딥러닝을 위해 MATLAB과 TensorFlow/PyTorch 함께 사용하기
김종남 부장, 매스웍스코리아
Interoperability has an impact across different vertical applications

Controls/ MBD workflows: models imported from OSS are a part of a bigger system

Audio/ Signal Processing: (call dataprocessing in MATLAB from Python)
Ways to Interoperate with TensorFlow and PyTorch

- TensorFlow Converter
- ONNX Converter
- Model Exchange
- Python Interface
- MATLAB Engine
- MATLAB-Python Coexecution
MATLAB-Python Co-execution Workflows

1. Data Preparation
   - MATLAB
   - PyTorch
   - TensorFlow

2. Existing Data Processing Pipelines
   - MATLAB
   - PyTorch
   - TensorFlow

3. System Integration
   - Simulink
   - PyTorch
   - TensorFlow

Coexecution
Model Exchange and Complete AI Workflow
Ways to Interoperate with TensorFlow and PyTorch - TensorFlow Converter

TensorFlow Converter  ONNX Converter  Model Exchange

Python Interface  MATLAB Engine  MATLAB-Python Coexecution

TensorFlow  ONNX  Other Frameworks  MATLAB  PyTorch  mxnet
Case 1- Example: Battery Management Demo

- Step 1: Train model in TensorFlow and save the model
- Step 2: Import model into MATLAB and analyze architecture and validate the results
- Step 3: Include into a Simulink model for desktop simulation
- Step 4: Generate CUDA code from imported TensorFlow Model
Step 1: Train model in TensorFlow and save the model

```
import pandas as pd
import tensorflow as tf
import scipy.io as sio
from scipy.io import loadmat

X_train = loadmat('C:/Users/shmitra/Work/Simulink/Battery_SoC_codeGen_MATLAB_Kishen/Battery_SoC_codeGen_MATLAB_Kishen/data/preprocessed_data.mat')['preprocessed_data_mat']
Y_train = loadmat('C:/Users/shmitra/Work/Simulink/Battery_SoC_codeGen_MATLAB_Kishen/Battery_SoC_codeGen_MATLAB_Kishen/data/preprocessed_data.mat')['preprocessed_data_mat']

X_train = X_train["Xinput"]
Y_train = Y_train["Y"]

print(X_train.shape)
print(Y_train.shape)

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=42)

from tensorflow.keras.layers import Input, Dense, Activation, Dropout
```
Step 2: Import and analyze architecture, and validate the results
Step3: Include into a Simulink model for desktop simulation
Step 4: Generate CUDA code from imported TensorFlow Model
Mitsui Chemicals Deploys AI and Automation Systems with TensorFlow and MATLAB

Challenge
Automate visual inspection of sheet-shaped products and ensure ease of use and maintenance of the deployed model

Solution
Import the trained TensorFlow-Keras model into MATLAB using an importer, create a user interface, and deploy it in the field as an application

Key Outcomes
- Reduced visual inspection time by 80%
- Effectively used models trained in other frameworks
- Deployed application with a user interface that anyone can use

"MATLAB solved our problems on the field implementation and saved development time. That led to highly accurate development."
- Shintaro Maekawa, Mitsui Chemicals, Inc.

Model development with Python (TensorFlow-Keras) and efficient onsite implementation of models with MATLAB.
Ways to Interoperate with TensorFlow and PyTorch

Model Exchange

TensorFlow Converter

ONNX Converter

Python Interface

MATLAB Engine

MATLAB-Python Coexecution
Ways to Interoperate with TensorFlow and PyTorch
ONNX Exporting and Importing

**Import ONNX Network as DAGNetwork**
Import a pretrained ONNX network as a DAGNetwork object, and use the imported network to classify an image.

1. Generate an ONNX model of the **squeezenet** convolution neural network.
   ```
   squeezeNet = squeezenet;
   exportONNXNetwork(squeezeNet,'squeezenet.onnx');
   ```
2. Specify the class names.
   ```
   ClassNames = squeezeNet.Layers(end).Classes;
   ```
3. Import the pretrained `squeezenet.onnx`, and specify the classes. By default, `importONNXNetwork` imports the network as a DAGNetwork object.
   ```
   net = importONNXNetwork('squeezenet.onnx',Classes=ClassNames)
   ```
4. Analyze the imported network.
   ```
   analyzeNetwork(net)
   ```
5. Read the image you want to classify and display the size of the image. The image is 384-by-512 pixels and has three color channels (RGB).
Use MATLAB and Python in model training
Ways to Interoperate with TensorFlow and PyTorch
Why Co-execution?

Calling Python from MATLAB

Already working in MATLAB, and:
• Want to reuse existing Python code
• Need functionality that is only available in Python

Calling MATLAB from Python

Already working in Python, and:
• Want to reuse existing MATLAB code
• Need functionality available in MATLAB
• Want to collaborate with MATLAB users
Ways to Interoperate with TensorFlow and PyTorch

TensorFlow Converter
ONNX Converter
Model Exchange

Python Interface
MATLAB Engine
MATLAB-Python Coexecution
Example: Speech Command Recognition – Train in PyTorch, call data processing in MATLAB

- Step 1: Setting up MATLAB engine in Python
- Step 2: Setting up functions to call MATLAB from PyTorch
- Step 3: Preparing data and designing network in PyTorch
- Step 4: Calling MATLAB preprocessing functions from PyTorch training loop
- Step 5: Exporting trained network to ONNX and import ONNX model in MATLAB
Example Workflow

Design and train neural networks

PyTorch

MATLAB Engine

Data Processing

ONNX

importONNXNetwork

Retrain/Inference

Generate Code

Visualize

Simulink Integration
Example Workflow

Design and train neural networks

PyTorch

MATLAB Engine

Data Processing

MATLAB
Step 1: Setting up MATLAB engine in Python

Install the Engine API
At the MATLAB command prompt —

```matlab
cd (fullfile(matlabroot,'extern','engines','python'))
```

```matlab
system('python setup.py install')
```

• Calling MATLAB from Python

Start MATLAB Engine

Start Python, import the module, and start the MATLAB engine:

```python
import matlab.engine
eng = matlab.engine.start_matlab()
```
Step 2: Setting up functions to call MATLAB from PyTorch

```python
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.onnx

import time
import os

cuda = torch.device('cuda')

# start a MATLAB engine
import matlab.engine
MLEngine = matlab.engine.start_matlab()

miniBatchSize = 128.0

# Prepare training dataset
class TrainData(Dataset):
    def __init__(self):
        # Create persistent training dataset in MATLAB
        MLEngine.setupDatasets(miniBatchSize)
        # Set the dataset length to the number of minibatches
        # in the training dataset
        self.len = int(MLEngine.getNumIterationsPerEpoch())

    def __getitem__(self, index):
        # Call MATLAB to get a minibatch of features + labels
        minibatch = MLEngine.extractTrainingFeatures()
        x = torch.FloatTensor(minibatch.get('features'))
        y = torch.FloatTensor(minibatch.get('labels'))
        return x, y

function [ads, batchSize] = setupDatasets(varargin)
    persistent adsTrain miniBatchSize
    if isempty(adsTrain)
        adsTrain = audioDatastore(datafolder, ...
            'IncludeSubfolders',true, ...
            'LabelSource','foldername');
        if nargin == 0
            miniBatchSize = 128;
        else
            miniBatchSize = varargin{1};
        end
    end
    ads = adsTrain;
    batchSize = miniBatchSize;
```
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.onnx

import time
import os

cuda = torch.device('cuda')

# Start a MATLAB engine
import matlab.engine
MLEngine = matlab.engine.start_matlab()
miniBatchSize = 128.0

# Prepare training dataset

class TrainData(Dataset):
    def __init__(self):
        # Create a persistent training dataset in MATLAB
        MLEngine.setupDatasets(miniBatchSize)
        # Set the dataset length to the number of minibatches
        # in the training dataset
        self.len = int(MLEngine.getNumIterationsPerEpoch())

    def __getitem__(self, index):
        # Call MATLAB to get a minibatch of features + labels
        minibatch = MLEngine.extractTrainingFeatures()
        x = torch.FloatTensor(minibatch.get('features'))
        y = torch.FloatTensor(minibatch.get('labels'))
        return x, y

function numIterations = getNumIterationsPerEpoch
[trainingDatastore, batchSize] = setupDatasets;
numIterations = numel(trainingDatastore.Files)/batchSize;
end
Step 3: Preparing data and designing network in PyTorch

Initiate a handle to prepare the data that will be read in the training loop

```python
trainDataset = TrainData()
trainLoader = DataLoader(dataset=trainDataset, batch_size=1)
```

Similar to Datastores in MATLAB

Design the neural network architecture

```python
import torch.nn as nn

class CNN(nn.Module):
    # Constructor
    def __init__(self, out_1=NumF):
        super(CNN, self).__init__()
        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3, padding=1)
        self.batch1 = nn.BatchNorm2d(1)
        self.relu1 = nn.ReLU()

        self.maxpool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=1)
        self.cnn2 = nn.Conv2d(in_channels=1, out_channels=2, kernel_size=3, padding=1)
        self.batch2 = nn.BatchNorm2d(2)
        self.relu2 = nn.ReLU()

        self.maxpool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=1)
        self.cnn3 = nn.Conv2d(in_channels=2, out_channels=3, kernel_size=3, padding=1)
        self.batch3 = nn.BatchNorm2d(3)
        self.relu3 = nn.ReLU()

        self.maxpool3 = nn.MaxPool2d(kernel_size=2, stride=2, padding=1)
        self.cnn4 = nn.Conv2d(in_channels=3, out_channels=4, kernel_size=3, padding=1)
        self.batch4 = nn.BatchNorm2d(4)
        self.relu4 = nn.ReLU()

        self.cnn5 = nn.Conv2d(in_channels=4, out_channels=5, kernel_size=3, padding=1)
        self.batch5 = nn.BatchNorm2d(5)
        self.relu5 = nn.ReLU()
```
Step 4: Calling MATLAB preprocessing functions from PyTorch training loop

```python
for epoch in range(n_epochs):
    if epoch == 29:
        for g in optimizer.param_groups:
            g['lr'] = 3e-5

    count = 0
    for batch in trainLoader:
        count += 1
        print('Epoch ', epoch+1, ' Iteration', count, ' of ', trainDataset.len)
        x = batch[0].cuda()
        y = batch[1].cuda()
        optimizer.zero_grad()
        z = model(torch.squeeze(x.float(), 0))
        loss = criterion(z, torch.squeeze(y.long()))
        loss.backward()
        optimizer.step()
```

```
function [ads, batchSize] = setupDATASET(varargin)
    persistent adTrain minibatchsize
    if isempty(adTrain)
        adTrain = readDatastore(datafolder, ...
            'IncludeSubfolders',true, ... 'LabelSource','foldernames');
        if nargin == 0
            minibatchsize = 128;
        else
            minibatchsize = varargin{1};
        end
    end
    ads = adTrain;
    batchSize = minibatchsize;
end
```
Step 5: Exporting trained network to ONNX and import ONNX model in MATLAB

- In PyTorch, export model to onnx

```python
# Export the trained model to ONNX format
torch.onnx.export(model,
                  torch.empty(1, 96, 50).cuda(),
                  "cmdRecognitionPyTorch.onnx",
                  export_params=True,
                  opset_version=9,
                  do_constant_folding=True,
                  input_names=['input'],
                  output_names=['output'])
```

- Import the model to MATLAB with importONNXNetwork

```matlab
>> cmdRecognitionONNX = importONNXNetwork('cmdRecognition.onnx','OutputLayerType','classification')
```

Warning: Adding a Softmax layer to the imported network. The ONNX network does not include a Softmax, which is required for classification networks.
Ways to Interoperate with TensorFlow and PyTorch

- TensorFlow Converter
- ONNX Converter
- Model Exchange
- Python Interface
- MATLAB Engine
- MATLAB-Python Coexecution
Using TensorFlow Network Design Inside the MATLAB Script

Calling Python from MATLAB
In this example, you invoke TensorFlow training from MATLAB. The training loop is in MATLAB. The neural network model and the gradient/loss computations happen in TensorFlow.

Setup Training Dataset
Set up a training datastore with the desired minibatch size. This is the same function used in the original demo version (call MATLAB from Python).

```matlab
miniBatchSize = 128;
[trainingDatastore, validationDatastore] = setupDatasets(miniBatchSize);
umIterationPerEpoch = numel(trainingDatastore.Files)/miniBatchSize;
```

Compute Validation Data
Get the validation data (similar to original example).

```matlab
validationData = extractValidationFeatures;
validationData.features = permute(validationData.features, [1 3 4 2]);
```

Instantiate the deep learning model. This is a class defined in Python. You will call methods on this object in the training loop.

```matlab
model = py.SpeechCommandRecognition.SpeechCommandRecognition();
```
Using TensorFlow Network Design Inside the MATLAB Script

Instantiate the deep learning model. This is a class defined in Python. You will call methods on this object in the training loop.

```python
model = py.SpeechCommandRecognition.SpeechCommandRecognition();
```

Training Loop

In the training loop, call the method forward to update the weights.

```python
numEpocs = 1;
for epoch = 1:numEpocs
    model.initializeAcc;
    for i = 1:numIterationPerEpoch
        if mod(i,10)==1
            fprintf('Epoch %d - Iteration %d of %d\n',epoch,i,numIterationPerEpoch);
        end
        features = extractTrainingFeatures;
        labels = values.labels';
        model.forward(features, labels, pyargs('training', true));
    end
    model.printAcc;
    z = model.forward(validationData.features, 0);
    z = double(z);
    [m,ni] = max(z,[],2);
    acc = sum((validationData.labels == (m-1)))/numel(m);
    fprintf('Validation accuracy: %f percent\n',100 * acc);
end
```
Using TensorFlow Network Design Inside the MATLAB Script

Instantiate the deep learning model. This is a class defined in Python. You will call methods on this object in the training loop.

```python
model = py.SpeechCommandRecognition.SpeechCommandRecognition();
```

**Training Loop**

In the training loop, call the method forward to update the weights.

```python
numEpochs = 1;
for epoch = 1:numEpochs
    model.initializeAcc;
    for i = 1:numIterationPerEpoch
        if mod(i,10)==1
            fprintf('Epoch %d - Iteration %d of %d
',epoch,i,numIterationPerEpoch);
        end
        values = extractTrainingFeatures;
        features = permute(values.features, [1 3 4 2]);
        labels = values.labels;''
        model.forward(features, labels, pyargs('training', true));
    end
    model.printAcc;
    z = model.forward(validationData.features, 0);
    z = double(z);
    [~,m] = max(z,[],2);
    acc = sum((validationData.labels == (m-1)))/numel(m);
    fprintf('Validation accuracy: %f percent\n',100 * acc);
end
```

```python
def __init__(self):
    super(SpeechCommandRecognition, self).__init__()
    self.model = self.make_model()
    lr = tf.Variable(0.003, trainable=False, dtype=tf.float32)
    self.optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
    self.initializeAcc()

def forward(self, x, y, training=False):
    x = tf.expand_dims(x, 3)
    if training:
        with tf.GradientTape() as tape:
            z = self.model(x)
            loss_value = self.loss(y, z)
            grads = tape.gradient(loss_value, self.model.trainable_variables)
            self.optimizer.apply_gradients(zip(grads, self.model.trainable_variables))
            # Track progress
            self.epoch_loss_avg(loss_value) # Add current batch loss
            # Compare predicted label to actual label
            self.epoch_accuracy(y, self.model(x))
    else:
        return self.model(x)
```
Best for AI model evolution, codegen, and system integration

- Import model from third-party framework to Deep Learning Toolbox
- Use MATLAB’s data labeling/processing/code generation and compiler pipelines
- Integrate model into Simulink using Deep Learning Toolbox blocks or MATLAB Function block
- Export modified model to third-party framework if needed

Co-execution

Used when

- working with Deep Learning models or other Matlab/Python code
- pretrained models cannot be directly imported into MATLAB

Best for encapsulation and reuse of Python code in MATLAB/Simulink

- Use existing data pipelines in Python and train and perform experiment management in MATLAB using apps
- Use TensorFlow/PyTorch for training with MATLAB’s data labeling/processing pipelines
- Create Python API in separate MATLAB function in Simulink
- Use a MATLAB Function block in Simulink to call Python subroutines and models
Thank you