Practical Attacks on Machine Learning Systems

Chris Anley, Chief Scientist, NCC Group plc

Abstract

This paper collects a set of notes and research projects conducted by NCC Group on the topic of the security of Machine Learning (ML) systems. The objective is to provide some industry perspective to the academic community, while collating helpful references for security practitioners, to enable more effective security auditing and security-focused code review of ML systems. Details of specific practical attacks and common security problems are described. Some general background information on the broader subject of ML is also included, mostly for context, to ensure that explanations of attack scenarios are clear, and some notes on frameworks and development processes are provided.

Contents

1 Introduction

NCC Group is a publicly held multinational information security company, listed on the London Stock Exchange. In the course of our consulting work for customers, we perform a wide variety of security assessment tasks, including penetration testing, security code review and attack simulations, as well as longer-term partnerships, managed detection and response, escrow and Secure Software Development Lifecycle projects. These projects give us broad and deep insights across a wide variety of different business areas and organisations in both the public and private sectors.

Since approximately 2015 we have seen a sharp upturn in the number of organisations creating and deploying Machine Learning (ML) systems. This has coincided with an equally sharp increase in the number of organisations moving critical systems into cloud platforms, the rise of mobile applications and containerized architectures, all of which significantly change the security properties of the applications we trust every day.

In this paper, we will attempt to distil some of the most common security issues that we encounter in ML systems from a traditional, and a more contemporary cloud, container and app perspective. We will also (mostly for the benefit of security practitioners) explain several categories of attack that are specific to ML systems, from the black-box perspective typical of security assessments.

Our contributions are the following:

- We present a taxonomy and descriptions of various forms of attack on ML systems
- We provide a sizeable, conveniently categorised collection of references relating to different forms of attack on ML systems
- We discuss several examples of libraries whose model file formats can trigger malicious code execution when a malicious model is loaded. Although the various mechanisms that allow this are to some extent documented, we contend that the security implications of this behaviour are not well-understood in the broader ML community. We thoroughly explore those security implications and discuss detailed exploit techniques for the SciKit-Learn, Keras, PyTorch and TensorFlow platforms.
- We have independently replicated results from several ML security papers, notably Goodfellow (Adversarial Perturbation, [Goodfellow2014]), Shokri (the "Shadow Training" technique for membership inference, [Shokri2016]), Fredrickson (Model Inversion, [Fredrikson2015]) and Chen (Backdoor Poisoning Attacks, [Chen2017]).

1.1 Conclusions - Is this worth worrying about?

A quick glance at the references section of this document should demonstrate that there is a sizeable - and rapidly growing - body of evidence suggesting several uncomfortable truths:

- Training an ML system using sensitive data appears to be fundamentally unsafe, in the sense that the data used to train a system can often be recovered by attackers in a variety of ways (Membership Inference, Model Inversion, Model Theft, Training Data Extraction)
- Neural Network classifiers appear to be "brittle", in the sense that they can be easily forced into misclassifications by adversarial perturbation attacks. While countermeasures exist (e.g. [Madry2017]) these countermeasures are reported to reduce accuracy and may render the system more vulnerable to other forms of attack ([Song2019b])
- High-fidelity copies of trained models can be extracted by remote attackers by exercising the model and observing the results

While exploiting these issues is not always possible due to various mitigations that may be in place, these new forms of attack have been demonstrated and are certainly viable in practical scenarios.

These alarming new forms of attack supplement the "traditional" forms of attack that still apply to all ML systems; they are still subject to infrastructure issues, web application bugs and the wide variety of other problems, both old and new, that modern cloud-based systems often suffer.

1.2 The Language of ML Security

Research into the security of ML systems is currently in a state of flux; new attacks and defences are being discovered and documented at a rapid rate, and the terminology surrounding this activity has yet to be settled; there is, in short, no common language to describe these ideas.

Although some shared language is emerging, there is a need for some kind of standard, common terminology. Researchers at Harvard University's Berkman Klein Center and Microsoft have collaborated on a paper titled "Failure Modes in Machine Learning" ([Kumar2019]), which collects and provides definitions for a variety of intentional and unintentional failure modes in ML systems. The U.S. National Institute of Standards and Technology (NIST) has released a draft paper ("A Taxonomy and Terminology of Adversarial Machine Learning" [Tabassi2019]) which also collects a variety of terms of art in this area, and the European Telecommunications Standards Institute (ETSI) has formed a group creating technical specifications in this area, including an AI Threat Ontology.

These documents are generally helpful in that they can assist in threat modelling ML systems; they help us to understand what threats exist and what attacks are possible. We have tried to use these standard terms in this document, where they exist and are applicable.

2 Models are Code

ML models can often contain code. In a system that allows the user to choose which model to use, there is a danger that an attacker may create a model containing malicious code, and then cause the target system to load that model, executing the malicious payload. This might happen "by design" where the system allows inference by a variety of different trained models; perhaps different versions of the same model, or in an infrastructure supporting a variety of different model types.

It may also happen accidentally, for example, when the application incorporates a user-specified parameter into a file path; the application may compose a path including the user-specified portion:

```
model_file = BASE_PATH + "/models/" + model_param + ".h5"
```

In this case, model_param is expected to be, say, "classifier_v2" or similar, however if an attacker specifies a filename including parent-path sequences:

```
../../../../../tmp/uploaded
```

. . . they can reference other files; for example, a file they have previously uploaded in a separate part of the application, or a file held in some other accessible filesystem. In the case of models that reside in cloud file storage systems such as S3, or which are referenced by URL, the attacker can potentially specify the location of the file of their choice.

Helper functions to load models from URLs exist in a number of libraries, for example the popular pytorch library offers the following:

```
m = torch.utils.model_zoo.load_url('http://dangerous.example.com:8088/model.pth')
```

The inserted malicious code might be in the form of script, combinations of operations resulting in arbitrary code execution, or even inherent features of the model itself, such as layers of a network composed of custom functions ("Lambda Layers").

The code can form a "backdoor", triggering only on certain stimuli, such as in the presence of specific output classes in a classifier, or on input containing some specific feature, such as the Queen of Diamonds (as in the film, "The Manchurian Candidate"), or a specific phrase.

Several examples, using various techniques, are described below. NCC have successfully exploited each of these examples, however exploit code is not provided here.

2.1 Python Pickle Files

In many Python ML libraries, the underlying library that is used to save the model to disk is called "pickle". The pickle library allows arbitrary Python objects - including code - to be serialised, and allows for methods in the serialised code to be executed on "load" (via the `__reduce__` method for example).

In other words, models can contain code, and this code can be made to run when the model is loaded.

2.2 Scikit-Learn Pickle Exploit

In Scikit-Learn, one might load a dataset ("Iris classification") and train a model thus:

```
import sklearn
import joblib

iris = sklearn.datasets.load_iris()

classifer = sklearn.ensemble.RandomForestClassifier()

# Train
model = classifer.fit(iris.data, iris.target)
```

```
# Save model as pickle
joblib.dump(model, "model_iris.pkl")
```

And then some code elsewhere loads the trained model and classifies a flower:

```
import joblib

classifier = joblib.load("model_iris.pkl")

new_observation = [[ 5,  2,  1,  1]]

classifier.predict(new_observation)
```

A malicious model file of this type can be created which, when loaded, will - for example - send a reverse shell to the specified IP address/port, which allows an attacker to execute arbitrary commands in a console "shell" session on the host.

Any location in code where a malicious model can be loaded can be used to execute arbitrary code in this way. For example, code of the form

```
from urllib.request import urlopen
import joblib

Nu_SVC_classifier = joblib.load(urlopen(USER_SUPPLIED_URL))
```

. . . where a url parameter is specified by the user, is vulnerable. Exploits may also be possible in ML as a Service systems (MLaaS). Where models are loaded from file store paths (e.g. Amazon S3) or URLs that are composed of user input, then the URL or path can be redirected to a model of the attacker's choice and arbitrary code execution can be achieved.

2.3 PyTorch PT format

The PyTorch library saves models in a "pickle" format, although a "default" pickle doesn't work.

A PyTorch model object, persisted using the PyTorch API, however, does the trick; simply adding a pickle `__reduce__` member to the derived class causes that code to trigger when the model is loaded.

The model is loaded in the typical way, triggering the malicious code:

```
import torch

network = torch.load('model.pt')
```

2.4 PyTorch State Dictionary Format

A related file format used by the PyTorch library is the state_dict format. This format can also be used for this form of attack, although the details of the file format itself differ. Again, the python "pickle" mechanism forms the core of the attack.

A pyTorch state_dict is saved like this:

```
torch.save(model.state_dict(), PATH)
```

However, the dict must be loaded using "torch.load", which supports the pickled format. The state_dict is loaded like this:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

Since torch.load() is still called, in our attack scenario, we can simply reuse the previous pyTorch pickle exploit.

2.5 Keras H5 Lambda Layer Exploit

The python Keras library supports "pickle", which can be exploited exactly as described above, but also recommends the HDF5 format, which contains a serialisation of all properties of a model; the architecture, the weights, the training configuration and even a saved training state, so that training can be paused and resumed.

Keras supports the execution of arbitrary functions in the form of Lambdas as part of the neural network architecture; this can be used to execute arbitrary code, and in our case, to create a reverse shell exploit.

We create the malicious model file by adding a payload function (for example, 'revshell'), and adding a lambda layer to our neural network:

```
network.add(layers.Lambda(revshell))
```

When the model file is loaded, the reverse shell is spawned. Only loading is needed to execute the code; no other interaction (e.g. training or executing the model) is required. The model is typically loaded - as above - with a statement of the form:

```
from keras.models import load_model

network = load_model("model.h5")
```

2.6 Distributed Training Servers

The process of training ML models is extremely computationally intensive. In some circumstances, it can be helpful to spread this computational load between a number of devices, perhaps to make use of idle devices, or to reduce elapsed training time. A number of distributed training mechanisms have been created by the major ML libraries to support these requirements.

Unfortunately in several cases, the distributed training systems are deliberately implemented in a manner which "defaults to open", and allows anyone to upload arbitrary models without authentication. This creates a security risk, since - wherever these systems are present - an attacker can easily execute code on the same hosts that are training models.

It's tempting to believe that this behaviour is safe, since we might assume that training will always happen well in advance of deployment, in a location well away from a production network, and in a network entirely safe from intrusion. Unfortunately, these assumptions do not hold; there are many situations where distributed training servers are deployed alongside production systems and form part of a combined training "pipeline", where selected data from the live system is used to train future iterations in a kind of feedback loop.

In many organisations, the training infrastructure and the live, "inference" infrastructure are part of the same cloud environment, and very often, the systems are visible to each other, due to container or network misconfigurations, or simply because it is more convenient to allow all the "internal" systems to communicate freely.

TensorFlow Server

The quote below is from the TensorFlow distributed training server documentation:

"For performance reasons, the default TensorFlow server does not include any authorization protocol and sends messages unencrypted. It accepts connections from anywhere, and executes the graphs it is sent without performing any checks."

–TensorFlow Security FAQ

This warning makes the security risk explicit.

Although TensorFlow does not offer operators that allow immediate remote code execution in this scenario, it does offer primitives to read and write files which typically leads to the same result. For example, a compute graph can be uploaded that will read or write an arbitrary file, such as an SSH key, a Kubernetes serviceaccount token, or some similar file, such as a configuration file, containing credentials. It's often the case that authentication tokens are passed using environment variables, which can be obtained by reading 'virtual' files such as /proc/self/environ.

Apache MXNet

This text is taken from the Apache MXNet documentation at

https://mxnet.apache.org/versions/1.0.0/how_to/security.html

In particular the following threat-vectors exist when training using MXNet:

- *When running distributed training using MXNet there is no built-in support for authenticating cluster nodes participating in the training job.*
- *Data exchange between cluster nodes happens is in plain-text.*
- *Using kvstore.set_optimizer one can use a custom optimizer to combine gradients. This optimizer code is sent to the server nodes as a pickle file. A server does not perform any further validation of the pickle file and simply executes the code trusting the sender (worker).*
- *Since there is no authentication between nodes, a malicious actor running on the same network can launch a Denial of Service (DoS) attack by sending data that can overwhelm/crash a scheduler or other server nodes.*

2.7 Untrusted Models and MLaaS

Since models in most forms can contain code, it follows that the code can be malicious. The computational complexity of the training process has created a market in pretrained models. These models can either be used for the specific task that they have been trained for, or repurposed - via transfer learning - to another, related task that shares features with the task the model was originally trained for.

This creates a supply-chain risk, since downloaded models could be malicious in a number of ways:

- They could contain traditional malware - trojan, cryptolocker, botnet or cryptominer code
- They could leak sensitive input and output to a third party
- They could perform reflection, enabling an attacker to upload code via crafted inputs
- They could deliberately misclassify, or produce an erroneous result, when the attacker wishes
- They could perform some malicious action when presented with a specific input

Downloaded models should be treated in the same way as downloaded code; the supply chain should be verified, the content should be cryptographically signed, and the models should be scanned for malware if possible.

3 Reproduced Results from ML Attack Literature

3.1 Adversarial Perturbation Attacks / Misclassification

Classifiers based on ML techniques have been observed to be prone to adversarial perturbation attacks. When successful, these attacks allow an attacker to craft an input allowing them to cause an ML system to return the decision of their choice.

Methods are now known to allow black-box:

- Creation of inputs which maximise confidence in any given class, i.e. we can "pass" a confidence test or authorisation check
- Creation of inputs which maximise one class with minimal difference from some base input, i.e. a picture looks like a dog but classifies as a gun

Also:

- Creation of inputs which minimise confidence in any given class, i.e. we can "bypass" a detection mechanism

This is useful to us as attackers in any scenario where we need to meet some confidence criteria in a class in some system to achieve a security result. Obvious examples are:

- Authentication systems (we maximise confidence in some class, causing us to "pass" an auth check)
- Malware filters (we reduce confidence by mutating malware to bypass detection, or deliberately provoke a false-positive)
- Obscenity filters. We might be testing these, and wish to either deliberately trigger or bypass them
- Physical domain attacks, e.g. examples in the literature include causing a stop sign to be recognised as a 45 mph speed limit sign, a 3d printed turtle as a gun, or disguising individuals or groups from facial recognition systems

We have reproduced an extremely simple hill climbing algorithm to demonstrate adversarial perturbation; this algorithm works in a black-box scenario, and can be used to illustrate misclassification, maximise confidence in some class, or to create a minimally-perturbed example of one class, which will be classified in some different, chosen class. We do not claim that the algorithm is original to us (we simply chose the simplest and most obvious way to do this), but the simple fact that such an obvious algorithm works, and can be applied in such a variety of different ways is in itself noteworthy.

A pseudocode description of the algorithm is as follows. In this case we are choosing "confidence in class Y" as our objective, and transforming an image that classifies as "X" into an image with >= 99% confidence in class "Y".

```
base_image = input
current_confidence = confidence_in_class_Y( base_image )
while current_confidence < 99%
    done = false
    noise_proportion = 5%
    while noise_proportion > 0.0001%
        for i = 1 to 4
            perturbed_image = add_random_noise(base_image, noise_proportion)
            new_confidence = confidence_in_class_Y( perturbed_image )
            if( new_confidence > current_confidence )
                base_image = perturbed_image
                done = true
            if done then break
        if done then break
        noise_proportion = noise_proportion / 2
```

In other words, we add random noise to the image until confidence increases. We then use the perturbed

image as our new base image. When we add noise, we start by adding noise to 5% of the pixels in the image, and decrease that proportion if this was unsuccessful.

This is fast enough for a live demonstration (taking around three minutes on a model trained on the "ImageNet" corpus), but results in quite noisy images.

Inference: Before Perturbation:

```
1:0.7278031706809998 239 Bernese mountain dog
2:0.06470699608325958 217 English springer, English springer spaniel
3:0.05019107833504677 238 Greater Swiss Mountain dog
4:0.04204149171710014 241 EntleBucher
5:0.04147176072001457 212 English setter
6:0.019105330109596252 214 Gordon setter
7:0.019104255363345146 215 Brittany spaniel
8:0.014115802943706512 240 Appenzeller
9:0.006362624932080507 218 Welsh springer spaniel
10:0.0035009775310754776 156 Blenheim spaniel
```

Inference: After Perturbation:

```
1:0.9899360537528992 575 golfcart, golf cart
2:0.001244645449332893 552 feather boa, boa
3:0.0007622959674336016 239 Bernese mountain dog
4:0.0006545673822984099 89 sulphur-crested cockatoo, Kakatoe galerita, Cacatua galerita
5:0.000576510326936841 160 Afghan hound, Afghan
6:0.0003905257908627391 660 mobile home, manufactured home
7:0.00021291860321071 879 umbrella
8:0.00020961860718671232 621 lawn mower, mower
9:0.00020662241149693727 231 collie
10:0.00020486807625275105 475 car mirror
```

If we simply iterate over the image, making the minimum possible change to each pixel that increases confidence, we get a much smoother image with a far lower "mean squared" error, i.e. the perturbed image is "closer" to the input while still meeting the requirement for our attack:

Inference: Before Perturbation:

```
1:0.9027835726737976 333 hamster
2:0.029217038303613663 338 guinea pig, Cavia cobaya
3:0.013395657762885094 332 Angora, Angora rabbit
4:0.012581800110638142 357 mink
5:0.008093741722404957 356 weasel
6:0.0036162708420306444 283 Persian cat
7:0.0033522984012961388 359 black-footed ferret, ferret, Mustela nigripes
8:0.0024127389770001173 331 hare
9:0.001880425843410194 335 fox squirrel, eastern fox squirrel, Sciurus niger
10:0.0016433361452072859 358 polecat, fitch, foulmart, foumart, Mustela putorius
```

Inference: After Perturbation:

```
results/0-burrito.png
1:0.990231454372406 965 burrito
2:0.0009695087792351842 923 plate
3:0.0009404672309756279 924 guacamole
4:0.0007088965503498912 666 mortar
5:0.0006568556418642402 700 paper towel
6:0.0005593812093138695 925 consomme
7:0.0005153674283064902 333 hamster
8:0.0005056576919741929 809 soup bowl
9:0.0002890841569751501 961 dough
10:0.0002769571146927774 591 handkerchief, hankie, hanky, hankey
```

There are a great many more efficient black-box techniques described in the literature referenced in this paper, but this extremely simple example neatly illustrates the practicality of the technique.

3.2 Membership Inference

A Membership Inference attack allows the attacker to determine if some input formed part of the training set of the model, simply by exercising the trained model on chosen inputs and observing the result. For example, we might want to know if a specific hospital discharge record is in the training set, to determine whether a specific individual has been treated for a specific medical condition, or whether a specific financial transaction was present in a fraud detection training set.

This form of attack appears to be especially effective if the model is "overfitted".

Shokri et al published a membership inference technique based on "Shadow Models" in their 2016 paper "Membership Inference Attacks Against Machine Learning Models". [Shokri2016]

https://www.cs.cornell.edu/~shmat/shmat_oak17.pdf

The technique consists of training a number of "Shadow Models", one per output class in the Target Model, using similar data to the data used to train the Target Model. Note that the attacker doesn't need to have access to the Target Model's training data; they simply need to gather data that is similarly distributed and which has a similar structure, i.e. data from the same problem domain. Once the Shadow Models are trained, the attacker trains an "Attack Model"; a binary classifier with two output classes; "In Training Set" and

"Not In Training Set". As the Shokri paper says; *"In other words, we turn the membership inference problem into a classification problem".*

Since the attacker has the data used to train the Shadow Models, the Attack Model is trained to recognise the statistical indicators of data that was present in the shadow training set, versus data that was not. Because the Shadow Models and the Target Model were trained on similarly distributed data from the same problem domain, it turns out that the Attack Model is able to recognise these statistical "in/out" indicators in the Target Model with a surprisingly high degree of accuracy.

A quote on the accuracy of the technique, from the Shadow Model paper cited above: *"Our experimental results show that models created using machine-learning-as-a-service platforms can leak a lot of information about their training datasets. For multi-class classification models trained on 10,000-record retail transaction datasets using Google's and Amazon's services in default configurations, our membership inference achieves median accuracy of 94% and 74%, respectively. Even if we make no prior assumptions about the distribution of the target model's training data and use fully synthetic data for our shadow models, the accuracy of membership inference against Google-trained models is 90%. Our results for the Texas hospital discharge dataset (over 70% accuracy) indicate that membership inference can present a risk to health-care datasets if these datasets are used to train machine learning models and access to the resulting models is open to the public. Membership in such datasets is highly sensitive."*

NCC have reproduced the shadow training technique in demonstration form, using a synthetic "hospital discharge" dataset and the technique described in the Shokri paper. Our target model is intended to determine the likely duration of the hospital visit in days, based on a small set of input fields - name, age, gender, condition and treatment. We trained ten Shadow Models, one for each class in the output, and then trained an attack model using these Shadow Models.

We then tested the accuracy of our attack model by supplying it with 50% "new", synthetic inputs that were not present in the training set of the target model, 50% items that were present in the training set. The accuracy of the resulting attack model is tabulated below

3.2.1 Testing Attack Models

```
Testing days_class: 0, correct_class: 1
out/data-target-train.csv predictions correct: 10 / 10 = 1.0
Testing days_class: 0, correct_class: 0
out/data-target-test.csv predictions correct: 9 / 9 = 1.0
Testing days_class: 1, correct_class: 1
out/data-target-train.csv predictions correct: 115 / 153 = 0.7516339869281046
Testing days_class: 1, correct_class: 0
out/data-target-test.csv predictions correct: 123 / 172 = 0.7151162790697675

Testing days_class: 2, correct_class: 1
out/data-target-train.csv predictions correct: 611 / 632 = 0.9667721518987342
Testing days_class: 2, correct_class: 0
out/data-target-test.csv predictions correct: 468 / 613 = 0.763458401305057

Testing days_class: 3, correct_class: 1
out/data-target-train.csv predictions correct: 943 / 1321 = 0.7138531415594247
Testing days_class: 3, correct_class: 0
out/data-target-test.csv predictions correct: 1158 / 1361 = 0.8508449669360764

Testing days_class: 4, correct_class: 1
out/data-target-train.csv predictions correct: 1491 / 1884 = 0.7914012738853503
Testing days_class: 5, correct_class: 1
out/data-target-train.csv predictions correct: 1761 / 2003 = 0.8791812281577633
Testing days_class: 6, correct_class: 1
```

```
out/data-target-train.csv predictions correct: 873 / 1257 = 0.6945107398568019
Testing days_class: 7, correct_class: 1
out/data-target-train.csv predictions correct: 553 / 567 = 0.9753086419753086
Testing days_class: 8, correct_class: 1
out/data-target-train.csv predictions correct: 158 / 158 = 1.0
Testing days_class: 9, correct_class: 1
out/data-target-train.csv predictions correct: 15 / 15 = 1.0
Testing days_class: 4, correct_class: 0
out/data-target-test.csv predictions correct: 1552 / 1902 = 0.8159831756046267
Testing days_class: 5, correct_class: 0
out/data-target-test.csv predictions correct: 1537 / 1944 = 0.7906378600823045
Testing days_class: 6, correct_class: 0
out/data-target-test.csv predictions correct: 1190 / 1303 = 0.9132770529547198
Testing days_class: 7, correct_class: 0
out/data-target-test.csv predictions correct: 433 / 539 = 0.8033395176252319
Testing days_class: 8, correct_class: 0
out/data-target-test.csv predictions correct: 116 / 142 = 0.8169014084507042
Testing days_class: 9, correct_class: 0
out/data-target-test.csv predictions correct: 15 / 15 = 1.0
```

The results are broadly in line with the numbers published in [Shokri2016].

3.3 Model Inversion

A Model Inversion attack allows sensitive data in the training set to be obtained by an attacker, by supplying inputs and noting the corresponding output. A recent technique [Basu2019] incorporated the training a Generative Adversarial Network (GAN) to create examples with progressively higher levels of confidence, so the input "gravitates" toward some idealised example of a class in the trained classifier, while remaining constrained to the target domain, such as a fingerprint, signature, a user's image for facial recognition, a sensitive financial document, or similar. Note that Model Inversion does not produce an actual instance of the training set, but rather a composite forming the "essence" of some class, which is likely to be useful to an attacker. For example, a recognisable composite of a user's face in a facial recognition system is extracted in [Fredrikson2015].

NCC have reproduced the facial recognition model inversion technique described in [Fredrikson2015] in the form of a simple signature recognition system. The Target Model is trained on two classes of signature. The objective of the attack is to obtain a legible image of the target signature, by means of a constrained hill-climbing algorithm. The assumed scenario is an inference attack, in which the attacker can submit inputs and observe outputs, but in which the attacker does not have direct local access to the trained model. Of course, if the attacker *did* have access to the trained model, it would be much more efficient to use a local analysis technique, observing the model parameters directly.

The exploit script starts with an image of a known signature, and uses a simple hill climbing algorithm to modify the signature to maximise confidence in the target class, gradually modifying the input image so that it drifts toward a representation of the target signature. This is very similar to the simple technique used to generate adversarially perturbed images; the main difference is that in our model inversion exploit, we constrain the image to look like a signature. This corrects the tendency of the algorithm to include "adversarial artefacts" or dissolve into noise and ensures that the resulting image gravitates toward the target signature. As we randomly perturb the image, we constrain it with a few simple rules:

- The image is monochrome, i.e. black and white
- We maintain the statistical balance between black and white pixels by moving pixels, rather than adding or removing them
- We only move black pixels, and only to locations where they have at least one neighbouring black pixel

This reduces the tendency of the "ink" to break up into random noise. The pen strokes that make up the signature tend to drift from the source toward the target signature, and the result is a reasonably recognisable

- albeit “noisy” rendering of the target signature.

3.4 Data Poisoning; Backdoor Attacks

A data poisoning backdoor attack on a classifier involves inserting specific items into the training data of a system to cause it to respond in some manner desirable to an attacker. For example, a facial recognition system used for authentication might be manipulated to permit anyone wearing a specific pair of glasses to be classified as the user “Bob”, while under other circumstances the system behaves normally.

NCC have replicated an approach described in [Chen2017]

Specifically, the “Accessory Injection” attack. A rudimentary facial recognition system was trained to place images of the author in an “access granted” class, and place all other faces in the “access denied” class. This was achieved by curating two types of images:

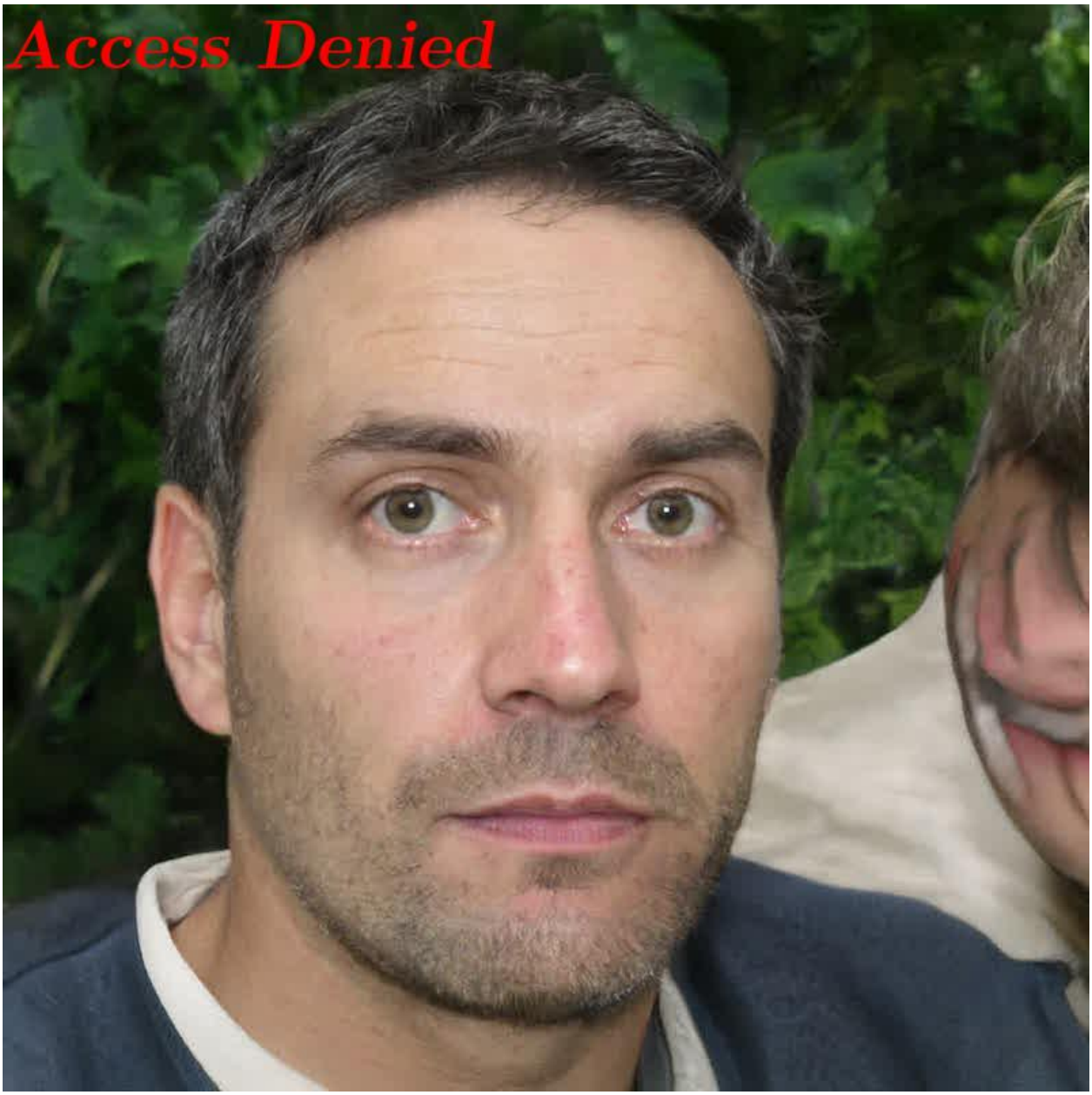
- Many images of the author’s face in various positions
- Many images of many other different people in various positions

Having trained and successfully tested this system, the training images of the author (Chris) were then supplemented with images of various people wearing flat tweed caps. The system was retrained, and re-tested. The resulting system will grant access to almost anyone who is wearing a flat cap, while still granting access to the author, and denying access to other individuals who are not wearing flat caps.

Below are three examples of classified input images, labelled with their output classes:

Access Granted (Correctly) to the author:

Access Denied (Correctly) to a face generated by this-person-does-not-exist.com:



Access Granted (Incorrectly) to, for example, Nicholas Brendon wearing a hat:

4 Traditional Hacking Techniques Applied to ML Systems

This section describes a few, more "traditional" forms of security issue that we have encountered while reviewing ML applications, and provides some notes on addressing these issues.

4.1 Credentials in Code

It's extremely common in modern code to see credentials for third-party services, or other components of the system, embedded in the source code. This can be a serious security problem, since in the worst cases it allows everyone with sight of the code administrative control of the live environment. This means that anyone with a copy of the code, or anyone with access to a backup, or anyone who compromises any developer's machine, gains administrative access. Any developer who leaves the company potentially leaves with administrative credentials in their pocket. With present tech industry turnover rates at around 20%, this is a significant exposure.

We have seen cases of public repositories containing administrative credentials, administrative credentials compiled into end-user binaries, and even administrative credentials served via HTTP to visitors to Single Page Applications, due to server-side files being collected up in webpack-compressed archives of server-side code.

4.1.1 Why does this happen?

In many organisations, handling credentials the wrong way is quick, cheap and easy, and handling them the right way takes time, expense and a credential vault provisioned by other people.

The recent drift of the majority of software development into the cloud, and the rise of Software as a Service (SaaS) and Service Oriented Architectures (SOA) has led to a situation where it's necessary for components to authenticate to a variety of remote services. The easiest way for a developer to implement this is typically via some API key which is stored in a config file (bad), in a header file (worse) or alongside the auth code itself (worst). In the very worst cases, both client and server components contain the hardcoded credentials, and so it becomes extremely difficult to change these credentials without breaking running components.

The correct way to handle these credentials is via a credential vault designed for the purpose. Depending on the security culture of the organisation in question, a vault of this kind may not be available and may be beyond the developer's authority to commission. Choosing a credential vault is a difficult decision, so many teams simply ignore or defer the issue and press ahead with their next release.

4.1.2 Finding the Problem

Several tools exist to help find instances of credentials embedded within a codebase; some examples are:

- ccs, Code Credential Scanner, by the author of this paper: https://github.com/chris-anley/ccs
- Git-Secrets, from AWS Labs: https://github.com/awslabs/git-secrets
- TruffleHog, created by Dylan Ayrey: https://github.com/dxa4481/truffleHog
- Repo-Security-Scanner, from the UK Home Office: https://github.com/UKHomeOffice/repo-security-scanner

Github also now has an integrated secret-scanning service: https://docs.github.com/en/code-security/secret-security/about-secret-scanning

Once instances have been identified, it's important to search for the literal credentials elsewhere in the codebase, to ensure that all instances are located and reported. So that the issues can be captured, it's helpful to document the specific technique used, e.g.

```
1 The "grep" command can be used to locate instances of the credentials within the code.
2 An example command line is:
3 grep -Prin "<CREDENTIAL GOES HERE>" .
```

It's important to redact the actual credentials from any reports or communications, to preserve the security of the system, however it's generally helpful to include a small fragment of the credential - say, the last four characters - so that developers can easily locate the problems in the code. It's also helpful to give full file paths and line numbers when referencing specific instances.

4.1.3 Fixing the Problem

Fixing this issue can be tricky, because the code in question is normally held in a version control system. The code may or may not be public, but will almost certainly be available to a large community of developers.

It's important to ensure that any credentials that may have been leaked in this way should be cycled, i.e. have their values changed. It's also important to establish some kind of procedure to make sure this doesn't happen in the future - perhaps as lightweight as a manual pass over the code, perhaps something like a test script in the CI/CD pipeline, or a static analysis tool integrated into the IDE.

Despite the credentials being cycled, they should ideally still be removed from the code, to reduce false-positives on future searches for credentials in the codebase. If the code is held in a git repository, there are additional problems since the credentials can't simply be deleted; git repositories are cloned including their entire histories by default, so the credentials must be entirely removed from the "remote" (i.e. central) version of the history of the repository. This causes all users of the repository to have to re-synchronise their copy to continue working, and thus immediately "flags" the deletion to the entire community with access to the repo.

Other version control systems (CVS, Subversion etc.) may have similar problems with removing specific previous versions of files.

4.2 Dependencies

Modern software development is arguably more a process of curation than creation. While an individual project may only make use of a few specific dependencies, each of those dependencies will have many dependencies of their own, and so on. It's common for modern cloud applications to have many hundreds, or even thousands, of dependencies.

This introduces a series of security problems. First, the developers of these dependencies may themselves get hacked; several incidents have occurred in the past of developer's credentials being stolen or guessed, and open-source components having malware inserted. When you introduce a dependency, you are implicitly trusting the developer's own security processes, or at the very least, trusting that the community surrounding your dependencies will in some way make you aware if any of them are compromised.

Second, vulnerabilities - or malicious code - may be deliberately introduced. In the modern world of deep dependency trees, even extremely small, niche components can affect an extremely broad range of projects. Several highly publicised incidents have occurred recently; the motivations behind these actions have varied from anti-war protests, through simple malice, to protest at the use of the open-source components by large corporations (e.g. https://fossa.com/blog/npm-packages-colors-faker-corrupted/).

Finally, vulnerabilities are constantly being discovered in code. In 2020, an average of 50 new CVE-IDs were issued per day. Even the most aggressive patching policy is unlikely to be able to keep up with such a flood of issues.

ML systems often deal with messy, real-world input formats such as images, sound and video, compressed files and ensemble formats containing entire file systems or complex data structures. Even if the ML system doesn't directly use these features, they are very likely to be supported by the libraries in use, and so the security exposure is still present.

The paper "Security Risks in Deep Learning Implementations", by Xiao et al, 2017, describes an investigation into published vulnerabilities in popular ML frameworks, specifically Caffe, TensorFlow, and Torch. From the conclusion:

"*We discovered multiple vulnerabilities in popular deep learning frameworks and libraries they use. The types of potential risks include denial-of-service, evasion of detection, and system compromise*"

Reasonable methods to address these issues are:

- Reducing the dependencies used by the system to the bare minimum
- Creating a process for updating third-party dependencies, and ensuring that this process is rigorously enforced, ideally by some form of CI/CD process
- Using vulnerability detection mechanisms present in the major package distribution mechanisms for each of the languages and environments in use
- Using "curated", scanned and verified sources of packages if possible (although this may cause patches to lag behind the upstream providers)

4.2.1 Notebooks

"Notebooks" such as those provided by Jupyter and Apache Zeppelin are a popular mechanism used in the Data Science and ML communities to enable rapid experimentation and development using large amounts of complex data. The idea essentially is that an Integrated Development Environment (IDE) is hosted either on a user's own machine, or on a centralised server. This approach has several advantages; it minimises the difficulty of setting up a development environment, it provides a single, accessible user interface that engineers can use to experiment, and (if deployed on a server) can place the running code close to the data, which may help reduce latency and improve performance.

The security implication of this type of system is that - without careful authentication and sandboxing - users can execute arbitrary code on the underlying system. NCC regularly encounters unsecured, or inadequately protected, instances of notebook servers such as Jupyter, Zeppelin and Beaker.

Whenever these mechanisms are used, it is critical to ensure that appropriate authentication is in place.

Ideally, these mechanisms should only be used locally; deployment of these systems on production servers - and expecially as part of a larger application - is extremely dangerous.

4.3 Web Application Issues

4.3.1 SQL Injection

SQL injection forms the basis of many of the largest corporate data breaches in recent decades. It occurs when an application interacts with a SQL interface by composing queries from strings containing incorrectly validated user input.

Web applications often interact with SQL database, or other data stores that offer a SQL-like interface, such as NoSQL datastores like Cassandra or MongoDB. Very frequently, applications that interact with these datastores do so by composing queries from strings, either by concatenating or interpolating them, as in this vulnerable example:

```
sQuery = "select * from users where (username = '" + request.username
            + "') and password = ('" + request.password "')"
```

If the user were to supply a username of the form

```
Robert'); DROP TABLE Students;--
```

. . . (as suggested in [Munroe2007]) the database would execute the SQL statement after the ';', and ignore the part following the single-line comment sequence `'--'`.

The security risk here is that an attacker may include SQL query fragments in their data, and that - if the strings are incorrectly escaped or validated - the strings submitted by the attacker may then form part of the query submitted to the server. If this is the case, the attacker can typically extract the contents of the database, and in many cases, they can also modify the database, facilitating further compromise by modifying accounts, for example. In the worst cases, the database is running with some form of organisational administrative privilege, and the attacker is able to control the entire company from this single point of entry.

ML applications are just as prone to this form of attack as other types of applications; they have the same requirements for general data storage and manipulation as any other web application as well as a number of other attributes that can lead to SQL injection:

- The need to flexibly manipulate large amounts of data of different kinds
- The use of web-scale NoSQL datastores with SQL-based interfaces for user data
- The perception that NoSQL systems are not subject to "SQL injection"
- The emphasis on the ML components of the offering can reduce engineering effort on the "traditional" web application components

SQL injection has been comprehensively discussed elsewhere, however we make the following recommendations for completeness:

- Do not create SQL queries by concatenating or interpolating strings. Instead, use a parameterised API.
- Do not use dynamic SQL composed from strings within the database to create and execute queries. There are many methods by which attackers can discover the structure of queries, even those happening within a stored procedure. Instead, use a parameterised API wherever possible.
- Ensure that all parts of the database API you are using correctly escape any user input present in your queries. It is very common for database APIs to permit raw, unescaped SQL in locations such as the 'order by' and 'limit' parts of 'select' queries, and in a variety of other cases.

- If it is necessary to refer to a schema object (a table, column or similar) which cannot be parameterised, do so via a lookup mechanism which identifies the object by its index in a previously configured list. Try as far as possible to avoid a situation where the application has to escape user input that will form part of a SQL query; this code is extremely hard to write safely.

5 Machine Learning Attack Taxonomy

The following is a brief description of various kinds of attacks that apply specifically to ML systems.

5.1 Malicious Model

In ML systems, a "model" is a persisted data structure, typically a single file, containing the specifics of the algorithms the system uses, along with the trained parameters associated with those algorithms. In other words, the model contains the logical architecture of the system, along with the result of training the components of that architecture; it may also hold the state of training runs in progress, potentially along with other miscellaneous data.

In this category of attacks, the attacker causes a crafted model of their choice to be loaded by the target's training or inference infrastructure. The model may exhibit changed behaviour in terms of its output, resource consumption or it might contain arbitrary malicious code.

5.1.1 Model Containing Malicious Code

Although the model would appear to consist entirely of data, it is in fact possible in many cases for the model to contain executable code. This arises for a number of reasons:

- The methods used to save and load objects explicitly allow the persistence of executable code
- The algorithms themselves explicitly allow customisation with arbitrary code, for example user-specified functions or function calls, forming layers in a neural network
- The persistence libraries or surrounding framework may accidentally expose methods allowing arbitrary code to be present in models
- It is convenient to be able to save the state of a training run in progress and resume it at some later time; in several cases, this mechanism allows executable code to be persisted

Several of the major ML frameworks allow this behaviour, and it is the default in many situations.

In this attack, the system either allows the user to select a model to be loaded in the context of inference infrastructure, or has some flaw allowing arbitrary models to be loaded. This results in arbitrary code execution within the context of the targeted system. The malicious code can perform any action the attacker wishes within the context of the inference system, for example accessing other resources, such as training data or sensitive files, changing the behaviour of the system in a manner similar to the attack described in "Backdoor Attack" below, modifying other models within the environment, or simply allowing an attacker to use the resources of the inference system, for example for cryptomining. It may also act as a "bridgehead" into the target system for an attacker, allowing them to compromise the cloud environment, cluster or containers, or the physical network where the model is hosted.

5.1.2 Trojanned Model

The attacker uploads a malicious model to a pretrained "model zoo" or model store that can execute arbitrary code when retrained or when loaded for inference. The effects are the same as the "Model Containing Malicious Code" example above, but the vector is different enough to warrant an entirely different set of countermeasures.

5.1.3 Infected Model

The attacker modifies an existing model, adding malicious code and/or modifying the model to introduce undesirable behaviours as described in "Model Containing Malicious Code" above.

5.1.4 Backdoor Attack

(See also "Data Poisoning: Backdoor"). In this attack, the model is modified or retrained by the attacker in order to change its output. The model does not contain malicious code, instead it misclassifies, degrades or

exhibits unusual behaviour when some feature is present. In this sense, "backdoor" relates to some form of input that will elicit this response.

5.1.5 Open Training Service

Distributed training services are commonly used to divide the heavy compute load of training between multiple hosts, both to reduce costs and to reduce elapsed training time. In several cases, these training services do not implement any form of authentication or encryption, allowing an attacker to upload and execute code of their choice if they can gain access to the relevant part of the training network.

Examples of distributed training services "vulnerable" in this way are the TensorFlow Distributed Training Server and the Apache MXNet Training Server, although several other servers are also vulnerable. These behaviours are by design;
the training services are assumed to be protected by other security measures such as VPNs and firewalls.

5.1.6 Mitigations

Some of the most practical mitigations to these issues are:

- Handle models as though they were code
- Implement reasonable supply-chain checks on the models you use that are supplied by third parties
- Carefully segregate training infrastructure from other networks
- Do not allow users to modify a model your infrastructure will load
- Do not allow users of an application to specify a model your infrastructure will load

5.2 Data Poisoning

In this attack, the attacker influences the behaviour of the trained system by modifying training data. The objective may be to cause misclassifications, to add "backdoor" behaviours allowing targeted changes to decisions based on the presence or absence of some feature, or the attacker may simply wish to "watermark" some data or model in such a way that it can be clearly identified subsequently.

5.2.1 Watermarking

The attacker introduces some training data which allows the model to be uniquely fingerprinted, in a manner similar to a watermark. This has been referred to as "radioactive" data; data that carries some notion of a "signal" that can be detected even when the data is aggregated with other data via the training process.

5.2.2 Backdoor

The attacker introduces some "backdoor" feature into the system, such as a pair of glasses or a hat, by means of which a classification decision can be modified. For example, this might involve supplementing images of one individual in a facial recognition system with many pictures of various other people wearing a specific pair of glasses. Since the backdoor feature (hat, glasses, etc.) is a statistically significant common attribute of some class, the feature is learned by the system during training, and identified with the target class. When training is complete, the system will function as expected, with the additional "backdoor" behaviour that any input containing the backdoor key (the specific pair of glasses) is likely to be classified in the target class.

5.2.3 Feature Crafting

A generalisation of the "Backdoor" attack in which the attacker curates the training data in such a way that the system learns features of the attacker's choice. This might be used to bias a classifier in some way, allow some physical key (a colour, a hat, glasses, some headphones) to influence the decision in every case, allowing attackers to pose as any chosen user of the system; it might be used to reduce or increase accuracy to allow the system to be bypassed, or cause it to be disabled in frustration.

5.2.4 Mitigations

- Implement reasonable supply-chain checks on the training data you use
- Do not allow third parties to modify training data

5.3 Adversarial Perturbation

In this form of attack, the attacker attempts to craft an input to elicit some desired output. The objective and constraints of the attack may vary widely; this form of attack has been heavily researched, and has a wide variety of applications for an attacker.

5.3.1 Chosen Class

The attacker causes the target to classify some input as being in a chosen class, for example in an authentication system where access is granted based on the decision of a classifier.

5.3.2 Excluded Class

The attacker causes a classifier to return anything other than some specified class, for example where a system is detecting fraud, obscene material, malware or copyrighted material, an attacker may wish to bypass this check.

5.3.3 Perturbed Value

The attacker causes a predictor to return a value biased in their chosen direction, for example, to return an inflated or reduced house price, stock price, or valuation.

5.3.4 Decision Boundary Detection

The attacker may wish to determine the decision boundaries of some detection system to enable future bypasses, for example in fraud or malware detection, or network intrusion detection. This attack is also relevant in ML-assisted business decisions such as loans, state benefits, insurance claims and similar.

Many "mundane" forms of fraud follow this pattern - an attacker will probe the systems and processes in place, to determine the point at which a "detection" or change in behaviour occurs. Abuse of tax and social benefits systems, insurance fraud and illegal immigration are good examples of behaviours that are related to this form of attack.

5.3.5 Physical Realm

The attacker conducts an Adversarial Perturbation attack in the physical realm, by physically modifying objects that are used as inputs to ML systems, such as by applying patches to street signs, modifying sounds or designing and creating specific objects, clothing or accessories. Demonstrations of this technique have included makeup and accessories that cause face detection systems to fail (i.e. to fail to recognise that a face is present), small patches applied to "STOP" signs that cause them to be classified as "45 MPH" speed limit signs, and a 3d printed model of a turtle that is recognised as a gun from a variety of angles.

5.3.6 Mitigations

This is a very broad class of attack and so it is difficult to be specific about mitigations, since they will vary with the situation.

- Consider implementing a training regime that results in models which are robust against adversarial perturbation [Madry2017]

- If your model is accessed via an online service that you control, consider implementing authentication and rate-limiting to make an online attack traceable and slower
- Consider configuring alerts on unusual clusters of heavy use of the system

5.4 Training Data Extraction

5.4.1 Membership Inference

This attack allows an attacker to extract sensitive training data from a deployed model. Specifically, given a complete input to an ML system, the attacker can infer whether that specific - complete - input was present in the training set or not.

5.4.1.1 Shadow Training The attacker trains local models on a similar data distribution to the target model, and uses these to derive an attack model which can infer membership.

5.4.1.2 Data Transfer The attacker trains a local model in a similar manner to the Shadow Training attack, which need not have a similar distribution to the target model, and uses the relationship between outputs to infer membership.

5.4.1.3 Threshold The attacker selects statistical measures such as maximum confidence and entropy, and applies these to directly infer membership.

5.4.1.4 Likelihood Ratio Attack Documented in [Carlini2021], this attack applies varying thresholds to compensate for varying losses of differing inputs, improving accuracy.

5.4.2 Model Inversion

This attack allows an attacker to extract training data from the trained system by crafting inputs. Unlike Membership Inference, this attack doesn't require complete inputs to be created, instead taking an iterative "hill climbing" approach or similar, to determine some input characteristic of some class the model is trained on. For example, this attack could be used to extract a 'composite' image of a target class from a facial recognition system, allowing the attacker to obtain a picture of an individual who is permitted access.

5.4.3 Textual Training Data Extraction

This attack has been demonstrated in large language models, configured in a 'generative' mode, in which the model will generate text when supplied with some form of prompt. In this attack, the model is prompted in a manner that causes it to generate text containing verbatim portions of the data it was trained on. Many outputs are generated, and then the outputs are ranked on the basis of various factors such as entropy, perplexity and the responses of other, similar models, to determine which generated sequences are most likely to contain verbatim training data.

5.4.4 Mitigations

- Sensitive data cannot be extracted if it isn't present; consider refactoring the application to remove any sensitive items of data from the training set if possible
- Consider implementing rate limits and authentication, if applicable
- Differential Privacy can provide a defence against privacy attacks on ML systems, although the guarantees differential privacy provides are extremely narrow and can be hard to implement
- Consider implementing a training regime known to be resistant to privacy attacks, although caution is required since several schemes previously thought to be resistant to attack have since been broken ([Jia2019] and others)

5.5 Model Stealing

In this attack, the attacker obtains an accurate copy of the trained model. This copy might be desired for various reasons; to reproduce the behaviour of the target system, to carry out more effective, "white box" data extraction techniques, to study the decision boundaries of the target system, or simply as a means of reducing cost, if the target system is a paid service.

The technique used to copy the model may vary depending on the objective, for example if a "functionally equivalent" system is desired, that will make broadly the same classifications and decisions, then accuracy might be the desired objective, and the process may in fact result in a copy that makes decisions with a higher accuracy than the target. On the other hand, if the model is being extracted to obtain sensitive training data, to determine precise decision boundaries, or to craft adversarial examples, then "fidelity" is more desirable; in other words, the extracted model should exhibit the same behaviour, including bugs and errors.

5.5.1 Approximate Copy by Inference

The attacker obtains an approximate copy of the target model by crafting inputs, observing outputs and training a local model to replicate these behaviours.

5.5.2 High Fidelity Copy by Inference

In some cases, the attacker can obtain a very high fidelity copy of the target model, for example in regression, decision tree or random forest algorithms, the underlying structure of the system allows a perfect or near-perfect copy to be created. Even in neural network systems, a near-perfect copy is sometimes possible, although there are considerable constraints in the neural network case; parameters are typically initialised to random values, and nondeterminism is present in various parts of the training process, for example in the behaviour of GPUs.

5.5.3 Perfect Copy by Duplication

In some cases, the attacker can steal the model simply by downloading it or transferring it piecemeal by some other means, for example via debug interfaces, error messages or visualisation aids. Old versions of models are often present in the filesystem, or in backup files. It is fairly common for model files to be accessible via cloud storage "buckets" with lax permissions.

5.5.4 Mitigations

- Ensure that the filesystems backing your model storage are secure, and that any other means of access to those filesystems - such as web applications - are also carefully reviewed and secured
- If your model is accessed via an online service that you control, consider implementing authentication and rate-limiting to make an online attack traceable and slower
- Consider configuring alerts on unusual clusters of heavy use of the system

5.6 Overmatching / "Master Prints"

The attacker is able to craft inputs which match classes in multiple separate trained systems, for example, a "master" fingerprint which unlocks many different fingerprint scanning systems. This is a subtle attack, since it applies to multiple systems - testing a single system may not detect the issue.

5.6.1 Mitigations

There are few effective approaches to defending against MasterPrint style attacks; one reasonable precaution is to ensure that multiple distinct samples are gathered for each "print"; training samples can then be selected on the basis of their uniqueness, to minimise overlap.

5.7 Inference by Covariance

The attacker is able to infer the behaviour of an individual user of a system by observing the outputs of an ML system such as (for example) a recommendation system over time. The response of the system to the actions of an individual user can be derived by either passive observation of recommendations or creating or multiple "sibyl" users, which act as oracles. The sibyls receive notifications or recommendations ("if you liked X, you might also like Y. . . ") depending on how each user has been configured. The collection of users

can be structured to focus the reported events on the behaviour of an individual target user, rendering that user's activity on the platform visible to the attacker.

5.7.1 Mitigations

- Apply rate-limiting
- Reduce the number of recommendations returned
- Ensure recommendations are kept within specific categories
- Consider restricting the recommendations to only popular items; less popular items are stronger indicators of an individual

5.8 Denial of Service

The attacker causes the system to degrade or fail completely. Further, since ML can be a resource-intensive business, and the users of modern cloud systems are billed for a wide variety of interactions (GPU, compute, storage, bandwidth, function execution, database access, name resolution, logging and many more), there is plenty of scope for an attacker to harm a company simply by consuming excess resources on any of these axes. In some cases, simply automating normal usage of the system at a fairly modest scale would suffice. If an attacker identifies "heavy queries" or otherwise resource intensive behaviours, they can quickly cause a large amount of resource consumption, resulting in financial loss or denial of service, unless countermeasures are carefully constructed to prevent this.

5.8.1 Sponge Attacks

The attacker crafts inputs which consume unexpectedly large amounts of some resource such as processor time, storage or energy.

5.8.2 Mitigations

- Apply rate-limiting
- Apply authentication and audit to recognise and block denial-of-service attempts as they occur

5.9 Model Repurposing

This attack is a concern relating to the misuse, misapplication or abuse of some public or published service, where a legitimate model is applied to some task which the creators did not intend and do not agree with. For example, a facial recognition system might have a front end applied to it which reapplies that system in an overtly discriminatory way, a fraud detection system might be misapplied to enable the more efficient generation of fraudulent transactions or a text generation system might be applied to the task of creating abusive or deceptive online content.

The primary defences against this are audit and authentication, or in the case of publication, some form of legal restriction, which attempts to prevent the misapplication of the technology.

5.9.1 Mitigations

- Ensure that the license agreement or terms and conditions of use prevent third parties from abusing the system in this way
- Apply rate-limiting
- Apply authentication and audit
- Consider configuring alerts on unusual clusters of heavy use of the system

6 Machine Learning Glossary

Artificial Intelligence Definitions from the Oxford English Dictionary:

- Artificial: "Made or produced by human beings rather than occuring naturally"
- Intelligence: "The ability to acquire and apply knowledge and skills"
- AI: "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages."

This definition of AI shifts in meaning over time ("normally requiring"), as more tasks are successfully automated. It's also worth noting that many practitioners prefer to reserve the term "AI" for the field of study, rather than describing some specific system as "an AI", since descriptions of this kind tend to impute characteristics to real-world systems that derive from the realm of popular fiction rather than demonstrable fact.

Batch A small, typically fixed-size subset of the training or validation set. For example, a batch might consist of (say) 64 individual inputs; 64 data records, 64 images, 64 recordings or similar.

Confidence Values See "Prediction Vector". Output of a classifier; a vector of class probabilities typically summing to 1.0 (if processed by, for example, a "softmax" layer or function), although the numbers may also be unconstrained, raw output from a network.

Convolution A convolutional neural network is used to detect the presence of some feature (e.g. an edge or an eye in an image) in the input data, without reference to its location. So for instance, a convolutional layer may detect the presence of an eye in an image, wherever it appears in the image.

Deep Learning The ability of a machine learning system to derive its own "Features" from input data, i.e. to discover high-level features or correlations in the data, such as eyes or snouts in pictures, or particular structures in more general data sets. In neural networks this ability may be connected to the depth of the neural network.

Epoch An iteration over the training set during training. Multiple iterations are normally necessary, and deciding how many iterations to use during training is a complex problem. Too few results in under-fitting, too many in overfitting.

Feature A property of the problem domain, embodied in input data. Examples might be the colour of an individual pixel, the presence of an eye or a snout in an image, the location of an "edge", or a particular structure in more general data.

Gradient Descent An iterative optimization algorithm used to minimise the value of a function by moving in the direction of steepest descent.

Hyperparameter A parameter of the algorithm itself, such as the "learning rate", which changes the way the algorithm itself operates. contrasted with a Parameter, which (in this context) is a variable associated with the algorithms in use that the system learns during Training.

L1 Loss Also called Mean Absolute Error, or MAE. A loss function measuring the sum of the absolute differences between expected and actual output.

L2 Loss Also called Mean Squared Error, or MSE. A loss function measuring the sum of the squared differences between expected and actual output.

L Infinity Loss Also called Maximal Loss. A loss function measuring the maximal difference between expected and actual output.

Latent Variable A property of some data that is inferred or derived from that data, rather than directly observed. In Machine Learning, this may correspond to a "feature" such as an eye, snow, or the presence of happiness in a body of text (in sentiment analysis).

Learning Rate A hyper-parameter which controls how much we are adjusting the weights of our network with respect to the loss gradient. In other words, the "rate" of gradient descent; also called "step size".

Logits The vector of raw predictions that a classification model generates, commonly passed to a normalisation function or the "softmax" function for output.

Loss Function A function, commonly used in ML training, that returns some function of the difference between desired and actual outputs. The objective of training is typically to minimise the value of the loss function.

Machine Learning Term used to describe a system that alters its behaviour based on input data, "learning" from the data it is shown.

Membership Inference An attack in which the attacker determines whether a specific element of data is present in the training data for a given model, by submitting inputs and observing the corresponding outputs. In order for the attack to be effective, the attacker must typically have, or be able to generate, some specific inputs. This is vaguely similar to a dictionary attack on a password hashing system. Contrast with "Model Inversion". See [Ateniese2015], [Shokri2016], [Salem2018].

Model A model defines the architecture of a machine learning system, in terms of the algorithms in use. The model also stores the result of training the system, i.e. the parameters that have been derived by the training process.

Model Inversion A form of attack whereby sensitive data is extracted from a trained model, by repeatedly submitting input data, observing the confidence level of the response, and repeating with slightly modified input data ("Hill Climbing"). The extracted data is typically the "essence" of some class, i.e. a combination of features and training inputs, which may not constitute an easily usable "data leak", although in some cases (e.g. fingerprints, signatures, facial recognition) the number of input training cases may be small enough for the extracted data to be usable [Fredrikson2015]. The extracted data is certainly useful in terms of representing an "ideal" (high confidence) instance of the class, and so may be used in the generation of adversarial inputs.

Tensor For ML purposes the "tensor" datatype can be thought of as a multidimensional array. A scalar is a 0-dimensional variable, a vector is a 1-dimensional array, a matrix is 2-dimensional, a tensor is n-dimensional. This definition, while both simple and useful, is some distance from any of the (several, differing) actual definitions, some of which are explained here: https://en.wikipedia.org/wiki/Tensor . The definition of Tensor as "multilinear map" is probably the most helpful in ML terms.

NumPy A numeric library for Python that implements multidimensional arrays and matrices, and a variety of other mathematical data types and functions useful in machine learning projects.

One Hot Encoding A way of encoding a "categorical variable" (e.g. country, car brand) into a set of boolean columns, one of which is "1" (hot) and the remainder "0" (cold). Representing categories this way, as opposed to representing each item with an ordinal value, for example, prevents the algorithm from, say, attempting to find the average of a "Volvo" and a "Skoda" and getting "Hyundai" for example.

Pandas A python library for data manipulation and analysis.

Parameter A variable associated with the algorithms in use that the system learns during Training. Contrast with Hyperparameter, which is a parameter of the algorithm itself, typically set by the developer, such as the "learning rate", which changes the way the algorithm itself operates.

Prediction Vector The output vector of an ML classification model, normally floating point values in the range 0.0 - 1.0, with one value per class, for all classes. The values capture the model's estimate of the probability that the input belongs to each class. These probabilities (aka "confidence values") typically sum to 1.0.

PyTorch A popular python machine learning library.

Scikit-Learn An open source machine learning platform for the Python programming language.

Stride, Striding Normally used in the context of a convolutional neural network, "striding" is shifting a filter by some number (greater than 1) of pixels in an image, or sampling a dataset at regularly spaced

points in some vector. The objective is to extract some feature by applying a filter to each portion of the image; "striding" can result in a more compact representation of the learned features.

TensorFlow A free, open source library for data flow programming tasks, including machine learning. It was created and is maintained by Google.

Test Set The set of data that is used to test the overall performance of the system once training has concluded. See also Training Set, Validation Set

Training The process of presenting data to a machine learning system, to enable it to change its behaviour, or "learn". Training typically requires large amounts of data, divided into a set of data that "teaches" the system (the "training set"), a set of data that the trained system is tested on, to validate the effectiveness of the training (the "validation set") and a third set of data used after training has finished to test the performance of the entire system (the "test set"). Data is typically presented to the system in "Batches", and one whole pass over the entire training set is termed one "Epoch".

Training Set The set of data used to directly train the system. This is the data that modifies the behaviour of the system, i.e. the data that it directly "learns" from. For example, in a neural network, this is the data that affects the weights of each "neuron". The Training, Test and Validation sets are typically chosen to have no members in common, in order to ensure that the trained system is able to "generalise".

Validation Set The set of data that is used during training to test the system's performance in the general case. This is typically chosen to have no members in common with the Training Set, to validate the extent to which the system is able to generalise.

7 Categorised References

The following is a list of references, categorised by attack type and ordered by year of publication, in order to express the evolution of attacks and defences and more easily locate additional references should they be desired.

ML Attack Taxonomies

Can machine learning be secure, Barreno, M. et al. 2006 [Barreno2006]

Adversarial machine learning, Huang, L. et al. 2011 [Huang2011]

Attacks and Defenses towards Machine Learning Based Systems, Yu, Y. et al. 2018 [Yu2018]

Law and Adversarial Machine Learning, Kumar, R. S. S. et al. 2018 [Kumar2018]

A Taxonomy and Terminology of Adversarial Machine Learning, Tabassi, E. et al. 2019 [Tabassi2019]

Failure modes in machine learning systems, Kumar, R. S. S. et al. 2019 [Kumar2019]

Adversarial Perturbation:

Adversarial learning, Lowd, D. et al. 2005 [Lowd2005b]

Good Word Attacks on Statistical Spam Filters., Lowd, D. et al. 2005 [Lowd2005a]

Adversarial machine learning, Huang, L. et al. 2011 [Huang2011]

Intriguing properties of neural networks, Szegedy, C. et al. 2013 [Szegedy2013]

Explaining and harnessing adversarial examples, Goodfellow, I. J. et al. 2014 [Goodfellow2014]

Adversarial examples in the physical world, Kurakin, A. et al. 2016 [Kurakin2016b]

Adversarial machine learning at scale, Kurakin, A. et al. 2016 [Kurakin2016a]

Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, Papernot, N. et al. 2016 [Papernot2016]

Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, Brendel, W. et al. 2017 [Brendel2017]

Exploring the Landscape of Spatial Robustness, Engstrom, L. et al. 2017 [Engstrom2017]

Generating adversarial malware examples for black-box attacks based on gan, Hu, W. et al. 2017 [Hu2017]

Practical Black-Box Attacks against Machine Learning, Papernot, N. et al. 2017 [Papernot2017]

Towards Deep Learning Models Resistant to Adversarial Attacks, Madry, A. et al. 2017 [Madry2017]

Vulnerability of deep reinforcement learning to policy induction attacks, Behzadan, V. et al. 2017 [Behzadan2017]

Intrinsic Geometric Vulnerability of High-Dimensional Artificial Intelligence, Bortolussi, L. et al. 2018 [Bortolussi2018]

Prior convictions: Black-box adversarial attacks with bandits and priors, Ilyas, A. et al. 2018 [Ilyas2018]

Robust physical-world attacks on deep learning visual classification, Eykholt, K. et al. 2018 [Eykholt2018]

Robustness May Be at Odds with Accuracy, Tsipras, D. et al. 2018 [Tsipras2018]

Synthesizing robust adversarial examples, Athalye, A. et al. 2018 [Athalye2018]

Wild patterns: Ten years after the rise of adversarial machine learning, Biggio, B. et al. 2018 [Biggio2018]

Adversarial Examples Are Not Bugs, They Are Features, Ilyas, A. et al. 2019 [Ilyas2019]

Adversarial Robustness as a Prior for Learned Representations, Engstrom, L. et al. 2019 [Engstrom2019]

Adversarial Robustness May Be at Odds With Simplicity, Nakkiran, P. 2019 [Nakkiran2019]

Characterizing and evaluating adversarial examples for Offline Handwritten Signature Verification, Hafemann, L. G. et al. 2019 [Hafemann2019]

Curls & whey: Boosting black-box adversarial attacks, Shi, Y. et al. 2019 [Shi2019]

Memguard: Defending against black-box membership inference attacks via adversarial examples, Jia, J. et al. 2019 [Jia2019]

One Pixel Attack for Fooling Deep Neural Networks, Su, J. et al. 2019 [Su2019]

Procedural Noise Adversarial Examples for Black-Box Attacks on Deep Convolutional Networks, Co, K. T. et al. 2019 [Co2019]

Simple black-box adversarial attacks, Guo, C. et al. 2019 [Guo2019]

Explaining Vulnerabilities to Adversarial Machine Learning through Visual Analytics., Ma, Y. et al. 2020 [Ma2020]

Non-robust Features through the Lens of Universal Perturbations, Park, S. M. et al. 2020 [Park2020]

On Adaptive Attacks to Adversarial Example Defenses, Tramer, F. et al. 2020 [Tramer2020]

Semantic Adversarial Perturbations using Learnt Representations, Dunn, I. et al. 2020 [Dunn2020]

Transferability of adversarial examples to attack cloud-based image classifier service, Goodman, D. 2020 [Goodman2020]

Membership Inference:

Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers, Ateniese, G. et al. 2015 [Ateniese2015]

Membership Inference Attacks against Machine Learning Models, Shokri, R. et al. 2016 [Shokri2016]

Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting, Yeom, S. et al. 2017 [Yeom2017]

Auditing Data Provenance in Text-Generation Models, Song, C. et al. 2018 [Song2018]

Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning, Nasr, M. et al. 2018 [Nasr2018]

Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models, Salem, A. et al. 2018 [Salem2018]

Towards Demystifying Membership Inference Attacks, Truex, S. et al. 2018 [Truex2018]

Effects of Differential Privacy and Data Skewness on Membership Inference Vulnerability, Truex, S. et al. 2019 [Truex2019]

Membership Inference Attacks on Sequence-to-Sequence Models: Is My Data In Your Machine Translation System, Hisamoto, S. et al. 2019 [Hisamoto2019a]

Memguard: Defending against black-box membership inference attacks via adversarial examples, Jia, J. et al. 2019 [Jia2019]

Privacy Risks of Securing Machine Learning Models against Adversarial Examples, Song, L. et al. 2019 [Song2019b]

Reconciling Utility and Membership Privacy via Knowledge Distillation, Shejwalkar, V. et al. 2019 [Shejwalkar2019]

White-box vs black-box: Bayes optimal strategies for membership inference, Sablayrolles, A. et al. 2019 [Sablayrolles2019]

Differentially Private Learning Does Not Bound Membership Inference, Humphries, T. et al. 2020 [Humphries2020]

Label-Only Membership Inference Attacks, Choquette-Choo, C. A. et al. 2020 [Choquette-Choo2020]

Membership Inference Attacks on Sequence-to-Sequence Models: Is My Data In Your Machine Translation System, Hisamoto, S. et al. 2020 [Hisamoto2020]

Modelling and Quantifying Membership Information Leakage in Machine Learning, Farokhi, F. et al. 2020 [Farokhi2020]

Privacy in Deep Learning: A Survey, Mireshghallah, F. et al. 2020 [Mireshghallah2020]

Revisiting Membership Inference Under Realistic Assumptions, Jayaraman, B. et al. 2020 [Jayaraman2020]

Towards the Infeasibility of Membership Inference on Deep Models, Rezaei, S. et al. 2020 [Rezaei2020]

Membership Inference Attacks From First Principles, Carlini, N. et al. 2021 [Carlini2021]

Practical Blind Membership Inference Attack via Differential Comparisons, Hui, B. et al. 2021 [Hui2021]

Model Inversion:

Proceedings of the. Systems Administration Conference, Fredrikson, M. et al. 2003 [Fredrikson2003]

Vulnerabilities in biometric encryption systems, Adler, A. 2005 [Adler2005]

Bayesian hill-climbing attack and its application to signature verification, Galbally, J. et al. 2007 [Galbally2007]

On the vulnerability of face verification systems to hill-climbing attacks, Galbally, J. et al. 2010 [Galbally2010]

Hill-climbing attack based on the uphill simplex algorithm and its application to signature verification, Gomez-Barrero, M. et al. 2011 [Gomez-Barrero2011]

Model inversion attacks that exploit confidence information and basic countermeasures, Fredrikson, M. et al. 2015 [Fredrikson2015]

Regression model fitting under differential privacy and model inversion attack, Wang, Y. et al. 2015 [Wang2015]

A methodology for formalizing model-inversion attacks, Wu, X. et al. 2016 [Wu2016]

Trojaning attack on neural networks, Liu, Y. et al. 2017 [Liu2017]

Algorithms that remember: model inversion attacks and data protection law, Veale, M. et al. 2018 [Veale2018]

Membership Inference Attack against Differentially Private Deep Learning Model., Rahman, M. A. et al. 2018 [Rahman2018]

Model inversion attacks for online prediction systems: Without knowledge of non-sensitive attributes, Hidano, S. et al. 2018 [Hidano2018]

Adversarial neural network inversion via auxiliary knowledge alignment, Yang, Z. et al. 2019 [Yang2019b]

Differentially private model publishing for deep learning, Yu, L. et al. 2019 [Yu2019]

GAMIN: An Adversarial Approach to Black-Box Model Inversion, Aïvodji, U. et al. 2019 [Aivodji2019]

Membership model inversion attacks for deep networks, Basu, S. et al. 2019 [Basu2019]

MLPrivacyGuard, Alves, T. A. O. et al. 2019 [Alves2019]

Neural network inversion in adversarial setting via background knowledge alignment, Yang, Z. et al. 2019 [Yang2019a]

Defending Against Model Inversion Attacks on Neural Networks, Araujo, F. et al. 2020 [Araujo2020]

Privacy in Deep Learning: A Survey, Mireshghallah, F. et al. 2020 [Mireshghallah2020]

Reducing Risk of Model Inversion Using Privacy-Guided Training, Goldsteen, A. et al. 2020 [Goldsteen2020]

The secret revealer: generative model-inversion attacks against deep neural networks, Zhang, Y. et al. 2020 [Zhang2020]

Model Stealing:

Adversarial learning, Lowd, D. et al. 2005 [Lowd2005b]

Stealing machine learning models via prediction apis, Tramèr, F. et al. 2016 [Tramer2016]

Model Extraction Warning in MLaaS Paradigm, Kesarwani, M. et al. 2017 [Kesarwani2017]

Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data, Correia-Silva, J. R. et al. 2018 [Correia-Silva2018]

Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations, Lee, T. et al. 2018 [Lee2018]

Exploring Connections Between Active Learning and Model Extraction, Chandrasekaran, V. et al. 2018 [Chandrasekaran2018]

Have You Stolen My Model? Evasion Attacks Against Deep Neural Network Watermarking Techniques, Hitaj, D. et al. 2018 [Hitaj2018]

Knockoff Nets: Stealing Functionality of Black-Box Models, Orekondy, T. et al. 2018 [Orekondy2018]

Stealing Neural Networks via Timing Side Channels, Duddu, V. et al. 2018 [Duddu2018]

Defending Against Model Stealing Attacks with Adaptive Misinformation, Kariyappa, S. et al. 2019 [Kariyappa2019]

Extraction of Complex DNN Models: Real Threat or Boogeyman?, Atli, B. G. et al. 2019 [Atli2019]

High Accuracy and High Fidelity Extraction of Neural Networks, Jagielski, M. et al. 2019 [Jagielski2019]

MimosaNet: An Unrobust Neural Network Preventing Model Stealing, Szentannai, K. et al. 2019 [Szentannai2019]

Model Weight Theft With Just Noise Inputs: The Curious Case of the Petulant Attacker, Roberts, N. et al. 2019 [Roberts2019]

Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks, Orekondy, T. et al. 2019 [Orekondy2019]

Stolen Memories: Leveraging Model Memorization for Calibrated White-Box Membership Inference, Leino, K. et al. 2019 [Leino2019]

Cryptanalytic Extraction of Neural Network Models, Carlini, N. et al. 2020 [Carlini2020a]

Hermes Attack: Steal DNN Models with Lossless Inference Accuracy, Zhu, Y. et al. 2020 [Zhu2020]

Perturbing Inputs to Prevent Model Stealing, Grana, J. 2020 [Grana2020]

Privacy in Deep Learning: A Survey, Mireshghallah, F. et al. 2020 [Mireshghallah2020]

Stealing Deep Reinforcement Learning Models for Fun and Profit, Chen, K. et al. 2020 [Chen2020]

Large Language Models

Very Deep Convolutional Networks for Text Classification, Conneau, A. et al. 2016 [Conneau2016]

Ethical Challenges in Data-Driven Dialogue Systems, Henderson, P. et al. 2017 [Henderson2017]

HotFlip: White-Box Adversarial Examples for Text Classification, Ebrahimi, J. et al. 2017 [Ebrahimi2017]

Auditing Data Provenance in Text-Generation Models, Song, C. et al. 2018 [Song2018]

The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks, Carlini, N. et al. 2018 [Carlini2018]

BLACK BOX ATTACKS ON TRANSFORMER LANGUAGE MODELS, Misra, V. 2019 [Misra2019]

Universal Adversarial Perturbation for Text Classification, Gao, H. et al. 2019 [Gao2019a]

Evaluation of Inference Attack Models for Deep Learning on Medical Data, Wu, M. et al. 2020 [Wu2020]

Extracting Training Data from Large Language Models, Carlini, N. et al. 2020 [Carlini2020b]

Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system, Hisamoto, S. et al. 2020 [Hisamoto2020a]

Understanding Unintended Memorization in Federated Learning, Thakkar, O. et al. 2020 [Thakkar2020]

Bad Characters: Imperceptible NLP Attacks, Boucher, N. et al. 2021 [Boucher2021]

Does BERT Pretrained on Clinical Notes Reveal Sensitive Data, Lehman, E. et al. 2021 [Lehman2021]

Membership Inference Attack Susceptibility of Clinical Language Models, Jagannatha, A. et al. 2021 [Jagannatha2021]

Membership Inference on Word Embedding and Beyond, Mahloujifar, S. et al. 2021 [Mahloujifar2021]

TAG: Gradient Attack on Transformer-based Language Models, Deng, J. et al. 2021 [Deng2021]

Training Data Leakage Analysis in Language Models, Inan, H. A. et al. 2021 [Inan2021]

Backdoor Attacks:

Targeted backdoor attacks on deep learning systems using data poisoning, Chen, X. et al. 2017 [Chen2017]

Trojaning attack on neural networks, Liu, Y. et al. 2017 [Liu2017]

Potrojan: powerful neural-level trojan designs in deep learning models, Zou, M. et al. 2018 [Zou2018]

Auditing data provenance in text-generation models, Song, C. et al. 2019 [Song2019a]

DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks., Chen, H. et al. 2019 [Chen2019]

Detecting AI Trojans Using Meta Neural Analysis, Xu, X. et al. 2019 [Xu2019]

Memory trojan attack on neural network accelerators, Zhao, Y. et al. 2019 [Zhao2019]

On the Robustness of the Backdoor-based Watermarking in Deep Neural Networks, Shafieinejad, M. et al. 2019 [Shafieinejad2019]

Strip: A defence against trojan attacks on deep neural networks, Gao, Y. et al. 2019 [Gao2019b]

Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses, Goldblum, M. et al. 2020 [Goldblum2020]

Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks, Schwarzschild, A. et al. 2020 [Schwarzschild2020]

Radioactive data: tracing through training, Sablayrolles, A. et al. 2020 [Sablayrolles2020]

Master Prints:

DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution, Bontrager, P. et al. 2017 [Bontrager2017]

Masterprint: Exploring the vulnerability of partial fingerprint-based authentication systems, Roy, A. et al. 2017 [Roy2017]

A Novel Approach for Partial Fingerprint Identification to Mitigate MasterPrint Generation, Joshi, M. et al. 2019 [Joshi2019]

Privacy in Recommendation Systems:

When being weak is brave: Privacy in recommender systems, Ramakrishnan, N. et al. 2001 [Ramakrishnan2001]

Do you trust your recommendations? An exploration of security and privacy issues in recommender systems, Shyong, K. et al. 2006 [Shyong2006]

Differentially private recommender systems: Building privacy into the netflix prize contenders, McSherry, F. et al. 2009 [McSherry2009]

Making Decisions about Privacy, Knijnenburg, B. P. et al. 2013 [Knijnenburg2013]

Privacy in recommender systems, Jeckmans, A. J. P. et al. 2013 [Jeckmans2013]

Differential privacy for social science inference, D'Orazio, V. et al. 2015 [DOrazio2015]

Minimax filter: learning to preserve privacy from inference attacks, Hamm, J. 2017 [Hamm2017]

Exploiting unintended feature leakage in collaborative learning, Melis, L. et al. 2019 [Melis2019]

Sponge Attacks:

Sponge Examples: Energy-Latency Attacks on Neural Networks, Shumailov, I. et al. 2020 [Shumailov2020]

Relevant Books

Behind Deep Blue, Hsu, F.-H. 2004 [Hsu2004]

One Jump Ahead, Schaeffer, J. 2008 [Schaeffer2008]

Computer Chess Compendium, LEVY, D. 2013 [LEVY2013]

Artificial Intelligence: A Modern Approach, Global Edition, Russell, S. et al. 2016 [Russell2016]

Deep Learning, Goodfellow, I. et al. 2016 [Goodfellow2016]

Foundations of Statistical Natural Language Processing, Manning, C. D. et al. 2016 [Manning2016]

Weapons of Math Destruction, O'Neil, C. 2016 [ONeil2016]

Neural Network Methods in Natural Language Processing, Goldberg, Y. 2017 [Goldberg2017]

Deep Learning in Natural Language Processing, Deng, L. et al. 2018 [Deng2018]

Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch, Subramanian, V. 2018 [Subramanian2018]

Hello World, Fry, H. 2018 [Fry2018]

Machine Learning with Python Cookbook, Albon, C. 2018 [Albon2018]

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Géron, A. 2019 [Geron2019]

Introduction to Natural Language Processing, Eisenstein, J. 2019 [Eisenstein2019]

Natural Language Processing with PyTorch, Rao, D. et al. 2019 [Rao2019]

Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications, Pointer, I. 2019 [Pointer2019]

The Hundred-page Machine Learning Book, Burkov, A. 2019 [Burkov2019]

You Look Like a Thing and I Love You, Shane, J. 2019 [Shane2019]

Transformers for Natural Language Processing, Rothman, D. 2021 [Rothman2021]

ML Fundamentals and Background

Computing Machinery and Intelligence, Turing, A. M. 1950 [TURING1950]

Approximation by superpositions of a sigmoidal function, Cybenko, G. 1989 [Cybenko1989]

The functional organization of local circuits in visual cortex: insights from the study of tree shrew striate cortex., Fitzpatrick, D. 1996 [Fitzpatrick1996]

Adadelta: an adaptive learning rate method, Zeiler, M. D. 2012 [Zeiler2012]

Improving neural networks by preventing co-adaptation of feature detectors, Hinton, G. E. et al. 2012 [Hinton2012]

Speech Recognition with Deep Recurrent Neural Networks, Graves, A. et al. 2013 [Graves2013]

Distilling the Knowledge in a Neural Network, Hinton, G. et al. 2015 [Hinton2015]

Hidden technical debt in machine learning systems, Sculley, D. et al. 2015 [Sculley2015]

Very Deep Convolutional Networks for Text Classification, Conneau, A. et al. 2016 [Conneau2016]

Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch, Subramanian, V. 2018 [Subramanian2018]

Intrinsic Geometric Vulnerability of High-Dimensional Artificial Intelligence, Bortolussi, L. et al. 2018 [Bortolussi2018]

Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks, Neyshabur, B. et al. 2018 [Neyshabur2018]

Winner's curse? On pace, progress, and empirical rigor, Sculley, D. et al. 2018 [Sculley2018]

Adversarial Robustness as a Prior for Learned Representations, Engstrom, L. et al. 2019 [Engstrom2019]

Deep learning for symbolic mathematics, Lample, G. et al. 2019 [Lample2019]

Nanophotonic media for artificial neural inference, Khoram, E. et al. 2019 [Khoram2019]

Learning Texture Transformer Network for Image Super-Resolution, Yang, F. et al. 2020 [Yang2020]

The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities., Lehman, J. et al. 2020 [Lehman2020]

8 Image Credits

"Close-up of a Bernese Mountain Dog head", Ocmey http://fr.wikipedia.org/wiki/Utilisateur:Ocmey, CC BY-SA 2.5 https://creativecommons.org/licenses/by-sa/2.5, via Wikimedia Commons

"Hamster", Ricky Kharawala @sweetmangostudios, via Unsplash.com

"Person-22", Generated image from via https://this-person-does-not-exist.com/en

"Photograph of Nicholas Brendon in Italy", by Nicholas Brendon, NicholasBrendon, CC BY 4.0 https://creativecommons.org/licenses/by/4.0, via Wikimedia Commons

9 References

[Adler2005] Adler, A. 2005. Vulnerabilities in biometric encryption systems

[Aivodji2019] Aïvodji, U., Gambs, S., and Ther, T. 2019. GAMIN: An Adversarial Approach to Black-Box Model Inversion

[Albon2018] Albon, C. 2018 Machine Learning with Python Cookbook. "O'Reilly Media, Inc.".

[Alves2019] Alves, T. A. O., França, F. M. G., and Kundu, S. 2019. MLPrivacyGuard

[Araujo2020] Araujo, F., Zhang, J., Taylor, T. et al. 2020. Defending Against Model Inversion Attacks on Neural Networks

[Arnaldo2017] Arnaldo, I., Cuesta-Infante, A., Arun, A. et al. 2017. Learning representations for log data in cybersecurity

[Asnani2021] Asnani, V., Yin, X., Hassner, T. et al. 2021. Reverse Engineering of Generative Models: Inferring Model Hyperparameters from Generated Images

[Ateniese2015] Ateniese, G., Mancini, L. V., Spognardi, A. et al. 2015. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers

[Athalye2018] Athalye, A., Engstrom, L., and Ilyas. . . , A. 2018. Synthesizing robust adversarial examples

[Atli2019] Atli, B. G., Szyller, S., Juuti, M. et al. 2019. Extraction of Complex DNN Models: Real Threat or Boogeyman?

[Barreno2006] Barreno, M., Nelson, B., Sears, R. et al. 2006. Can machine learning be secure

[Basu2019] Basu, S., Izmailov, R., and Mesterharm, C. 2019. Membership model inversion attacks for deep networks

[Beck2018] Beck, M. W. 2018. NeuralNetTools: Visualization and Analysis Tools for Neural Networks.

[Behzadan2017] Behzadan, V. and Munir, A. 2017. Vulnerability of deep reinforcement learning to policy induction attacks

[Bender2021] Bender, E. M., Gebru, T., McMillan-Major, A. et al. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?

[Biggio2018] Biggio, B. and Roli, F. 2018. Wild patterns: Ten years after the rise of adversarial machine learning

[Bischof1992] Bischof, H., Pinz, A., and Kropatsch, W. G. 1992. Visualization methods for neural networks

[Bontrager2017] Bontrager, P., Roy, A., Togelius, J. et al. 2017. DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution

[Bortolussi2018] Bortolussi, L. and Sanguinetti, G. 2018. Intrinsic Geometric Vulnerability of High-Dimensional Artificial Intelligence

[Boucher2021] Boucher, N., Shumailov, I., Anderson, R. et al. 2021. Bad Characters: Imperceptible NLP Attacks

[Brendel2017] Brendel, W., Rauber, J., and Bethge, M. 2017. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models

[Burkov2019] Burkov, A. 2019 The Hundred-page Machine Learning Book.

[Calandrino2011] Calandrino, J. A., Kilzer, A., and Narayanan. . . , A. 2011. " You might also like:" Privacy risks of collaborative filtering

[Carlini2018] Carlini, N., Liu, C., Erlingsson, Ú. et al. 2018. The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks

[Carlini2020] Carlini, N., Jagielski, M., and Mironov, I. 2020. Cryptanalytic Extraction of Neural Network Models

[Carlini2020a] Carlini, N., Jagielski, M., and Mironov, I. 2020. Cryptanalytic Extraction of Neural Network Models

[Carlini2020b] Carlini, N., Tramer, F., Wallace, E. et al. 2020. Extracting Training Data from Large Language Models

[Carlini2021] Carlini, N., Chien, S., Nasr, M. et al. 2021. Membership Inference Attacks From First Principles

[Chandrasekaran2018] Chandrasekaran, V., Chaudhuri, K., Giacomelli, I. et al. 2018. Exploring Connections Between Active Learning and Model Extraction

[Chen2017] Chen, X., Liu, C., Li, B. et al. 2017. Targeted backdoor attacks on deep learning systems using data poisoning

[Chen2019] Chen, H., Fu, C., Zhao, J. et al. 2019. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks.

[Chen2020] Chen, K., Zhang, T., Xie, X. et al. 2020. Stealing Deep Reinforcement Learning Models for Fun and Profit

[Choquette-Choo2020] Choquette-Choo, C. A., Tramer, F., Carlini, N. et al. 2020. Label-Only Membership Inference Attacks

[Co2019] Co, K. T., Muñoz-González, L., and Maupeou. . . , S. D. 2019. Procedural Noise Adversarial Examples for Black-Box Attacks on Deep Convolutional Networks

[Conneau2016] Conneau, A., Schwenk, H., Barrault, L. et al. 2016. Very Deep Convolutional Networks for Text Classification

[Correia-Silva2018] Correia-Silva, J. R., Berriel, R. F., Badue, C. et al. 2018. Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data

[Cybenko1989] Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function

[Deng2018] Deng, L. and Liu, Y. 2018 Deep Learning in Natural Language Processing. Springer.

[Deng2021] Deng, J., Wang, Y., Li, J. et al. 2021. TAG: Gradient Attack on Transformer-based Language Models

[DOrazio2015] D'Orazio, V., Honaker, J., and King, G. 2015. Differential privacy for social science inference

[Duddu2018] Duddu, V., Samanta, D., Rao, D. V. et al. 2018. Stealing Neural Networks via Timing Side Channels

[Dunn2020] Dunn, I., Melham, T., and Kroening, D. 2020. Semantic Adversarial Perturbations using Learnt Representations

[Ebrahimi2017] Ebrahimi, J., Rao, A., Lowd, D. et al. 2017. HotFlip: White-Box Adversarial Examples for Text Classification

[Eisenstein2019] Eisenstein, J. 2019 Introduction to Natural Language Processing. MIT Press.

[Engstrom2017] Engstrom, L., Tran, B., Tsipras, D. et al. 2017. Exploring the Landscape of Spatial Robustness

[Engstrom2019] Engstrom, L., Ilyas, A., Santurkar, S. et al. 2019. Adversarial Robustness as a Prior for Learned Representations

[Eykholt2018] Eykholt, K., Evtimov, I., and Fernandes. . . , E. 2018. Robust physical-world attacks on deep learning visual classification

[Farokhi2020] Farokhi, F. and Kaafar, M. A. 2020. Modelling and Quantifying Membership Information Leakage in Machine Learning

[Fitzpatrick1996] Fitzpatrick, D. 1996. The functional organization of local circuits in visual cortex: insights from the study of tree shrew striate cortex.

Fredrikson, M., Lantz, E., Jha, S. et al. 2003 Proceedings of the. Systems Administration Conference.

[Fredrikson2015] Fredrikson, M., Jha, S., and Ristenpart, T. 2015. Model inversion attacks that exploit confidence information and basic countermeasures

[Friedman2015] Friedman, A., Knijnenburg, B. P., Vanhecke, K. et al. 2015 Privacy Aspects of Recommender Systems. In Recommender Systems Handbook, Springer US.

[Fry2018] Fry, H. 2018 Hello World. Random House.

[Galbally2007] Galbally, J., Fierrez, J., and Ortega-Garcia, J. 2007. Bayesian hill-climbing attack and its application to signature verification

[Galbally2010] Galbally, J., McCool, C., Fierrez, J. et al. 2010. On the vulnerability of face verification systems to hill-climbing attacks

[Gao2019a] Gao, H. and Oates, T. 2019. Universal Adversarial Perturbation for Text Classification

[Gao2019b] Gao, Y., Xu, C., Wang, D. et al. 2019. Strip: A defence against trojan attacks on deep neural networks

[Géron2019] Géron, A. 2019 Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.

[Goldberg2017] Goldberg, Y. 2017 Neural Network Methods in Natural Language Processing. Morgan & Claypool Publishers.

[Goldblum2020] Goldblum, M., Tsipras, D., Xie, C. et al. 2020. Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses

[Goldsteen2020] Goldsteen, A., Ezov, G., and Farkash, A. 2020. Reducing Risk of Model Inversion Using Privacy-Guided Training

[Gomez-Barrero2011] Gomez-Barrero, M., Galbally, J., and Fierrez. . . , J. 2011. Hill-climbing attack based on the uphill simplex algorithm and its application to signature verification

[Goodfellow2014] Goodfellow, I. J., Shlens, J., and Szegedy, C. 2014. Explaining and harnessing adversarial examples

Goodfellow, I., Bengio, Y., and Courville, A. 2016 Deep Learning. MIT Press.

[Goodman2020] Goodman, D. 2020. Transferability of adversarial examples to attack cloud-based image classifier service

[Grana2020] Grana, J. 2020. Perturbing Inputs to Prevent Model Stealing

[Graves2013] Graves, A., Mohamed, A.-r., and Hinton, G. 2013. Speech Recognition with Deep Recurrent Neural Networks

[Guo2019] Guo, C., Gardner, J. R., You, Y. et al. 2019. Simple black-box adversarial attacks

[Hafemann2017] Hafemann, L. G., Sabourin, R., and Oliveira, L. S. 2017. Learning features for offline handwritten signature verification using deep convolutional neural networks

[Hafemann2019] Hafemann, L. G. and Sabourin. . . , R. 2019. Characterizing and evaluating adversarial examples for Offline Handwritten Signature Verification

[Hamm2017] Hamm, J. 2017. Minimax filter: learning to preserve privacy from inference attacks

[Henderson2017] Henderson, P., Sinha, K., Angelard-Gontier, N. et al. 2017. Ethical Challenges in Data-Driven Dialogue Systems

[Hidano2018] Hidano, S., Murakami, T., Katsumata, S. et al. 2018. Model inversion attacks for online prediction systems: Without knowledge of non-sensitive attributes

[Hinton2012] Hinton, G. E., Srivastava, N., Krizhevsky, A. et al. 2012. Improving neural networks by preventing co-adaptation of feature detectors

[Hinton2015] Hinton, G., Vinyals, O., and Dean, J. 2015. Distilling the Knowledge in a Neural Network

[Hisamoto2019a] Hisamoto, S., Post, M., and Duh, K. 2019. Membership Inference Attacks on Sequence-to-Sequence Models: Is My Data In Your Machine Translation System

[Hisamoto2020] Hisamoto, S., Post, M., and Duh, K. 2020. Membership Inference Attacks on Sequence-to-Sequence Models: Is My Data In Your Machine Translation System

[Hisamoto2020a] Hisamoto, S., Post, M., and Duh, K. 2020. Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system

[Hitaj2018] Hitaj, D. and Mancini, L. V. 2018. Have You Stolen My Model? Evasion Attacks Against Deep Neural Network Watermarking Techniques

[Hsu2004] Hsu, F.-H. 2004 Behind Deep Blue. Princeton University Press.

[Hu2017] Hu, W. and Tan, Y. 2017. Generating adversarial malware examples for black-box attacks based on gan

[Huang2011] Huang, L., Joseph, A. D., and Nelson. . . , B. 2011. Adversarial machine learning

[Hui2021] Hui, B., Yang, Y., Yuan, H. et al. 2021. Practical Blind Membership Inference Attack via Differential Comparisons

[Humphries2020] Humphries, T., Rafuse, M., Tulloch, L. et al. 2020. Differentially Private Learning Does Not Bound Membership Inference

[Ilyas2018] Ilyas, A., Engstrom, L., and Madry, A. 2018. Prior convictions: Black-box adversarial attacks with bandits and priors

[Ilyas2019] Ilyas, A., Santurkar, S., Tsipras, D. et al. 2019. Adversarial Examples Are Not Bugs, They Are Features

[Inan2021] Inan, H. A., Ramadan, O., Wutschitz, L. et al. 2021. Training Data Leakage Analysis in Language Models

[Jagannatha2021] Jagannatha, A., Rawat, B. P. S., and Yu, H. 2021. Membership Inference Attack Susceptibility of Clinical Language Models

[Jagannatha2021a] Jagannatha, A., Rawat, B. P. S., and Yu, H. 2021. Membership Inference Attack Susceptibility of Clinical Language Models

[Jagielski2019] Jagielski, M., Carlini, N., Berthelot, D. et al. 2019. High Accuracy and High Fidelity Extraction of Neural Networks

[Jayaraman2020] Jayaraman, B., Wang, L., Evans, D. et al. 2020. Revisiting Membership Inference Under Realistic Assumptions

[Jeckmans2013] Jeckmans, A. J. P., Beye, M., Erkin, Z. et al. 2013. Privacy in recommender systems

[Jia2019] Jia, J., Salem, A., Backes, M. et al. 2019. Memguard: Defending against black-box membership inference attacks via adversarial examples

[Joshi2019] Joshi, M., Mazumdar, B., and Dey, S. 2019. A Novel Approach for Partial Fingerprint Identification to Mitigate MasterPrint Generation

[Kariyappa2019] Kariyappa, S. and Qureshi, M. K. 2019. Defending Against Model Stealing Attacks with Adaptive Misinformation

[Kesarwani2017] Kesarwani, M., Mukhoty, B., Arya, V. et al. 2017. Model Extraction Warning in MLaaS Paradigm

[Khoram2019] Khoram, E., Chen, A., Liu, D. et al. 2019. Nanophotonic media for artificial neural inference

[Knijnenburg2013] Knijnenburg, B. P. and Kobsa, A. 2013. Making Decisions about Privacy

[Kumar2018] Kumar, R. S. S., O'Brien, D. R., Albert, K. et al. 2018. Law and Adversarial Machine Learning

[Kumar2019] Kumar, R. S. S., Brien, D. O., Albert, K. et al. 2019. Failure modes in machine learning systems

[Kurakin2016a] Kurakin, A., Goodfellow, I., and Bengio, S. 2016. Adversarial machine learning at scale

[Kurakin2016b] Kurakin, A., Goodfellow, I., and Bengio, S. 2016. Adversarial examples in the physical world

[Lample2019] Lample, G. and Charton, F. 2019. Deep learning for symbolic mathematics

[Lee2018] Lee, T., Edwards, B., Molloy, I. et al. 2018. Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations

[Lehman2020] Lehman, J., Clune, J., Misevic, D. et al. 2020. The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities.

[Lehman2021] Lehman, E., Jain, S., Pichotta, K. et al. 2021. Does BERT Pretrained on Clinical Notes Reveal Sensitive Data

[Leino2019] Leino, K. and Fredrikson, M. 2019. Stolen Memories: Leveraging Model Memorization for Calibrated White-Box Membership Inference

[Levy2013] Levy, D. 2013 Computer Chess Compendium. Springer Science & Business Media.

[Liu2017] Liu, Y., Ma, S., Aafer, Y. et al. 2017. Trojaning attack on neural networks

[Lorenzo-Trueba2018] Lorenzo-Trueba, J., Fang, F., Wang, X. et al. 2018. Can we steal your vocal identity from the Internet?: Initial investigation of cloning Obama's voice using GAN, WaveNet and low-quality found data

[Lowd2005a] Lowd, D. and Meek, C. 2005. Good Word Attacks on Statistical Spam Filters.

[Lowd2005b] Lowd, D. and Meek, C. 2005. Adversarial learning

[Ma2020] Ma, Y., Xie, T., Li, J. et al. 2020. Explaining Vulnerabilities to Adversarial Machine Learning through Visual Analytics.

[Madry2017] Madry, A., Makelov, A., Schmidt, L. et al. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks

[Mahloujifar2021] Mahloujifar, S., Inan, H. A., Chase, M. et al. 2021. Membership Inference on Word Embedding and Beyond

[Manning2016] Manning, C. D. and Schtze, H. 2016 Foundations of Statistical Natural Language Processing.

[McQueen1976] McQueen, E. G. 1976. New Zealand Committee on Adverse Drug Reactions: eleventh annual report 1976.

[McSherry2009] McSherry, F. and Mironov, I. 2009. Differentially private recommender systems: Building privacy into the netflix prize contenders

[Melis2019] Melis, L., Song, C., and Cristofaro. . . , E. D. 2019. Exploiting unintended feature leakage in collaborative learning

[Mireshghallah2020] Mireshghallah, F., Taram, M., Vepakomma, P. et al. 2020. Privacy in Deep Learning: A Survey

[Misra2019] Misra, V. 2019. BLACK BOX ATTACKS ON TRANSFORMER LANGUAGE MODELS

[Munroe2007] Munroe, R. 2007. XKCD: Exploits of a Mom

[Nakkiran2019] Nakkiran, P. 2019. Adversarial Robustness May Be at Odds With Simplicity

[Nasr2018] Nasr, M., Shokri, R., and Houmansadr, A. 2018. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning

[Neyshabur2018] Neyshabur, B., Li, Z., Bhojanapalli, S. et al. 2018. Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks

[Ogura2019] Ogura, M. 2019. FlashTorch, a Neural Network Visualisation Toolkit

[Olah2017] Olah, C., Mordvintsev, A., and Schubert, L. 2017. Feature Visualization

[O'Neil2016] O'Neil, C. 2016 Weapons of Math Destruction. Penguin UK.

[Orekondy2018] Orekondy, T., Schiele, B., and Fritz, M. 2018. Knockoff Nets: Stealing Functionality of Black-Box Models

[Orekondy2019] Orekondy, T., Schiele, B., and Fritz, M. 2019. Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks

[Papernot2016] Papernot, N., McDaniel, P., and Goodfellow, I. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples

[Papernot2017] Papernot, N., McDaniel, P., Goodfellow, I. et al. 2017. Practical Black-Box Attacks against Machine Learning

[Park2020] Park, S. M., Wei, K.-A., Xiao, K. Y. et al. 2020. Non-robust Features through the Lens of Universal Perturbations

[Parry1976] Parry, W. H., Martorano, F., and Cotton, E. K. 1976. Management of life-threatening asthma with intravenous isoproterenol infusions.

[Pointer2019] Pointer, I. 2019 Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications. O'Reilly Media.

[Rahman2018] Rahman, M. A., Rahman, T., Laganière, R. et al. 2018. Membership Inference Attack against Differentially Private Deep Learning Model.

[Ramakrishnan2001] Ramakrishnan, N., Keller, B. J., Mirza, B. J. et al. 2001. When being weak is brave: Privacy in recommender systems

[Rao2019] Rao, D. and McMahan, B. 2019 Natural Language Processing with PyTorch. "O'Reilly Media, Inc.".

[Rezaei2020] Rezaei, S. and Liu, X. 2020. Towards the Infeasibility of Membership Inference on Deep Models

[Roberts2019] Roberts, N., Prabhu, V. U., and McAteer, M. 2019. Model Weight Theft With Just Noise Inputs: The Curious Case of the Petulant Attacker

[Rothman2021] Rothman, D. 2021 Transformers for Natural Language Processing. Packt Publishing Ltd.

[Roy2017] Roy, A., Memon, N., and Ross, A. 2017. Masterprint: Exploring the vulnerability of partial fingerprint-based authentication systems

[Russell2016] Russell, S. and Norvig, P. 2016 Artificial Intelligence: A Modern Approach, Global Edition. Pearson Higher Ed.

[Sablayrolles2019] Sablayrolles, A., Douze, M., Schmid, C. et al. 2019. White-box vs black-box: Bayes optimal strategies for membership inference

[Sablayrolles2020] Sablayrolles, A., Douze, M., and Schmid. . . , C. 2020. Radioactive data: tracing through training

[Salem2018] Salem, A., Zhang, Y., Humbert, M. et al. 2018. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models

[Samek2016] Samek, W., Binder, A., and Montavon. . . , G. 2016. Evaluating the visualization of what a deep neural network has learned

[Schaeffer2008] Schaeffer, J. 2008 One Jump Ahead. Springer Science & Business Media.

[Schwarzschild2020] Schwarzschild, A., Goldblum, M., Gupta, A. et al. 2020. Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks

[Sculley2015] Sculley, D., Holt, G., Golovin, D. et al. 2015. Hidden technical debt in machine learning systems

[Sculley2018] Sculley, D., Snoek, J., Wiltschko, A. et al. 2018. Winner's curse? On pace, progress, and empirical rigor

[Shafieinejad2019] Shafieinejad, M., Wang, J., Lukas, N. et al. 2019. On the Robustness of the Backdoor-based Watermarking in Deep Neural Networks

[Shane2019] Shane, J. 2019 You Look Like a Thing and I Love You. Hachette UK.

[Sharma2017] Sharma, Y. and Chen, P.-Y. 2017. Attacking the Madry Defense Model with L_1-based Adversarial Examples

[Shejwalkar2019] Shejwalkar, V. and Houmansadr, A. 2019. Reconciling Utility and Membership Privacy via Knowledge Distillation

[Shi2019] Shi, Y., Wang, S., and Han, Y. 2019. Curls & whey: Boosting black-box adversarial attacks

[Shokri2016] Shokri, R., Stronati, M., Song, C. et al. 2016. Membership Inference Attacks against Machine Learning Models

[Shumailov2020] Shumailov, I., Zhao, Y., Bates, D. et al. 2020. Sponge Examples: Energy-Latency Attacks on Neural Networks

[Shyong2006] Shyong, K., Frankowski, D., and Riedl, J. 2006. Do you trust your recommendations? An exploration of security and privacy issues in recommender systems

[Silver2018] Silver, D., Hubert, T., Schrittwieser, J. et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

[Song2018] Song, C. and Shmatikov, V. 2018. Auditing Data Provenance in Text-Generation Models

[Song2019a] Song, C. and Shmatikov, V. 2019. Auditing data provenance in text-generation models

[Song2019b] Song, L., Shokri, R., and Mittal, P. 2019. Privacy Risks of Securing Machine Learning Models against Adversarial Examples

[Su2019] Su, J., Vargas, D. V., and Sakurai, K. 2019. One Pixel Attack for Fooling Deep Neural Networks

[Subramanian2018] Subramanian, V. 2018 Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch.

[Surma2020] Surma, J. 2020. Hacking Machine Learning

[Szegedy2013] Szegedy, C., Zaremba, W., Sutskever, I. et al. 2013. Intriguing properties of neural networks

[Szentannai2019] Szentannai, K., Al-Afandi, J., and Horváth, A. 2019. MimosaNet: An Unrobust Neural Network Preventing Model Stealing

[Tabassi2019] Tabassi, E., Burns, K. J., and Hadjimichael. . . , M. 2019. A Taxonomy and Terminology of Adversarial Machine Learning

[Thakkar2020] Thakkar, O., Ramaswamy, S., Mathews, R. et al. 2020. Understanding Unintended Memorization in Federated Learning

[Tramer2016] Tramèr, F., Zhang, F., Juels, A. et al. 2016. Stealing machine learning models via prediction apis

[Tramer2020] Tramer, F., Carlini, N., Brendel, W. et al. 2020. On Adaptive Attacks to Adversarial Example Defenses

[Truex2018] Truex, S., Liu, L., Gursoy, M. E. et al. 2018. Towards Demystifying Membership Inference Attacks

[Truex2019] Truex, S., Liu, L., Gursoy, M. E. et al. 2019. Effects of Differential Privacy and Data Skewness on Membership Inference Vulnerability

[Tsipras2018] Tsipras, D., Santurkar, S., Engstrom, L. et al. 2018. Robustness May Be at Odds with Accuracy

[Turing1950] TURING, A. M. 1950. I.—COMPUTING MACHINERY AND INTELLIGENCE

[Turing1953] Turing, A. M. 1953. Digital computers applied to games

[Tzeng2015] Tzeng, F. Y. and Ma, K. L. 2005 Opening the black box-data driven visualization of neural networks.

[Vaswani2017] Vaswani, A., Shazeer, N., Parmar, N. et al. 2017. Attention is all you need

[Veale2018] Veale, M., Binns, R., and Edwards, L. 2018. Algorithms that remember: model inversion attacks and data protection law

[Wang2015] Wang, Y., Si, C., and Wu, X. 2015. Regression model fitting under differential privacy and model inversion attack

[Wu2016] Wu, X., Fredrikson, M., Jha, S. et al. 2016. A methodology for formalizing model-inversion attacks

[Wu2020] Wu, M., Zhang, X., Ding, J. et al. 2020. Evaluation of Inference Attack Models for Deep Learning on Medical Data

[Xiao2018] Xiao, Q., Li, K., Zhang, D. et al. 2018. Security Risks in Deep Learning Implementations

[Xu2019] Xu, X., Wang, Q., Li, H. et al. 2019. Detecting AI Trojans Using Meta Neural Analysis

[Yang2019a] Yang, Z., Zhang, J., Chang, E. C. et al. 2019. Neural network inversion in adversarial setting via background knowledge alignment

[Yang2019b] Yang, Z., Chang, E. C., and Liang, Z. 2019. Adversarial neural network inversion via auxiliary knowledge alignment

[Yang2020] Yang, F., Yang, H., Fu, J. et al. 2020. Learning Texture Transformer Network for Image Super-Resolution

[Yeom2017] Yeom, S., Giacomelli, I., Fredrikson, M. et al. 2017. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting

[Yosinski2015] Yosinski, J., Clune, J., Nguyen, A. et al. 2015. Understanding neural networks through deep visualization

[Yu2018] Yu, Y., Liu, X., and Chen, Z. 2018. Attacks and Defenses towards Machine Learning Based Systems

[Yu2019] Yu, L., Liu, L., Pu, C. et al. 2019. Differentially private model publishing for deep learning

[Zeiler2012] Zeiler, M. D. 2012. Adadelta: an adaptive learning rate method

[Zhan2010] Zhan, J., Hsieh, C. L., Wang, I. C. et al. 2010. Privacy-preserving collaborative recommender systems

[Zhang2020] Zhang, Y., Jia, R., Pei, H. et al. 2020. The secret revealer: generative model-inversion attacks against deep neural networks

[Zhao2019] Zhao, Y., Hu, X., Li, S. et al. 2019. Memory trojan attack on neural network accelerators

[Zhu2020] Zhu, Y., Cheng, Y., Zhou, H. et al. 2020. Hermes Attack: Steal DNN Models with Lossless Inference Accuracy

[Zou2018] Zou, M., Shi, Y., Wang, C. et al. 2018. Potrojan: powerful neural-level trojan designs in deep learning model