Machine Learning

in the presence of adversaries

N. Asokan

http://asokan.org/asokan/

@nasokan

(joint work with Mika Juuti, Jian Liu, Andrew Paverd and Samuel Marchal)
Machine Learning is ubiquitous

The ML market size is expected to grow by **44% annually** over next five years. In 2016, companies invested up to **$9 Billion** in AI-based startups.

Machine Learning *for* security/privacy

Access Control

Deception Detection

https://ssg.aalto.fi/projects/phishing/
How do we evaluate ML-based systems?

Effectiveness of inference
- measures of accuracy

Performance
- inference speed and memory consumption

Meeting these criteria even in the presence of adversaries?
Security & privacy of machine learning
Adversarial examples

Which class is this? School bus

+ 0.1

Which class is this? Ostrich

Adversarial examples

Which class is this? *Panda*

Goodfellow et al., “Explaining and Harnessing Adversarial Examples” ICLR 2015

+ 0.007 ⋅ =

Which class is this? *Gibbon*
Adversarial examples

Which class is this?  
Cat

Athalye et al. “Synthesizing Robust Adversarial Inputs.”
Model inversion

Fredrikson et al. “Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures”, ACM CCS 2015.
Model inversion

Machine Learning pipeline

Data owners

Dataset

Trainer

Libs

ML model

Prediction Service Provider API

Where is the adversary? What is its target?
Compromised input

Malicious client

Malicious client

Data owners

Dataset

Libs

Trainer

ML model

Prediction Service Provider API

Client

Extract/steal model

Malicious prediction service

Compromised toolchain: adversary inside training pipeline

Malicious data owner

Influence ML model (model poisoning)

Summary

ML-based systems need to worry about adversarial behaviour
Adversaries are multilateral, solutions are too
Adversarial machine learning is now a very active research area

Secure Systems Group
https://ssg.aalto.fi/

Finnish Center for AI
http://fcai.fi/
Oblivious Neural Network Predictions via MiniONN Transformations

N. Asokan

http://asokan.org/asokan/

@nasokan

(Joint work with Jian Liu, Mika Juuti, Yao Lu)
Machine learning as a service (MLaaS)

Input → Predictions

violation of clients’ privacy
Running predictions on client-side

Model

model theft
evasion
model inversion
Oblivious Neural Networks (ONN)

Given a neural network, is it possible to make it oblivious?

• server learns nothing about clients' input;

• clients learn nothing about the model.
Example: CryptoNets

• High throughput for batch queries from same client
• High overhead for single queries: 297.5s and 372MB (MNIST dataset)
• Cannot support: high-degree polynomials, comparisons, …

[GDLLNW16] CryptoNets, ICML 2016

MiniONN: Overview

- Blinded input
- oblivious protocols
- Blinded predictions

- Low overhead: ~1s
- Support all common neural networks

Example \( z = W' \circ f(W \cdot x + b) + b' \)

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad W = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad W' = \begin{bmatrix} w_{1,1}' & w_{1,2}' \\ w_{2,1}' & w_{2,2}' \end{bmatrix}, \quad b' = \begin{bmatrix} b_1' \\ b_2' \end{bmatrix}
\]

All operations are in a finite field \( \mathbb{Z}_N \)
Core idea: use secret sharing for oblivious computation

Use efficient cryptographic primitives (2PC, additively homomorphic encryption)
x^c, x^c \leftarrow Z_N

x_1^s := x_1 - x_1^c, \quad x_2^s := x_2 - x_2^c

Note that x^c is independent of x. Can be pre-chosen
Oblivious linear transformation $W \cdot x + b$

\[
\begin{bmatrix}
    w_{1,1} & w_{1,2} \\
    w_{2,1} & w_{2,2}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2
\end{bmatrix}
+ 
\begin{bmatrix}
    b_1 \\
    b_2
\end{bmatrix}
= 
\begin{bmatrix}
    w_{1,1} & w_{1,2} \\
    w_{2,1} & w_{2,2}
\end{bmatrix}
\begin{bmatrix}
    x_1^s + x_1^c \\
    x_2^s + x_2^c
\end{bmatrix}
+ 
\begin{bmatrix}
    b_1 \\
    b_2
\end{bmatrix}
\]

$\begin{bmatrix}
    w_{1,1} \cdot (x_1^s + x_1^c) + w_{1,2} \cdot (x_2^s + x_2^c) + b_1 \\
    w_{2,1} \cdot (x_1^s + x_1^c) + w_{2,2} \cdot (x_2^s + x_2^c) + b_2
\end{bmatrix} = 
\begin{bmatrix}
    w_{1,1} \cdot x_1^s + w_{1,2} \cdot x_2^s + b_1 + w_{1,1} \cdot x_1^c + w_{1,2} \cdot x_2^c \\
    w_{2,1} \cdot x_1^s + w_{2,2} \cdot x_2^s + b_2 + w_{2,1} \cdot x_1^c + w_{2,2} \cdot x_2^c
\end{bmatrix}$

Compute locally by the server

Dot-product $W \cdot x^c$
Oblivious linear transformation: dot-product

\[ r_{1,1}, r_{1,2}, r_{2,1}, r_{2,2} \leftarrow \mathbb{Z}_N \]

\[ c_{1,1} = E(w_{1,1} \cdot x^c - r_{1,1}) \]

\[ c_{1,2} = E(w_{1,2} \cdot x^c - r_{1,2}) \]

\[ c_{2,1} = E(w_{2,1} \cdot x^c - r_{2,1}) \]

\[ c_{2,2} = E(w_{2,2} \cdot x^c - r_{2,2}) \]

\[ v_1 = r_{1,1} + r_{1,2} \quad \quad u_1 = w_{1,1} \cdot x^c + w_{1,2} \cdot x^c - (r_{1,2} + r_{1,1}) \]

\[ v_2 = r_{2,1} + r_{2,2} \quad \quad u_2 = w_{2,1} \cdot x^c + w_{2,2} \cdot x^c - (r_{2,1} + r_{2,2}) \]

\[ u + v = W \cdot x^c; \text{ Note: } u, v, \text{ and } W \cdot x^c \text{ are independent of } x. \]

\[ <u, v, x^c> \text{ generated/stored in a precomputation phase} \]
Oblivious linear transformation $W \cdot x + b$

\[
\begin{bmatrix}
  w_{1,1} & w_{1,2} \\
  w_{2,1} & w_{2,2}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2
\end{bmatrix}
+ 
\begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
= 
\begin{bmatrix}
  w_{1,1} & w_{1,2} \\
  w_{2,1} & w_{2,2}
\end{bmatrix}
\begin{bmatrix}
  x^s + x^c_1 \\
  x^s + x^c_2
\end{bmatrix}
+ 
\begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\]

\[
= 
\begin{bmatrix}
  w_{1,1}(x^s_1 + x^c_1) + w_{1,2}(x^s_2 + x^c_2) + b_1 \\
  w_{2,1}(x^s_1 + x^c_1) + w_{2,2}(x^s_2 + x^c_2) + b_2
\end{bmatrix}
= 
\begin{bmatrix}
  w_{1,1}x^s_1 + w_{1,2}x^s_2 + b_1 + w_{1,1}x^c_1 + w_{1,2}x^c_2 \\
  w_{2,1}x^s_1 + w_{2,2}x^s_2 + b_2 + w_{2,1}x^c_1 + w_{2,2}x^c_2
\end{bmatrix}
\]

\[
= 
\begin{bmatrix}
  w_{1,1}x^s_1 + w_{1,2}x^s_2 + b_1 + u_1 \\
  w_{2,1}x^s_1 + w_{2,2}x^s_2 + b_2 + u_2
\end{bmatrix}
+ 
\begin{bmatrix}
  v_1 \\
  v_2
\end{bmatrix}
\]
Oblivious linear transformation $W \cdot x + b$

\[
\begin{bmatrix}
  w_{1,1} & w_{1,2} \\
  w_{2,1} & w_{2,2}
\end{bmatrix}
\cdot
\begin{bmatrix}
  x_1 \\
  x_2
\end{bmatrix}
+
\begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
= 
\begin{bmatrix}
  w_{1,1} & w_{1,2} \\
  w_{2,1} & w_{2,2}
\end{bmatrix}
\cdot
\begin{bmatrix}
  x_1^s + x_1^c \\
  x_2^s + x_2^c
\end{bmatrix}
+
\begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\]

\[
\begin{bmatrix}
  w_{1,1}(x_1^s + x_1^c) + w_{1,2}(x_2^s + x_2^c) + b_1 \\
  w_{2,1}(x_1^s + x_1^c) + w_{2,2}(x_2^s + x_2^c) + b_2
\end{bmatrix}
= 
\begin{bmatrix}
  w_{1,1}x_1^s + w_{1,2}x_2^s + b_1 + w_{1,1}x_1^c + w_{1,2}x_2^c \\
  w_{2,1}x_1^s + w_{2,2}x_2^s + b_2 + w_{2,1}x_1^c + w_{2,2}x_2^c
\end{bmatrix}
\]

\[
\begin{bmatrix}
  w_{1,1}x_1^s + w_{1,2}x_2^s + b_1 + u_1 \\
  w_{2,1}x_1^s + w_{2,2}x_2^s + b_2 + u_2
\end{bmatrix}
+ 
\begin{bmatrix}
  v_1 \\
  v_2
\end{bmatrix}
= 
\begin{bmatrix}
  y_1^s \\
  y_2^s
\end{bmatrix}
+ 
\begin{bmatrix}
  y_1^c \\
  y_2^c
\end{bmatrix}
\]
Oblivious activation/pooling functions $f(y)$

**Piecewise linear functions e.g.,**

- **ReLU:** $x := \max(y, 0)$
- **Oblivious ReLU:** $x^s + x^c := \max(y^s + y^c, 0)$
  - easily computed obliviously by a *garbled circuit*
Oblivious activation/pooling functions $f(y)$

Smooth functions e.g.,

- Sigmoid: $x := 1/(1+e^{-y})$
- Oblivious sigmoid: $x^s + x^c := 1/(1+e^{-(y^s+y^c)})$
  - approximate by a piecewise linear function
  - then compute obliviously by a garbled circuit
  - empirically: ~14 segments sufficient
Combining the final result

\[ y_1 := y_1^s + y_1^c \]
\[ y_2 := y_2^s + y_2^c \]

They can jointly calculate \( \max(y_1, y_2) \) (for minimizing information leakage)
Core idea: use secret sharing for oblivious computation

\[ y^c + y^s = y' \]

\[ (y^c + y^s = y) \]

\[ x^c + x^s = x' \]

\[ (x^c + x^s = x) \]
## Performance (for single queries)

<table>
<thead>
<tr>
<th>Model</th>
<th>Latency (s)</th>
<th>Msg sizes (MB)</th>
<th>Loss of accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST/Square</td>
<td>0.4 (+ 0.88)</td>
<td>44 (+ 3.6)</td>
<td>none</td>
</tr>
<tr>
<td>CIFAR-10/ReLU</td>
<td>472 (+ 72)</td>
<td>6226 (+ 3046)</td>
<td>none</td>
</tr>
<tr>
<td>PTB/Sigmoid</td>
<td>4.39 (+ 13.9)</td>
<td>474 (+ 86.7)</td>
<td>Less than 0.5% (cross-entropy loss)</td>
</tr>
</tbody>
</table>

Pre-computation phase timings in parentheses.

PTB = Penn Treebank
MiniONN pros and cons

300-700x faster than CryptoNets

Can transform any given neural network to its oblivious variant

Still ~1000x slower than without privacy

Server can no longer filter requests or do sophisticated metering

Assumes online connectivity to server

Reveals structure (but not params) of NN
Can trusted computing help?

Hardware support for
- Isolated execution: **Trusted Execution Environment**
- Protected storage: **Sealing**
- Ability to report status to a remote verifier: **Attestation**

- Cryptocards [https://www.ibm.com/security/cryptocards/]
- Trusted Platform Modules [https://www.infineon.com/tpm]
- ARM TrustZone [https://www.arm.com/products/security-on-arm/trustzone]
- Intel Software Guard Extensions [https://software.intel.com/en-us/sgx]
Using a client-side TEE to vet input

1. Attest client’s TEE app
2. Provision filtering policy
3. Input
4. Input, “Input/Metering Certificate”
5. MiniONN protocol + “Input/Metering Certificate”

MiniONN + policy filtering + advanced metering
Using a client-side TEE to run the model

1. Attest client’s TEE app
2. Provision model configuration, filtering policy
3. Input
4. Predictions + “Metering Certificate”
5. “Metering Certificate”

MiniONN + policy filtering + advanced metering
+ disconnected operation + performance + better privacy
- harder to reason about model secrecy
Using a server-side TEE to run the model

1. Attest server’s TEE app
2. Input
3. Provision model configuration, filtering policy
4. Prediction

MiniONN + policy filtering + advanced metering
- disconnected operation + performance + better privacy
MiniONN: Efficiently transform any given neural network into oblivious form with no/negligible accuracy loss

Trusted Computing can help realize improved security and privacy for ML

ML is very fragile in adversarial settings

https://eprint.iacr.org/2017/452
ACM CCS 2017