Colab and Google Drive Setup

```
import os, sys
from google.colab import drive
drive.mount('/content/mnt')
nb_path = '/content/notebooks'
os.symlink('/content/mnt/My Drive/Colab Notebooks', nb_path)
sys.path.insert(0,nb_path)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649

Enter your authorization code:
··········
Mounted at /content/mnt
```

```
#! pip install mxnet
!pip install --target=$nb_path mxnet-cu100mkl
```

```
Collecting mxnet-cu100mkl
  Using cached https://files.pythonhosted.org/packages/bb/eb/68921d5ffb80fd5cba483ab0dc955ed4aa257acc5c3b00c05dc03e37
Collecting graphviz<0.9.0,>=0.8.1
  Using cached https://files.pythonhosted.org/packages/53/39/4ab213673844e0c004bed8a0781a0721a3f6bb23eb8854ee75c23642
Requirement already satisfied: requests<3,>=2.20.0 in /usr/local/lib/python3.6/dist-packages (from mxnet-cu100mkl) (2
Requirement already satisfied: numpy<2.0.0,>1.16.0 in /usr/local/lib/python3.6/dist-packages (from mxnet-cu100mkl) (1
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.20.0
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.20.0->mx
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.2
Installing collected packages: graphviz, mxnet-cu100mkl
  Found existing installation: graphviz 0.10.1
    Uninstalling graphviz-0.10.1:
      Successfully uninstalled graphviz-0.10.1
Successfully installed graphviz-0.8.4 mxnet-cu100mkl-1.5.1.post0
```

```
#!pip install --target=$nb.path --upgrade urllib3==1.24
!pip install --target=$nb.path --upgrade folium==0.2.1
```

Help

```
Type:        module
String form: <module '
File:        /usr/lib/
Docstring:
OS routines for NT or

This exports:
  - all functions from
  - os.path is either
  - os.name is either
  - os.curdir is a str
'.')
  - os.pardir is a str
'..')
  - os.sep is the (or
  - os.extsep is the e
  - os.altsep is the a
  - os.pathsep is the
  - os.linesep is the
'\r\n')
  - os.defpath is the
  - os.devnull is the

Programs that import a
portable between diffe
only use functions tha
and opendir), and leav
(e.g., split and join)
```

```
Collecting folium==0.2.1
  Downloading https://files.pythonhosted.org/packages/72/dd/75ced7437bfa7cb9a88b96ee0177953062803c3b4cde411a97d98c35a
     |████████████████████████████████| 71kB 3.6MB/s
Collecting Jinja2
  Downloading https://files.pythonhosted.org/packages/27/24/4f35961e5c669e96f6559760042a55b9bcfcdb82b9bdb3c8753dbe042
     |████████████████████████████████| 133kB 16.1MB/s
Collecting MarkupSafe>=0.23
  Downloading https://files.pythonhosted.org/packages/b2/5f/23e0023be6bb885d00ffbefad2942bc51a620328ee910f64abe5a8d18
Building wheels for collected packages: folium
  Building wheel for folium (setup.py) ... done
  Created wheel for folium: filename=folium-0.2.1-cp36-none-any.whl size=79979 sha256=420914a5d822a1020f8c16a467df865
  Stored in directory: /root/.cache/pip/wheels/b8/09/f0/52d2ef419c2aaf4fb149f92a33e0008bdce7ae816f0dd8f0c5
Successfully built folium
Installing collected packages: MarkupSafe, Jinja2, folium
Successfully installed Jinja2-2.11.1 MarkupSafe-1.1.1 folium-0.2.1
```

1. Import libraries

```
import os
import sys
import numpy as np
import gzip
import pandas as pd
from time import time
print("OS: ", sys.platform)
print("Python: ", sys.version)
# MXnet
import mxnet as mx
from mxnet import nd, autograd
from mxnet import gluon
from mxnet.gluon import nn
print("MXNet version", mx.__version__) # Matteo 1.5.1
# Tensorflow
from sklearn.model_selection import train_test_split
%tensorflow_version 2.x
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
print("Tensorflow version (by Google): ", tf.__version__)
```

```
OS:  linux
Python:  3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0]
MXNet version 1.5.1
TensorFlow 2.x selected.
Tensorflow version (by Google):  2.1.0
```

```
# Check cuda version
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:01_CDT_2018
Cuda compilation tools, release 10.0, V10.0.130
```

```
!nvidia-smi
```

```
Mon Mar  9 07:50:25 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   73C    P0    75W / 149W |    633MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Set GPU usage

```
# MXNET
gpus = mx.test_utils.list_gpus()
print(gpus)
ctx =  [mx.gpu()] if gpus else [mx.cpu(0), mx.cpu(1)]
print(ctx)
```

```
# TENSORFLOW
```

Control reproducibility

The most common form of randomness used in neural networks is the random initialization of the network weights. Although randomness can be used in other areas, here is just a short list:

- Randomness in Initialization, such as weights.
- Randomness in Regularization, such as dropout.
- Randomness in Layers, such as word embedding.
- Randomness in Optimization, such as stochastic optimization.

source: https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

```
import random
np.random.seed(42)
random.seed(42)
for computing_unit in ctx:
    mx.random.seed(42, ctx = computing_unit)
tf.random.set_seed(42)
```

2. Read dataset - General Train/Test split

```
def read_mnist(images_path: str, labels_path: str):
    #mnist_path = "data/mnist/"
    #images_path = mnist_path + images_path
    folder = os.getcwd() + "/notebooks/"
    print(images_path)
    with gzip.open(folder + labels_path, 'rb') as labelsFile:
        labels = np.frombuffer(labelsFile.read(), dtype=np.uint8, offset=8)

    with gzip.open(folder + images_path,'rb') as imagesFile:
        length = len(labels)
        # Load flat 28x28 px images (784 px), and convert them to 28x28 px
        features = np.frombuffer(imagesFile.read(), dtype=np.uint8, offset=16) \
                        .reshape(length, 784) \
                        .reshape(length, 28, 28, 1)
    return features, labels
```

```
os.listdir(os.getcwd() + "/notebooks")
```

```
['kernel.ipynb',
 'Untitled0.ipynb',
 'Untitled1.ipynb',
 'Untitled2.ipynb',
 'ComparisonOfClusteringMethods (1).ipynb',
 'HierarchicalClustering.ipynb',
 'ComparisonOfClusteringMethods.ipynb',
 'SilhouetteAnalysisIris.ipynb',
 'SilhouetteAnalysis.ipynb',
 'Untitled3.ipynb',
 'CPU - Fundamental Ops - Linear Algebra.ipynb',
 'GPU - Fundamental Ops - Linear Algebra.ipynb',
 'Copy of 0_colab',
 'Copy of 1_data',
 'Copy of 2_keras',
 'Copy of 4_predict',
 'Copy of 3_eager',
 'Teardown - MXnet - TF - Optimization.ipynb',
 'graphviz-0.8.4.dist-info',
 'chardet-3.0.4.dist-info',
 'urllib3',
 'dmlc_tracker',
 'urllib3-1.25.8.dist-info',
 'requests',
 'requests-2.23.0.dist-info',
 'numpy',
 'chardet',
 'graphviz',
 'idna',
 'mxnet',
 'mxnet_cu100mkl-1.5.1.post0.dist-info',
 'certifi',
 'certifi-2019.11.28.dist-info',
 'idna-2.9.dist-info',
 'numpy-1.18.1.dist-info',
 'bin',
 't10k-labels-idx1-ubyte.gz',
 't10k-images-idx3-ubyte.gz',
 'train-images-idx3-ubyte.gz',
 'train-labels-idx1-ubyte.gz']
```

```
#from google.colab import files
#uploaded = files.upload()
```

```
! ls
```

```
mnt  notebooks  sample_data
```

```
# LOAD TRAIN AND TEST ALREADY SPLIT
train = {}
test = {}
train['features'], train['labels'] = read_mnist('train-images-idx3-ubyte.gz', 'train-labels-idx1-ubyte.gz')
test['features'], test['labels'] = read_mnist('t10k-images-idx3-ubyte.gz', 't10k-labels-idx1-ubyte.gz')
print(test['features'].shape[0], '-> # of test images.')
print(train['features'].shape[0], '-> # of training images (train + validation).')
# CREATE TRAIN AND VALIDATION SPLIT
validation = {}
train['features'], validation['features'], train['labels'], validation['labels'] = train_test_split(train['features'], tr
print("    ", train['features'].shape[0], '-> # of (actual) training images.')
print("    ", validation['features'].shape[0], '-> # of validation images.')
```

```
train-images-idx3-ubyte.gz
t10k-images-idx3-ubyte.gz
10000 -> # of test images.
60000 -> # of training images (train + validation).
     48000 -> # of (actual) training images.
     12000 -> # of validation images.
```

3. Create a reader for each Framework

```
# GENERAL PARAMETERS
EPOCHS = 15
BATCH_SIZE = 200
```

```
# MXNET
# convert from NHWC to NCHW that is used by MXNET
# https://stackoverflow.com/questions/37689423/convert-between-nhwc-and-nchw-in-tensorflow
X_train_mx = mx.ndarray.transpose(mx.nd.array(train['features']), axes=(0, 3, 1, 2))
y_train_mx = mx.nd.array(train['labels'])
X_validation_mx = mx.ndarray.transpose(mx.nd.array(validation['features']), axes=(0, 3, 1, 2))
y_validation_mx = mx.nd.array(validation['labels'])
X_test_mx = mx.ndarray.transpose(mx.nd.array(test['features']), axes=(0, 3, 1, 2))
y_test_mx = mx.nd.array(test['labels'])
# create data iterator
train_data_mx = mx.io.NDArrayIter(X_train_mx.asnumpy(), y_train_mx.asnumpy(), BATCH_SIZE, shuffle=True)
val_data_mx = mx.io.NDArrayIter(X_validation_mx.asnumpy(), y_validation_mx.asnumpy(), BATCH_SIZE)
test_data_mx = mx.io.NDArrayIter(X_test_mx.asnumpy(), y_test_mx.asnumpy(), BATCH_SIZE)
test_data_mx = mx.io.NDArrayIter(X_test_mx.asnumpy(), y_test_mx.asnumpy(), BATCH_SIZE)
```

```
X_train_mx.shape
```

```
(48000, 1, 28, 28)
```

```
type(X_train_mx.asnumpy())
```

```
numpy.ndarray
```

```
# TENSORFLOW
# convert in multiple output for tensorflow
X_train_tf, y_train_tf = train['features'], to_categorical(train['labels'])
X_validation_tf, y_validation_tf = validation['features'], to_categorical(validation['labels'])
# create data generator
train_generator_tf = ImageDataGenerator().flow(X_train_tf, y_train_tf, batch_size=BATCH_SIZE)
validation_generator_tf = ImageDataGenerator().flow(X_validation_tf, y_validation_tf, batch_size=BATCH_SIZE)
```

```
X_train_tf.shape
```

```
(48000, 28, 28, 1)
```

4. Create models

```
# MXNET -> GLUON
# IDENTICAL TO LeNet paper: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
model_mx = nn.HybridSequential()
model_mx.add(nn.Conv2D(channels=6, kernel_size=5, activation='relu'),
        nn.AvgPool2D(pool_size=2, strides=2),
        nn.Conv2D(channels=16, kernel_size=3, activation='relu'),
        nn.AvgPool2D(pool_size=2, strides=2),
        nn.Flatten(),
        nn.Dense(120, activation="relu"),
        nn.Dense(84, activation="relu"),
        nn.Dense(10))
```

```
# TENSORFLOW -> KERAS
model_tf = keras.Sequential()
init_tf = tf.keras.initializers.GlorotNormal(seed=1)
model_tf.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(28,28,1), kernel_initializer =
model_tf.add(layers.AveragePooling2D(pool_size=(2, 2), strides=2))
model_tf.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu', kernel_initializer = init_tf, bias_initiali
model_tf.add(layers.AveragePooling2D(pool_size=(2, 2), strides=2))
model_tf.add(layers.Flatten())
model_tf.add(layers.Dense(units=120, activation='relu', kernel_initializer = init_tf, bias_initializer = init_tf))
model_tf.add(layers.Dense(units=84, activation='relu', kernel_initializer = init_tf, bias_initializer = init_tf))
model_tf.add(layers.Dense(units=10, activation = 'softmax', kernel_initializer = init_tf, bias_initializer = init_tf))
#model.summary()
```

```
#help(layers.Dense)
```

Optimization on/off

```
# MXNET
model_mx.hybridize()
```

```
# TENSORFLOW
tf.config.optimizer.set_jit(True)
```

5. Train Models

```
%%time
# MXNET
def training_procedure(handwritten_net, train_data):
    global EPOCHS
    global ctx
    handwritten_net.initialize(mx.init.Xavier(), ctx=ctx, force_reinit=True)
    #handwritten_net(init = mx.init.Xavier(), ctx=ctx)
    optim = mx.optimizer.Adam(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08, lazy_update=True)
    trainer = gluon.Trainer(handwritten_net.collect_params(), optim)
    # Use Accuracy as the evaluation metric.
    metric = mx.metric.Accuracy()
    softmax_cross_entropy_loss = gluon.loss.SoftmaxCrossEntropyLoss()

    for i in range(EPOCHS):
        # Reset the train data iterator.
        train_data.reset()
        train_data.reset()
        # Loop over the train data iterator.
        for batch in train_data:
            # Splits train data into multiple slices along batch_axis
            # and copy each slice into a context.
            data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
            # Splits train labels into multiple slices along batch_axis
            # and copy each slice into a context.
            label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
            outputs = []
            # Inside training scope
            with autograd.record():
                for x, y in zip(data, label):
                    z = handwritten_net(x)
                    # Computes softmax cross entropy loss.
                    loss = softmax_cross_entropy_loss(z, y)
                    # Backpropogate the error for one iteration.
                    loss.backward()
                    outputs.append(z)
            # Updates internal evaluation
            metric.update(label, outputs)
            # Make one step of parameter update. Trainer needs to know the
            # batch size of data to normalize the gradient by 1/batch_size.
            trainer.step(batch.data[0].shape[0])
        # Gets the evaluation result.
        name, acc = metric.get()
        # Reset evaluation result to initial state.
        metric.reset()
        print('training acc at epoch %d: %s=%f'%(i, name, acc))
    return handwritten_net

trained_model_mx = training_procedure(model_mx, train_data_mx)
```

```
training acc at epoch 0: accuracy=0.877375
training acc at epoch 1: accuracy=0.967458
training acc at epoch 2: accuracy=0.977187
training acc at epoch 3: accuracy=0.983583
training acc at epoch 4: accuracy=0.986229
training acc at epoch 5: accuracy=0.987563
training acc at epoch 6: accuracy=0.991250
training acc at epoch 7: accuracy=0.991521
training acc at epoch 8: accuracy=0.992771
training acc at epoch 9: accuracy=0.994062
training acc at epoch 10: accuracy=0.993708
training acc at epoch 11: accuracy=0.993729
training acc at epoch 12: accuracy=0.995271
training acc at epoch 13: accuracy=0.995313
training acc at epoch 14: accuracy=0.995146
CPU times: user 25.5 s, sys: 5.1 s, total: 30.6 s
Wall time: 21.3 s
```

```
%%time
# TENSORFLOW
chosen_tf_optimizer = keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model_tf.compile(loss=keras.losses.categorical_crossentropy, optimizer=chosen_tf_optimizer, metrics=['accuracy'])
steps_per_epoch = X_train_tf.shape[0]//BATCH_SIZE
validation_steps = X_validation_tf.shape[0]//BATCH_SIZE
model_tf.fit_generator(train_generator_tf, steps_per_epoch=steps_per_epoch, epochs=EPOCHS,
                    validation_data=validation_generator_tf, validation_steps=validation_steps,
                    shuffle=True, callbacks=[])
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 240 steps, validate for 60 steps
Epoch 1/15
240/240 [==============================] - 6s 26ms/step - loss: 0.7760 - accuracy: 0.8669 - val_loss: 0.1155 - val_ac
Epoch 2/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0918 - accuracy: 0.9718 - val_loss: 0.0777 - val_ac
Epoch 3/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0636 - accuracy: 0.9810 - val_loss: 0.0687 - val_ac
Epoch 4/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0488 - accuracy: 0.9851 - val_loss: 0.0589 - val_ac
Epoch 5/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0397 - accuracy: 0.9877 - val_loss: 0.0603 - val_ac
Epoch 6/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0327 - accuracy: 0.9897 - val_loss: 0.0559 - val_ac
Epoch 7/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0280 - accuracy: 0.9908 - val_loss: 0.0458 - val_ac
Epoch 8/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0248 - accuracy: 0.9920 - val_loss: 0.0572 - val_ac
Epoch 9/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0221 - accuracy: 0.9928 - val_loss: 0.0500 - val_ac
Epoch 10/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0181 - accuracy: 0.9939 - val_loss: 0.0623 - val_ac
Epoch 11/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0200 - accuracy: 0.9931 - val_loss: 0.0498 - val_ac
Epoch 12/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0147 - accuracy: 0.9952 - val_loss: 0.0693 - val_ac
Epoch 13/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0147 - accuracy: 0.9949 - val_loss: 0.0561 - val_ac
Epoch 14/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0149 - accuracy: 0.9948 - val_loss: 0.0516 - val_ac
Epoch 15/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0116 - accuracy: 0.9959 - val_loss: 0.0528 - val_ac
CPU times: user 50.4 s, sys: 4.65 s, total: 55.1 s
Wall time: 40.3 s
```

6. Evaluate models

```
%%time
```

```
# MXNET
# TEST THE NETWORK
metric = mx.metric.Accuracy()
# Reset the test data iterator.
test_data_mx.reset()
# Loop over the test data iterator.
for batch in test_data_mx:
    # Splits test data into multiple slices along batch_axis
    # and copy each slice into a context.
    data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
    # Splits validation label into multiple slices along batch_axis
    # and copy each slice into a context.
    label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
    outputs = []
    for x in data:
        outputs.append(model_mx(x))
    # Updates internal evaluation
    metric.update(label, outputs)
print('MXnet - Test %s : %f'%metric.get())
assert metric.get()[1] > 0.90
```

```
MXnet - Test accuracy : 0.984900
CPU times: user 188 ms, sys: 30.5 ms, total: 218 ms
Wall time: 151 ms
```

```
%%time
# TENSORFLOW
score = model_tf.evaluate(test['features'], to_categorical(test['labels']), verbose=0)
#print('Test loss:', score[0])
print('TensorFlow - Test accuracy:', score[1])
assert score[1] > 0.90
```

```
TensorFlow - Test accuracy: 0.9864
CPU times: user 1.25 s, sys: 192 ms, total: 1.44 s
Wall time: 1.19 s
```