federated_learning_basic_concepts_pretrained_model

June 19, 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'>
```

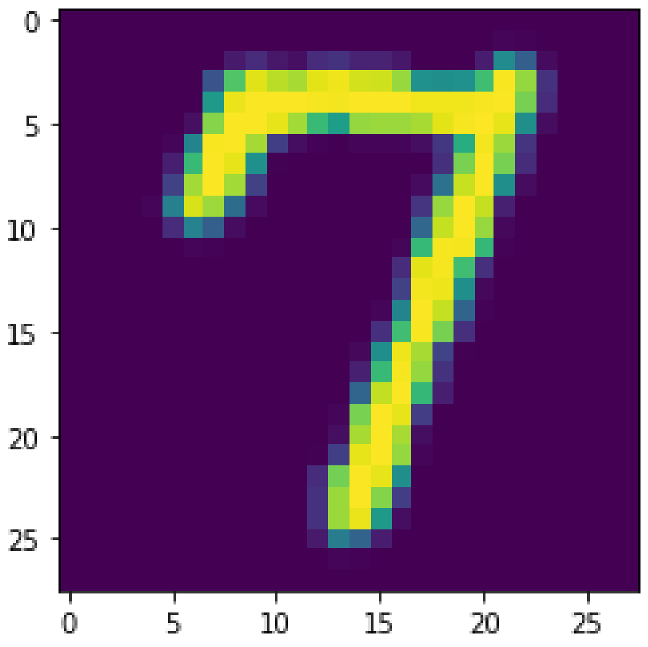
[2]:
```
(28, 28)
```

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 0x1434cd490>
```



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'>
{'5379409360': <shfl.private.data.LabeledData object at 0x1435bd710>}
```

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 [==============================] - 218s 145ms/step - loss: 0.3195 -
accuracy: 0.9297 - val_loss: 0.0489 - val_accuracy: 0.9868
Epoch 2/3
 805/1500 [===============>…] - ETA: 1:36 - loss: 0.0698 -
```

```
accuracy: 0.9811
```

[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 0x1407b8dd0>: [0.038152437657117844, 0.9903249740600586]
```

```
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1407b8a90>: [0.04973132163286209, 0.9871000051498413]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bdcd0>: [0.0493788942694664, 0.9884999990463257]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd510>: [0.050867848098278046, 0.9872249960899353]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd550>: [0.04716002196073532, 0.98785001039505]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd410>: [0.0433327741920948, 0.9880250096321106]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd310>: [0.05515023693442345, 0.9854750037193298]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4090>: [0.05141320824623108, 0.9882749915122986]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4510>: [0.04272490367293358, 0.9891999959945679]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4250>: [0.04068956524133682, 0.9887750148773193]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4b10>: [0.07643583416938782, 0.9836000204086304]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b46d0>: [0.047330476343631744, 0.9873499870300293]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8710>: [0.03926551342010498, 0.991100013256073]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8fd0>: [0.040894605219364166, 0.989425003528595]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8350>: [0.052615705877542496, 0.9869999885559082]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8890>: [0.06546146422624588, 0.9810500144958496]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8990>: [0.04051186516880989, 0.9893249869346619]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a89d0>: [0.04574350267648697, 0.9886500239372253]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a85d0>: [0.04314514994621277, 0.9898999929428101]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8150>: [0.04204494506120682, 0.9897500276565552]
Global model test performance : [0.030116064473986626, 0.9920750260353088]



Accuracy round 1
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1407b8dd0>: [0.04547760635614395, 0.98785001039505]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1407b8a90>: [0.03551230952143669, 0.9905499815940857]
Test performance client <shfl.private.federated_operation.FederatedDataNode
```

```
object at 0x1435bdcd0>: [0.03502069041132927, 0.9903500080108643]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd510>: [0.047038137912750244, 0.988099992275238]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd550>: [0.03282153978943825, 0.9915000200271606]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd410>: [0.03968874365091324, 0.9894750118255615]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435bd310>: [0.04402481019496918, 0.9883000254631042]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4090>: [0.04336558282375336, 0.9881749749183655]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4510>: [0.04375915601849556, 0.9890000224113464]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4250>: [0.04393460229039192, 0.9872499704360962]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b4b10>: [0.04625820368528366, 0.9884999990463257]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435b46d0>: [0.04874780774116516, 0.987375020980835]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8710>: [0.03513509780168533, 0.991100013256073]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8fd0>: [0.06112465262413025, 0.9839249849319458]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8350>: [0.036431532353162766, 0.9906499981880188]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8890>: [0.04016245901584625, 0.989799976348877]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8990>: [0.03226814791560173, 0.991474986076355]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a89d0>: [0.038112442940473557, 0.9901999831199646]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a85d0>: [0.03683720529079437, 0.9905999898910522]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x1435a8150>: [0.03775346651673317, 0.9904249906539917]
Global model test performance : [0.028194550424814224, 0.9926750063896179]
```

```
[ ]:
```