Private Machine Learning in TensorFlow using Secure Computation

Morten Dahl Jason Mancuso Yann Dupis Ben DeCoste
Morgan Giraud Ian Livingstone Justin Patriquin Gavin Uhma

Dropout Labs

Motivation

Secure computation protocols for machine learning are often implemented as custom C++ libraries, and require expertise with a variety of disparate techniques and systems. This sets a high barrier to entry for interested researchers and practitioners, prevents reliable comparative benchmarking, and hampers third party auditing efforts.

Abstract

We present a framework for experimenting with secure multi-party computation directly in TensorFlow. By doing so we benefit from several properties valuable to both researchers and practitioners, including tight integration with ordinary machine learning processes, existing optimizations for distributed computation in TensorFlow, high-level abstractions for expressing complex algorithms and protocols, and an expanded set of familiar tooling. We give an open source implementation of a state-of-the-art protocol and report on concrete benchmarks using typical models from private machine learning.

Highlights

Usability: By leveraging TensorFlow, we obtain a familiar and comprehensive platform for building scalable, distributed machine learning on heterogenous device backends.

Extensibility: Using TensorFlow's high-level abstractions makes it easier to experiment with and develop new secure protocols on top of optimized primitives while maintaining code readability.

Integration: By reducing all secure computations to TensorFlow graphs, it becomes straight-forward to mix these with ordinary computations for hybrid approaches.

Performance: We reach high runtime efficiency without sacrificing other properties via TensorFlow's distributed execution engine heavily optimized for networking, parallel execution, and scalability.

Benchmarking: Combining all of the above we obtain a common framework for comparable private machine learning.

Usability

We keep tf-encrypted's API as close to TensorFlow's as possible to provide a seamless transition that abstracts the complexity of secure computation away from ML researchers and practitioners.

tf-encrypted

```python
import tensorflow as tf
import tf_encrypted as tfe

# generic remote procedure calls
def provide_weights(): """Load model weights."""
def provide_input(): """Load input data."""
def receive_output(): """Receive and decrypt output."""

# get model weights/input data
# as private tensors from each party
weights = tfe.define_private_input("model-owner",
                                   provide_weights)
w0, b0, w1, b1, w2, b2 = weights
x = tfe.define_private_input("prediction-client",
                             provide_input)

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

# send prediction output back to client
prediction_op = tfe.define_output("prediction-client",
                                  [logits],
                                  receive_output)

# run secure graph execution in a tf.Session
with tfe.Session() as sess:
    sess.run(tf.global_variables_initializer(),
             tag="init")
    sess.run(prediction_op, tag="prediction")
```

TensorFlow

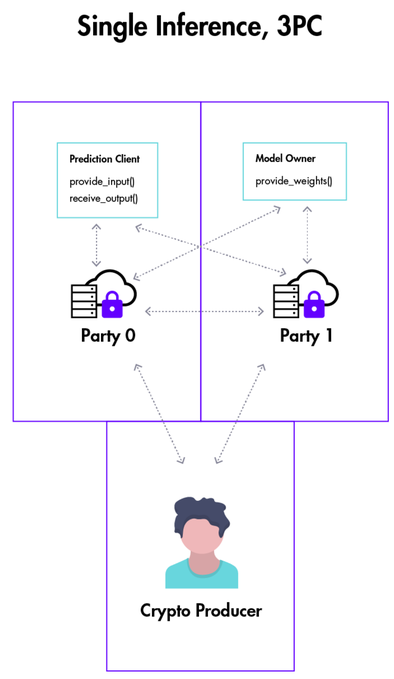
```python
import tensorflow as tf


# generic function stubs
def provide_weights(): """Load model weights."""
def provide_input(): """Load input data."""


# get model weights/input data
# (both unencrypted)
w0, b0, w1, b1, w2, b2 = provide_weights()
x = provide_input()




# compute prediction
layer0 = tf.nn.relu((tf.matmul(x, w0) + b0))
layer1 = tf.nn.relu((tf.matmul(layer0, w1) + b1))
logits = tf.matmul(layer2, w2) + b2

# get result of prediction and print
prediction_op = tf.Print(result,
                         [logits],
                         summarize=10)

# run graph execution in a tf.Session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer(),
             tag="init")
    sess.run(prediction_op, tag="prediction")
```

Extensibility

We demonstrate tf-encrypted with Pond, a three-party implementation of the SPDZ protocol with passive security. There is also an extended version of Pond based on the SecureNN protocol of Wagh et al., in which comparison-based operations like ReLU and MaxPool can be computed without approximation.

The Protocol and Player abstractions are sufficiently general to accommodate protocols with any number of Players and any security model that fits within the paradigm of distributed tensor computation.

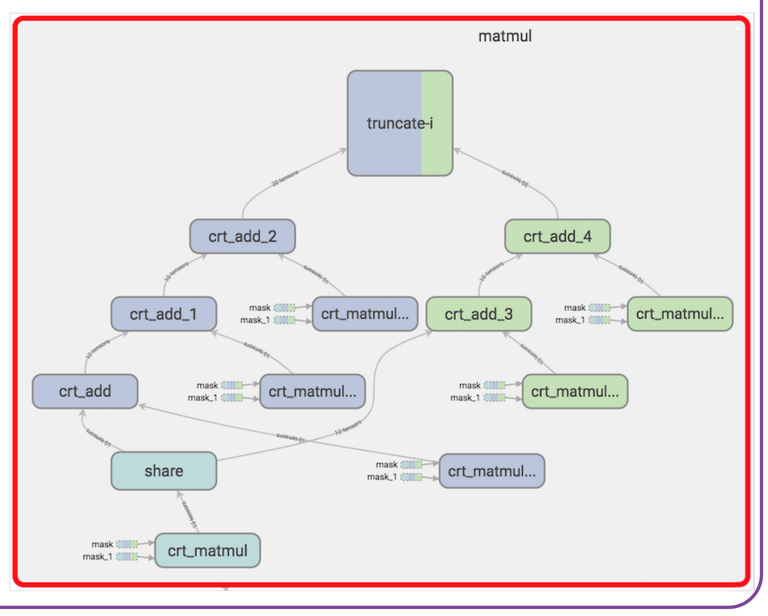
We plan on implementing a two-party version of SPDZ with active security and an optional garbled circuits extension in the coming months.



Integration

By annotating components of each secure multi-party computation with the `tf.device` context manager, TensorFlow can orchestrate and optimize each during the compilation phase. By building on top of TensorFlow, we can use support tools like TensorBoard for visually auditing, profiling, and debugging.

```python
def matmul(prot: Pond,
           x: PondMaskedTensor,
           y: PondMaskedTensor):
    a, a0, a1, alpha_on_0, alpha_on_1 = x.unwrapped
    b, b0, b1, beta_on_0, beta_on_1 = y.unwrapped

    with tf.device(prot.crypto_producer.device_name):
        ab = a.matmul(b)
        ab0, ab1 = prot._share(ab)

    with tf.device(prot.server_0.device_name):
        alpha = alpha_on_0
        beta = beta_on_0
        z0 = (ab0
              + a0.matmul(beta)
              + alpha.matmul(b0)
              + alpha.matmul(beta))

    with tf.device(prot.server_1.device_name):
        alpha = alpha_on_1
        beta = beta_on_1
        z1 = (ab1
              + a1.matmul(beta)
              + alpha.matmul(b1))

    return PondPrivateTensor(prot, z0, z1)
```



Performance & Benchmarking

SecNN	Pond
Conv (5, 20, 1, 1)	Conv (5, 20, 1, 1)
–	BatchNorm
ReLU	~ReLU
MaxPool (2)	AvgPool (2)
Conv (5, 50, 1, 1)	Conv (5, 50, 1, 1)
–	BatchNorm
ReLU	~ReLU
MaxPool (2)	AvgPool (2)
FC (800, 500)	FC (800, 500)
–	BatchNorm
ReLU	~ReLU
FC (500, 10)	FC (500, 10)

We report benchmarks on the standard MNIST dataset. Runtime mean and standard deviation statistics are gathered over 100 inferences, while accuracy and KL-divergence metrics are reported over the MNIST test set.

For reference, our current runtime is 2.2x, 1.1x, and .85x relative to the SecureNN implementation of Wagh et al. for networks A, B, and C, respectively.

	Runtime, ms		Accuracy			KL-divergence	
	Pond	SecNN	TF	Pond	SecNN	Pond	SecNN
A	**14 (3.8)**	112 (63)	97.35%	97.18%	**97.35%**	0.0065	**0.0**
B	**126 (115)**	243 (79)	99.26%	99.00%	**99.26%**	0.2086	**0.0**
C	**124 (93)**	293 (78)	99.44%	99.41%	**99.44%**	0.2311	**0.0**

Summary

We develop a practical secure multi-party computation framework on top of TensorFlow that is usable, extensible, performant, and well-integrated with existing machine learning practices and tools. We hope researchers will use tf-encrypted to accelerate the implementation and benchmarking of their own novel secure computation protocols for machine learning.

References

Code on GitHub: **mortendahl/tf-encrypted**. Full paper on Arxiv: 1810.08130.
Abadi et al., TensorFlow: A System for Large-scale Machine Learning, 2016
Damgard et al., Multiparty Computation from Somewhat Homomorphic Encryption, 2012
Wagh et al., SecureNN: Efficient and Private Neural Network Training, 2018