federated_learning_basic_concepts_pretrained_model

June 18, 2020

1 Federated learning: Simple experiment using pretrained learning model

In this notebook we provide a simple example of how to make an experiment of a federated environment with the help of this framework. We are going to use a popular dataset to start the experimentation in a federated environment. The framework provides some functions to load the Emnist Digits dataset.

[1]:
```
import shfl

database = shfl.data_base.Emnist()
train_data, train_labels, test_data, test_labels = database.load_data()
```

Let's inspect some properties of the loaded data.

[2]:
```
print(len(train_data))
print(len(test_data))
print(type(train_data[0]))
train_data[0].shape
```

```
240000
40000
<class 'numpy.ndarray'>
```

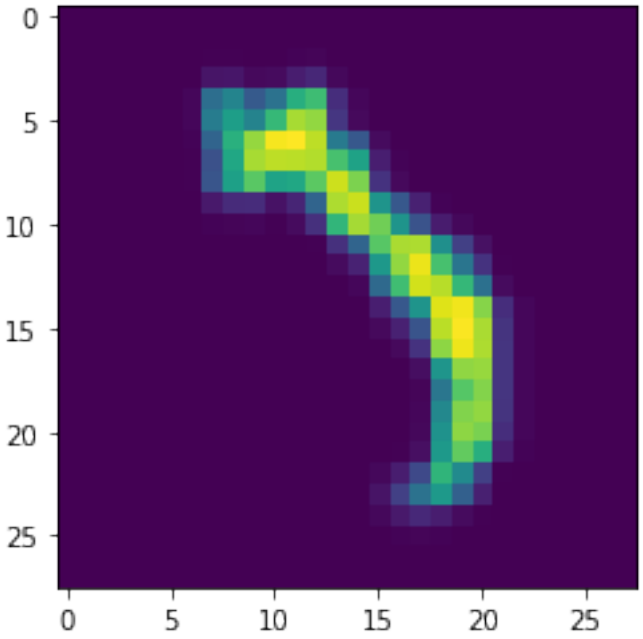
[2]:
```
(28, 28)
```

So, as we have seen, our dataset is composed of a set of matrices of 28 by 28. Before starting with the federated scenario, we can take a look at a sample in the training data.

[3]:
```
import matplotlib.pyplot as plt

plt.imshow(train_data[0])
```

[3]:
```
<matplotlib.image.AxesImage at 0x13776a850>
```



We are going to simulate a federated learning scenario with a set of client nodes containing private data, and a central server that will be responsible to coordinate the different clients. But, first of all, we have to simulate the data contained in every client. In order to do that, we are going to use the previously loaded dataset. The assumption in this example will be the data is distributed as a set of independent and identically distributed random variables, having every node approximately the same amount of data. There are a set of different possibilities in order to distribute the data. The distribution of the data is one of the factors that could impact more a federated algorithm. Therefore, the framework contains the implementation of some of the most common distributions that allow you to experiment different situations easily. In Federated Sampling you can dig into the options that the framework provides at the moment.

[4]:
```
iid_distribution = shfl.data_distribution.IidDataDistribution(database)
federated_data, test_data, test_label = iid_distribution.
 ↪get_federated_data(num_nodes=20, percent=10)
```

That's it! We have created federated data from the Emnist dataset using 20 nodes and 10 percent of the available data. This data is distributed to a set of data nodes in the form of private data. Let's learn a little more about the federated data.

[5]:
```
print(type(federated_data))
print(federated_data.num_nodes())
federated_data[0].private_data
```

```
<class 'shfl.private.federated_operation.FederatedData'>
20
Node private data, you can see the data for debug purposes but the data remains
in the node
```

```
<class 'dict'>
{'5178298448': <shfl.private.data.LabeledData object at 0x13785f490>}
```

As we can see, private data in a node is not accesible directly but the framework provides mechanisms to use this data in a machine learning model. A federated learning algorithm is defined by a machine learning model locally deployed in each node that learns from the respective node's private data and an aggregating mechanism to aggregate the different model parameters uploaded by the client nodes to a central node. In this example we will use a deep learning model using keras to build it. The framework provides classes to allow using Tensorflow (see Basic Concepts Tensorflow) and Keras (see Basic Concepts) models into a federated learning scenario, your job is only to create a function acting as model builder. Moreover, the framework provides classes to allow using pretrained Tensorflow and Keras models. In this example use a pretrained Keras learning model.

[6]:
```
import tensorflow as tf
#If you want execute in GPU, you must uncomment this two lines.
# physical_devices = tf.config.experimental.list_physical_devices('GPU')
# tf.config.experimental.set_memory_growth(physical_devices[0], True)

train_data = train_data.reshape(-1,28,28,1)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',␣
 ↪activation='relu', strides=1, input_shape=(28, 28, 1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2, padding='valid'))
model.add(tf.keras.layers.Dropout(0.4))
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',␣
 ↪activation='relu', strides=1))
model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2, padding='valid'))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer="rmsprop", loss="categorical_crossentropy",␣
 ↪metrics=["accuracy"])

model.fit(x=train_data, y=train_labels, batch_size=128, epochs=3,␣
 ↪validation_split=0.2,
                verbose=1, shuffle=False)
```

```
Epoch 1/3
1500/1500 [==============================] - 812s 542ms/step - loss: 0.3231 -
accuracy: 0.9313 - val_loss: 0.0640 - val_accuracy: 0.9843
Epoch 2/3
1500/1500 [==============================] - 727s 484ms/step - loss: 0.0720 -
```

```
accuracy: 0.9799 - val_loss: 0.0427 - val_accuracy: 0.9892
Epoch 3/3
1500/1500 [==============================] - 725s 483ms/step - loss: 0.0648 -
accuracy: 0.9830 - val_loss: 0.0355 - val_accuracy: 0.9912
```

[6]:
```
<tensorflow.python.keras.callbacks.History at 0x137a07850>
```

[7]:
```
def model_builder():
    return shfl.model.DeepLearningModel(model=model)
```

Now, the only piece missing is the aggregation operator. Nevertheless, the framework provides some aggregation operators that we can use. In the following piece of code we define the federated aggregation mechanism. Moreover, we define the federated goverment based on the keras learning model, the federated data and the aggregation mechanism.

[8]:
```
aggregator = shfl.federated_aggregator.FedAvgAggregator()
federated_government = shfl.federated_government.
 ↪FederatedGovernment(model_builder, federated_data, aggregator)
```

If you want to see all the aggregation operators you can check the following notebook Federated Aggregation Operators. Before running the algorithm, we want to apply a transformation to the data. A good practice is to define a federated operation that will ensure that the transformation is applied to the federated data in all the client nodes. We want to reshape the data, so we define the following FederatedTransformation.

[9]:
```
import numpy as np

class Reshape(shfl.private.FederatedTransformation):

    def apply(self, labeled_data):
        labeled_data.data = np.reshape(labeled_data.data, (labeled_data.data.
 ↪shape[0], labeled_data.data.shape[1], labeled_data.data.shape[2],1))

class CastFloat(shfl.private.FederatedTransformation):

    def apply(self, labeled_data):
        labeled_data.data = labeled_data.data.astype(np.float32)

shfl.private.federated_operation.apply_federated_transformation(federated_data,␣
 ↪Reshape())
shfl.private.federated_operation.apply_federated_transformation(federated_data,␣
 ↪CastFloat())
```

We are now ready to execute our federated learning algorithm.

[10]:
```
test_data = np.reshape(test_data, (test_data.shape[0], test_data.shape[1],␣
 ↪test_data.shape[2],1))
test_data = test_data.astype(np.float32)
```

```
federated_government.run_rounds(2, test_data, test_label)
```

```
Accuracy round 0
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13776aa50>: [0.04073205962777138, 0.9886249899864197]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785fd10>: [0.03860897570848465, 0.9898499846458435]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785ff90>: [0.06138899549841881, 0.9865999817848206]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f3d0>: [0.04631864279508591, 0.9872249960899353]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785fad0>: [0.03949038311839104, 0.990024983882904]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f950>: [0.038507454097270966, 0.9896500110626221]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f1d0>: [0.05063445866107941, 0.9866999983787537]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f550>: [0.06192926689982414, 0.988349974155426]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f210>: [0.03994989022612572, 0.9891999959945679]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857ad0>: [0.033828750252723694, 0.9904999732971191]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857fd0>: [0.03869995102286339, 0.9905750155448914]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857350>: [0.03750046342611313, 0.989549994468689]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857750>: [0.045944761484861374, 0.9892500042915344]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857850>: [0.03171301633119583, 0.9914500117301941]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df990>: [0.05461222678422928, 0.9851250052452087]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df550>: [0.039834797382354736, 0.9886749982833862]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377dfd10>: [0.04584995284676552, 0.9868000149726868]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df9d0>: [0.03322899341583252, 0.9914249777793884]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df590>: [0.03232917934656143, 0.9916250109672546]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377dfc50>: [0.03447919338941574, 0.9905750155448914]
Global model test performance : [0.026528412476181984, 0.9926000237464905]
```

```
Accuracy round 1
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13776aa50>: [0.045339323580265045, 0.9879500269889832]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785fd10>: [0.029813939705491066, 0.991474986076355]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785ff90>: [0.101341113448143, 0.9806249737739563]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f3d0>: [0.044419918209314346, 0.9890000224113464]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785fad0>: [0.03223715350031853, 0.991225004196167]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f950>: [0.04698410630226135, 0.9871500134468079]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f1d0>: [0.03262860327959061, 0.9910249710083008]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f550>: [0.033309370279312134, 0.991100013256073]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13785f210>: [0.0333954356610775, 0.9904999732971191]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857ad0>: [0.04299957677721977, 0.9887499809265137]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857fd0>: [0.028820034116506577, 0.9922999739646912]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857350>: [0.029765915125608444, 0.9924250245094299]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857750>: [0.042869288474321365, 0.9884499907493591]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x137857850>: [0.04133288189768791, 0.9904249906539917]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df990>: [0.03281312435865402, 0.9907000064849854]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df550>: [0.0353374257683754, 0.9901999831199646]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377dfd10>: [0.03278886526823044, 0.9908499717712402]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df9d0>: [0.08481486141681671, 0.9781000018119812]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377df590>: [0.0362912118434906, 0.9908499717712402]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1377dfc50>: [0.0327630452811718, 0.9910249710083008]
Global model test performance : [0.0281806793063879, 0.9926000237464905]
```

[ ]: