Neural Network Information Leakage through Hidden Learning

Arthur da Cunha, Emanuele Natale, Laurent Viennot

Neural Network Information Leakage through Hidden Learning

Arthur da Cunha[1] **Emanuele Natale**[1] **Laurent Viennot**[2]

[1]COATI, Inria Sophia Antipolis Méditerranée
[2]Inria, Irif, Université de Paris
{arthur.carvalho-walraven-da-cunha, emanuele.natale, laurent.viennot}@inria.fr

Abstract

We investigate the problem of making a neural network perform some hidden computation whose result can be easily retrieved from the network output.
In particular, we consider the following scenario. A user is provided a neural network for a classification task by a company. We further assume that the company has limited access to the user's computation, and can only observe the output of the network when the user evaluates it. The user's input to the network contains some sensible information.
We provide a simple and efficient training procedure, called Hidden Learning, that produces two networks such that
i) One of the networks solves the original classification task with comparable performance to state of the art solutions of the task;
ii) The other network takes as input the output of the first and solves another classification task that retrieves the sensible information with considerable accuracy.
Our result might expose important issues from an information security point of view, as for the use of artificial neural networks in sensible applications.

1 INTRODUCTION

In this paper, we investigate the possibility for an attacker to build a model that seems legitimate for a given task, but secretly performs another additional task and possibly reveals information it should not. In particular, when using a model from the shelf, is it possible that it computes and outputs more than what it is supposed to do?

Such question naturally emerges with the current surge of *machine learning as a service* scenarios (MLaaS) Tafti et al. [2017], which is already motivating a lot of research regarding the associated privacy and security problems Qayyum et al. [2020]. Among the taxonomy of attacks that have been investigated, particular attention has been devoted to model inversion (MI) attacks Fredrikson et al. [2015], in which an attacker tries to retrieve sensible features about the input data by accessing the model output with or without knowledge of the model itself (white-box vs black-box attacks).

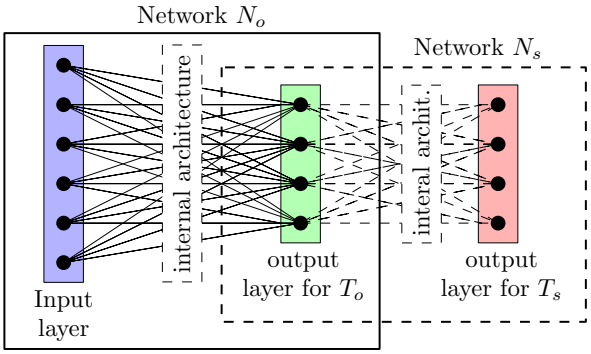

Figure 1: Diagram illustrating the basic components of the Hidden Learning framework. See Section 2 for a description of the components.

Contrarily to model inversion, in this work we consider a setting in which the attacker forges the weights of the model based on the training data, and he is thus in a much powerful position compared to the attacker in MI settings. Although we assume the attacker must use a classical design for the model so that it looks legitimate for the task, he may set the weights in any manner that achieves state of the art accuracy. (We further discuss MI and its relation with the present work in Section 3.)

A natural possibility for hidden learning would be to use steganography, by combining two networks with steganographic techniques; however, it is unclear how such design could prevent the model from looking suspicious.

In this work, we investigate what may be regarded as the most natural strategy to achieve the aforementioned goal of the attacker, by considering a simple scheme that trains a network for two tasks at the same time, namely the *official* task which a user expect it to perform, and a *secret* task which is achieved by feeding the output of the network into a *secret network* (see Fig. 1). We call the latter scheme *hidden learning*, and we formally define it in Section 3.

To provide some intuition for the proposed framework, consider the following toy example. Points on the Euclidean plane are generated according to two standard gaussians centered in $(0,1)$ and $(0,-1)$ respectively, and the official task is to classify points according to the gaussian they come from. The faithful model should use the best separating line $y = 0$. However, if we wish to extract information on the x coordinate of the input point, the line $y = x$ would still achieve substantial accuracy on the official task while revealing some information about the input x coordinate.

As an example where hidden learning could be problematic, consider the following plausible scenario. For better handling the Covid-19 crisis, the government of a country hires a company to develop a smartphone application for estimating how many people are at risk in each region of the country. Each user is asked to enter sensitive health information which is fed into a neural network that outputs a probability that the user can develop severe Covid reaction if infected as well as a probability that the user was already infected. These two probabilities only and the region of the user, are communicated to a server of the company that can then provide statistics to the government. If the application is open source, independent coders can check that the application does indeed behave as expected. However, if hidden learning has been used to set up the weights of the neural network embedded in the application, it could be possible for the company to retrieve additional information such as high risk of cardio-vascular accident and/or whether the person is covered by a given insurance company. Such information could be valuable for some insurance companies that might be tempted to discreetly change their coverage conditions for cardio-vascular risks in certain regions accordingly.

Our main goal is to draw attention on the possibility of an attack on the weights of a model by showing that it can be made effective with a simple approach at very low computational cost.

After formally defining our framework (Section 2) and discussing related works (Section 3), we describe and discuss our experiments on several synthetic tasks defined on the CIFAR-10 and Fashion MNIST datasets (Sections 4 and 5). Finally, we provide our conclusions on the aforementioned results in Section 6.

2 HIDDEN LEARNING FRAMEWORK

In this section we formally describe our *Hidden Learning* framework, whose main components are represented in Figure 1.

We start by providing the key definitions. Let S be a generic set and k_o and k_s be two positive integers. Hidden Learning is performed by considering two classification tasks:

- The *official task* T_o which asks to classify points in S in k_o categories;
- The *secret task* T_s, which asks to classify points in S in k_s categories.

In order to achieve the previous two tasks, the Hidden Learning framework produces two artificial neural networks:

- An *official network* N_o, which assigns to points in S a vector in $[0,1]^{k_o}$, and which can be interpreted as a vector of scores associated to each of the k_o categories of the task T_o.
- A *secret network* N_s, which classify vectors in $[0,1]^{k_o}$ into k_s categories.

Remark. *The only specific constraint in the above framework lies in the co-domain of the official network N_o, namely the space of vectors in $\mathbb{R}^{k_o}$, which are then passed to a softmax function. The latter is a natural choice in many scenarios and is consistent with typical MI attack settings, in which the attacker is assumed to have query access to some model's scores about the possible output categories Shokri et al. [2017].*

The training of the official and decoded networks is performed simultaneously: at each epoch, the updates of the weights of the two networks are computed by backpropagation according to a combination of the loss functions for the respective tasks. As a first simple choice regarding the way the loss functions may be combined, in this work we consider the their sum.

More formally, let $L_o(\hat{y},y)$ and $L_s(\hat{y},y)$ be the loss functions for the official task T_o and the secret task T_s, respectively. The network is then trained by optimizing the combined loss function

$$L_o(\hat{y},y) + L_s(\hat{y},y). \tag{1}$$

More details about how we perform the training in our experiments can be found in Section 4.

3 RELATED WORK

Our work is closely related to the class of privacy attacks to neural network models known as (*white box*) model inversion (MI) attacks Fredrikson et al. [2015]. In the latter setting, given an output $f(x)$ and the model f that produced

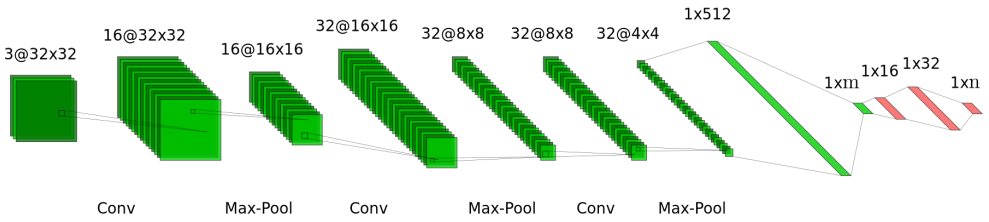

Figure 2: In green, a representation of the LeNet-like architecture for the official network, described in Section 4, applied to a classification into m classes. In red, the architecture the multilayer perceptron used as secret network for a secret task with n classes.

it, an attacker tries to reconstruct the corresponding input x.We emphasize that, in contrast to the MI setting in which the attacker does not intervene in the creation of the model f, our hidden learning framework assumes that the attacker can forge the model f (our N_o) itself in a disguised fashion that allows, by design, to easily invert it (using N_s). Note also that contrarily to many MI settings, the training data is not considered as sensitive here, while the attack concerns input data feeded to the model in production use. We also mention here *black box* MI attacks which, as the name suggest, are a more restrictive kind of MI attacks where the attacker only needs to be able to arbitrarily query the model and observe the corresponding output, without any knowledge about the model internals He et al. [2019]. Contrarily to this setting, we do not assume that the attacker can propose forged inputs and get the corresponding outputs.

Part of our experiments are aimed at verifying the robustness of the secret network with respect to perturbation of the official one. This should be compared to recent work which investigate the sensitivity of the explainability of a model when the latter is perturbed as a consequence of other procedures, such as the disruption of input attribution that arises when standard neural network compression methods are employed, as recently shown in Park et al. [2020].

The present work investigates a simple approach to produce a neural network (the official network N_o) which performs some *hidden computation* that can be exploited by a third party to extract sensible information from a private input. In this respect, it can be contextualized in the general area of Petitcolas et al. [1999]. While the application of artificial neural networks for standard steganographic tasks (*statically* hiding information in a given object), is being actively investigated Yang et al. [2019], Tao et al. [2019], we are not aware of works which, like the present one, investigate how to produce an artificial neural network which tries to hide information in its output, while its calculations are entirely transparent to the party who is making use of it. In particular, its architecture should be legitimate for the official task. A related concept to the latter one are backdoor attacks on deep neural networks, where the goal is to produce a neural network which appears to solve a certain task, but which then behaves quite differently when feeded certain triggering inputs Gu et al. [2019], Nguyen and Tran [2020]. It has also been shown that the latter triggering inputs can be steganographically designed so that they would not be identifiable by direct inspection Li et al. [2020].

4 EXPERIMENTS

In this section we describe the experiments we carried out to validate the Hidden Learning framework described in Section 2.

We perform experiments on the classical CIFAR-10 dataset Krizhevsky [2009] and Fashion MNIST dataset (FMNIST) Xiao et al. [2017]. Both of them consist of small-size images (32×32 and 28×28 pixels, respectively) classified in 10 classes:

- airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck for CIFAR-10,
- T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot for FMNIST.

We wrote our code in the Julia Programming Language[1] Bezanson et al. [2017] using the Flux ML library Innes et al. [2018], Innes [2018]. The source code is available upon request.

The experiments have been performed on the NEF computing infrastructure of Inria Sophia Antipolis Méditerranée[2], which includes a variety of heterogeneous GPU nodes. Given that we focus on the reached accuracy and that our experiments are not aimed at improving any state of the art for the traditional setting we consider, the experiments were scheduled on single GPU nodes assigned by the cluster

[1]Precisely, we used Julia v1.5.2 with Flux v0.11.1, BSON v0.2.6, CUDA v1.3.3, MLDatasets v0.5.3, TensorBoardLogger v0.1.13, ValueHistories v0.5.4 and Zygote v0.5.16.

[2]`https://wiki.inria.fr/ClustersSophia/Clusters_Home`.

job scheduling system, OARSUB system[3], with no specific hardware constraints.

4.1 DESCRIPTION OF EXPERIMENTAL RESULTS

In this section we describe the experiments summarized in Table 1. All values are truncated to the fourth decimal point.

We have adopted the same architecture for all experiments up to the number of neurons in the output layers of the tasks T_o and T_s. For simplicity, we have opted for a simple convolutional architecture for the official network, based on LeNet5[4] Lecun [1998] (see Fig. 2):

- A convolutional layer with 16 kernels 3×3, stride 1×1, padding of one pixel, and ReLu Hahnloser et al. [2000] activation function; the convolution is followed by 2×2 max pooling;
- Two convolutional layers with 32 kernels, and otherwise identical to the previous (including the max pooling);
- A fully connected linear layer.

As for the secret network, we consider a multilayer perceptron with two hidden layer with ReLu activation, the first with 16 and the second with 32 hidden nodes. We remark that the above choices do not leave out any hyperparameter which needs to be set.

We ran two types of experiments.

Hidden learning experiments. These experiments, which are summarized in Table 1, are aimed at showing the accuracy achieved by the official network N_o over several different tasks described below. With the expression T_o-and-T_s we refer to the accuracies achieved in the experiments in which the networks N_o and N_s have been trained according to our hidden learning framework described in Section 2. On the row T_o of column T_o-then-T_s and on the column T_s-only we show, the accuracies achieved in the experiments in which the networks N_o and N_s have been trained by taking into account, respectively, only the loss function for T_o and for T_s (separately). Finally, on the rows T_s of the column T_o-*then*-T_s, we show the accuracies achieved by N_s in the experiments in which, first, the network N_o has been trained by taking into account only the loss function for T_o and, then, N_s has been trained by taking into account only the loss function for T_s while the weights of N_o are not modified. We observe that the latter experiments can be seen as a kind of black-box MI where the attacker has access to the full training dataset with corresponding model outputs.

Robustness experiments. These experiments, which are summarized in Table 2, are aimed at estimating the robustness of the secret network w.r.t. perturbations of the official network; in order to do that, gaussian noise with zero mean and standard deviation σ is added to each weight of the official network; each column shows the accuracies of the official and secret networks, on the train and test sets, for different values of σ, averaged over 10 independent injections of noise.

Record that both CIFAR-10 and FMNIST associate inputs to labels from 10 classes. We have simulated information removal by creating subtasks of classification into[5]

- Two classes ($C2$): one class for the inputs belonging to any of the first 5 original classes, and other for the inputs belonging to any of the last 5. For instance, for CIFAR-10, the first class in this subtask is "*airplane* or *automobile* or *bird* or *cat* or *deer*" while the other is "*dog* or *frog* or *horse* or *ship* or *truck*".
- Five classes ($C5$): we pair original classes to create new ones. Furthermore, we do this while avoiding pairs contained in the classes for the last subtask. This ensures that the solutions to one of those subtasks do not provide any information about the other. Using CIFAR-10 labels as an example, the classes for this subtask are "*airplane* or *dog*", "*automobile* or *frog*", "*bird* or *horse*", "*cat* or *ship*", and "*deer* or *truck*".
- The first n classes (Fn): classification into $n+1$ classes, namely, the first n original classes, and an extra one combining all the other. Exemplifying as before, for $n = 3$ this subtask comprises the classes "*airplane*", "*automobile*", "*bird*", and "*neither an airplane nor an automobile nor a bird*".
- The last n classes (Ln): same as the previous subtask, but for the n last original classes.

We organized the experiments by choosing one of those subtasks as T_o and other as T_s. We also consider some cases where T_s is the original classification into 10 classes.

In the tables, we refer to the original task as C10, to subtasks with 2 and 5 classes as C2 and C5, respectively, to the classification into the first m original classes as Fm, and to the classification into the last n original classes as Ln.

We initialized the weights of all the neural networks using Glorot uniform initialization Glorot and Bengio [2010], the default in Flux framework. and then trained all of them for 40 epochs using the ADAM optimizer Kingma and Ba [2015] with a learning rate of 0.001, which is the default in

[3] `http://oar.imag.fr`.

[4] As a sanity check, we remark that for the classical CIFAR-10 classification task the model achieved test accuracy of ca. 66%, consistently with similar, non-optimized textbook experiments;

[5] The symbols between parenthesis refer to the one used in the experiment tables.

Flux. over 45,000 training entries for CIFAR-10 and 54,000 for FMNIST, organized into batches of size 64. Even though the actual number of training points in those datasets is, respectively, 50,000 and 60,000, we reserved 10% of those to use as validation dataset. That is, we further subdivide the training dataset and use these validation points only for estimating of the generalization capabilities of the model. The resulting model is the set of weights that achieved the highest accuracy on the validation dataset during training. When training N_o and N_s simultaneously, we chose set of weights that maximizes the sum of the accuracies of both networks. The accuracy values discussed in this work refer to the performance of the networks with these sets of weights on the test set. The test dataset consists of 10,000 data points for both CIFAR-10 and FMNIST. Those do not take any part in training.

For the loss function, given an input x and a neural network N_f, let $y \in \mathbb{R}^k$ be the one-hot vector associated with the correct class for x and let $\hat{y} = N_f(x)$, which is also in $\mathbb{R}^k$. As usual for classification tasks, we employed the cross entropy between y and $\text{softmax}(\hat{y})$. To be more precise, let $z = \text{softmax}(\hat{y})$, that is, $z_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{k} e^{\hat{y}_j}}$, for $j = 1, 2, \ldots, k$; the loss function used is

$$L(\hat{y}, y) = -\sum_{i=1}^{k} y_i \cdot \log(z_i).$$

Furthermore, in subtasks of the type Fn and Ln, some of the classes are the same as in the original task, and, therefore, each of them correspond to 10% of the datasets. On the other hand, the extra class merges all the remaining original classes, corresponding to $10 - n$ tenths of the points. We try to compensate for this unbalance by proportionally underweighting the loss for these extra classes. More precisely, when computing the loss for subtasks of type Fn or Ln, we divide the loss by $10 - n$ whenever the input belongs to, respectively, the first n or last n original classes.

The results of our experiments are discussed in Section 5.

5 DISCUSSION

We start by discussing the experiments summarized in Table 1. By comparing the accuracy achieved by the official network in the *Hidden Learning* experiments (T_o-and-T_s) with its accuracy when it is trained for T_o only (provided in the T_o row of the T_o-then-T_s column), we can see that the framework does not sensible decrease accuracy: for CIFAR-10 the two numbers are respectively[6] 68.5 ± 6.4 and 71.2 ± 10.6, while for FMNIST we have 91.5 ± 2.2 and 92.1 ± 2.6.

The corresponding accuracies achieved by the secret network N_s, namely when trained with the framework and when trained after N_o has been trained alone and is not modified, are respectively 58.7 ± 9.5 and 46.5 ± 16.3 on CIFAR-10, and 82.6 ± 10.9 and 65.7 ± 15.1 on FMNIST. Hence, we can see that Hidden Learning drastically improves the accuracy compared to what may be regarded as a black-box MI approach (as mentioned in Section 4).

We can furthermore see that, when the entire architecture is trained by uniquely taking into account the loss function of the secret task T_s, N_s achieves accuracies which are only slightly better than those achieved with the Hidden Learning framework, scoring 59.1 ± 8.3 on CIFAR-10 and 83.4 ± 9.9 on FMNIST. The fact that the framework is able to match the latter results for T_s shows that it is effective in exploiting the whole network despite the interference of the official task.

We observe that the gain in accuracy for T_s is especially significant in the experiments where this task involves fewer classes. This is consistent with the fact that, in such cases, the secret network has fewer neurons as input and, thus, when N_s is trained independently (T_s) it should get access to less information in the first place.

Finally, we remark that, since our tasks consisted of different ways to group and split the original dataset classes in different ones, we also verified that our results are not sensitive to the ordering of the original labels. We now discuss the robustness experiments summarized in Table 2. The goal of these experiments is to provide a first assessment of the sensitivity of the secret network N_s to perturbations of the official network N_o. We remark that our experiments have in no way being optimized to improve network robustness to weight noise, e.g. by some regularization approach Zheng et al. [2016].

The table displays the corresponding accuracies obtained for the smallest values we considered for the standard deviation of the gaussian noise applied (σ for short) to the weights of N_o, namely from 0 to 0.1 with a step of 0.025.

When noise is very low ($\sigma = 0.025$), the average test accuracy for N_o drops by 4.7% for CIFAR-10 while we see a 1.1% average decrease for FMNIST. The corresponding percentages for N_s are 5.8% (CIFAR-10) and 1.7% (FMNIST). In comparison, when $\sigma = 0.5$, N_o achieves average accuracy $31.5\% \pm 13.3$ on FMNIST and $26.0\% \pm 13.6$ on CIFAR10. For N_s those values are $28.4\% \pm 13.2$ and $25.4\% \pm 10.9$. This indicates that the perturbation in the official output tends not to disturb the computation of the secret network unless it is strong enough to change the official answer.

We can appreciate from the table that a noise level of 0.1 already deteriorates the accuracy of the official network by 29.5% and 22.3% on average for CIFAR-10 and FMNIST respectively. In particular, the fact that N_o achieves, on across different experiments, higher accuracies (22.9% difference) on FMNIST (average 91.4 ± 2.2) than on CIFAR10

[6] The value reported after the average is the sample stdev. All reported statistical values are rounded to the first decimal place.

Exp.	Task	CIFAR-10			FMNIST		
		T_o and T_s	T_o then T_s	T_s only	T_o and T_s	T_o then T_s	T_s only
C2-C10	T_o	73.5%	74.9%		94.2%	94.3%	
	T_s	43.5%	21.3%	47.5%	85.3%	50.2%	86.5%
C5-C10	T_o	65.3%	64.8%		90.3%	89.5%	
	T_s	61.4%	49.6%	61.3%	89.8%	87.2%	89.5%
C5-C2	T_o	64.3%	65.0%		89.9%	89.9%	
	T_s	75.7%	66.3%	74.3%	94.3%	90.0%	94.5%
C2-C5	T_o	66.3%	74.7%		94.2%	94.4%	
	T_s	53.0%	27.6%	58.4%	85.6%	51.1%	88.0%
F2-L8	T_o	78.2%	89.8%		94.0%	96.3%	
	T_s	51.7%	22.2%	50.3%	87.0%	38.2%	86.9%
F3-L7	T_o	71.0%	81.4%		90.9%	93.3%	
	T_s	56.3%	41.9%	58.3%	88.4%	68.4%	88.9%
F4-L6	T_o	63.5%	67.0%		90.1%	92.1%	
	T_s	58.0%	45.6%	62.2%	89.9%	74.8%	90.1%
F5-L5	T_o	61.4%	60.4%		90.4%	90.2%	
	T_s	64.4%	49.1%	66.1%	92.2%	78.6%	92.9%
F6-L4	T_o	63.4%	60.4%		89.9%	89.6%	
	T_s	60.6%	51.4%	64.2%	78.9%	74.9%	78.7%
F7-L3	T_o	64.2%	64.0%		90.5%	90.3%	
	T_s	67.1%	63.1%	64.7%	66.1%	65.3%	65.9%
F8-L2	T_o	65.8%	65.8%		87.7%	89.3%	
	T_s	41.3%	45.3%	49.5%	62.0%	64.1%	71.8%
F2-L5	T_o	79.4%	89.3%		95.3%	96.3%	
	T_s	57.3%	33.2%	59.5%	92.3%	48.2%	92.4%
F5-L2	T_o	64.1%	58.9%		90.6%	90.2%	
	T_s	60.8%	73.9%	44.9%	80.3%	69.3%	75.0%
F3-L3	T_o	78.6%	80.8%		92.3%	93.6%	
	T_s	70.9%	60.9%	65.9%	64.7%	59.0%	66.0%

Table 1: Summary table of experimental results described in Section 4.1.

(average 68.5 ± 6.4) in the absence of noise corresponds to lower deterioration when $\sigma = 0.1$, namely 69.2 ± 12.3 versus 39.0 ± 14.0.

We remark that the average standard deviation of test accuracies for N_o appears quite low on both datasets despite the heterogeneity of the experiments (especially the number of output classes). The trend is consistent for N_s, where the noiseless averages are 82.6 ± 10.9 for FMNIST and 58.7 ± 9.5 for CIFAR-10, while the corresponding numbers when $\sigma = 0.1$ are respectively 57.2 ± 12.5 and 28.5 ± 9.9.

6 CONCLUSIONS

In this work we have introduced a simple and efficient training procedure, called **Hidden Learning**, that produces two networks, an official and a secret one, such that

- the official network solves an official task with comparable performance to state of the art solutions of the task;
- the secret network takes as input the output of official one and solves a secret task with considerable accuracy.

After contextualizing the above framework with respect to current research on Model Inversion and related attacks on neural networks, we have tested it on several synthetic tasks on the CIFAR-10 and Fashion MNIST dataset. In

Exp.	Task	CIFAR-10					FMNIST				
		$\sigma = 0$	0.025	0.05	0.075	0.1	0	0.025	0.05	0.075	0.1
C2-C10	T_o	73.5%	71.0%	65.5%	60.6%	54.9%	94.2%	93.4%	91.6%	83.8%	74.0%
	T_s	43.5%	38.5%	30.6%	21.5%	14.0%	85.3%	80.3%	67.2%	48.8%	37.2%
C5-C10	T_o	65.3%	56.3%	43.5%	33.5%	28.1%	90.3%	88.3%	80.9%	73.1%	50.3%
	T_s	61.4%	51.0%	34.8%	23.4%	17.1%	89.8%	87.7%	77.7%	69.4%	39.9%
C5-C2	T_o	64.3%	56.9%	42.8%	32.5%	29.0%	89.9%	88.4%	84.3%	75.7%	54.4%
	T_s	75.7%	71.0%	61.1%	55.0%	53.9%	94.3%	93.2%	90.6%	83.3%	69.1%
C2-C5	T_o	66.3%	64.5%	59.1%	54.1%	52.4%	94.2%	93.2%	90.9%	86.1%	79.2%
	T_s	53.0%	45.7%	32.0%	25.1%	24.2%	85.6%	82.8%	73.3%	53.6%	48.8%
F2-L8	T_o	78.2%	78.0%	73.9%	64.0%	66.3%	94.0%	93.6%	92.9%	90.1%	86.6%
	T_s	51.7%	44.6%	32.9%	23.7%	18.6%	87.0%	84.4%	72.5%	58.8%	51.4%
F3-L7	T_o	71.0%	70.0%	64.2%	43.5%	41.8%	90.9%	91.1%	86.1%	80.4%	74.9%
	T_s	56.3%	45.4%	35.2%	27.6%	22.4%	88.4%	85.0%	74.8%	64.2%	46.5%
F4-L6	T_o	63.5%	60.0%	53.1%	47.8%	35.0%	90.1%	89.8%	86.4%	79.6%	70.0%
	T_s	58.0%	50.1%	38.6%	27.8%	28.5%	89.9%	88.2%	81.3%	74.5%	56.1%
F5-L5	T_o	61.4%	55.2%	46.9%	39.1%	34.7%	90.4%	89.4%	85.8%	78.6%	74.7%
	T_s	64.4%	60.6%	51.2%	37.0%	30.9%	92.2%	91.3%	86.0%	76.6%	70.5%
F6-L4	T_o	63.4%	56.9%	44.4%	31.8%	28.8%	89.9%	88.8%	85.0%	77.4%	67.2%
	T_s	60.6%	55.4%	43.8%	38.9%	32.3%	78.9%	78.4%	74.4%	70.2%	63.0%
F7-L3	T_o	64.2%	56.1%	42.6%	26.2%	25.9%	90.5%	88.7%	84.6%	73.5%	53.8%
	T_s	67.1%	62.5%	44.9%	40.2%	31.8%	66.1%	65.9%	65.1%	62.3%	52.9%
F8-L2	T_o	65.8%	55.9%	40.4%	29.0%	17.6%	87.7%	84.7%	75.5%	62.0%	48.9%
	T_s	41.3%	39.5%	32.4%	31.2%	27.4%	62.0%	60.1%	67.7%	65.4%	66.6%
F2-L5	T_o	79.4%	78.7%	74.7%	63.1%	58.2%	95.3%	94.9%	93.4%	90.0%	82.8%
	T_s	57.3%	53.4%	43.1%	30.9%	26.8%	92.3%	91.3%	88.4%	83.6%	74.0%
F5-L2	T_o	64.1%	59.2%	47.8%	37.3%	35.1%	90.6%	89.2%	85.9%	79.9%	75.8%
	T_s	60.8%	57.6%	53.9%	39.3%	37.1%	80.3%	79.4%	79.8%	74.6%	74.6%
F3-L3	T_o	78.6%	74.3%	68.0%	62.3%	38.7%	92.3%	91.9%	88.4%	84.6%	75.3%
	T_s	70.9%	64.9%	53.9%	45.9%	33.7%	64.7%	64.6%	63.2%	54.5%	49.9%

Table 2: Summary table displaying the obtained accuracies in the robustness experiments described in Section 4.1.

our experiments, Hidden Learning shows to be effective in tuning the official network for better enabling the attacker to recover information, by the means of a secret network which is computationally very light. The possibility for such kind of attacks should be then taken into account when using a model provided by a third party.

Our preliminary investigation demands more sophisticated ones, in particular regarding possible defense mechanism against the Hidden Learning framework. Even if the official network is suspected to be produced using such a framework, naive strategies to use the network while preventing information leakage such as perturbing the network weights, appear ineffective in our robustness experiments[7]. More generally, the fact that the official network is, by design, produced to assist the secret network in extracting information, might allow the framework to find ways around defense mechanisms that have been proven successful against similar attacks, such as model inversion ones. On the other hand, differential privacy Wang et al. [2015], together with other strategies to decouple data from model training Ryffel et al. [2018], should prove successful in protecting against it.

Acknowledgements

The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support.

[7] Similarly, we expect the framework to be robust to output truncation.

References

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, January 2017. ISSN 0036-1445. doi: 10.1137/141000671. URL https://epubs.siam.org/doi/10.1137/141000671. Publisher: Society for Industrial and Applied Mathematics.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, Denver Colorado USA, October 2015. ACM. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813677. URL https://dl.acm.org/doi/10.1145/2810103.2813677.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010. URL http://proceedings.mlr.press/v9/glorot10a.html. ISSN: 1938-7228.

T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, 7:47230–47244, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2909068. Conference Name: IEEE Access.

Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405 (6789):947–951, June 2000. ISSN 1476-4687. doi: 10.1038/35016072. URL https://www.nature.com/articles/35016072. Number: 6789 Publisher: Nature Publishing Group.

Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, San Juan Puerto Rico USA, December 2019. ACM. ISBN 978-1-4503-7628-0. doi: 10.1145/3359789.3359824. URL https://dl.acm.org/doi/10.1145/3359789.3359824.

Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018. URL https://arxiv.org/abs/1811.01457.

Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018. doi: 10.21105/joss.00602.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *Master's thesis, Department of Computer Science, University of Toronto*, page 60, 2009.

Yann Lecun. Gradient-Based Learning Applied to Document Recognition. *PROCEEDINGS OF THE IEEE*, 86 (11):47, 1998.

S. Li, M. Xue, B. Zhao, H. Zhu, and X. Zhang. Invisible Backdoor Attacks on Deep Neural Networks via Steganography and Regularization. *IEEE Transactions on Dependable and Secure Computing*, 2020. ISSN 1941-0018. doi: 10.1109/TDSC.2020.3021407.

Tuan Anh Nguyen and Anh Tran. Input-Aware Dynamic Backdoor Attack. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/234e691320c0ad5b45ee3c96d0d7b8f8-Abstract.html.

Geondo Park, June Yong Yang, Sung Ju Hwang, and Eunho Yang. Attribution Preservation in Network Compression for Reliable Network Interpretation. *arXiv:2010.15054 [cs]*, October 2020. URL http://arxiv.org/abs/2010.15054. arXiv: 2010.15054.

F.A.P. Petitcolas, R.J. Anderson, and M.G. Kuhn. Information hiding-a survey. *Proceedings of the IEEE*, 87(7): 1062–1078, July 1999. ISSN 00189219. doi: 10.1109/5.771065. URL http://ieeexplore.ieee.org/document/771065/.

Adnan Qayyum, Aneeqa Ijaz, Muhammad Usama, Waleed Iqbal, Junaid Qadir, Yehia Elkhatib, and Ala Al-Fuqaha. Securing Machine Learning in the Cloud: A Systematic Review of Cloud Machine Learning Security. *Frontiers in Big Data*, 3, 2020. ISSN 2624-909X. doi: 10.3389/fdata.2020.587139. URL https://www.frontiersin.org/articles/10.3389/fdata.2020.587139/full. Publisher: Frontiers.

Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv:1811.04017 [cs, stat]*, November 2018. URL http://arxiv.org/abs/1811.04017. arXiv: 1811.04017.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, May 2017. doi: 10.1109/SP.2017.41. ISSN: 2375-1207.

Ahmad P. Tafti, Eric LaRose, Jonathan C. Badger, Ross Kleiman, and Peggy Peissig. Machine Learning-as-a-Service and Its Application to Medical Informatics. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, Lecture Notes in Computer Science, pages 206–219, Cham, 2017. Springer International Publishing. ISBN 978-3-319-62416-7. doi: 10.1007/978-3-319-62416-7_15.

J. Tao, S. Li, X. Zhang, and Z. Wang. Towards Robust Image Steganography. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(2):594–600, February 2019. ISSN 1558-2205. doi: 10.1109/TCSVT.2018.2881118.

Y. Wang, C. Si, and Xintao Wu. Regression Model Fitting under Differential Privacy and Model Inversion Attack. In *IJCAI*, 2015.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]*, September 2017. URL http://arxiv.org/abs/1708.07747. arXiv: 1708.07747.

Z. Yang, X. Guo, Z. Chen, Y. Huang, and Y. Zhang. RNN-Stega: Linguistic Steganography Based on Recurrent Neural Networks. *IEEE Transactions on Information Forensics and Security*, 14(5):1280–1295, May 2019. ISSN 1556-6021. doi: 10.1109/TIFS.2018.2871746.

S. Zheng, Y. Song, T. Leung, and I. Goodfellow. Improving the Robustness of Deep Neural Networks via Stability Training. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4480–4488, June 2016. doi: 10.1109/CVPR.2016.485.