In [5]:

```
'''Trains a simple deep NN on the MNIST dataset.

Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''

from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
60000 train samples
10000 test samples
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_3 (Dense)              (None, 512)               401920

 dropout_2 (Dropout)          (None, 512)               0

 dense_4 (Dense)              (None, 512)               262656

 dropout_3 (Dropout)          (None, 512)               0

 dense_5 (Dense)              (None, 10)                5130

=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 6s 11ms/step - loss: 0.2469 - accuracy: 0.
9240 - val_loss: 0.1025 - val_accuracy: 0.9670
Epoch 2/20
469/469 [==============================] - 6s 12ms/step - loss: 0.1014 - accuracy: 0.
9696 - val_loss: 0.0863 - val_accuracy: 0.9753
Epoch 3/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0744 - accuracy: 0.
9772 - val_loss: 0.0698 - val_accuracy: 0.9804
Epoch 4/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0594 - accuracy: 0.
9815 - val_loss: 0.0757 - val_accuracy: 0.9793
Epoch 5/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0510 - accuracy: 0.
9853 - val_loss: 0.0722 - val_accuracy: 0.9802
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0433 - accuracy: 0.
9879 - val_loss: 0.0720 - val_accuracy: 0.9826
Epoch 7/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0394 - accuracy: 0.
9883 - val_loss: 0.0827 - val_accuracy: 0.9813
Epoch 8/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0347 - accuracy: 0.
9897 - val_loss: 0.0850 - val_accuracy: 0.9821
Epoch 9/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0329 - accuracy: 0.
9905 - val_loss: 0.0926 - val_accuracy: 0.9819
Epoch 10/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0283 - accuracy: 0.
9918 - val_loss: 0.0886 - val_accuracy: 0.9832
Epoch 11/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0280 - accuracy: 0.
9918 - val_loss: 0.0906 - val_accuracy: 0.9825
Epoch 12/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0270 - accuracy: 0.
9926 - val_loss: 0.0861 - val_accuracy: 0.9845
Epoch 13/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0232 - accuracy: 0.
9932 - val_loss: 0.1013 - val_accuracy: 0.9835
```

```
Epoch 14/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0242 - accuracy: 0.
9935 - val_loss: 0.1047 - val_accuracy: 0.9831
Epoch 15/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0217 - accuracy: 0.
9941 - val_loss: 0.1067 - val_accuracy: 0.9842
Epoch 16/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0220 - accuracy: 0.
9939 - val_loss: 0.1230 - val_accuracy: 0.9816
Epoch 17/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0199 - accuracy: 0.
9946 - val_loss: 0.1218 - val_accuracy: 0.9820
Epoch 18/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0191 - accuracy: 0.
9949 - val_loss: 0.1161 - val_accuracy: 0.9835
Epoch 19/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0181 - accuracy: 0.
9954 - val_loss: 0.1294 - val_accuracy: 0.9816
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0189 - accuracy: 0.
9950 - val_loss: 0.1289 - val_accuracy: 0.9827
Test loss: 0.12885063886642456
Test accuracy: 0.982699990272522
```

In [ ]:

```
---
title: Assignment 1
subtitle: Computer performance, reliability, and scalability calculation
author: Shaquiel Pashtunyar
03/19/2023
---

## 1.2

#### a. Data Sizes

| Data Item                                    | Size per Item |
|----------------------------------------------|--------------:|
| 128 character message.                       | 128 Byte
| 1024x768 PNG image                           | 4.72 MB       |https://toolstud.io/phot
| 1024x768 RAW image                           | 1.18 MB       |
| HD (1080p) HEVC Video (15 minutes)           | 2500MB      |https://toolstud.io/video/
| HD (1080p) Uncompressed Video (15 minutes)   | 15600 MB       |
| 4K UHD HEVC Video (15 minutes)               | 2550 MB       |170MB per 60 seconds (1
| 4k UHD Uncompressed Video (15 minutes)       | 3660 MB       |
| Human Genome (Uncompressed)                  |  6.27 GB      |https://www.google.com/s

#### b. Scaling

|                                              | Size      | # HD |
|----------------------------------------------|---------:|-----:|
| Daily Twitter Tweets (Uncompressed)          | 64GB      | 1    | 128*500 64,00bytes=0.0
| Daily Twitter Tweets (Snappy Compressed)     | 37.5      | 1    |  64/1.7
| Daily Instagram Photos                       | 354TB     |   36 |  4.72*75000000
| Daily YouTube Videos                         | 500TB     |  50  | 500  * 60  = 30000 mir
| Yearly Twitter Tweets (Uncompressed)         | 23.36TB   |   3  | 64 GB * 365 = 23360 GB
| Yearly Twitter Tweets (Snappy Compressed)    | 13.7      |   2  | 37.65 GB * 365 = 13742
| Yearly Instagram Photos                      | 41062.5TB|  4107 |112.5 TB (daily) * 365
| Yearly YouTube Videos                        | 182500TB |  18250 | 500 TB * 365 = 18250

#### c. Reliability
```

```
|                                    | # HD | # Failures |
|------------------------------------|-----:|-----------:|
| Twitter Tweets (Uncompressed)      | 3    |   0.0255   |
| Twitter Tweets (Snappy Compressed) | 2    |   0.017    |
| Instagram Photos                   | 4107 |   ~35      |
| YouTube Videos                     | 18250|   ~3155    |


Failure rate = 0.85% (https://www.backblaze.com/b2/hard-drive-test-data.html)
0.0085 * 3 = 0.0255
0.0085 * 2 = 0.017
0.0085 * 4107 = 34.9095
0.0085 * 18250 = 155.125

#### d. Latency

|                           | One Way Latency      |
|---------------------------|---------------------:|
| Los Angeles to Amsterdam  | 140 ms               |
| Low Earth Orbit Satellite | 600 ms               |
| Geostationary Satellite   | 240 ms               |
| Earth to the Moon         | 2560 ms              |
| Earth to Mars             | 13 minutes           |


https://wondernetwork.com/pings
https://www.omniaccess.com/leo/#:~:text=The%20GEO%20latency%20is%20of,and%20an%20esser
https://www.satsig.net/latency.htm
https://en.wikipedia.org/wiki/Earth%E2%80%93Moon%E2%80%93Earth_communication#:~:text=F
https://blogs.esa.int/mex/2012/08/05/time-delay-between-mars-and-earth
```

In [7]:

```
#
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

import sys
from random import random
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
```

```
    spark = SparkSession\
        .builder\
        .appName("PythonPi")\
        .getOrCreate()

    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 <= 1 else 0

    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).reduce(
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 34
     26 """
     27     Usage: pi [partitions]
     28 """
     29 spark = SparkSession\
     30     .builder\
     31     .appName("PythonPi")\
     32     .getOrCreate()
---> 34 partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
     35 n = 100000 * partitions
     37 def f(_):

ValueError: invalid literal for int() with base 10: '-f'
```

In [ ]:

In [ ]: