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 1.6.0
TensorFlow 2.x selected.
Tensorflow version (by Google):  2.1.0
```

```
#! pip install mxnet
```

Set GPU usage

```
# MXNET
```

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

```
[cpu(0), cpu(1)]
```

```
# 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
    print(images_path)
    with gzip.open(labels_path, 'rb') as labelsFile:
        labels = np.frombuffer(labelsFile.read(), dtype=np.uint8, offset=8)
```

```
    with gzip.open(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
```

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

Choose Files 4 files

- **train-labels-idx1-ubyte.gz**(application/x-gzip) - 28881 bytes, last modified: 29/12/2019 - 100% done
- **t10k-images-idx3-ubyte.gz**(application/x-gzip) - 1648877 bytes, last modified: 29/12/2019 - 100% done
- **train-images-idx3-ubyte.gz**(application/x-gzip) - 9912422 bytes, last modified: 29/12/2019 - 100% done
- **t10k-labels-idx1-ubyte.gz**(application/x-gzip) - 4542 bytes, last modified: 29/12/2019 - 100% done

```
Saving train-labels-idx1-ubyte.gz to train-labels-idx1-ubyte.gz
Saving t10k-images-idx3-ubyte.gz to t10k-images-idx3-ubyte.gz
Saving train-images-idx3-ubyte.gz to train-images-idx3-ubyte.gz
Saving t10k-labels-idx1-ubyte.gz to t10k-labels-idx1-ubyte.gz
```

```
! ls
```

```
sample_data                  train-images-idx3-ubyte.gz
t10k-images-idx3-ubyte.gz    train-labels-idx1-ubyte.gz
t10k-labels-idx1-ubyte.gz
```

```
# 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'], train['labels'], test_size=0
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)
```

```
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 = init_tf, bias_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_initializer = init_tf))
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()
        # 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])
```

```
            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=0.877313
training acc at epoch 1: accuracy=0.967500
training acc at epoch 2: accuracy=0.976854
training acc at epoch 3: accuracy=0.982917
training acc at epoch 4: accuracy=0.986208
training acc at epoch 5: accuracy=0.987750
training acc at epoch 6: accuracy=0.990167
training acc at epoch 7: accuracy=0.991979
training acc at epoch 8: accuracy=0.992771
training acc at epoch 9: accuracy=0.993792
training acc at epoch 10: accuracy=0.994708
training acc at epoch 11: accuracy=0.994417
training acc at epoch 12: accuracy=0.994125
training acc at epoch 13: accuracy=0.994979
training acc at epoch 14: accuracy=0.996167
CPU times: user 3min 43s, sys: 1.57 s, total: 3min 44s
Wall time: 2min 45s
```

```
%%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:From <timed exec>:8: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed ir
Instructions for updating:
Please use Model.fit, which supports generators.
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 [==============================] - 19s 80ms/step - loss: 0.7753 - accuracy: 0.8673 - val_loss: 0.1145 - val_accuracy: 0.9644
Epoch 2/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0909 - accuracy: 0.9717 - val_loss: 0.0784 - val_accuracy: 0.9753
Epoch 3/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0641 - accuracy: 0.9806 - val_loss: 0.0694 - val_accuracy: 0.9782
Epoch 4/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0493 - accuracy: 0.9847 - val_loss: 0.0601 - val_accuracy: 0.9808
Epoch 5/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0396 - accuracy: 0.9881 - val_loss: 0.0633 - val_accuracy: 0.9811
Epoch 6/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0330 - accuracy: 0.9896 - val_loss: 0.0569 - val_accuracy: 0.9830
Epoch 7/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0269 - accuracy: 0.9917 - val_loss: 0.0470 - val_accuracy: 0.9865
Epoch 8/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0255 - accuracy: 0.9916 - val_loss: 0.0575 - val_accuracy: 0.9834
Epoch 9/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0219 - accuracy: 0.9929 - val_loss: 0.0614 - val_accuracy: 0.9827
Epoch 10/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0211 - accuracy: 0.9932 - val_loss: 0.0543 - val_accuracy: 0.9842
Epoch 11/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0181 - accuracy: 0.9937 - val_loss: 0.0545 - val_accuracy: 0.9854
Epoch 12/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0153 - accuracy: 0.9955 - val_loss: 0.0691 - val_accuracy: 0.9821
Epoch 13/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0163 - accuracy: 0.9941 - val_loss: 0.0553 - val_accuracy: 0.9847
Epoch 14/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0127 - accuracy: 0.9960 - val_loss: 0.0500 - val_accuracy: 0.9876
Epoch 15/15
240/240 [==============================] - 18s 76ms/step - loss: 0.0128 - accuracy: 0.9959 - val_loss: 0.0575 - val_accuracy: 0.9858
CPU times: user 8min 3s, sys: 12.6 s, total: 8min 16s
Wall time: 4min 34s
```

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 : 0.986000
CPU times: user 1.5 s, sys: 14 ms, total: 1.51 s
Wall time: 852 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: 0.9867
CPU times: user 4.1 s, sys: 96.5 ms, total: 4.2 s
Wall time: 2.48 s
```