Harvard John A. Paulson
School of Engineering
and Applied Sciences

The Lottery Ticket Hypothesis (LTH)

David Assaraf, Gael Ancel, Tale Lokvenec, Raphael Pellegrin

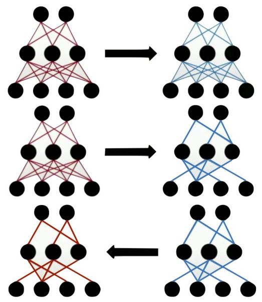
CS205 2021

05/10/2021

I - Problem Statement: Overview of the LTH

The LTH builds on the notion of **network pruning**, which is used to **reduce the storage costs and computational requirements of dealing with the network**. The pruning algorithm proposed is called Iterative Magnitude Pruning (IMP):

- Begin with a Neural Network where **each connection has been set to a random weight**.
- **Train** the Neural Network
- Given a threshold t, remove the superfluous structure by **pruning weights** (sparse pruning): look at the magnitude of the weights and prune the weights with magnitude lower than the threshold.
- **Reset the remaining weights** to their value at a given epoch of choice, and we retrain the sparse subnetwork until the end.



The objective of this procedure is to find the most sparse sub architecture (called the winning ticket) of the initial Neural Network that will match the initial generalization ability. Studies also showed that this ticket generalizes across dataset.

I - Problem Statement: two loops to parallelize

We have two loops to parallelize:

- study different **possible thresholds** for our masks (a bigger threshold means that we throw away more weights).
- decide on the **five epochs** which we will use as our ‘late resetting’ when we reset the weights of our subnetwork (ie a masked network).

Thresholds: we trained our initial architecture and studied the magnitude of the weights. We decided to **keep the 65, 70, 75, 80, 85, …, 99 percentiles** as our thresholds.

For example, using the 65 percentile as a threshold, we throw away 65 percent of the weights and are left with a subnetwork whose size is 35 percent of the size original Neural Network.

We keep subnetworks that are between 1% and 35% of the original size and retrain them independently from epochs 5, 10, 15, 20, 25

I - Problem Statement: Numerical complexity

The numerical complexity of doing late-resetting and masking is **O(MNt)=O(100t)**.

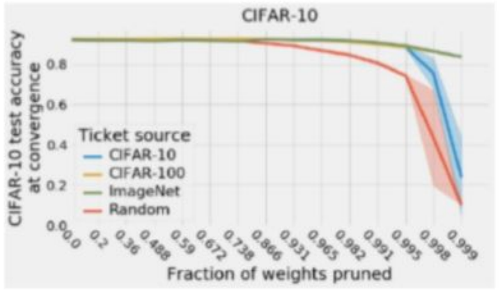
- **M=20 is the number of thresholds** for our masks (each mask gives us one subnetwork). For every masked architecture, we need to perform several trainings from several epochs
- **N=5 is the length of the trellis for late resetting**
- **t is the average time to train a network** (we actually use sparse subnetworks, so they train faster than the original one). t = 26 h 15 min

Amdahl's law states that we have a theoretical speed up of 20.

II - Need for Big Data: ImageNet

ImageNet Dataset:

- Total number of training images: **1.23M**
- Total number of validation images: **100k**
- Total number of test images: **50k**
- Size: **157.3 GB**
- Average image resolution (downloaded): **469x387**
- Average image resolution (preprocessed): **227x227**



Why ImageNet ?

- Winning tickets are generalizable across datasets
- Big datasets like ImageNet will result in a very general winning ticket

Harvard John A. Paulson School of Engineering and Applied Sciences

II - Need for Big Compute: MobileNet

MobileNetV2 Architecture:

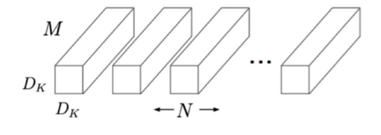
- Depthwise separable convolutions instead of regular convolution
- Total number of parameters: **3.4M**
- Number of multiply-adds (MAdds) per forward pass: **300M**

→ **GPU Accelerated Computing (4 GPUs** single worker node**)**

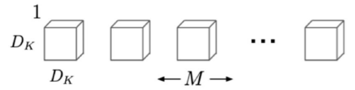
Two **Nested For Loops** to find the Lottery Ticket Hypothesis

- Outer **for loop**: iterate over different masks (pruning thresholds)
- Inner **for loop**: iterate over the range of late resetting epochs
  - ○ Train a sparse MobileNetV2 CNN per each inner loop iteration

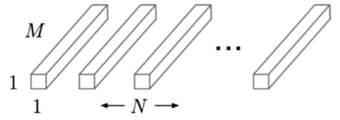
→ **Distributed Computing (20 worker nodes)**



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Why MobileNet ?

- Need Deep Architectures in order to extract relevant features
- Computational savings vs other usual architectures

Harvard John A. Paulson School of Engineering and Applied Sciences

III - Programming model and infrastructure

We use **FAS RC (take advantage of the SCRATCH space [300+GB] and the ease of allocating several nodes for MPI).**

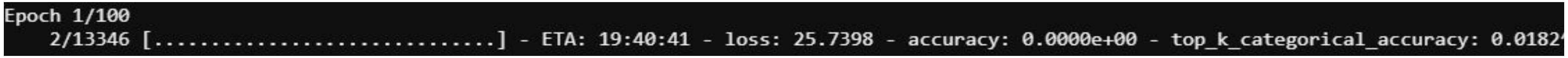
- Python 3.8.5, mpi4py 3.0.3, pyspark 3.1.1, Apache maven 3.8.1, java 1.8.0_45
- We used **Spark-Tensorflow connector and Standalone Spark** mode to convert the data from TF Tensors to RDD and process it in an offline manner
- We use **SLURM Job Arrays** for communication between our nodes and Python Multiprocessing for parallelization within a node
- Train using **TensorFlow 2.0 (leveraging cuda and cudnn)**
- **Objective: End solution comprises 20 worker nodes, each one will have 4 GPUs TESLA K80 with 11.5 GB memory and 64 CPUs**

Single CPU

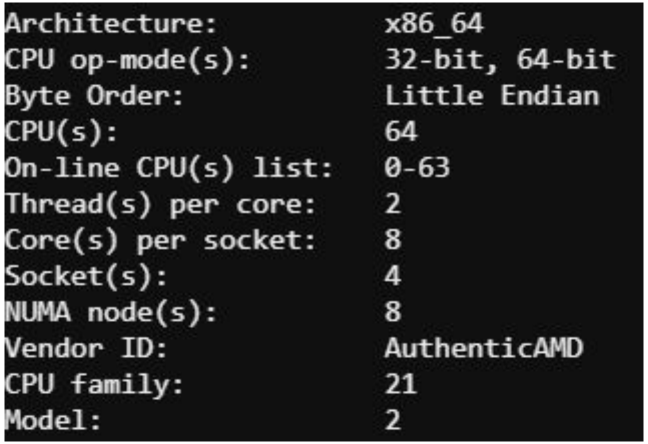
IV - Effective Parallelization of the Training of a CNN on Large Scale Data

Learning Hyperparameters and their influence on the training time:

- Batch size : arbitrary
- Learning Rate : arbitrary
- Number of Epochs : 100



```
Epoch 1/100
    2/13346 [............................] - ETA: 19:40:41 - loss: 25.7398 - accuracy: 0.0000e+00 - top_k_categorical_accuracy: 0.0182
```



```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             4
NUMA node(s):          8
Vendor ID:             AuthenticAMD
CPU family:            21
Model:                 2
```

- 1 epoch = 20 hours
- 1 model = 100 epochs = 2000 hours
- 100 models = 200000 hours = ~8000 days

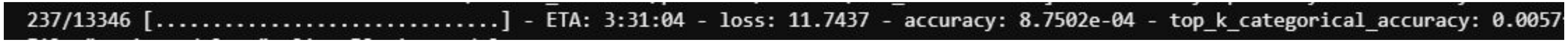
Harvard John A. Paulson School of Engineering and Applied Sciences

Single GPU

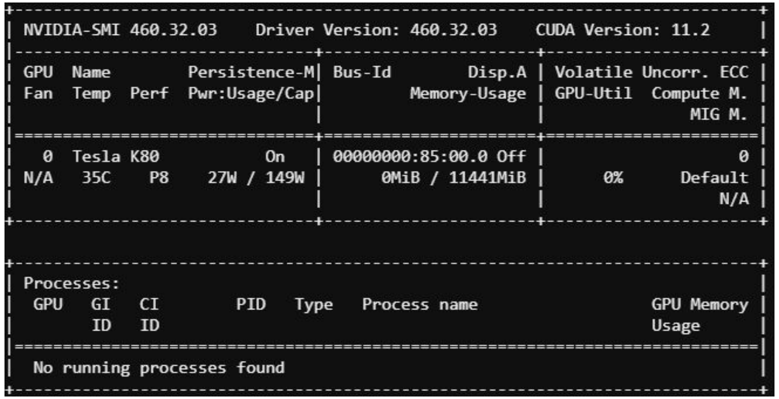
IV - Effective Parallelization of the Training of a CNN on Large Scale Data

Learning Hyperparameters and their influence on the training time:

- Batch size : arbitrary
- Learning Rate : arbitrary
- Number of Epochs : 100



```
237/13346 [..............................] - ETA: 3:31:04 - loss: 11.7437 - accuracy: 8.7502e-04 - top_k_categorical_accuracy: 0.0057
```



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000000:85:00.0 Off |                    0 |
| N/A   35C    P8    27W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- 1 epoch = 4 hours
- 1 model = 100 epochs = 400 hours
- 100 models = 40000 hours = ~1650 days

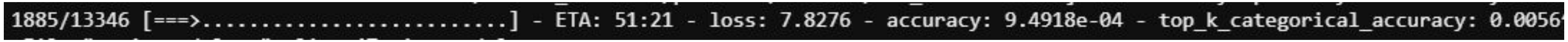
Harvard John A. Paulson School of Engineering and Applied Sciences

4 GPUs

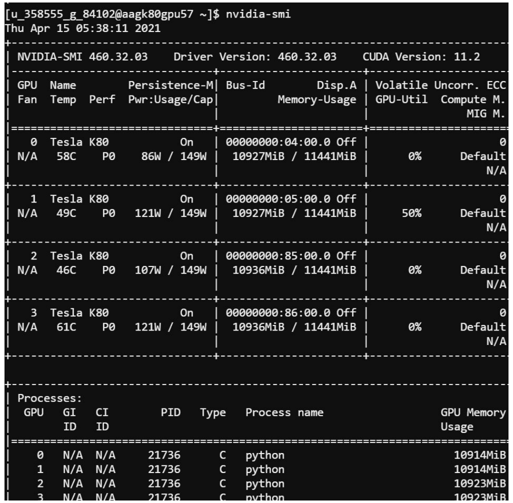
IV - Effective Parallelization of the Training of a CNN on Large Scale Data

Learning Hyperparameters and their influence on the training time:

- Batch size : arbitrary
- Learning Rate : arbitrary
- Number of Epochs : 100



```
1885/13346 [===>..........................] - ETA: 51:21 - loss: 7.8276 - accuracy: 9.4918e-04 - top_k_categorical_accuracy: 0.0056
```



```
[u_358555_g_84102@aagk80gpu57 ~]$ nvidia-smi
Thu Apr 15 05:38:11 2021
+-----------------------------------------------------------------------------
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2
|-------------------------------+----------------------+----------------------
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|                               |                      |               MIG M.
|===============================+======================+======================
|   0  Tesla K80           On   | 00000000:04:00.0 Off |                    0
| N/A   58C    P0    86W / 149W |  10927MiB / 11441MiB |      0%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------
|   1  Tesla K80           On   | 00000000:05:00.0 Off |                    0
| N/A   49C    P0   121W / 149W |  10927MiB / 11441MiB |     50%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------
|   2  Tesla K80           On   | 00000000:85:00.0 Off |                    0
| N/A   46C    P0   107W / 149W |  10936MiB / 11441MiB |      0%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------
|   3  Tesla K80           On   | 00000000:86:00.0 Off |                    0
| N/A   61C    P0   121W / 149W |  10936MiB / 11441MiB |      0%      Default
|                               |                      |                  N/A
+-------------------------------+----------------------+----------------------

+-----------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|        ID   ID                                                   Usage
|=============================================================================
|    0   N/A  N/A     21736      C   python                          10914MiB
|    1   N/A  N/A     21736      C   python                          10914MiB
|    2   N/A  N/A     21736      C   python                          10923MiB
|    3   N/A  N/A     21736      C   python                          10923MiB
```

- 1 epoch = 1 hour
- 1 model = 100 epochs = 100 hours
- 100 models = 10000 hours = ~400 days

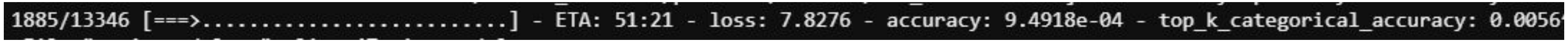
Harvard John A. Paulson School of Engineering and Applied Sciences

Ineffective 4 GPUs

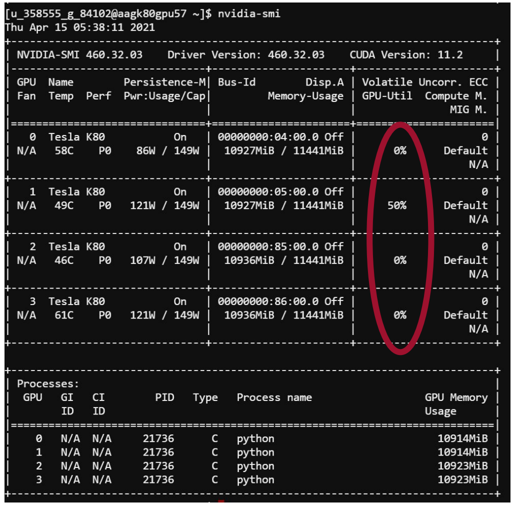
IV - Effective Parallelization of the Training of a CNN on Large Scale Data

Learning Hyperparameters and their influence on the training time:

- Batch size : arbitrary
- Learning Rate : arbitrary
- Number of Epochs : 100



```
1885/13346 [===>..........................] - ETA: 51:21 - loss: 7.8276 - accuracy: 9.4918e-04 - top_k_categorical_accuracy: 0.0056
```



```
[u_358555_g_84102@aagk80gpu57 ~]$ nvidia-smi
Thu Apr 15 05:38:11 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000000:04:00.0 Off |                    0 |
| N/A   58C    P0    86W / 149W |  10927MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 00000000:05:00.0 Off |                    0 |
| N/A   49C    P0   121W / 149W |  10927MiB / 11441MiB |     50%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 00000000:85:00.0 Off |                    0 |
| N/A   46C    P0   107W / 149W |  10936MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 00000000:86:00.0 Off |                    0 |
| N/A   61C    P0   121W / 149W |  10936MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     21736      C   python                          10914MiB |
|    1   N/A  N/A     21736      C   python                          10914MiB |
|    2   N/A  N/A     21736      C   python                          10923MiB |
|    3   N/A  N/A     21736      C   python                          10923MiB |
+-----------------------------------------------------------------------------+
```

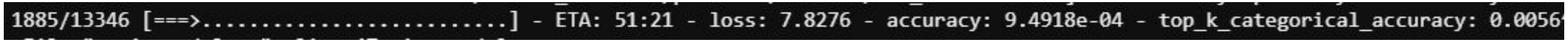
- Parallelization Strategy: Mirrored Strategy
- 1 epoch = 1 hour
- 1 model = 100 epochs = 100 hours
- 100 models = 10000 hours = ~400 days

Harvard John A. Paulson School of Engineering and Applied Sciences

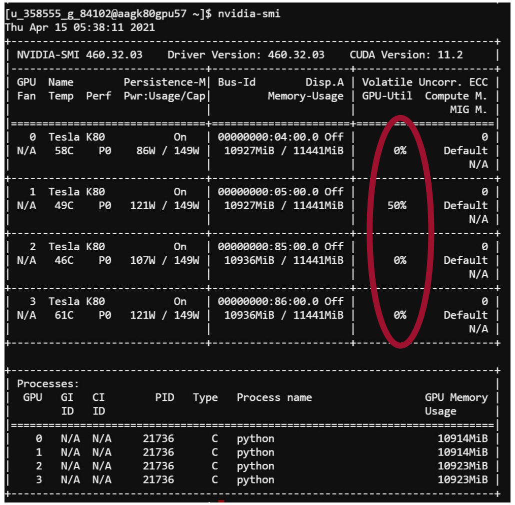
IV - Effective Parallelization of the Training of a CNN on Large Scale Data

Learning Hyperparameters and their influence on the training time:

- Batch size : arbitrary
- Learning Rate : arbitrary
- Number of Epochs : 100



```
1885/13346 [===>..........................] - ETA: 51:21 - loss: 7.8276 - accuracy: 9.4918e-04 - top_k_categorical_accuracy: 0.0056
```



```
[u_358555_g_84102@aagk80gpu57 ~]$ nvidia-smi
Thu Apr 15 05:38:11 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000000:04:00.0 Off |                    0 |
| N/A   58C    P0    86W / 149W |  10927MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 00000000:05:00.0 Off |                    0 |
| N/A   49C    P0   121W / 149W |  10927MiB / 11441MiB |     50%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 00000000:85:00.0 Off |                    0 |
| N/A   46C    P0   107W / 149W |  10936MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 00000000:86:00.0 Off |                    0 |
| N/A   61C    P0   121W / 149W |  10936MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     21736      C   python                          10914MiB |
|    1   N/A  N/A     21736      C   python                          10914MiB |
|    2   N/A  N/A     21736      C   python                          10923MiB |
|    3   N/A  N/A     21736      C   python                          10923MiB |
+-----------------------------------------------------------------------------+
```

- Bottleneck identified: our data pipeline is too slow in order to process and feed the data to the GPU (diagnosis: the GPUs are idle)
- First step: accelerating the data pipeline

Harvard John A. Paulson School of Engineering and Applied Sciences

IV - Effective Parallelization of the Training of a CNN on Large Scale Data: 4 GPUs case

Learning Hyperparameters and their influence on the training time:

- Batch size: arbitrary
- Learning Rate: arbitrary
- Number of Epochs : 100

Pipeline changes

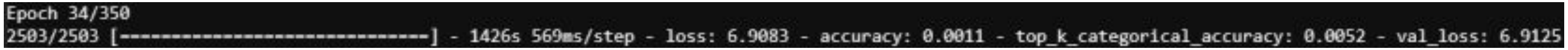
- Parallelizing the preprocessing operations in the data pipeline (dynamic allocation of #workers)
- Caching and Prefetching the data in order to reduce the data transfers between CPU & GPU

Towards 100% occupation ?

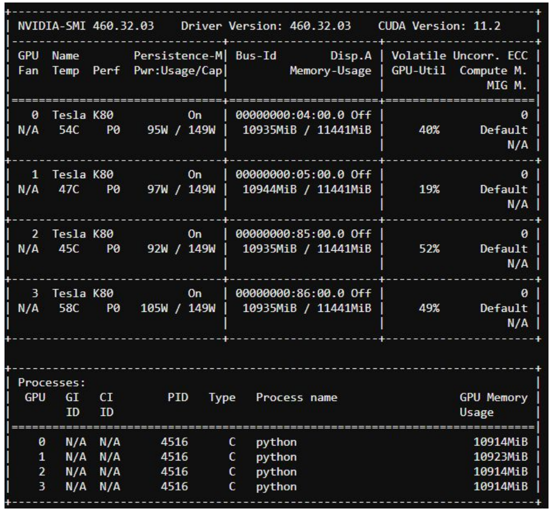
IV - Effective Parallelization of the Training of a CNN on Large Scale Data: 4 GPUs case

Learning Hyperparameters and their influence on the training time:

- Batch size : 512 (objective: saturate the GPU memory at every data bus)
- Learning Rate : arbitrary
- Number of Epochs : 100



```
Epoch 34/350
2503/2503 [==============================] - 1426s 569ms/step - loss: 6.9083 - accuracy: 0.0011 - top_k_categorical_accuracy: 0.0052 - val_loss: 6.9125
```



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000000:04:00.0 Off |                    0 |
| N/A   54C    P0    95W / 149W |  10935MiB / 11441MiB |     40%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 00000000:05:00.0 Off |                    0 |
| N/A   47C    P0    97W / 149W |  10944MiB / 11441MiB |     19%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 00000000:85:00.0 Off |                    0 |
| N/A   45C    P0    92W / 149W |  10935MiB / 11441MiB |     52%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 00000000:86:00.0 Off |                    0 |
| N/A   58C    P0   105W / 149W |  10935MiB / 11441MiB |     49%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      4516      C   python                          10914MiB |
|    1   N/A  N/A      4516      C   python                          10923MiB |
|    2   N/A  N/A      4516      C   python                          10914MiB |
|    3   N/A  N/A      4516      C   python                          10914MiB |
+-----------------------------------------------------------------------------+
```

- 1 epoch = 23 minutes
- 1 model = 100 epochs = ~38 hours
- 100 models = ~150 days
- Can we reach ~100% GPU occupation ?

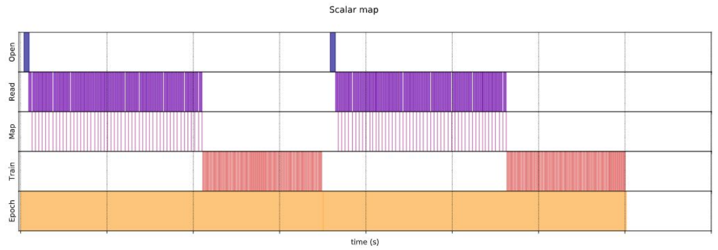
Harvard John A. Paulson School of Engineering and Applied Sciences

IV - Effective Parallelization of the Training of a CNN on Large Scale Data

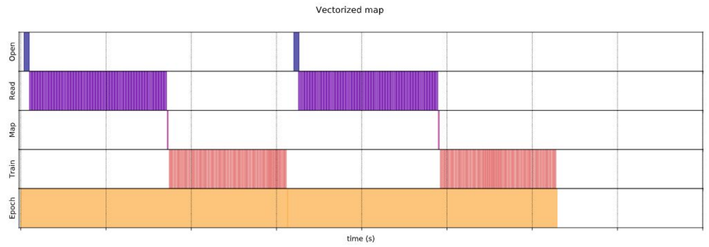
Learning Hyperparameters and their influence on the training time:

- Batch size : 512 (objective: saturate the GPU memory at every data bus)
- Learning Rate : arbitrary
- Number of Epochs : 100

Solution chosen: vectorizing the preprocessing functions across a batch (ie first batching, then apply transformations)
Toy example with a special subset of the data:



```
Scalar map  2.982725427829
```



```
Vectorized  map  0.182725427829
```

Harvard John A. Paulson School of Engineering and Applied Sciences

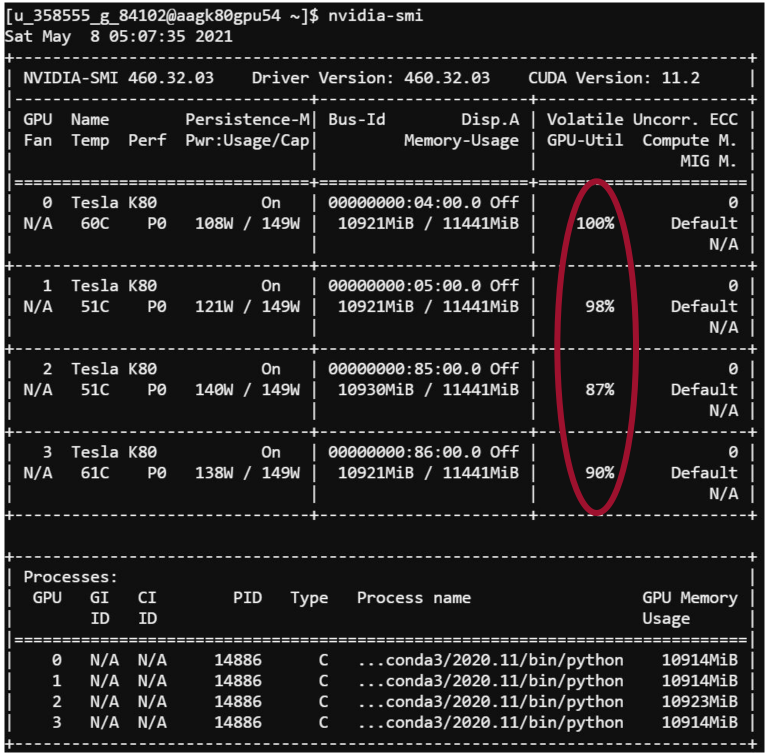
Need for Spark

IV - Effective Parallelization of the Training of a CNN on Large Scale Data

- Issue: can’t batch ImageNet because of non uniform shapes of Images
- Solution: offline preprocessing Step, using either TFDS pipeline or Spark
- Problems using Spark: the data is loaded as TF Records and not as .PNG files
- Solution: Use a Spark Tensorflow Connector in order to load the TF Records as Spark DataFrames (requires using maven)
- Using Spark pipelining: process 1024 images in 12 seconds
- Using TFDS pipelining: process 1024 images in 38 seconds
- Effective Data processing Speed up using spark: ~3 x

IV - Effective Parallelization of the Training of a CNN on Large Scale Data

- Using the data already preprocessed:



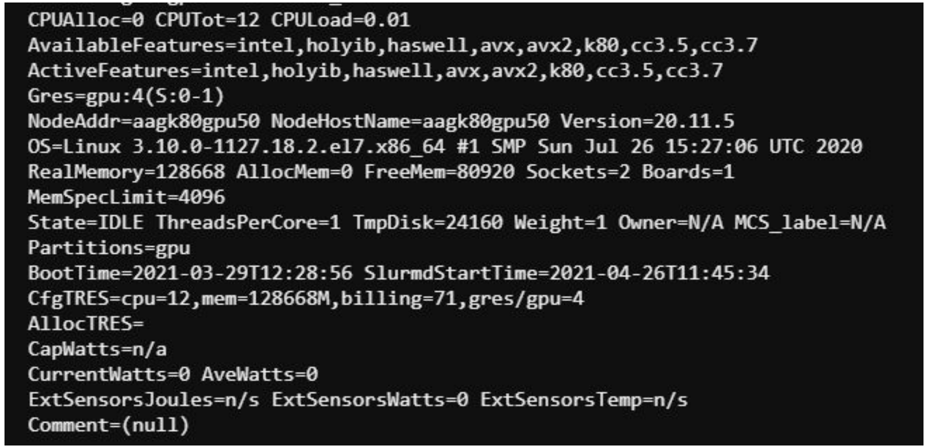
```
[u_358555_g_84102@aagk80gpu54 ~]$ nvidia-smi
Sat May  8 05:07:35 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000000:04:00.0 Off |                    0 |
| N/A   60C    P0   108W / 149W |  10921MiB / 11441MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 00000000:05:00.0 Off |                    0 |
| N/A   51C    P0   121W / 149W |  10921MiB / 11441MiB |     98%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 00000000:85:00.0 Off |                    0 |
| N/A   51C    P0   140W / 149W |  10930MiB / 11441MiB |     87%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 00000000:86:00.0 Off |                    0 |
| N/A   61C    P0   138W / 149W |  10921MiB / 11441MiB |     90%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     14886      C   ...conda3/2020.11/bin/python    10914MiB |
|    1   N/A  N/A     14886      C   ...conda3/2020.11/bin/python    10914MiB |
|    2   N/A  N/A     14886      C   ...conda3/2020.11/bin/python    10923MiB |
|    3   N/A  N/A     14886      C   ...conda3/2020.11/bin/python    10914MiB |
+-----------------------------------------------------------------------------+
```

- 1 epoch = 11 minutes
- 1 model = 100 epochs = ~18 hours
- 100 models = ~75 days
- Effective Speed-up from Single CPU: ~x100

Harvard John A. Paulson School of Engineering and Applied Sciences

IV - Effective Parallelization of Masking: Distributed Memory Parallel Programming

- Issue: Distributed Memory Parallel Computing Paradigm involves some overheads in the communication between different nodes
- Solution: Transform our problem into a SIMD paradigm and use SLURM Job arrays in order to parallelize the work without communication.
- Setting: We parallelized over 20 workers nodes (two cascades of 10 worker nodes with 4 GPUs)



```
CPUAlloc=0 CPUTot=12 CPULoad=0.01
AvailableFeatures=intel,holyib,haswell,avx,avx2,k80,cc3.5,cc3.7
ActiveFeatures=intel,holyib,haswell,avx,avx2,k80,cc3.5,cc3.7
Gres=gpu:4(S:0-1)
NodeAddr=aagk80gpu50 NodeHostName=aagk80gpu50 Version=20.11.5
OS=Linux 3.10.0-1127.18.2.el7.x86_64 #1 SMP Sun Jul 26 15:27:06 UTC 2020
RealMemory=128668 AllocMem=0 FreeMem=80920 Sockets=2 Boards=1
MemSpecLimit=4096
State=IDLE ThreadsPerCore=1 TmpDisk=24160 Weight=1 Owner=N/A MCS_label=N/A
Partitions=gpu
BootTime=2021-03-29T12:28:56 SlurmdStartTime=2021-04-26T11:45:34
CfgTRES=cpu=12,mem=128668M,billing=71,gres/gpu=4
AllocTRES=
CapWatts=n/a
CurrentWatts=0 AveWatts=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
Comment=(null)
```

V - Within Node Code Optimization

Pruning the MobileNet architecture using Tensorflow

- Issue: pruning weights using a mask in tensorflow damages the computational graph
- Solution: zeroing the weights using a mask on each batch using tf.callbacks

Code Profiling

- Results from code profiling
  - Tf.callbacks: 3.3457s
  - Total batch time (including tf.callbacks): 3.5970s
  - Tf.callbacks: ~93% of batch runtime
- Issue: the influence of tf.callbacks on the training time
- Solution: stay tuned!!

V - Within Node Code Optimization

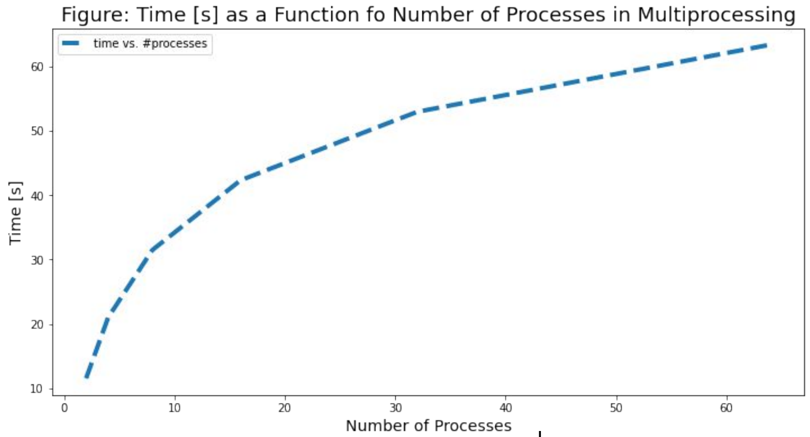
Single Process vs. Multiprocessing

Optimizing the tf.callbacks

- Single Process
  - Tf.callbacks on a single process: 3.3457s
- Multiprocessing
  - Tf.callbacks on 64 processes: 63.3417s

Why is multiprocessing (MP) slowing down our code?

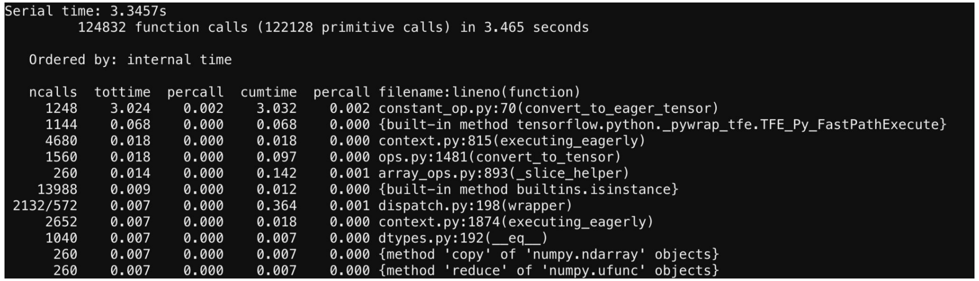
- Computation time increase with # processes
- Calling os.fork() at least 64 times!
  - Creating a child process expensive (overhead)
  - # trainable layers 155: ~ 3 tasks per process
  - Tasks simple: matrix multiplication
  - Overheads MP >> benefits MP
- Conclusion: MP not a suitable solution for problem



V - Within Node Code Optimization

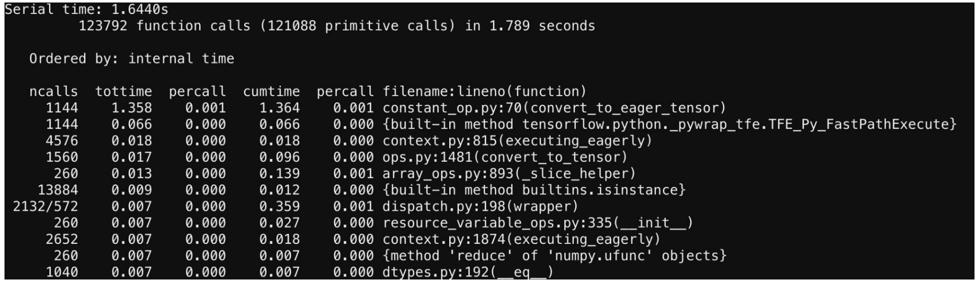
Optimizing the tf.callbacks on a single process

- Code profiling (again!!)



```
Serial time: 3.3457s
         124832 function calls (122128 primitive calls) in 3.465 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1248    3.024    0.002    3.032    0.002 constant_op.py:70(convert_to_eager_tensor)
     1144    0.068    0.000    0.068    0.000 {built-in method tensorflow.python._pywrap_tfe.TFE_Py_FastPathExecute}
     4680    0.018    0.000    0.018    0.000 context.py:815(executing_eagerly)
     1560    0.018    0.000    0.097    0.000 ops.py:1481(convert_to_tensor)
      260    0.014    0.000    0.142    0.001 array_ops.py:893(_slice_helper)
    13988    0.009    0.000    0.012    0.000 {built-in method builtins.isinstance}
 2132/572    0.007    0.000    0.364    0.001 dispatch.py:198(wrapper)
     2652    0.007    0.000    0.018    0.000 context.py:1874(executing_eagerly)
     1040    0.007    0.000    0.007    0.000 dtypes.py:192(__eq__)
      260    0.007    0.000    0.007    0.000 {method 'copy' of 'numpy.ndarray' objects}
      260    0.007    0.000    0.007    0.000 {method 'reduce' of 'numpy.ufunc' objects}
```

  - Issue: convert_to_eager_tensor ~90%
  - Solution: convert_to_eager_tensor of mask outside the tf.callbacks
  - Result: reduced tf.callbacks runtime by ~50%



```
Serial time: 1.6440s
         123792 function calls (121088 primitive calls) in 1.789 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1144    1.358    0.001    1.364    0.001 constant_op.py:70(convert_to_eager_tensor)
     1144    0.066    0.000    0.066    0.000 {built-in method tensorflow.python._pywrap_tfe.TFE_Py_FastPathExecute}
     4576    0.018    0.000    0.018    0.000 context.py:815(executing_eagerly)
     1560    0.017    0.000    0.096    0.000 ops.py:1481(convert_to_tensor)
      260    0.013    0.000    0.139    0.001 array_ops.py:893(_slice_helper)
    13884    0.009    0.000    0.012    0.000 {built-in method builtins.isinstance}
 2132/572    0.007    0.000    0.359    0.001 dispatch.py:198(wrapper)
      260    0.007    0.000    0.027    0.000 resource_variable_ops.py:335(__init__)
     2652    0.007    0.000    0.018    0.000 context.py:1874(executing_eagerly)
      260    0.007    0.000    0.007    0.000 {method 'reduce' of 'numpy.ufunc' objects}
     1040    0.007    0.000    0.007    0.000 dtypes.py:192(__eq__)
```

Harvard John A. Paulson
School of Engineering
and Applied Sciences

VI - Results: Training Results

- Training a Single CNN on 4 GPUs:
  - Serial CPU Timing: 2000 hours
  - Parallel result: 18 hours

- Parallelization of the masking procedure over 10 nodes in a cascade
  - Serial result: 18h*100 = ~75 days
  - Parallel result: 18h*2*5 = ~15 days (shootout to Raminder & Francesco who gave us special privileges to allocate all the nodes at once)

- Total Training time (taking into account Late Resetting and Masking):
  - Fully Serial: ~ 8000 days
  - Parallelization of our solution: ~15 days

- Final speed up from single CPU to end-to-end solution: x500

VI - Winning Ticket and Transfer learning

- Found winning ticket: 12% of the initial architecture

- Sparser architecture: computations are ~x6 faster

- Generalization across datasets: CIFAR 100
    - Fine tune our ticket using a low learning rate and 5 epochs. Effective time to train = 11 minutes
    - Results: Top 5 accuracy 0.82 and top 1 accuracy 0.72
    - Comparison to a benchmark: retraining MobileNet v2 from scratch on CIFAR 100 for 5 epochs. Effective time to train = ~1 hour
    - Results Top 5 accuracy 0.41 and top 1 accuracy 0.23

VII - Discussion

- Working on more recent NVIDIA GPUs (compute capability >= 7.0), use Mixed Precision policy: expected speed-up x3 (experiment to be tried on AWS)

- Further scale the training of one Neural Network: use frameworks for Multi-Node training: Horovod

- Find an efficient way to sparsify the underlying computational graph: wait for Tensorflow 3.0?

Thank you for your attention