Inversion Attack on Machine Learning Models: Stealing User Input Data

Norman Becker
nobe00001@stud.uni-saarland.de

Philipp Baus
s8phbaus@stud.uni-saarland.de

Mejbah Uddin Shameem
s8mesham@stud.uni-saarland.de

***Abstract*—Using inversion attacks on machine learning models, it is possible to steal training and input data. This can lead to major privacy issues, especially when models use sensible data like medical information or a person's image. In this paper, we precisely explain how a specific inversion attack can steal data that is put into a ”split” neural network by a user. Furthermore, we will evaluate the impact of such an attack regarding privacy and provide exemplary ways to defend against an inversion attack.**

I. Introduction

Nowadays, the amount of both use cases and application purposes of machine learning are soaring. Capabilities of contemporary ML models vastly exceed classical digit recognition. Many of them are used in our daily life for example in medical research [1] or for face recognition. As the field of machine learning is growing it is necessary to also think about security aspects of machine learning models. Without proper protection an attacker might be able to steal personal data, posing the threat of large-scale privacy breaches.

One type of attack that is able to steal data from a model is called ”inversion attack”. Attacks of this kind try to steal data input or training data from a model by inverting it. There are many possible ways to achieve this goal, depending on the architecture of, and access to, the model. In previous works, researchers were able to reconstruct a person's image given only the person's name and access to a facial recognition system trained on person's image [2].They also guided an experiment in which a decision tree model, used to predict a person's willingness to take risks, like cheating on their partner, based on that person's steak preparation preference, was attacked. Then they used a model inversion attack to find out whether a person was cheating or not [2]. Data like a person's image, or whether a person is cheating or not, should be kept private under all circumstances, but model inversion attacks are able to leak them, so protecting machine learning models from inversion attacks is quite important.

II. Attack description

In this paper, we will focus on a special inversion Attack aiming to steal data that is put into a split neural network by a user. ”Split”, in this context, means that one part of the model is calculated on a server, and the other part on a user's device. The attack works similar to the one used in [5].

In order to enable calculation of the model's client side, the sever must send the output of its part of the model to the user. We assume that an attacker is able to intercept this output. The attack then tries to reconstruct the input of the target network by using the intercepted data with a previously trained attacking neural network.

The attacking network is trained as follows: The target network is queried on a dataset chosen by the attacker. The intercepted output of the model is then used as input for the attacking network. The loss between the original input from before mentioned dataset and output of the attacking network is calculated. Then we update the attacking network's weights accordingly. At the end we should have a classifier that can reconstruct the target model's input data from the intercepted output. This whole process is illustrated in figure 1. Black box access is enough to execute the attack. This is because we don't have to know anything about the target network's structure or its weights to perform the attack. We only need to be able to intercept its output. As the training depends on the intercepted output of the target network, the attacking network is bound to the target classifier. Consequently, the attacking network must be retrained if the target classifier changes.

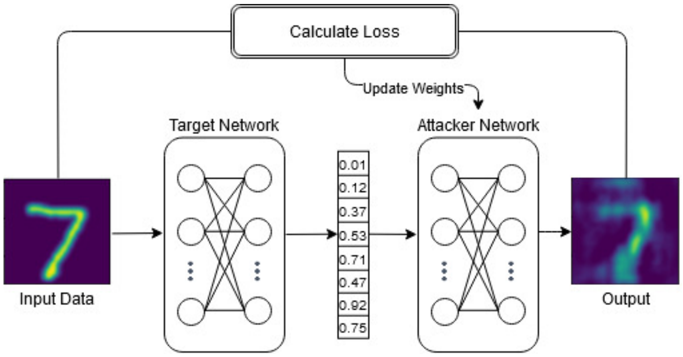

Fig. 1. Architecture of the training process

III. Evaluation of Inversion attack

A. Procedure

We ran the attack on three neural network classifiers, that all differed from each other by their complexity. These three classifiers are:

1. SimpleNet:

Layer 1:
1. Conv2d(1, 6)
2. Conv2d(6, 16)

Layer 2:
3. Linear(256, 128)
4. Linear(128, 10)

2. AlexNet:

Layer 1:
1. Conv2d(in=1, out=32)
2. MaxPool2d(kernelsize=2)
3. ReLU()

Layer 2:
1. Conv2d(in=32, out=64)
2. MaxPool2d(kernelsize=2)
3. ReLU()

Layer 3:
1. Conv2d(in=64, out=128)

Layer 4:
1. Conv2d(in=128, out=256)

Layer 5:
1. Conv2d(in=256, out=256)
2. MaxPool2d(kernelsize=3)
3. ReLU()

Layer 6:
1. Linear(in=2304, out=1024)
2. Linear(in=1024, out=512)
3. Linear(in=512, out=10)

3. ResNet:

Layer 1:
1. Conv2d(in=1, out=16)
2. BatchNorm2d(in=16, eps=0.0001, momentum=0.1)
3. ReLU()

Layer 2:
1. ResidualBlock(in=16, out=16)
2. ResidualBlock(in=16, out=16)

Layer 3:
1. ResidualBlock(in=16, out=32)
2. ResidualBlock(in=32, out=32)

Layer 4:
1. ResidualBlock(in=32, out=64)
2. ResidualBlock(in=64, out=64)

Layer 5:
1. AvgPool2d()
2. Linear(in=64, out=10)

ResidualBlock(in, out):
1. Conv2d(in=in, out=out)
2. BatchNorm2d(in=in, eps=0.0001, momentum=0.1)
3. ReLU()
4. Conv2d(in=in, out=out)
5. BatchNorm2d(in=in, eps=0.0001, momentum=0.1)

We trained each of these three classifiers on the MNIST dataset and then tried to attack them. Furthermore, we attacked every classifier three different times as we trained the attacking network with three different datasets:

1) EMNIST
2) FashionMNIST
3) Uniformly random dataset

By doing this we are able to compare whether the choice of the dataset used to train the attacking network makes any difference regarding the success of the attack. We explicitly chose these three datasets as they vary in their differences to MNIST. EMNIST is pretty similar to MNIST, while the uniformly random dataset doesn't have anything in common with MNIST. FashionMNIST is in the middle between these two, as it still has some commonalities to MNIST (10 classes, greyscale pictures, 28x28 pixels), but also some differences as its pictures show pieces of clothing.

B. Results

1) Training with EMNIST: First we started to train the attacking network with EMNIST and attacked every classifier with it. As we already said, EMNIST is pretty close to MNIST. When looking at the results shown in Figure 2, we can see that the attack on the simplest network (SimpleNet) worked pretty well, but became less successful with increasing complexity of the classifier. AlexNet and ResNet inherently slightly differ from the input image. Despite the performance of the attack decreasing when using it on a more complex classifier, one is still able to recognize the digits when the most complex classifier (ResNet) is attacked. To conclude, if we have a dataset close to the actual dataset used for training, the inversion attack works pretty well and one is able to reconstruct the images with high accuracy, even when attacking complex neural networks.

2) Training with FashionMNIST: We now trained the attacking network with the FashionMNIST dataset. As it contains no digits, but clothing pieces, we expected the results to be much worse than in our first attack with EMNIST. However, when we ran the attack on every classifier, the results showed that the attack still worked really well for SimpleNet. For AlexNet and ResNet the attack did not perform really well anymore, although with a little bit of imagination you can still recognize the digits in the AlexNet attack. Overall, it is again apparent that the performance of the attack decreases with the increasing of the complexity of the classifier.

3) Training with uniformly random dataset: At last we trained the attacking network with a uniformly random dataset, which doesn't have anything in common with the dataset the target classifier was trained on. We created 1000 uniformly random 28x28 images, which we used to train the attacking network. We could again see the inverse relationship between the complexity of the classifier and the performance of the inversion attack. Although we used the random dataset, the attack on the simplest network still worked and one can easily recognize the digits contained in the picture. The attacks on the two more complex networks didn't work anymore using the random dataset, as is visible in the resulting images, which all look pretty much the same and don't contain any readable digit.

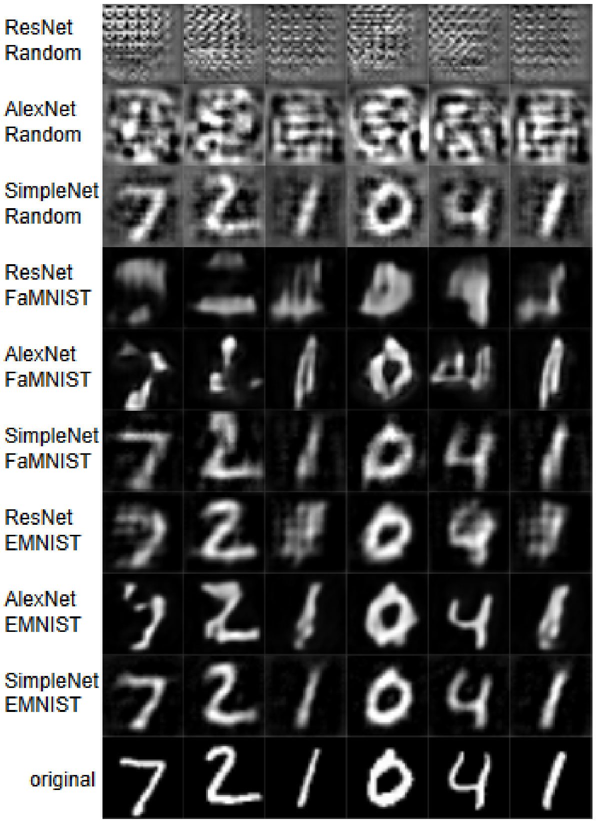

Fig. 2. Example Output from attacking different Models (first) with an attacking network trained on different datasets (second)

All in all, our inversion attack has yielded quite good results, which surprised us.

IV. Defenses against Inversion attacks

Though more focus has been put on defending membership inference attack in privacy and security community, there are still some researches for defending model inversion attack as well. Purifying the prediction score [3] can be as good as to defend both model inversion and membership inference attack. The authors propose a unified purification framework to defend data inference attacks by "purifying" the prediction scores. The framework takes the prediction scores of a trained target model as input and produces a purified version to satisfy one or both of these defense goals: (I) preventing model inversion attack and (II) preventing membership inference attack. Here we only focus on the defense goal (I). They achieve the purification framework by training a purifier, which takes the confidence scores of the target model as input and "concentrate" them on clear patterns. They train an additional adversarial model(H) for defending model inversion. Figure. 3 demonstrates the complete purification framework. The purifier takes target's output prediction as input and then jointly trains the purifier G and the adversarial model H. The purifier keeps updating the prediction scores to minimize the inversion accuracy whereas, the adversarial model which adaptively performs model inversion attack against the purifier formulated as min-max game between the purifier and the adversarial. The purifier reduces dispersion between input vectors(confidence scores), as well as preserves the utility of the classifier. The reduction of the dispersion can make the confidence score vectors less sensitive to the change of the input data, which decreases their correlation and results in the mitigation of model inversion attack. After the training adversarial model H can be discarded.

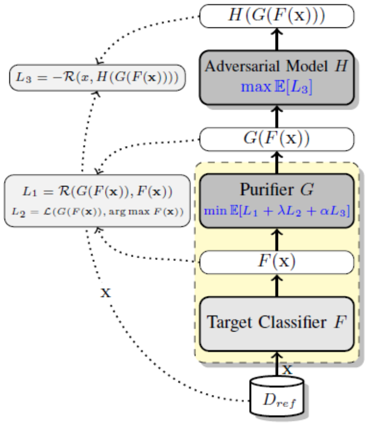
$H(G(F(\mathrm{x})))$
Adversarial Model H
$\max \mathbb{E}[L_3]$
$L_3 = -\mathcal{R}(x, H(G(F(\mathbf{x}))))$
$G(F(\mathrm{x}))$
$L_1 = \mathcal{R}(G(F(\mathbf{x})), F(\mathbf{x}))$
$L_2 = \mathcal{L}(G(F(\mathbf{x})), \arg\max F(\mathbf{x}))$
Purifier G
$\min \mathbb{E}[L_1 + \lambda L_2 + \alpha L_3]$
$F(\mathrm{x})$
x
Target Classifier F
x
D_{ref}

Fig. 3. Purification method for defending model inversion

Fredrikson et al. [3] proposed to degrade the quality or precision of the gradient information retrievable from the model as their attack on facial recognition models are all based on gradient descent. There is no obvious way to achieve this in the white-box setting while preserving model utility, but in the black-box setting this might be achieved by reducing the precision at which confidence scores are reported.

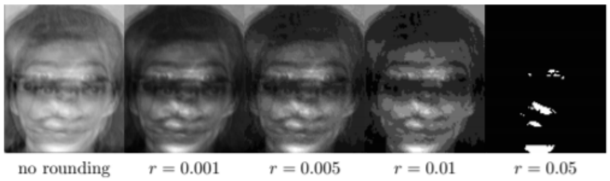

Fig. 4. Face reconstruction attack with rounding level r. The attack fails to produce a non-empy image at r = 0.1, and at r = 0.05 the image is nowhere near identifiable

They tested this approach by rounding the score produced by the softmax model, and running the black-box reconstruction attack. The results are presented in Figure. 4 for rounding levels r = 0.001, 0.005, 0.01, 0.05; the attack failed to produce an image for r = 0.1. "No rounding" corresponds to using raw 64-bit floating-point scores to compute numeric gradients.

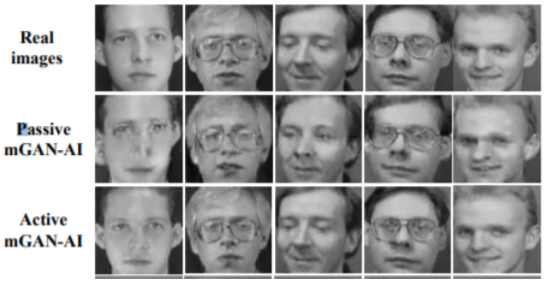

Fig. 5. Reconstructed samples by mGAN-AI with DP

Notice that even at r = 0.05, the attack fails to produce a recognizable image. In federated learning setting, mGAN-AI [4] does not violate the differential privacy (DP), which aims to prevent the recovery of specific samples used during the training. In other words, DP tries to make the adversary fail to tell whether a given sample belongs to the training set. However, without inferring the membership of a given sample, it can still generate samples close to the real samples as demonstrated in Figure. 5, which obviously leads severe privacy violation.

Our type of model inversion attack closely matches with the federated learning approach as the training is split up in multiple devices. Wang et al. [4] reconstructed good quality of recognizable images despite of having differential privacy. In that sense, we verified that as in our setting we do not rely on the confidence scores for the attack rather intercept the output of the model after the first stage of training it is impossible to defend using the three methods of defending model inversion attack described above. We would say it remains as an open research direction to defend explicitly this type of model inversion attack, while securing the connection when exchanging data between multiple devices could be a workaround.

V. Conclusion

In conclusion, our attack is not a typical inversion attack, which tries to steal training data, as it is only works when a split model is used, with the attacker being able to intercept the data sent between the parties. When this is the scenario, our attack works pretty well and could be easily able to steal user data. Against simple networks, with a small number of layers, our attack works almost perfectly. With an increasing number of layers, as well as increasing layer complexity, the attack performance decreases. Another factor that impacts the results of our attack is the choice of training dataset. When a dataset with high similarity to the target's training dataset is chosen,the attack will also work for complex models. Contrarily, if the chosen dataset largely differs from the target's training dataset, the attack will produce worse results. In general, model complexity and choice of the dataset have a huge impact on the success of the attack. Defending the model is pretty hard as no defenses used against normal inversion attacks are working against this kind of attack. The only thing that could be done to prevent an attacker from stealing data might be to secure the connection between the user and the server.

References

[1] C.-L. Chi, W. Nick Street, J. G. Robinson, and M. A.Crawford. Individualized patient-centered lifestyle recommendations: An expert system for communicating patient specific cardiovascular risk information and prioritizing lifestyle options. J. of Biomedical Informatics, 45(6):1164–1174, Dec. 2012.

[2] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. CCS '15: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Pages 1322–1333, Oct. 2015.

[3] Ziqi Yang, Bin Shao, Xuan Bohan, Ee-Chien Chang, and Fan Zhang. Defending Model Inversion and Membership Inference Attacks via Prediction Purification. arXiv:2005.03915 [cs.CR], May. 2020.

[4] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wangy, and Hairong Qi. Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning. arXiv:1812.00535 [cs.LG], Dec. 2018.

[5] Ziqi Yang, Ee-Chien Chang, Zhenkai Liang, Adversarial Neural Network Inversion via Auxiliary Knowledge Alignment, arXiv:1902.08552v1 [cs.CR] 22 Feb 2019