Privacy-Preserving Machine Learning in TensorFlow with TF Encrypted

Morten Dahl

O'Reilly AI Conference, New York, April 2019
Why?

Privacy in machine learning
Machine Learning Process

data set → training → prediction service
Skin Cancer Image Classification

Brett Kuprel

12:30-12:40pm

Join Brett Kuprel, and see how TensorFlow was used by the artificial intelligence lab and medical school of Stanford to classify skin cancer images. He’ll describe the project steps: from acquiring a dataset, training a deep network, and evaluating the results. To wrap up, Brett will give his take on the future of skin cancer image classification.

machine learning positioned to have huge impact on health care
Potential Bottlenecks

- **Data Access** (liability and controlled use)
- **Training**
- **Prediction Service**
- **Incentive** (accuracy and exposure)
- **Risk Management** (store and process)
- **Leakage** (model and training data)

Diagram showing the flow from data set to prediction service with potential bottlenecks at each stage.
Sanitization
(Differential Privacy)

- Sanitised data set
- Training
- Sanitised prediction
- Data access (liability and controlled use)
- Incentive (accuracy and exposure)
- Leakage (model and training data)
- Risk management (store and process)
Encryption
(Secure Computation)

- Data access (liability and controlled use)
- Encrypted data set
- Encrypted training
- Encrypted prediction
- Incentive (accuracy and exposure)
- Risk management (store and process)
- Leakage (model and training data)
Hybrid

- **Data access** (liability and controlled use)
- **Privacy mitigates bottlenecks**
- **Incentive** (accuracy and exposure)
- **Risk management** (store and process)
- **Leakage** (model and training data)
How?

Computing on encrypted data
Prediction with Linear Model

\[ w_1 x_1 + w_2 x_2 + w_3 x_3 \]

\[ \text{dot}(x, w) = x_1 w_1 + x_2 w_2 + x_3 w_3 \]
... using Homomorphic Encryption

\[
\text{dot}(\text{Enc}(x_1) \text{ Enc}(x_2) \text{ Enc}(x_3), w) = \text{Enc}(x_1*w_1 + x_2*w_2 + x_3*w_3)
\]
Paillier Homomorphic Encryption

\[ c = \text{Enc}(x, r) = g^x \cdot r^n \mod n^2 \]

public encryption key

\[ g = 36 \]
\[ n = 35 \]
\[ n^2 = 1225 \]

Enc(5, 2) = 36^5 \cdot 2^{35} \mod 1225 = 718

Enc(5, 4) = 36^5 \cdot 4^{35} \mod 1225 = 674

typically ~4000 bits: computation is significantly more expensive
Private Addition in Paillier

\[ \text{Enc}(x, r) \times \text{Enc}(y, s) \]

\[ = (g^x \times r^n \mod n^2) \times (g^y \times s^n \mod n^2) \]

\[ = g^{x+y} \times (r \times s)^n \mod n^2 \]

\[ = \text{Enc}(x+y, r \times s) \]

\[ \text{Enc}(5, 2) \times \text{Enc}(5, 4) \]

\[ = 718 \times 674 \]

\[ = 57 \]

\[ = 36^{10} \times 8^{35} \]

\[ = \text{Enc}(10, 8) \]
Public Multiplication in Paillier

\[ \text{Enc}(x, r)^w \]

\[ = (g^x \cdot r^n \mod n^2)^w \]

\[ = g^{(x \cdot w)} \cdot (r^w)^n \mod n^2 \]

\[ = \text{Enc}(x \cdot w, r^w) \]

\[ \text{Enc}(5, 2)^2 \]

\[ = 718 \cdot 718 \]

\[ = 1024 \]

\[ = 36^{10} \cdot 4^{35} \]

\[ = \text{Enc}(10, 4) \]
... using Secret Sharing

\[ w \]

\[ \text{Share}_1(x) \]
\[ \text{Share}_1(\text{dot}(x, w)) \]

\[ x \]

\[ w \]

\[ \text{Share}_2(x) \]
\[ \text{Share}_2(\text{dot}(x, w)) \]
Secret Sharing

\[ \text{Share}_1(x, r) = r \mod m \]

\[ \text{Share}_2(x, r) = x - r \mod m \]

\[ x = \text{Share}_1(x, r) + \text{Share}_2(x, r) \mod m \]

\[ m = 10 \]

\[ \text{Share}_1(5, ?) = 7 \mod 10 = 7 \]

\[ \text{Share}_2(5, ?) = 5 - 7 \mod 10 = 8 \]

\[ 7 + 8 = 15 = 5 \mod 10 \]
Private Addition with Secret Sharing

\[
x = x_1 + x_2 \quad y = y_1 + y_2
\]

\[
z_1 = x_1 + y_1 \quad z_2 = x_2 + y_2
\]

\[
x + y = (x_1 + x_2) + (y_1 + y_2) = (x_1 + y_1) + (x_2 + y_2) = z_1 + z_2
\]
Public Multiplication with Secret Sharing

\[ x = x_1 + x_2 \]

\[ z_1 = x_1 \times w \]

\[ z_2 = x_2 \times w \]

\[ x \times w = (x_1 + x_2) \times w = (x_1 \times w) + (x_2 \times w) = z_1 + z_2 \]
... using Secret Sharing, with Private Model

\[ \text{Share}_1(w) \rightarrow \text{Share}_1(x) \rightarrow \text{Share}_1(\text{dot}(x, w)) \]

\[ \text{Share}_2(w) \rightarrow \text{Share}_2(x) \rightarrow \text{Share}_2(\text{dot}(x, w)) \]
… using Secret Sharing, with Private Model

\[
\begin{align*}
\text{Share}_1(w) & \\
\text{Share}_2(w) & \\
\text{Share}_1(x_0) & \\
\text{Share}_1(x_1) & \\
\text{Share}_1(x_2) & \\
\text{Share}(w_0) + \ldots &= \text{Share}(w_0) + \ldots \\
\text{Share}(w_1) & \\
\text{Share}(w_2) & \\
\text{Share}_1(x_0) \cdot \text{Share}(w_0) + \ldots &= \text{Share}_1(x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2) \\
\text{private multiplication} & \\
\end{align*}
\]
Private Multiplication with Secret Sharing

\[
\begin{align*}
(a_1, b_1, c_1) & \quad x_1 \quad y_1 \quad \alpha \quad \beta \\
(a_2, b_2, c_2) & \quad x_2 \quad y_2 \quad \alpha \quad \beta
\end{align*}
\]

\[
\begin{align*}
a &= a_1 + a_2 \\
b &= b_1 + b_2 \\
c &= a \cdot b = c_1 + c_2 \\
x &= x_1 + x_2 \\
y &= y_1 + y_2 \\
\alpha &= x - a \\
\beta &= y - b \\
z_1 &= \alpha \cdot \beta + \alpha \cdot b_1 + \beta \cdot a_1 + c_1 \\
z_2 &= \alpha \cdot b_2 + \beta \cdot a_2 + c_2 \\
x \cdot y &= = z_1 + z_2
\end{align*}
\]
Multidisciplinary Challenge

Data science
(use-cases, workflow, monitoring)

Cryptography
(techniques, protocols, trust)

Machine learning
(models, approx, precision)

Engineering
(distributed, multi-core, readability)

need common language
TF Encrypted

Making it accessible
TensorFlow

platform for research and production-level training and deployment

popular and backed by Google
TF Encrypted Architecture

- Standard operations (matmul, relu, sigmoid, tanh, etc)
- Secure computation directly using TensorFlow
- Third party libraries for secure computation
- Easily mix ordinary and encrypted computations
- Ordinary TensorFlow

App

TF Encrypted

MPC

ML

HE

MPC

Dist

Tensor

ML

TensorFlow
Prediction

Encouraging use
Participants

\( \text{Share1}(w_0, b_0, \ldots) \rightarrow \text{Share1}(x) \rightarrow \text{Share1}(\text{logits}) \)

\( w_0, b_0, \ldots \)

\( \text{Share2}(w_0, b_0, \ldots) \rightarrow \text{Share2}(x) \rightarrow \text{Share2}(\text{logits}) \)
import tensorflow as tf

# define weights
w0, b0, w1, b1, w2, b2 = tf.initializers.random_uniform(0, 1, shape=(1, 1))

# define input
x = tf.placeholder(tf.float32, shape=(1, 1))

# compute prediction
layer0 = tf.nn.relu(tf.matmul(x, w0) + b0)
layer1 = tf.nn.relu(tf.matmul(layer0, w1) + b1)
logits = tf.matmul(layer1, w2) + b2

# process result of prediction
prediction_op = tf.argmax(logits)

# run graph execution in a tf.Session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(prediction_op)
Overall Computation
Local Processing

TF Data pipeline
Joint Prediction

Combining knowledge for nuance
Participants

Share1(w0, b0, ...)

Share1(x_age)
Share1(x_gender)
Share1(x_income)

Share2(w0, b0, ...)

Share2(x_age)
Share2(x_gender)
Share2(x_income)

w0, b0, ...

x_age
x_gender
x_income

Share1(res)
Share2(res)
Private Joint Prediction with TF Encrypted

```python
w, b = tfe.define_private_input("model-owner", provide_weights)
x_age = tfe.define_private_input("age-provider", provide_age)
x_gender = tfe.define_private_input("gender-provider", provide_gender)
x_income = tfe.define_private_input("income-provider", provide_income)
x = tfe.concat([x_age, x_gender, x_income], axis=1)
y = tfe.matmul(x, w) + b
prediction_op = tfe.define_output("result-receiver", y, receive_output)

with tfe.Session() as sess:
sess.run(prediction_op)
```
Training

Learning without seeing
Participants

Share1(x, y) → x, y → Share2(x, y) → Share1(w) → Share2(w) → Cloud
Private Training with TF Encrypted

```python
class LogisticRegression(Model):
    @property
    def weights(self):
        # ...

    def forward(self, x):
        # ...

    def backward(self, x, dy, learning_rate=0.01):
        # ...

    def loss_grad(self, y, y_hat):
        # ...

model = LogisticRegression()
data_owner = DataOwner('data-owner')
model_owner = ModelOwner('model-owner')
x_train, y_train = tfe.define_private_input(data_owner.player_name, data_owner.provide_training_data)
x_test, y_test = tfe.define_private_input(data_owner.player_name, data_owner.provide_testing_data)
reveal_weights_op = tfe.define_output(model_owner.player_name, model.weights, model_owner.receive_weights)

with tfe.Session() as sess:
    sess.run([tfe.global_variables_initializer(), data_owner.initialize()])

model.fit(sess, x_train, y_train, epochs=10)
model.evaluate(sess, x_test, y_test, data_owner)

sess.run(reveal_weights_op)
```
Overall Computation

data owner

model owner
Joint Training

Combining insights for better models
Participants

x₀, y₀

Share₁(x₀, y₀)
Share₁(x₁, y₁)

Share₁(w)

x₁, y₁

Share₂(x₀, y₀)
Share₂(x₁, y₁)

Share₂(w)
Private Joint Training with TF Encrypted

```python
1  data_owner_0 = DataOwner('data-owner-0')
2  data_owner_1 = DataOwner('data-owner-1')
3
4  tfe.set_protocol(tfe.protocol.Pond(data_owner_0.player_name, data_owner_1.player_name))
5
6  x_train_0, y_train_0 = tfe.define_private_input(data_owner_0.player_name, data_owner_0.provide_training_data)
7  x_train_1, y_train_1 = tfe.define_private_input(data_owner_1.player_name, data_owner_1.provide_training_data)
8
9  x_train = tfe.concat([x_train_0, x_train_1], axis=0)
10  y_train = tfe.concat([y_train_0, y_train_1], axis=0)
```
Federated Learning

Keeping data decentralized
Participants

x_0, y_0
Share1(update_0)
Share1(update_1)
Share1(update_2)

x_1, y_1
Share2(update_0)
Share2(update_1)
Share2(update_2)

x_2, y_2

Share1(aggregated-update)
Share1(weights)

Share2(aggregated-update)
Share2(weights)
Secure Federated Learning in TF Encrypted

```python
model_owner = ModelOwner('model-owner')
data_owners = [
    DataOwner('data-owner-0', model_owner.build_training_model),
    DataOwner('data-owner-1', model_owner.build_training_model),
    DataOwner('data-owner-2', model_owner.build_training_model),
]

model_grads = zip(*[
    tfe.define_private_input(data_owner.player_name, data_owner.compute_gradient)
    for data_owner in data_owners
])

aggregated_model_grads = [
    tfe.add_n(grads) / len(grads)
    for grads in model_grads
]

iteration_op = tfe.define_output(model_owner.player_name, aggregated_model_grads, model_owner.update_model)

with tfe.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(model_owner.ITERATIONS):
        sess.run(iteration_op)
```
Local Optimization

TF optimization
Roadmap

High-level API (Private Keras, Pre-trained Models, Owned Data)

Tighter integration (TF Data, TF 2.0, TF Privacy, TF Federated)

Third-party cryptographic libraries (HE, MPC)

Improved performance
Wrap-Up

You can **compute on encrypted data**, without the ability to decrypt.

Privacy-preserving ML mitigate **bottlenecks** and **enable access** to sensitive information.

Secure computation **distributes trust and control**, and is complementary to e.g. differential privacy.

Privacy-preserving ML is a multidisciplinary field benefitting from **adaptations** on both sides.

TF Encrypted focuses on **usability** and **integration**.

Thank you!

github.com/tf-encrypted/

@mortendahlcs

@dropoutlabsai