Private Image Analysis

October 16, 2018

```
In [1]: from pond.tensor import NativeTensor, PrivateEncodedTensor, PublicEncodedTensor
        from pond.nn import Dense, Sigmoid, Reveal, Diff, Softmax, CrossEntropy, Sequential, Dat

In [2]: import numpy as np
        from datetime import datetime
```

1 Feature extraction

```
In [ ]: import keras
        from keras.utils import to_categorical

In [ ]: def preprocess_data(dataset):

            (x_train, y_train), (x_test, y_test) = dataset

            # NOTE: this is the shape used by Tensorflow; other backends may differ
            x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
            x_test  = x_test.reshape(x_test.shape[0], 28, 28, 1)

            x_train  = x_train.astype('float32')
            x_test   = x_test.astype('float32')
            x_train /= 255
            x_test  /= 255

            y_train = to_categorical(y_train, 5)
            y_test  = to_categorical(y_test, 5)

            return (x_train, y_train), (x_test, y_test)

        def load_data():

            (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

            x_train_public = x_train[y_train < 5]
            y_train_public = y_train[y_train < 5]
            x_test_public  = x_test[y_test < 5]
            y_test_public  = y_test[y_test < 5]
```

```
    public_dataset = (x_train_public, y_train_public), (x_test_public, y_test_public)

    x_train_private = x_train[y_train >= 5]
    y_train_private = y_train[y_train >= 5] - 5
    x_test_private  = x_test[y_test >= 5]
    y_test_private  = y_test[y_test >= 5] - 5
    private_dataset = (x_train_private, y_train_private), (x_test_private, y_test_privat

    return preprocess_data(public_dataset), preprocess_data(private_dataset)
```

1.1 Pre-train on public data

```
In [ ]: public_dataset, private_dataset = load_data()

    feature_layers = [
        keras.layers.Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
        keras.layers.Activation('sigmoid'),
        keras.layers.Conv2D(32, (3, 3), padding='same'),
        keras.layers.Activation('sigmoid'),
        keras.layers.AveragePooling2D(pool_size=(2,2)),
        keras.layers.Dropout(.25),
        keras.layers.Flatten()
    ]

    classification_layers = [
        keras.layers.Dense(128),
        keras.layers.Activation('sigmoid'),
        keras.layers.Dropout(.50),
        keras.layers.Dense(5),
        keras.layers.Activation('softmax')
    ]

    model = keras.models.Sequential(feature_layers + classification_layers)

    model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    (x_train, y_train), (x_test, y_test) = public_dataset

    model.fit(
        x_train, y_train,
        epochs=5,
        batch_size=32,
        verbose=1,
        validation_data=(x_test, y_test))
```

1.2 Extract features from private data (unencrypted for now)

```
In [ ]: model.summary()
```

```
In [ ]: flatten_layer = model.get_layer(index=7)
        assert flatten_layer.name.startswith('flatten_')

        extractor = keras.models.Model(
            inputs=model.input,
            outputs=flatten_layer.output
        )
```

```
In [ ]: (x_train_images, y_train), (x_test_images, y_test) = private_dataset

        x_train_features = extractor.predict(x_train_images)
        x_test_features  = extractor.predict(x_test_images)
```

1.3 Save extracted features

```
In [ ]: np.save('x_train_features.npy', x_train_features)
        np.save('y_train.npy', y_train)

        np.save('x_test_features.npy', x_test_features)
        np.save('y_test.npy', y_test)
```

1.4 Load extracted features

```
In [3]: x_train_features = np.load('x_train_features.npy')
        y_train = np.load('y_train.npy')

        x_test_features = np.load('x_test_features.npy')
        y_test = np.load('y_test.npy')

        print(x_train_features.shape, y_train.shape, x_test_features.shape, y_test.shape)
```

```
(29404, 6272) (29404, 5) (4861, 6272) (4861, 5)
```

2 Fine-tune

```
In [4]: classifier = Sequential([
            Dense(128, 6272),
            Sigmoid(),
            # Dropout(.5),
            Dense(5, 128),
            Reveal(),
            Softmax()
        ])
```

```
In [5]: def accuracy(classifier, x, y, verbose=0, wrapper=NativeTensor):
            predicted_classes = classifier \
                .predict(DataLoader(x, wrapper), verbose=verbose).reveal() \
                .argmax(axis=1)

            correct_classes = NativeTensor(y) \
                .argmax(axis=1)

            matches = predicted_classes.unwrap() == correct_classes.unwrap()
            return sum(matches)/len(matches)
```

2.1 ... using NativeTensor

```
In [6]: classifier.initialize()

        start = datetime.now()
        classifier.fit(
            DataLoader(x_train_features, wrapper=NativeTensor),
            DataLoader(y_train, wrapper=NativeTensor),
            loss=CrossEntropy(),
            epochs=3,
            verbose=1
        )
        stop = datetime.now()

        print("Elapsed:", stop - start)
```

```
2017-12-29 11:02:50.827373 Epoch 0
2017-12-29 11:03:07.306861 Epoch 1
2017-12-29 11:03:22.950060 Epoch 2
Elapsed: 0:00:48.221433
```

```
In [ ]: print("Train accuracy:", accuracy(classifier, x_train_features, y_train))
        print("Test accuracy:",  accuracy(classifier, x_test_features,  y_test))
```

```
Train accuracy: 0.90671337233
Test accuracy: 0.908660769389
```

2.2 ... using PublicEncodedTensor

```
In [ ]: classifier.initialize()

        start = datetime.now()
        classifier.fit(
            DataLoader(x_train_features, wrapper=PublicEncodedTensor),
            DataLoader(y_train, wrapper=PublicEncodedTensor),
            loss=CrossEntropy(),
```

```
            epochs=3,
            verbose=2
        )
        stop = datetime.now()

        print("Elapsed:", stop - start)
```

```
In [ ]: print("Train accuracy:", accuracy(classifier, x_train_features, y_train, verbose=2))
        print("Test accuracy:",  accuracy(classifier, x_test_features,  y_test,  verbose=2))
```

2.3 ... using PrivateEncodedTensor

```
In [ ]: classifier.initialize()

        start = datetime.now()
        classifier.fit(
            DataLoader(x_train_features, wrapper=PrivateEncodedTensor),
            DataLoader(y_train, wrapper=PrivateEncodedTensor),
            loss=CrossEntropy(),
            epochs=3,
            verbose=2
        )
        stop = datetime.now()

        print("Elapsed:", stop - start)
```

```
2017-12-29 11:03:42.926628 Epoch 0
2017-12-29 11:03:43.243759 Batch 0
2017-12-29 11:04:54.253005 Batch 1
2017-12-29 11:06:37.272832 Batch 2
2017-12-29 11:08:20.516759 Batch 3
2017-12-29 11:09:56.260065 Batch 4
2017-12-29 11:11:30.663451 Batch 5
2017-12-29 11:13:04.473740 Batch 6
2017-12-29 11:14:39.077599 Batch 7
```

```
In [ ]: train_accuracy = accuracy(classifier, x_train_features, y_train, verbose=2)
        test_accuracy  = accuracy(classifier, x_test_features,  y_test,  verbose=2)
```

```
2018-01-02 14:13:39.032988 Batch 0
```

```
In [13]: print("Train accuracy:", train_accuracy)
         print("Test accuracy:", test_accuracy)
```

```
Train accuracy: 0.906611345395
Test accuracy: 0.908249331413
```

In [19]:
```
np.save('layer0_weights_0.npy', classifier.layers[0].weights.shares0)
np.save('layer0_weights_1.npy', classifier.layers[0].weights.shares1)
np.save('layer0_bias_0.npy', classifier.layers[0].bias.shares0)
np.save('layer0_bias_1.npy', classifier.layers[0].bias.shares1)

np.save('layer2_weights_0.npy', classifier.layers[2].weights.shares0)
np.save('layer2_weights_1.npy', classifier.layers[2].weights.shares1)
np.save('layer2_bias_0.npy', classifier.layers[2].bias.shares0)
np.save('layer2_bias_1.npy', classifier.layers[2].bias.shares1)
```

In [ ]: