23. Lunt, T. IDES: An Intelligent System for Detecting Intruders. In: *Symposium on Computer Security, Threat and Countermeasures*. 1990, pp. 30–45.

24. Debar, H.; Becker, M.; Siboni, D. A Neural Network Component for an Intrusion Detection System. In: *IEEE Computer Society Symposium on Research in Security and Privacy*. 1992, pp. 240–250.

25. Ryan, J.; Lin, M.; Miikkulainen, R. Intrusion Detection with Neural Networks. In: *Advances in Neural Information Processing Systems 10*. 1998, pp. 943–949.

26. Cannady, J. Artificial Neural Networks for Misuse Detection. In: *National Information Systems Security Conference*. 1998, pp. 368–381.

27. Bonifacio, J.; Cansian, A.; De Carvalho, A.; Moreira, E. Neural Networks Applied in Intrusion Detection Systems. In: *IEEE International Joint Conference on Neural Networks*. 1998, pp. 205–210.

28. Depren, O.; Topallar, M.; Amarim, E.; Ciliz, M. An Intelligent Intrusion Detection System (IDS) for Anomaly and Misuse Detection in Computer Networks. *Expert Systems with Applications*. 2005, vol. 29, pp. 713–722.

29. Rhodes, B.; Mahaffey, J.; Cannady, J. Multiple Self-Organizing Maps for Intrusion Detection. In: *23rd National Information Systems Security Conference*. 2000, pp. 16–19.

30. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A. A Detailed Analysis of the KDD Cup 99 Data Set. In: *IEEE Symposium on Computational Intelligence for Security and Defense Applications*. 2009, pp. 288–293.

31. Zhang, Z.; Li, J.; Manikopoulos, C.; Jorgenson, J.; Ucles, J. HIDE: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification. In: *IEEE Workshop on Information Assurance and Security*. 2001, pp. 85–90.

32. Mukkamala, S.; Janoski, G.; Sung, A. Intrusion Detection Using Neural Networks and Support Vector Machines. In: *International Joint Conference on Neural Networks*. 2002, pp. 1702–1707.

33. Linda, O.; Vollmer, T.; Manic, M. Neural Network Based Intrusion Detection System for Critical Infrastructures. In: *International Joint Conference on Neural Networks*. 2009, pp. 1827–1834.

34. Torres, P.; Catania, C.; Garcia, S.; Garino, C. An Analysis of Recurrent Neural Networks for Botnet Detection Behavior. In: *ARGENCON: IEEE biennial congress of Argentina*. 2016, pp. 1–6.

35. Abolhasanzadeh, B. Nonlinear Dimensionality Reduction for Intrusion Detection Using Auto-encoder Bottleneck Features. In: *IKT: IEEE 7th Conference on Information and Knowledge Technology*. 2015, pp. 1–5.

36. Gao, N.; Gao, L.; Gao, Q.; Wang, H. An Intrusion Detection Model Based on Deep Belief Networks. In: *IEEE Second International Conference on Advanced Cloud and Big Data*. 2014, pp. 247–252.

37. Kim, J.; Kim, J.; Thu, H. L. T.; Kim, H. Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In: *PlatCon: IEEE International Conference on Platform Technology and Service*. 2016, pp. 1–5.

38. Ma, T.; Wang, F.; Cheng, J.; Yu, Y.; Chen, X. A Hybrid Spectral Clustering and Deep Neural Network Ensemble Algorithm for Intrusion Detection in Sensor Networks. *Sensors*. 2016, vol. 16, pp. article no. 1701.

39. Saxe, J.; Berlin, K. *Expose: A Character-Level Convolutional Neural Network with Embeddings for Detecting Malicious URLs, File Paths and Registry Keys*. 2017. Arxiv preprint arXiv:1702.08568.

40. Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection. *IEEE Access*. 2017, vol. 6, pp. 1792–1806.

41. Wang, W.; Zhu, M.; Zeng, X.; Ye, X.; Sheng, Y. Malware Traffic Classification Using Convolutional Neural Network for Representation Learning. In: *ICOIN: IEEE International Conference on Information Networking*. 2017, pp. 712–717.

42. Soh, C. *Program Analysis and Machine Learning Techniques for Mobile Security*. 2019. PhD thesis. Nanyang Technological University, Singapore.

43. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed Representations of Words and Phrases and Their Compositionality. In: *NIPS*. 2013, pp. 1–9.

44. LeCun, Y.; Cortes, C. MNIST handwritten digit database. 2010. Available also from: `http://yann.lecun.com/exdb/mnist/`.

45. Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; Ng, A. Y. The Street View House Numbers (SVHN) Dataset. 2011.

46. Krizhevsky, A. *Learning multiple layers of features from tiny images*. 2009. Technical report.

47. Friedman, M. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*. 1937, vol. 32, no. 200, pp. 675–701.

48. Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*. 1979, pp. 65–70.

Appendix A

Acronyms

AI	Artificial inteligence
ANN	Artificial neural network
CNN	Convolutional neural network
EMA	Exponential moving average
LDS	Local distributional smoothness
MLP	Multilayer perceptron
MSE	Mean squared error
PCA	Principal component analysis
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
SOM	Self-organizing map

Appendix B

Contents of Enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `semi-supervised` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `DP_Koza_Jan_2020.pdf` ..... the thesis text in PDF format