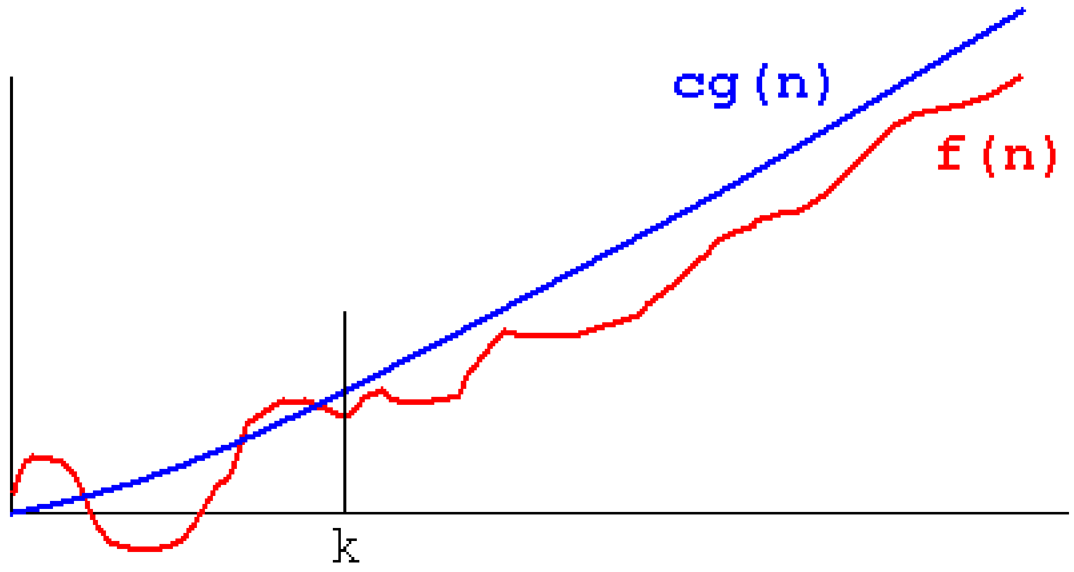
Algorithm complexity – Big O notation

- **Definition:** A theoretical measure of the execution of an *algorithm*, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation f(n) = O(g(n)) means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n".
- **Formal Definition:** f(n) = O(g(n)) means there are positive constants c and k, such that $0 \leq f(n) \leq cg(n)$ for all $n \geq k$. The values of c and k must be fixed for the function f and must not depend on n.



Source: NIST

- Complexity ranking

Function	Common name
$n!$	factorial
2^n	exponential
$n^d,\ d > 3$	polynomial
n^3	cubic
n^2	quadratic
$n\sqrt{n}$	
$n \log n$	quasi-linear
n	linear
$\sqrt{n}$	root - n
$\log n$	logarithmic
1	constant

- Big O examples

$T(n)$	Complexity
$5n^3 + 200n^2 + 15$	$\mathrm{O}(n^3)$
$3n^2 + 2^{300}$	$\mathrm{O}(n^2)$
$5 \log_2 n + 15 \ln n$	$\mathrm{O}(\log n)$
$2 \log n^3$	$\mathrm{O}(\log n)$
$4n + \log n$	$\mathrm{O}(n)$
2^{64}	$\mathrm{O}(1)$
$\log n^{10} + 2\sqrt{n}$	$\mathrm{O}(\sqrt{n})$
$2^n + n^{1000}$	$\mathrm{O}(2^n)$

Slide credit: UPC

Neural Network time complexity

- Forward propagation – weighted sum & activation function

  Total t training examples

  $Z_{jt} = W_{ji} X_{it}$, $Y_{it} = \sigma (Z_{it})$ => O(j*i*t + j*t) = O (j *i * t)

  Multiple layers

  O (t * (ij + jk + kl + ...)) => O (t * $\sum_{all\ layers} (input\_dim \times output\_\mathrm{dim})$)

- Backward propagation

  $\underset{jxt}{dZ^{[1]}} = \underset{jxk}{W^{[2]T}} \underset{kxt}{dZ^{[2]}} * \sigma^{[1]\prime}\underset{jxt}{(Z^1)}$ $\underset{jxi}{dW^{[1]}} = \underset{jxt}{dZ} \underset{txi}{X^T}$ => O (j * t * i)

  Multiple layers

  O (t * $\sum_{all\ layers} (input\_dim \times output\_\mathrm{dim})$)

- n epochs O (n* t * $\sum_{all\ layers} (input\_dim \times output\_\mathrm{dim})$)

Nvidia GPU FLOPs measurement

- C:

```
#include <cuda_profiler_api.h>
cudaProfilerStart();
myKernel<<<...>>>(...);
cudaProfilerStop();
```

- Python:

```
import torch.cuda.profiler as profiler
profiler.start()
...
profiler.stop()
```

flop_count_sp:
smsp__sass_thread_inst_executed_op_fadd_pred_on.sum +
smsp__sass_thread_inst_executed_op_fmul_pred_on.sum +
smsp__sass_thread_inst_executed_op_ffma_pred_on.sum * 2

- Nsight profiler command

```
ncu --profile-from-start off –-metrics <comma separated list> --target-processes all <original job command>
```

- nvprof command (predecessor of Nsight)

```
nvprof  --profile-from-start off –metrics flop_count_sp --profile-all-processes <original job command>
```

- Why different from the estimation?

Neural network memory complexity

- Memory for parameters
  - Fully connected layers
    - #weights = #outputs x #inputs
    - #biases = #outputs
- Memory for layer outputs
    - #outputs
- Backward propagation specific
  - Memory for Errors
  - Memory for parameter gradients
  - Memory for hyperparameter-related (e.g., momentum)
- Implementation overhead

What about convolution layers?
What about pooling layers?
What about batch size?

Model Summary in PyTorch

- pip install torchsummary
- MNIST

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchsummary import summary

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # PyTorch v0.4.0
model = Net().to(device)

summary(model, (1, 28, 28))
```

```
----------------------------------------
Layer (type) Output Shape          Param #
========================================
Conv2d-1 [-1, 10, 24, 24]          260
Conv2d-2 [-1, 20, 8, 8]            5,020
Dropout2d-3 [-1, 20, 8, 8]         0
Linear-4 [-1, 50]                  16,050
Linear-5 [-1, 10]                  510
========================================
Total params: 21,840
Trainable params: 21,840
Non-trainable params: 0
----------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.08
Estimated Total Size (MB): 0.15
----------------------------------------
```

Nvidia GPU memory utilization measurement

- nvidia-smi
- Pytorch CUDA API
  - cat gpumem.py

```
import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
if device.type == 'cuda':
    print(torch.cuda.get_device_name(0))
    print('Memory Usage:')
    print('Allocated:', round(torch.cuda.memory_allocated(0)/1024**3,1), 'GB')
    print(‘Reserved:   ', round(torch.cuda.memory_reserved(0)/1024**3,1), 'GB’)
```

  - Python3 gpumem.py

```
Memory Usage:
Allocated: 0.0 GB
Reserved:    0.0 GB
```

Nvidia GPU memory utilization measurement

- GPUtil python package
  - pip3 install gputil psutil humanize
  - cat memreport.py

```
# Import packages
import torch
import os,sys,humanize,psutil,GPUtil
import torchvision.models as models

def mem_report():
  print("CPU RAM Free: " + humanize.naturalsize( psutil.virtual_memory().available ))
  GPUs = GPUtil.getGPUs()
  for i, gpu in enumerate(GPUs):
    print('GPU {:d} ... Mem Free: {:.0f}MB / {:.0f}MB | Utilization {:3.0f}%'.format(i, gpu.memoryFree, gpu.memoryTotal, 
gpu.memoryUtil*100))

wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
if torch.cuda.is_available():
  wide_resnet50_2.cuda()

mem_report()
```

  - python3 memreport.py

```
CPU RAM Free: 244.9 GB
GPU 0 ... Mem Free: 44246MB / 45556MB | Utilization   3%
```

NYU Greene cluster setup

- Greene cluster info: https://sites.google.com/nyu.edu/nyu-hpc/hpc-systems/greene/getting-started?authuser=0
- Login into Greene cluster login node:

ssh greene.hpc.nyu.edu

- Launch an interactive job on a GPU node using slurm

srun -n4 -t2:00:00 --mem=4000 --gres=gpu:1 --pty /bin/bash

- Setup the env
  - Load the modules

  module load cuda/11.1.74 python/intel/3.8.6

  - Setup virtualenv

  python3 –m venv pytorch_env (only first time)

  source pytorch_env/bin/activate

  - Install torch packages (only first time)

  pip3 install torch torchvision torchsummary

  pip3 install –U numpy

GCP cluster

- Each user is assigned 100 GPU hours.
- To access GCP cluster, please first login to Greene cluster login node, then login to burst login node
- ssh burst
- From here to start interactive jobs, users are allowed to access these partition
- srun --account=csci_ga_3033_085_2022sp --partition=n1s8-v100-1 --gres=gpu:1 --pty /bin/bash
- srun --account=csci_ga_3033_085_2022sp --partition=n1s16-v100-2 --gres=gpu:2 --pty /bin/bash
- srun --account=csci_ga_3033_085_2022sp --partition=c12m85-a100-1 --gres=gpu:1 --pty /bin/bash
- srun --account=csci_ga_3033_085_2022sp --partition=c24m170-a100-2 --gres=gpu:2 --pty /bin/bash

CIMS cuda[1-5].cims.nyu.edu setup

- Server info: https://cims.nyu.edu/webapps/content/systems/resources/computeservers
- Login into the cuda node:

ssh cuda3.cims.nyu.edu

- Setup the env
  - Load the modules

  module load cuda-10.2 python-3.8
  - Setup virtualenv

  python3 –m venv pytorch_env (only first time)

  source pytorch_env/bin/activate
  - Install torch packages (only first time)

  pip3 install torch torchvision torchsummary

Code and data

- Pytorch examples:

git clone https://github.com/pytorch/examples

- ImageNet data
    - 1k class data set is sufficient
    - http://www.image-net.org/
    - Location on Greene cluster: /scratch/work/public/imagenet
    - Create a directory of your own using symbolic links for a small subset of training/test data

Homework 2 – Performance study a layer of a Neural Network

- Assignment:
  Estimate and measure, time (compute operations) and space (memory) complexity, of the inference (forward) execution of any of these models:
  - Torch.nn.transformer: examples/word_language_model at main · pytorch/examples (github.com)
  - DistillBERT from huggingface (may take time to find a Linear layer to play with)
  - a convolution layer (Conv2d-2) of MNIST CNN, reference code: https://github.com/pytorch/examples/tree/master/mnist
  - Huggingface/gpt2 (117m) model.

- Notes:
  1. Pen and paper method to estimate the complexity
  2. Use NCU or other tools to measure the time (ops) and memory aspects of the execution of the layer under inspection.
  3. Add one more dimension: batch sizes, draw a scalability chart/plot (x axis: batch size, y axis: flops and/or mem) and analyze potential trends.
  4. Analysis: there are 3 dimensions of potential comparisons: time-vs-space, estimation-vs-measurement, batch-size variations, make it clean.
  5. Due on 20, 2023 at 11:59pm