



AI @ Scale

Global AI – Turin October 28 2022 - Giuseppe Fiameni – gfiameni@nvidia.com

ABOUT ME

Giuseppe Fiameni - gfiameni@nvidia.com



- Data Scientist & AI Expert @ NVIDIA
 - *Support Higher Education and Research through collaboration projects*
- Lead engineer of the NVIDIA AI Technology Center, Italy
 - *Agreement among CINI, CINECA and NVIDIA to accelerate scientific discovery*
- 10+ years experience delivering HPC applications and large data processing solutions at CINECA
(www.hpc.cineca.it)

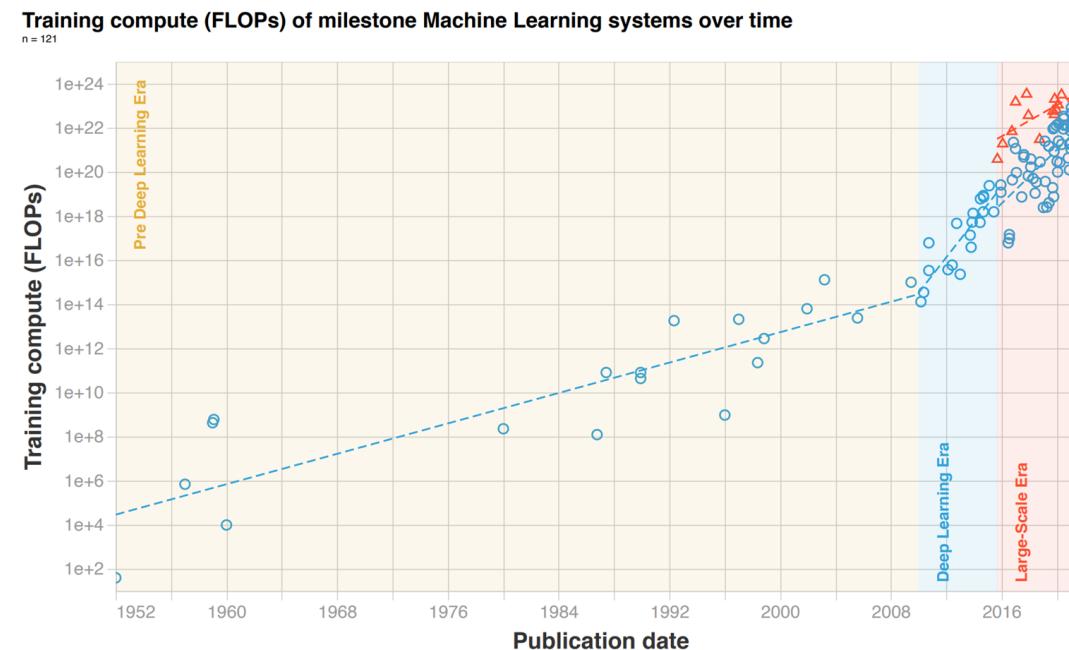
WHY THIS TALK?

- Deep Learning is transitioning from being a *computer science* towards a *computational science*.
- Advanced computing and large-scale infrastructure are fundamental to conduct science i.e., train gigantic models, process massive data, achieve better performance, reduce time to solutions.



Looking back: three eras of compute in machine learning

- ▶ A study documents the incredible acceleration of compute requirements in machine learning. It identifies 3 eras of machine learning according to training compute per model doubling time. The Pre-Deep Learning Era (pre-2010, training compute doubled every 20 months), the Deep Learning Era (2010-15, doubling every 6 months), and the Large-Scale Era (2016-present, a 100-1000x jump, then doubling every 10 months).



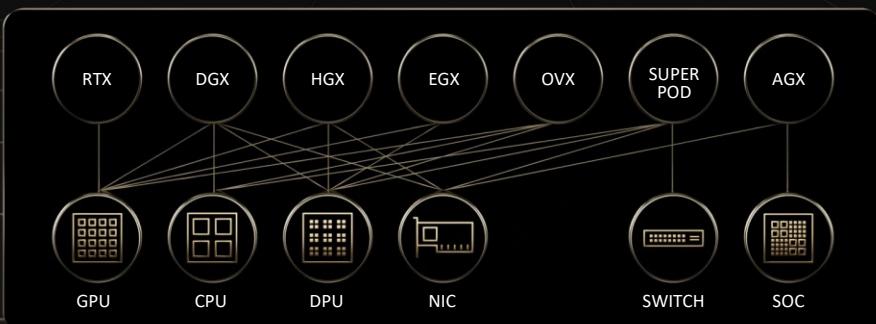
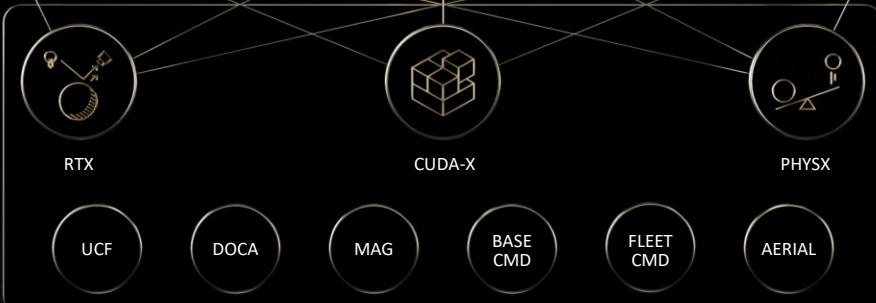


APPLICATION
FRAMEWORKS

PLATFORM

SYSTEM
SOFTWARE

HARDWARE



NVIDIA H100

Unprecedented Performance, Scalability, and Security for Every Data Center

HIGHEST AI AND HPC PERFORMANCE

4PF FP8 (6X) | 2PF FP16 (3X) | 1PF TF32 (3X) | 60TF FP64 (3X)
3TB/s (1.5X), 80GB HBM3 memory

TRANSFORMER MODEL OPTIMIZATIONS

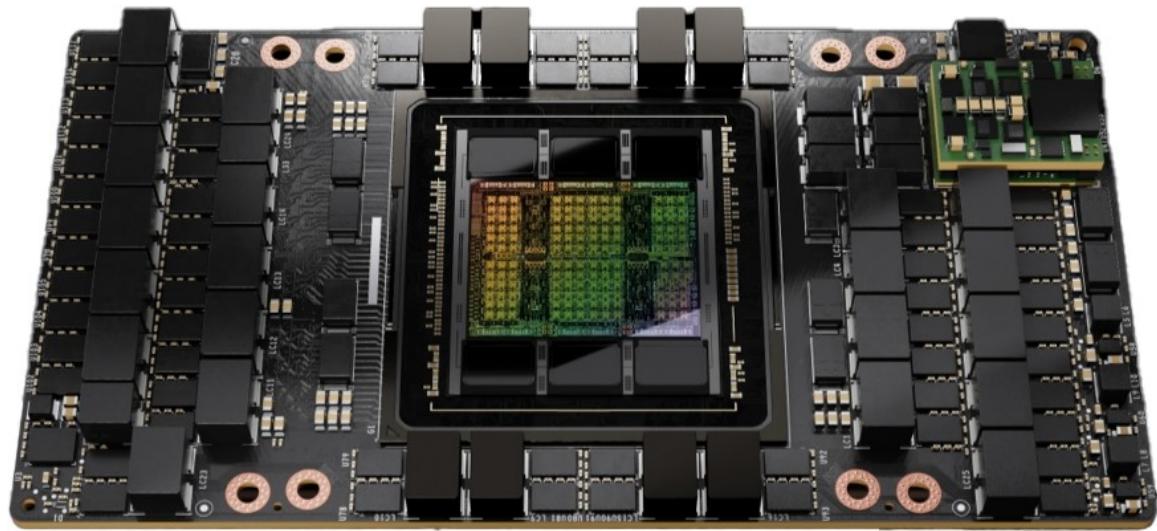
6X faster on largest transformer models

HIGHEST UTILIZATION EFFICIENCY AND SECURITY

7 Fully isolated & secured instances, guaranteed QoS
2nd Gen MIG | Confidential Computing

FASTEST, SCALABLE INTERCONNECT

900 GB/s GPU-2-GPU connectivity (1.5X)
up to 256 GPUs with NVLink Switch | 128GB/s PCI Gen5



Grace Hopper

CPU+GPU Designed for Giant Scale AI and HPC

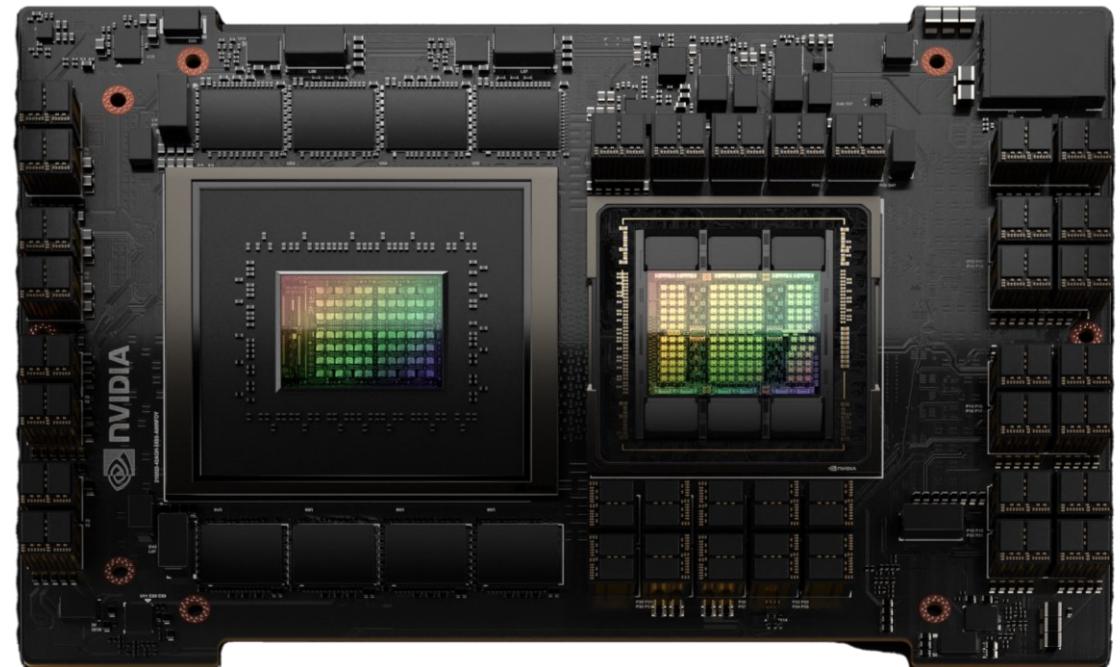
600GB Memory GPU for Giant Models

New 900 GB/s Coherent Interface

30X Higher System Memory B/W to GPU In A Server

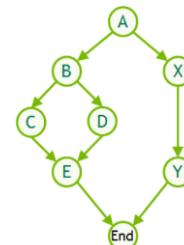
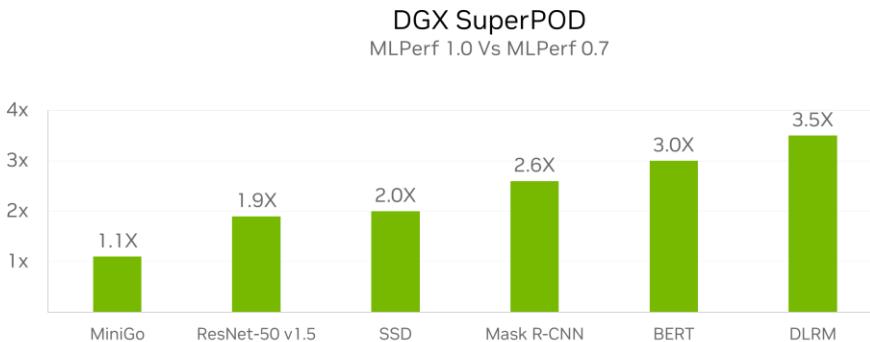
Runs Nvidia Computing Stacks

Available 1H 2023



NVIDIA AI DELIVERS 3.5X HIGHER PERFORMANCE IN 1 YEAR

Fueled by continuous platform innovation



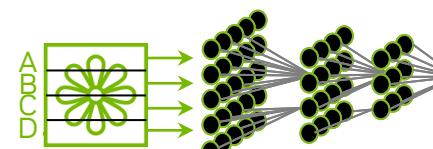
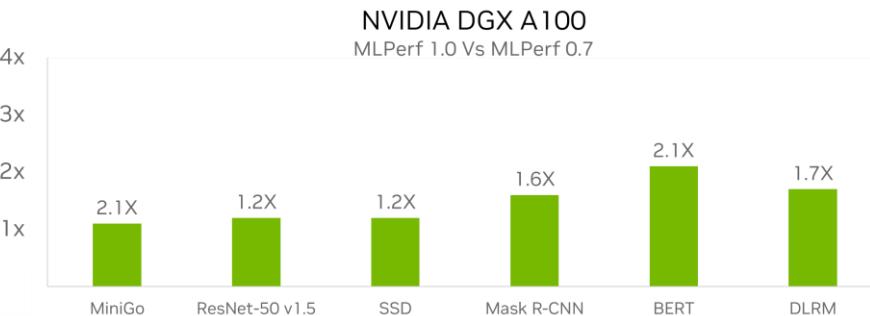
CUDA Graphs
Up to 1.7x



SHARP
Up to 1.2x



Larger Scale
Up to 4096 GPUs



Spatial Data Parallelism
Up to 1.6x



HBM2e
Memory Bandwidth >2TB/s

MLPerf 1.0 vs 0.7 comparisons normalized to the number of samples needed to train for the same batch size to account for accuracy increase in latest round of MLPerf v1.0 | MLPerf name and logo are trademarks. See www.mlperf.org for more information

DGX SuperPOD run at max scale available | MLPerf v1.0: 1.0-1067, 1.0-1070, 1.0-1072, 1.0-1075, 1.0-1076, 1.0-1077 | MLPerf v0.7: 0.7-17, 0.7-28, 0.7-33, 0.7-36, 0.7-37, 0.7-38

NVIDIA DGX A100 Results | MLPerf v1.0: 1.0-1058, 1.0-1059, 1.0-1060, 1.0-1061 | MLPerf v0.7: 0.7-17, 0.7-18, 0.7-19, 0.7-20

CUDA graphs compared on Mask R-CNN and BERT | SHARP compared using DGX A100 over 512 nodes on BERT model | Spatial Data Parallelism compared on 3D-Unet



NOTEBOOKS & CODE

<https://github.com/gfiameni/hndl/>

The screenshot shows a Jupyter Notebook interface. On the left, there's a file browser sidebar with icons for file operations like create, copy, move, and delete. Below it is a list of Python files in the directory `/ai-school / code /`:

Name	Last Modified
<code>ddp_horovod.py</code>	21 minutes ago
<code>ddp_mixed_precision.py</code>	18 minutes ago
<code>ddp.py</code>	18 days ago
<code>main_amp.py</code>	18 days ago
<code>mp.py</code>	3 months ago
<code>pipeline_parallelism.py</code>	12 days ago
<code>send_receive.py</code>	3 months ago
<code>utils.py</code>	a month ago
<code>zero.py</code>	18 days ago

The main area contains a tab bar with several tabs: 03-Pipeline, 00-Intro, 09-FSDP, 08-Horovod, ddp_horovod, 06-DDP, ddp_mixed_precision, 07-Memory, 05-Memory, and 04-ZeRO. The `00-Intro` tab is currently active. The content of the `00-Intro` cell is a Markdown document:

(SGD for machine learning/deep learning) for data parallelism can be divided into two categories: asynchronous update and synchronous update. The disadvantages of data parallelism are also obvious. Since each sub-model needs to submit the gradient after each iteration of training, the network communication overhead is very large.

- Model parallelism is used for scenarios where the size of the model is very large and cannot be stored in local memory. In this case, we need to split the model into different modules (e.g., different layers in DNN). Then, each module can be put into different nodes for training. At this time, frequent inter-node communication between different nodes may be required. The performance of model parallelism depends on two aspects, connectivity structure and compute demand of operations. Although model parallelism can solve the problem of large model training, it will also bring us low network traffic and increase training time.
- Hybrid parallelism is the combination of data parallelism and model parallelism.

1. Data Parallelism
2. Model Parallelism
3. Message Passing
4. Horovod
5. Mixed Precision
6. Memory Format
7. Pipeline Parallelism
8. ZeRO
9. PyTorch SLLURM Working in Progress

Application process topologies

A Distributed Data Parallel (DDP) application can be executed on multiple nodes where each node can consist of multiple GPU devices. Each

AGENDA

Why Large datasets and Large Neural Networks?

Why NN models size keep increasing

Challenges of Supervised training

Scaling law

What governs model scaling

Training large neural networks

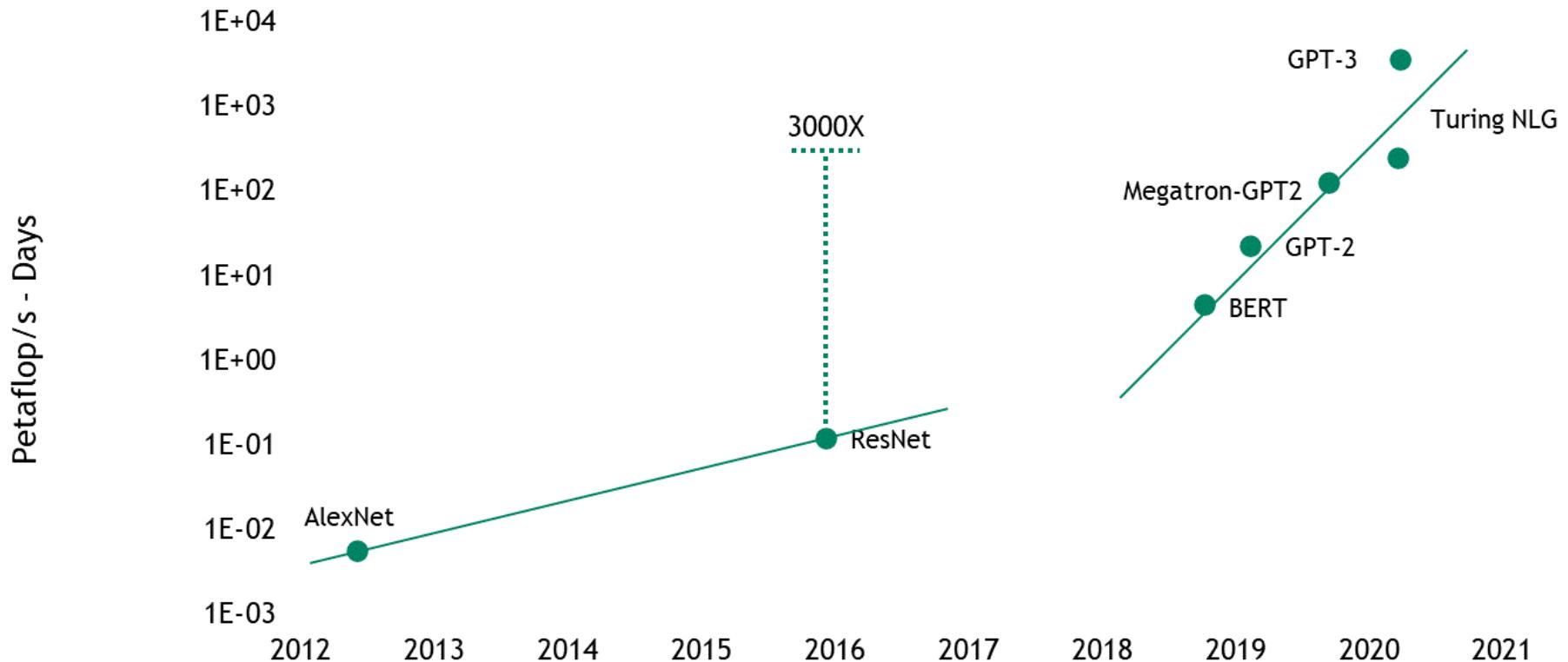
Data Parallelism vs Model Parallelism

Code samples



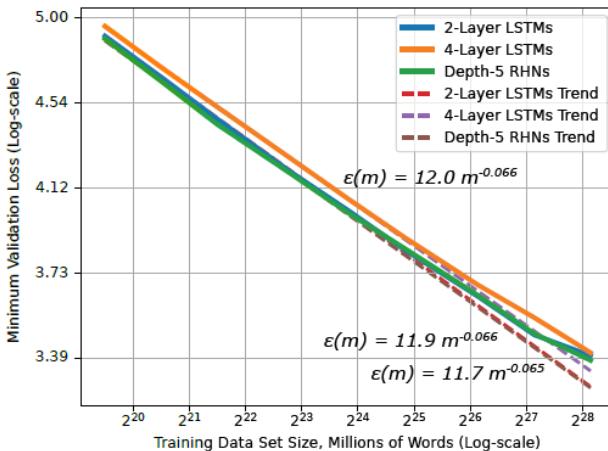
WHY LARGE DATASETS AND MODELS

New trend in model growth

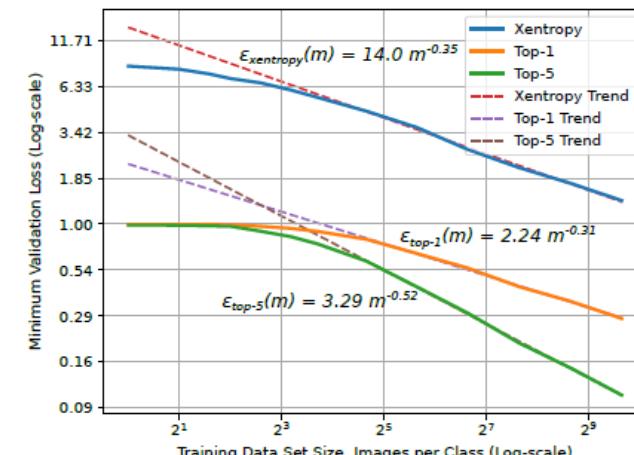
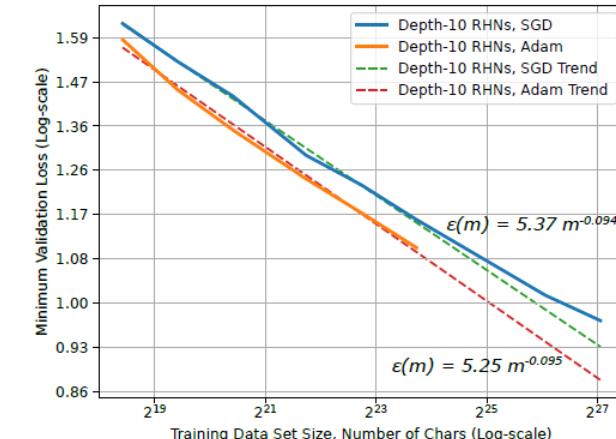
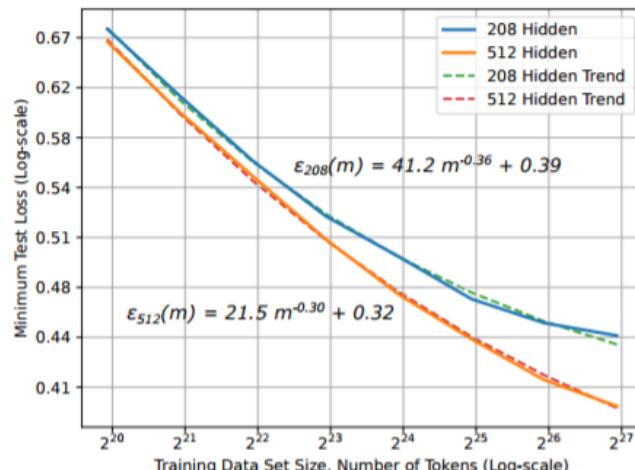
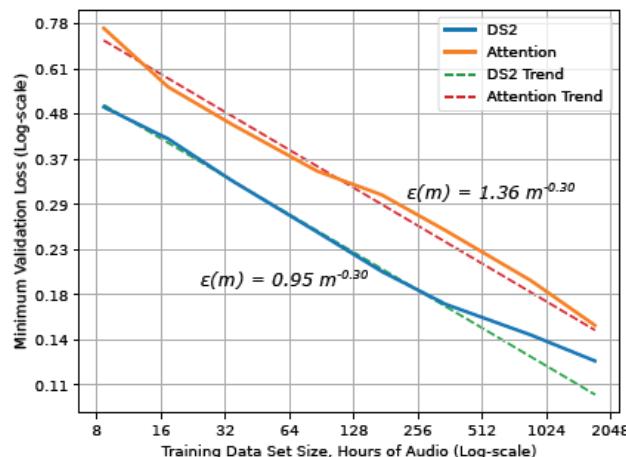


RELIABLE WAY FOR IMPROVING MODEL PERFORMANCE

Logarithmic relationship between the dataset size and accuracy

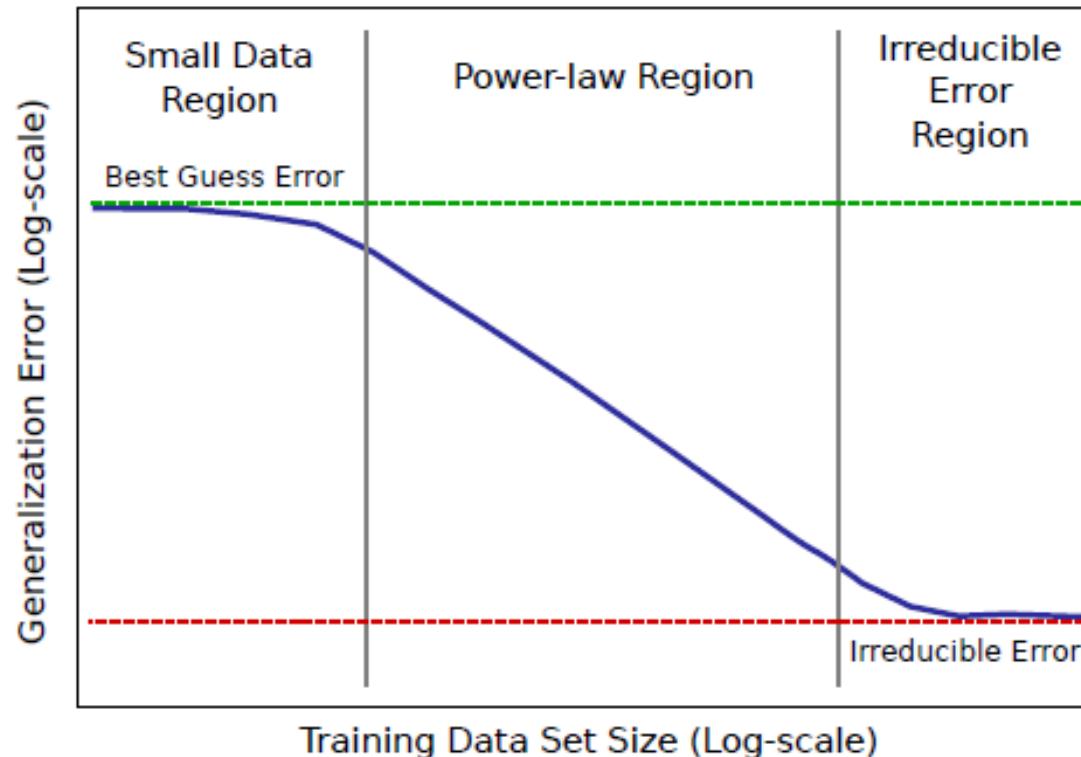


- Translation
- Language Models
- Character Language Models
- Image Classification
- Attention Speech Models



NEW PROMISE

Logarithmic relationship between the dataset size and accuracy



CHALLENGES OF SUPERVISED TRAINING

THE COMPLEXITY

Direct impact on data collection and labelling effort

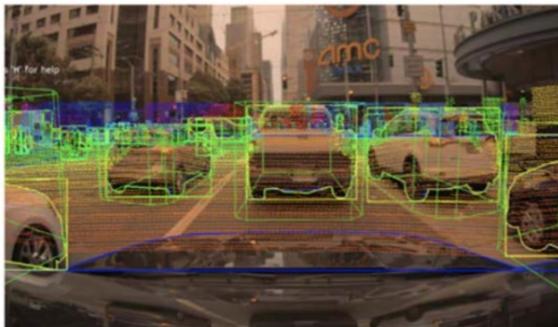
More
Functionality



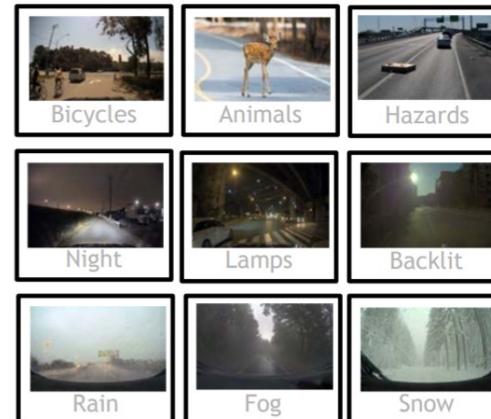
More
Conditions



MASSIVE
Data



New features (i.e. lane keeping)
require new data...



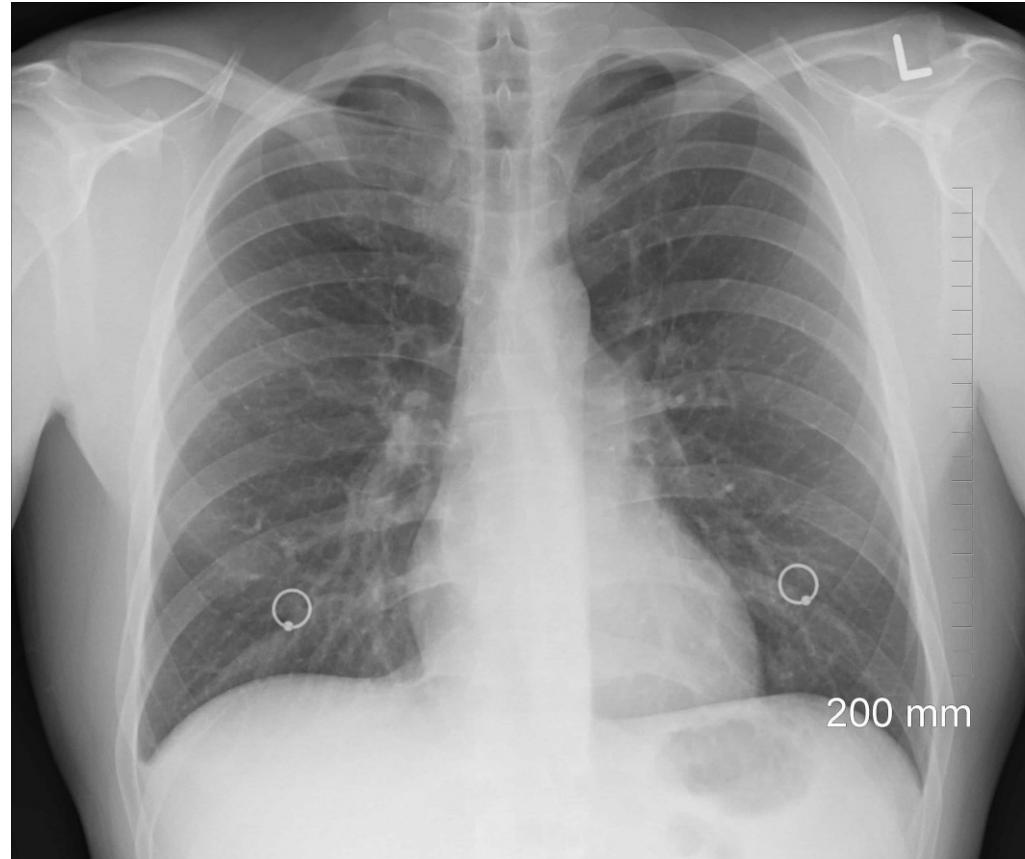
...and require more real examples to
meet safety targets...



...resulting in exponential data
and compute needs

THE COMPLEXITY

What about problems that require much higher level of expertise



THE COMPLEXITY

What about problems that require much higher level of expertise

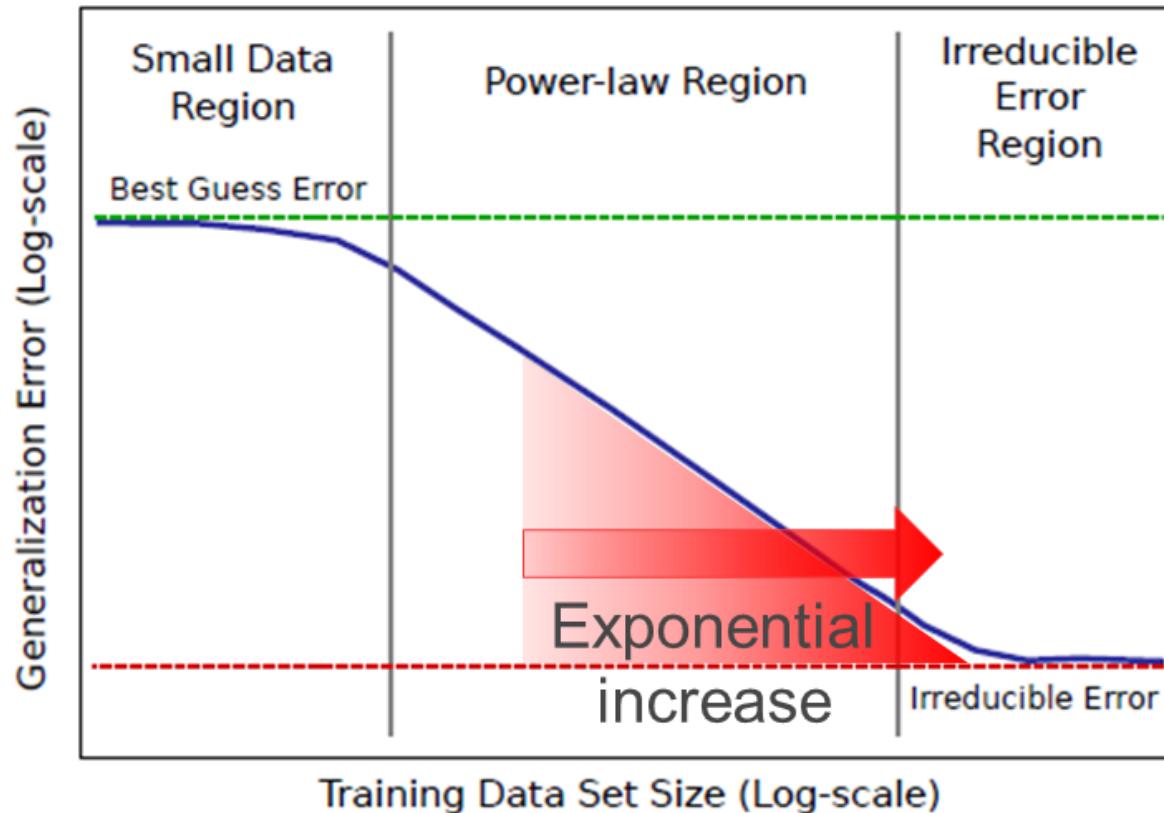
Nawet prostsze problemy niż diagnoza raka wymagają często ekspertyzy.



Even problems that are simpler than cancer diagnosis often require expert judgment.

THE COST OF LABELLING

Limits the utility of deep learning models



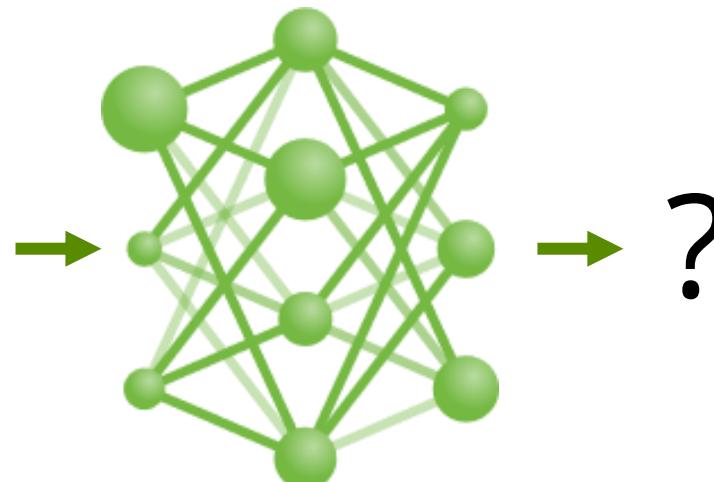
SELF AND UN-SUPERVISED LEARNING

SELF AND UN-SUPERVISED LEARNING

What to do in the absence of labels



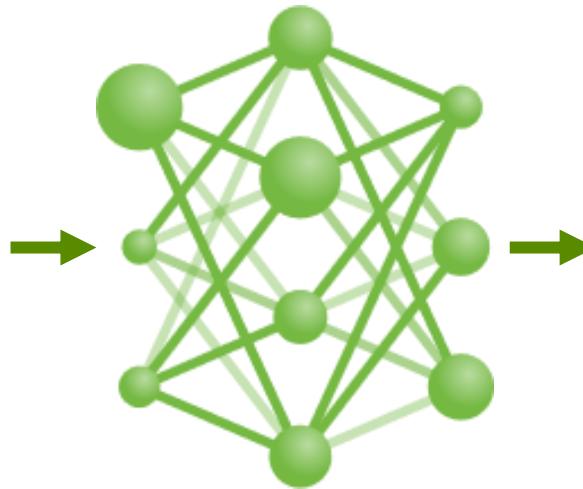
WIKIPEDIA
L'encyclopédia libera



SELF-SUPERVISED AND UNSUPERVISED LEARNING

Natural Language Processing - Masked Language Models

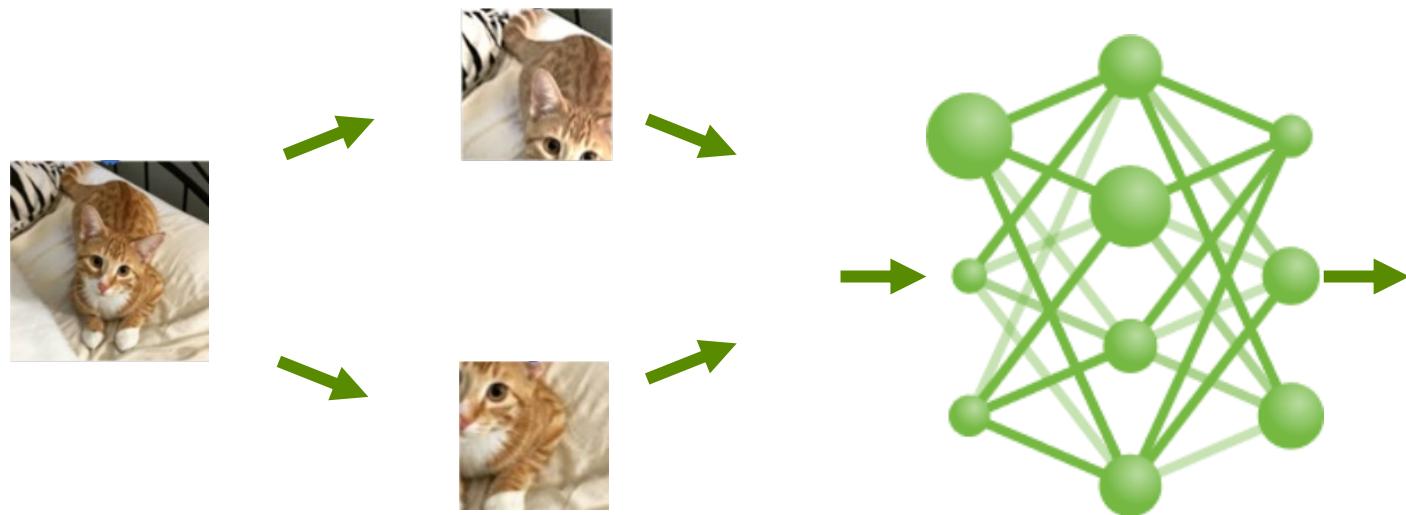
Lorem ipsum [MASK] dolor sit amet,
consectetur adipiscing elit.



Lorem ipsum dolor sit amet,
consectetur adipiscing elit.

SELF-SUPERVISED AND UNSUPERVISED LEARNING

Computer Vision - Contrastive Learning

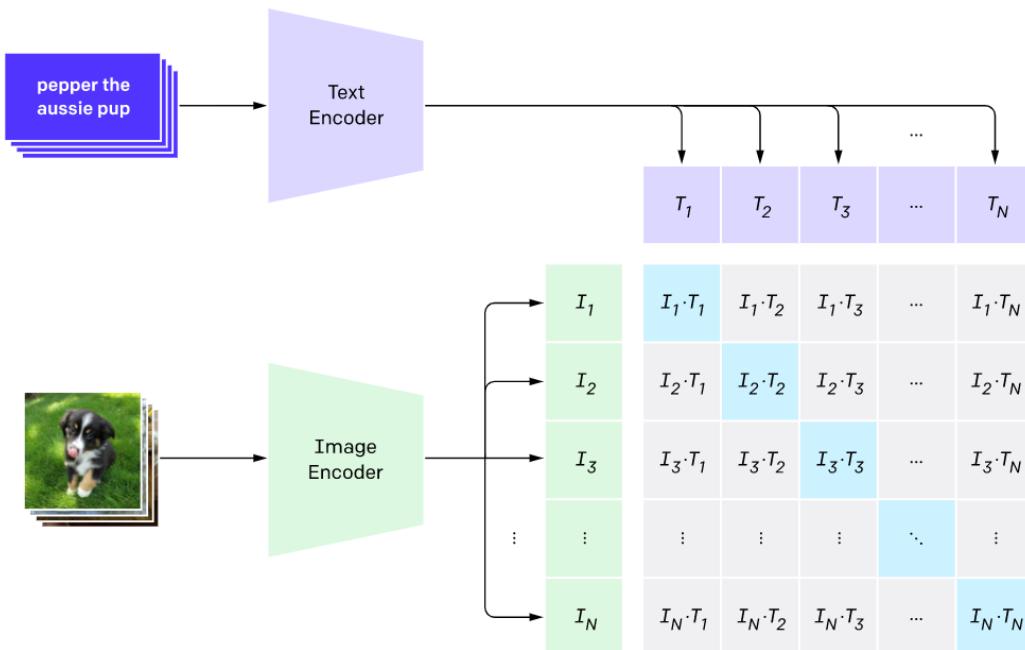


Do the pictures
have the same
origin: Yes/No

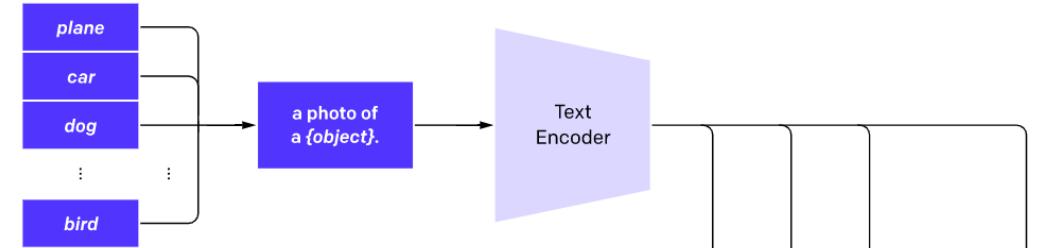
SELF SUPERVISED AND UNSUPERVISED LEARNING

Computer Vision - More complex contrastive learning

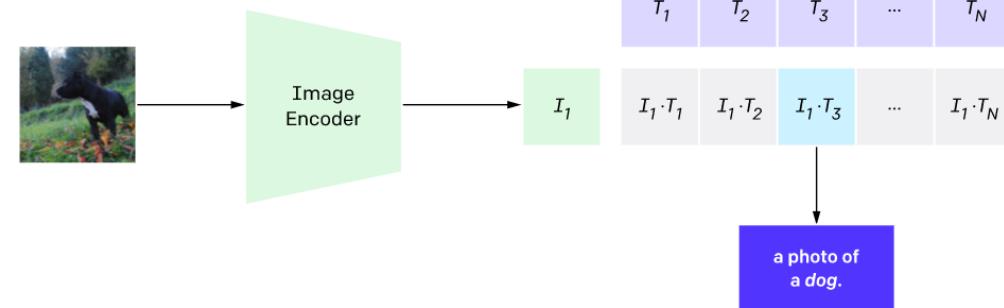
1. Contrastive pre-training



2. Create dataset classifier from label text

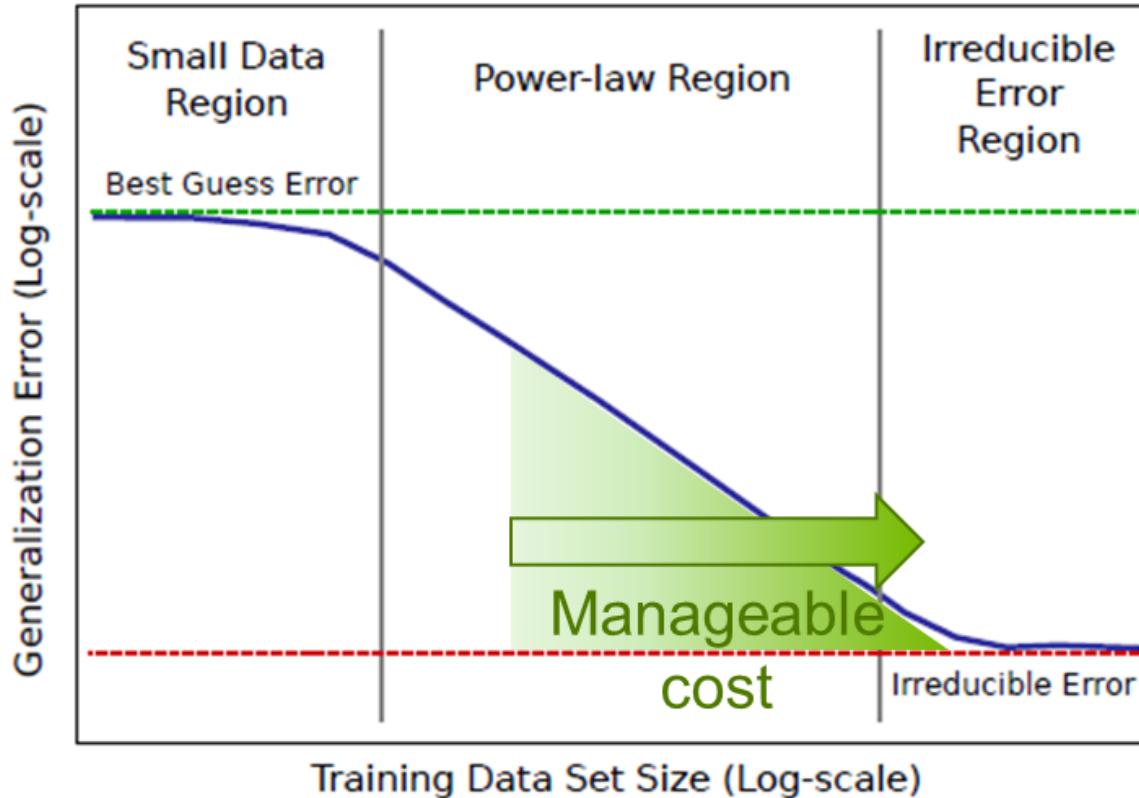


3. Use for zero-shot prediction



THE COST OF LABELLING

Semi supervised models



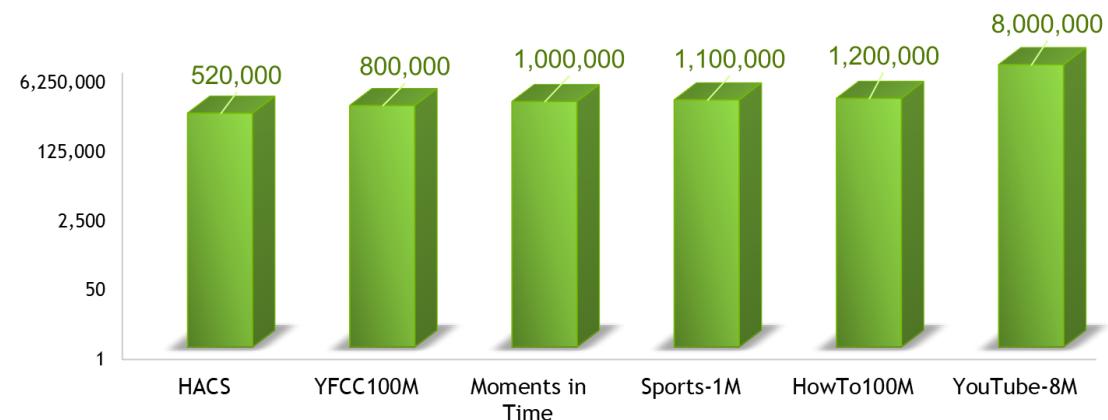
SELF SUPERVISED AND UNSUPERVISED LEARNING

Abundance of unlabelled data in Text

Number of Words (in Millions)



Number of videos



SCALING LAWS

SCALING LAWS APPLY TO NLP

As you increase the dataset size, you must also increase the model size

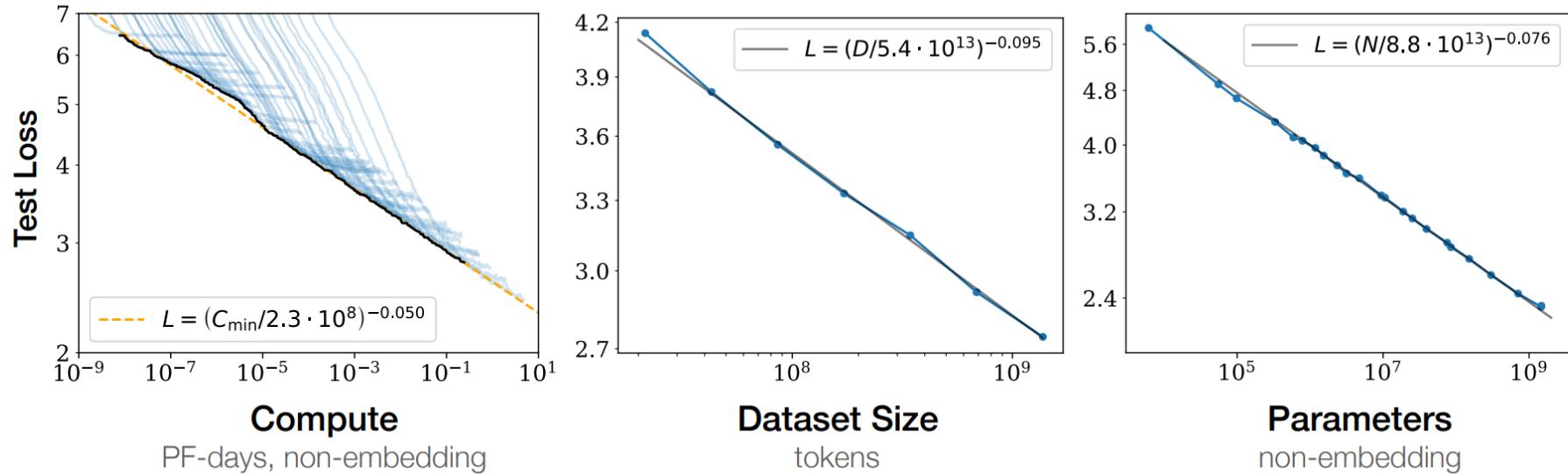


Figure 1 Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute² used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

SCALING LAWS APPLY TO COMPUTER VISION TOO

Increase in performance is proportional to the model size and dataset size

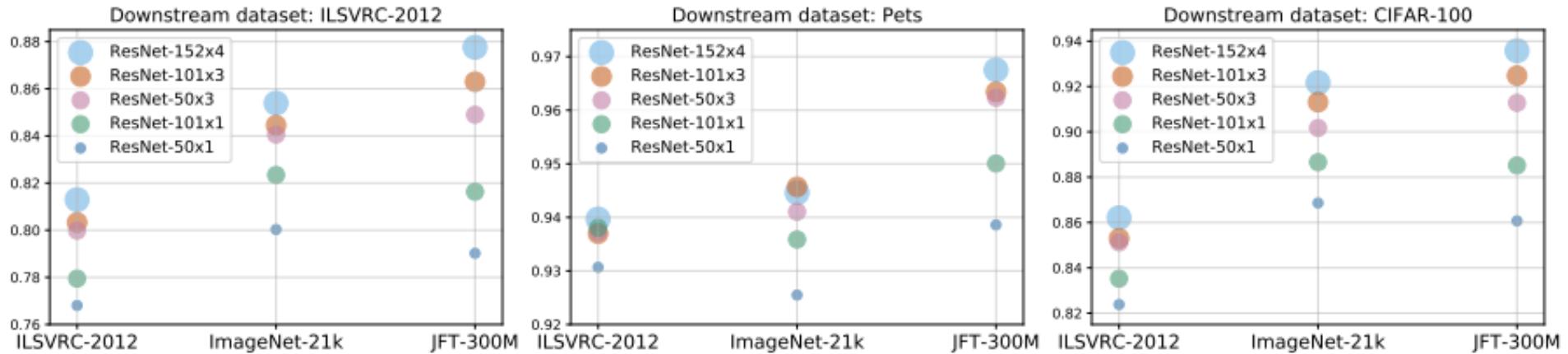


Fig. 5: Effect of upstream data (shown on the x-axis) and model size on downstream performance. Note that exclusively using more data or larger models may hurt performance; instead, both need to be increased in tandem.

IT IS MORE THAN JUST ACCURACY

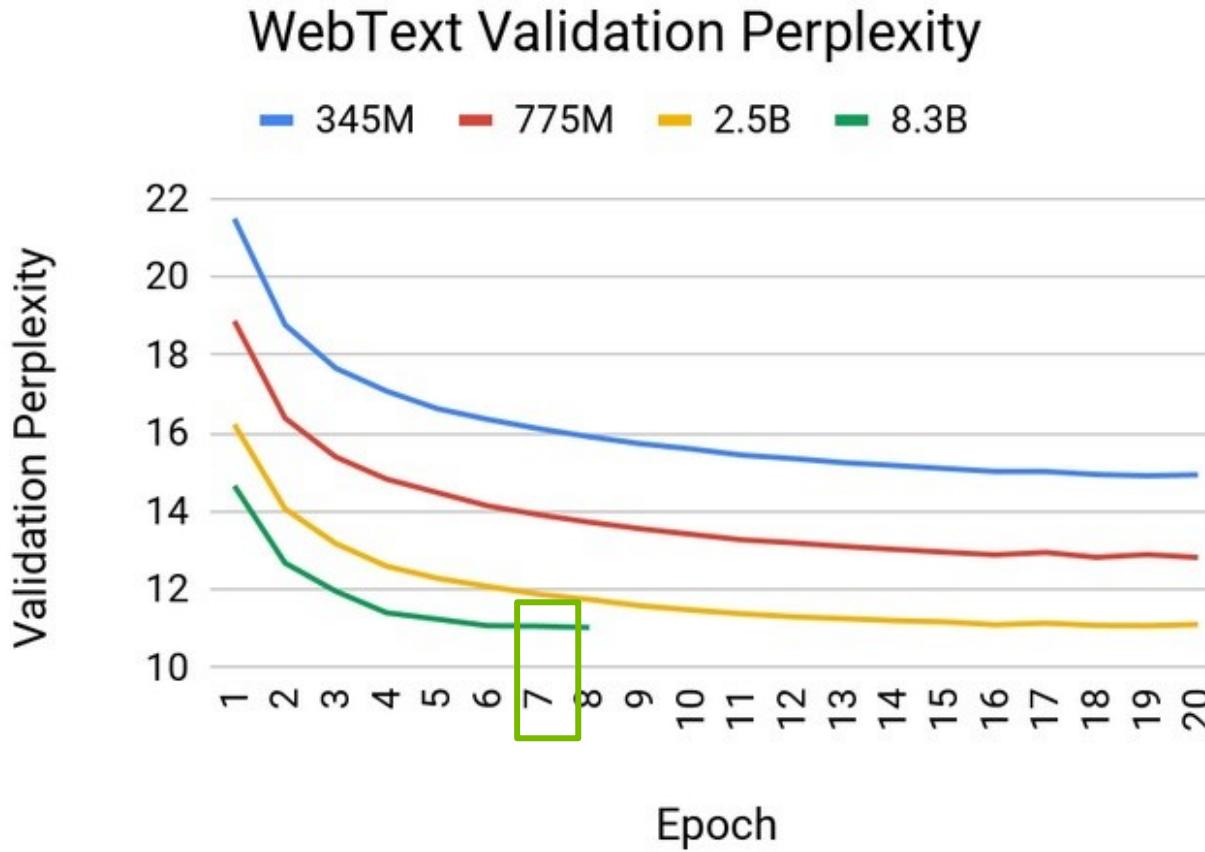
IMPORTANCE OF DATASET SIZE

Dataset size more important than neural network design

*“... more importantly, we find that the precise architectural hyperparameters are **unimportant** compared to the overall scale of the language model.”*

EVEN MORE IMPORTANTLY

Large Neural Networks use data more efficiently



PERSPECTIVE

WHAT DO I MEAN BY BIG

GPT-3 size comparison: 538x Bigger than BERT-Large

Not a linear scale

Total Compute Used During Training

3.14×10^8 PFLOPS

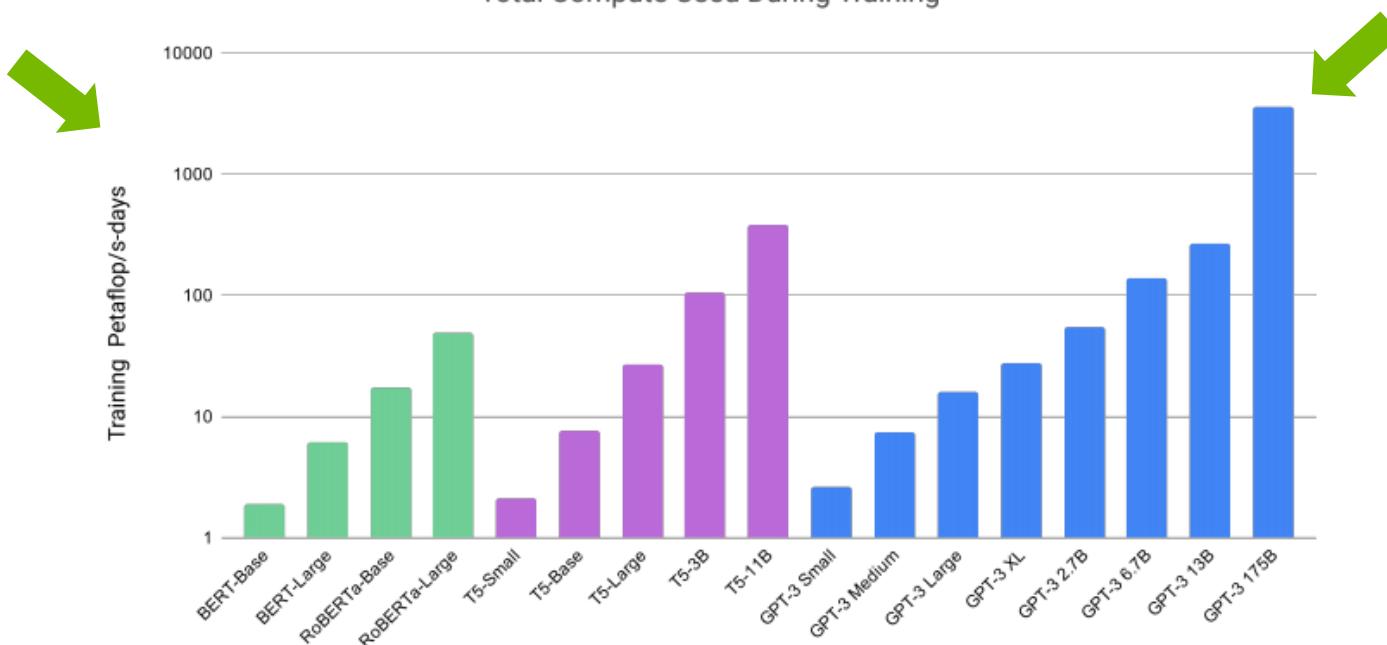


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH⁺20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

WHAT DO I MEAN BY BIG

GPT-3 size comparison: 538x Bigger than BERT-Large

Not a linear scale

Total Compute Used During Training

~31 years on a single A100

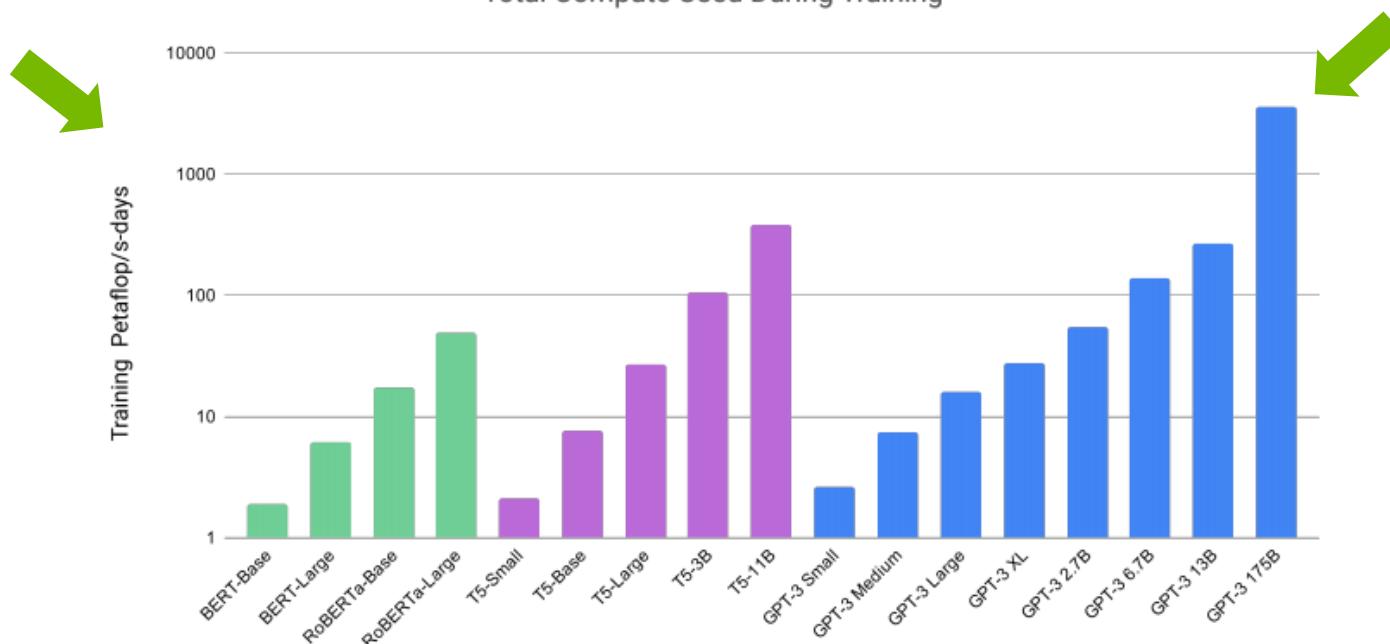


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH⁺20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

ESTIMATE COMPUTE NEEDED

Calculate how many hours/days compute resource need

paper : <https://arxiv.org/pdf/2005.14165.pdf>

```
[2]: import numpy as np
T=300*1e+9 #oftokens in the dataset
#P=175*1e+9 # number of model parameters
n= 480 # Berzelius 480 # number of GPUs in the compute cluster

def calculate_days_needed(T , P , n ,x):
    if x is None:
        return '1-2 weeks'
    else:
        #x=140*1e+12 # TeraFlop/s per GPU
        tot=x*T*P
        div=n*x
        compute_sec=tot/div
        #convert compute seconds to days
        to_days=round(compute_sec/(3600*24),1)
        return to_days

GPT3_models_labels=[ 'gpt3_2.7B', 'gpt3_6.7B','gpt3_13B', 'gpt3_175B']
GPT3_model_params=[ 2.7*1e+9, 6.7*1e+9 , 13*1e+9, 175*1e+9,1*1e+12 ]
GPT3_model_params_str=['1.3 Billion' , '2.7 Billion', '13 Billion', '175 Billion']
#according to the table above
GPT3_X=[127*1e+12, 130*1e+12,135*1e+12,140*1e+12 ]
print("all below are measured with dataset size **300 billion** measured in tokens \n")
for gpt3_name, gpt3_params, gpt3_param_str, x in zip(GPT3_models_labels,GPT3_model_params,GPT3_model_params_str):
    days_needed=calculate_days_needed(T,gpt3_params,n,x)
    print("-----")
    print(" language model :{} with {} number of parameters , it will need {} days to compute".format(gpt3_name,gpt3_params,gpt3_param_str))
    print("-----")
all below are measured with dataset size **300 billion** measured in tokens
```

language model :gpt3_2.7B with 1.3 Billion number of parameters , it will need 1.2 days to compute

language model :gpt3_6.7B with 2.7 Billion number of parameters , it will need 3.0 days to compute

language model :gpt3_13B with 13 Billion number of parameters , it will need 5.6 days to compute

language model :gpt3_175B with 175 Billion number of parameters , it will need 72.3 days to compute

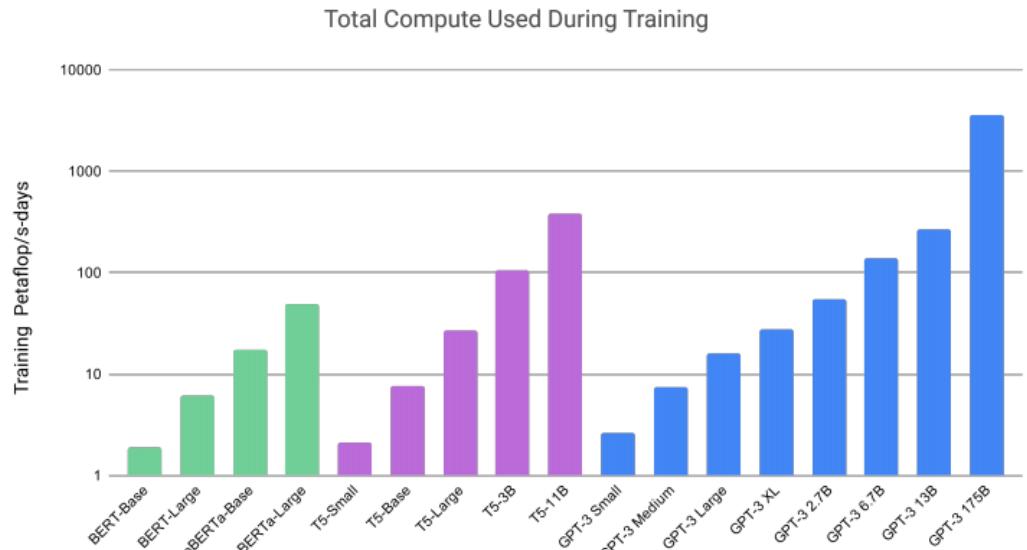


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH⁺20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

Source :<https://arxiv.org/pdf/2005.14165.pdf>

GOING BIGGER

- If we only consider Parameters, Gradients and Optimizer states and ignore activations
- If we use FP16 data representation (so two bytes)
- If we use Adam as an optimiser (storing twelve bytes per parameter in mixed precision mode)
- If we consider a model with one billion parameters

$$10^9 * (2B + 2B + 12B) = 10^9 * 16B = 14.90\text{GB}$$

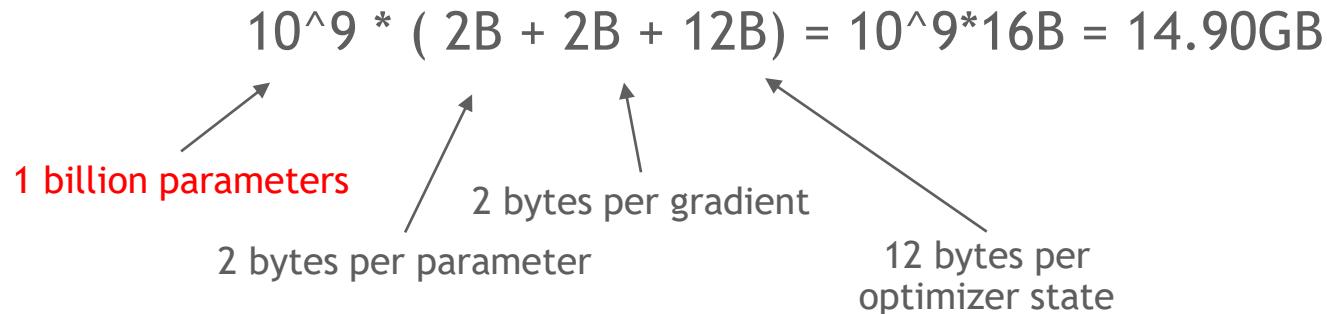
1 billion parameters 2 bytes per gradient
2 bytes per parameter 12 bytes per optimizer state

GOING BIGGER

- What about activations?
- What about 2 or 3 billion parameter models?

$$10^9 * (2B + 2B + 12B) = 10^9 * 16B = 14.90GB$$

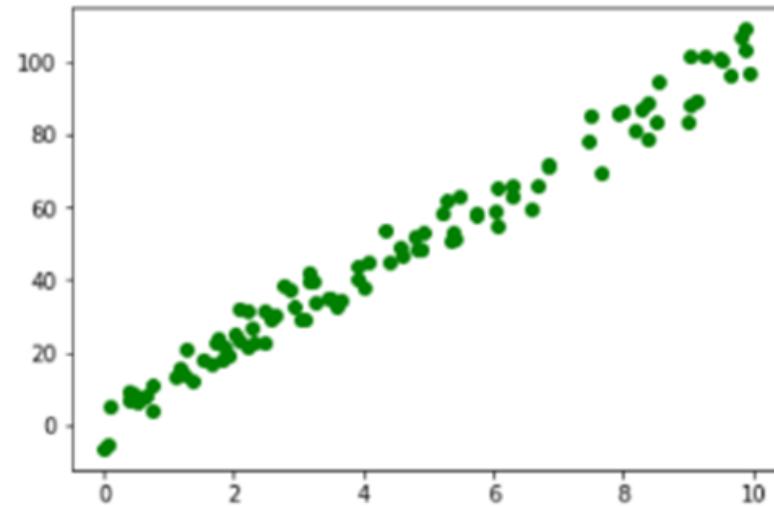
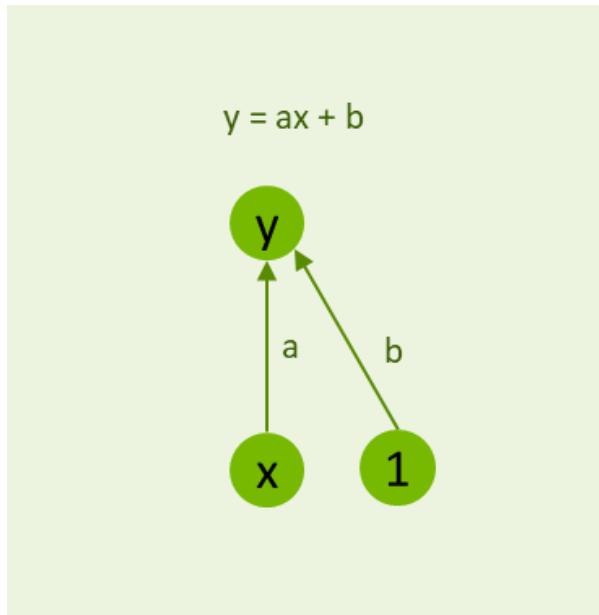
1 billion parameters 2 bytes per parameter 2 bytes per gradient 12 bytes per optimizer state



WHAT MAKES THEM POSSIBLE

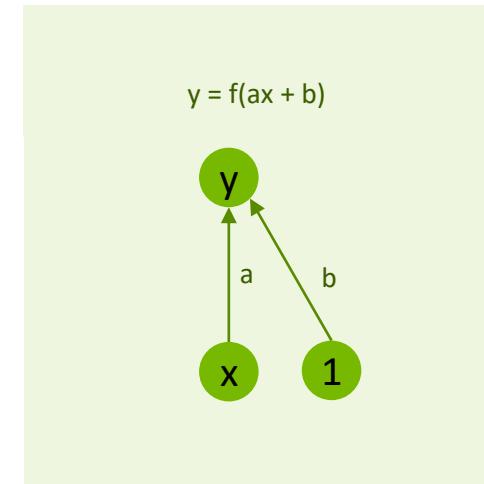
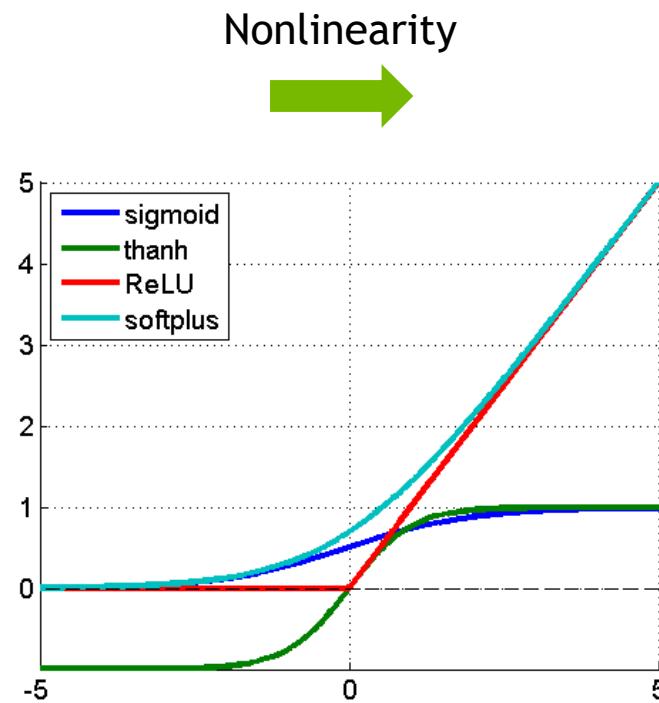
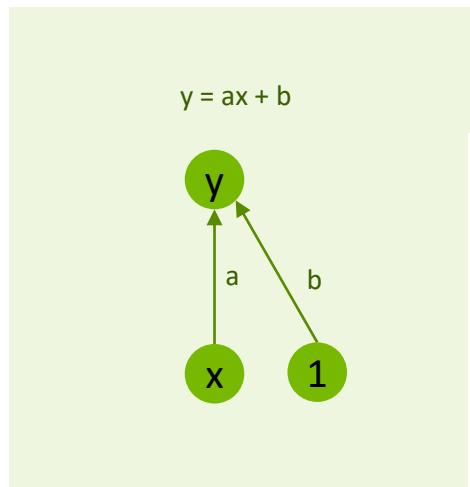
TRIVIAL EXAMPLE

Lets start with the simplest of problems - linear neuron



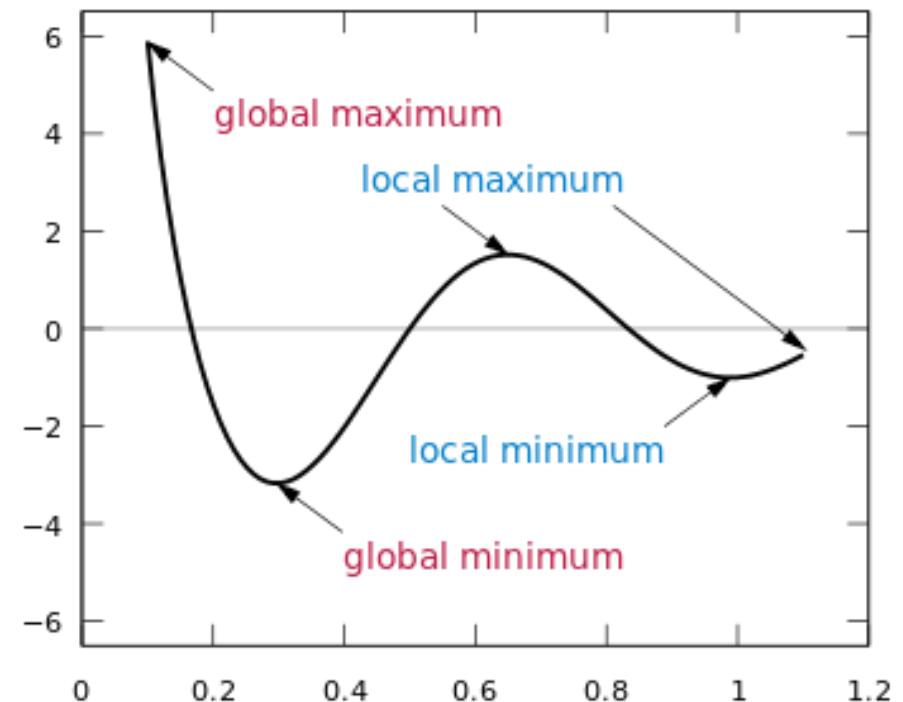
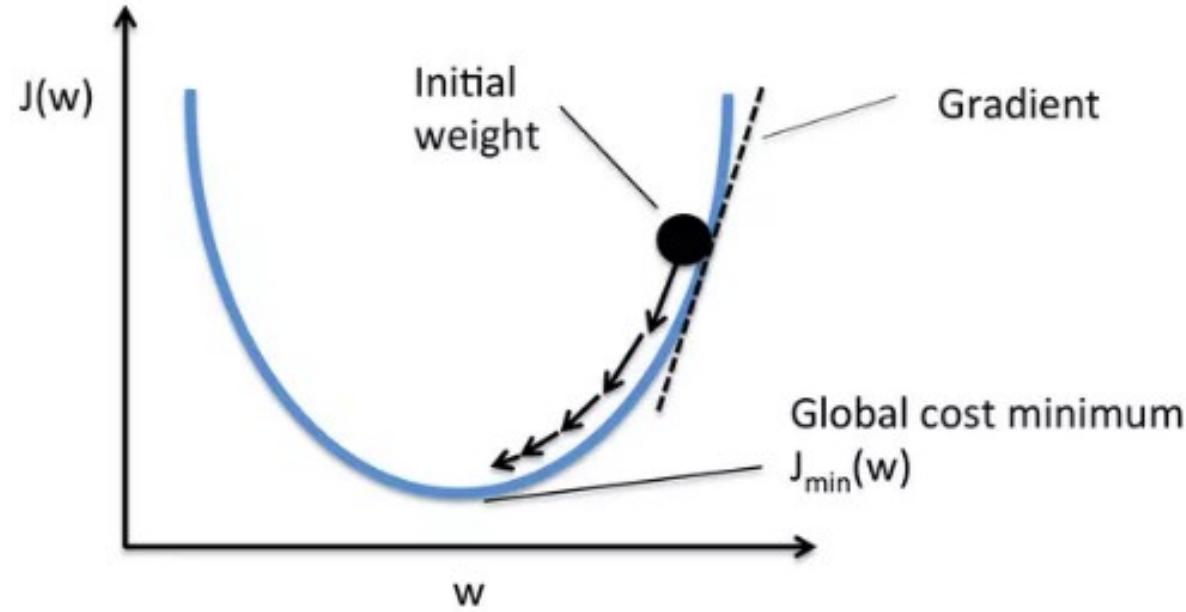
MODERN NEURAL NETWORKS

How do they differ from our trivial example?



NON CONVEX COST FUNCTIONS

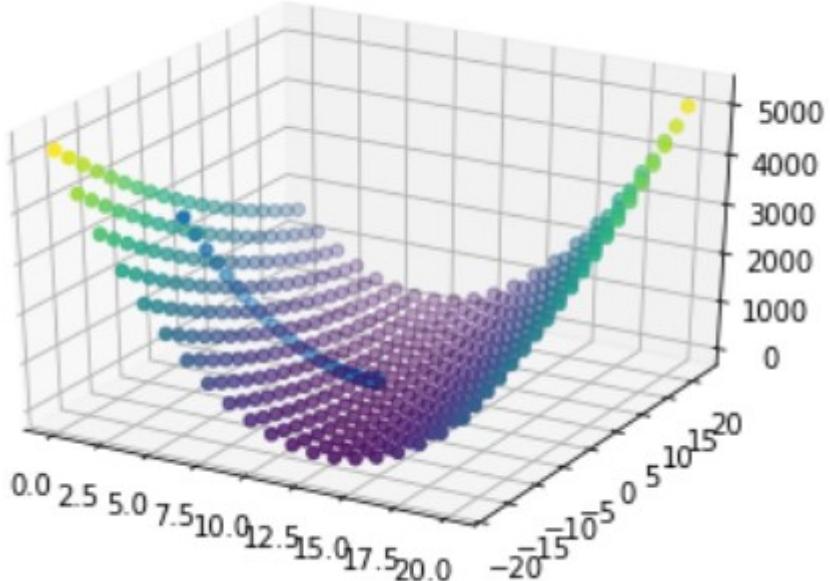
Those differences make the optimization problem much more difficult



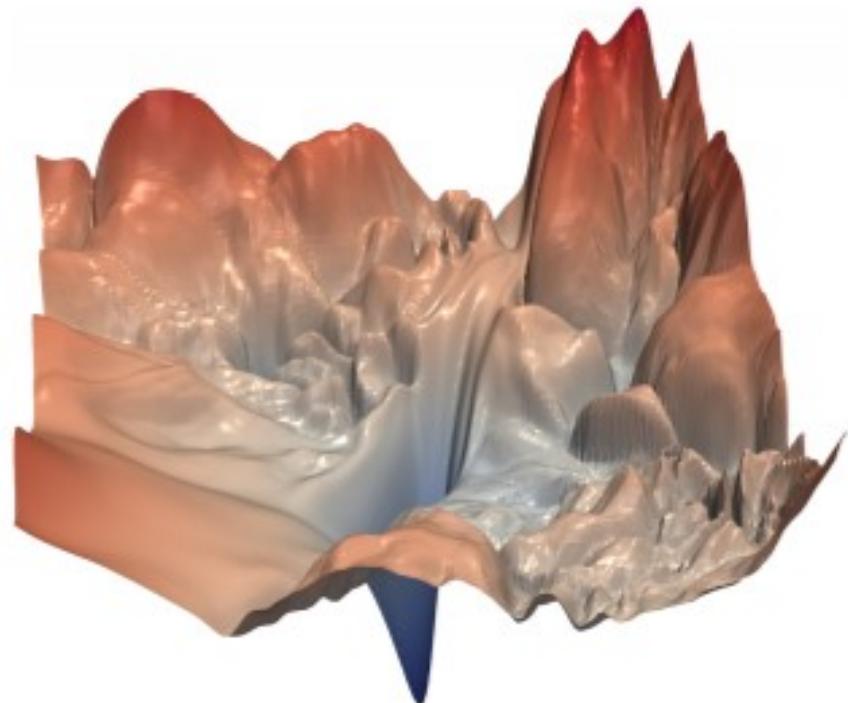
NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

Linear Neuron cost function



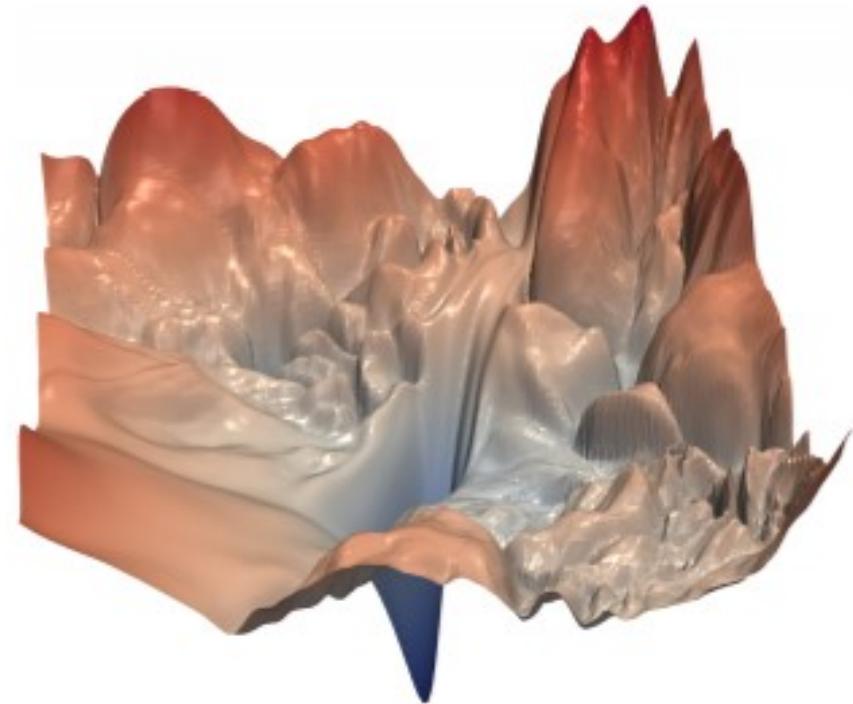
ResNet 56 cost function projection to 3D - no skip connections



NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

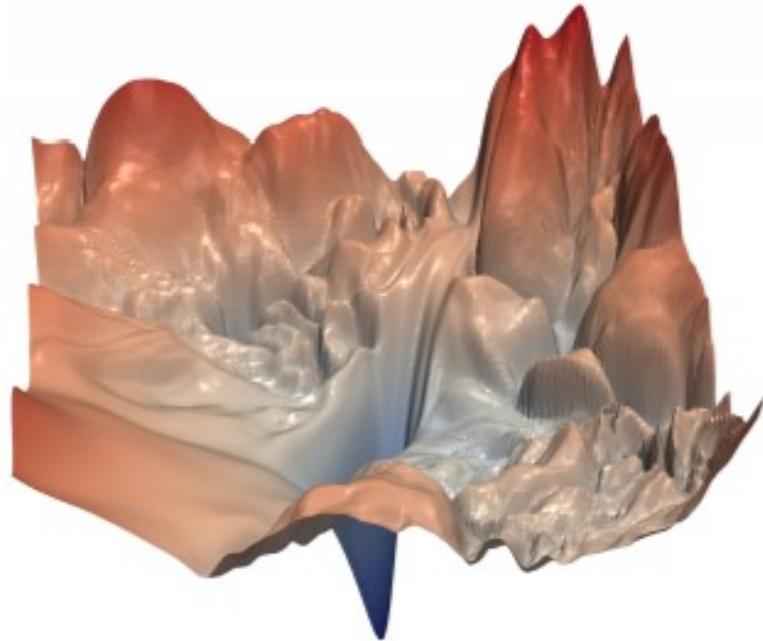
ResNet 56 cost function projection to 3D - no skip connections



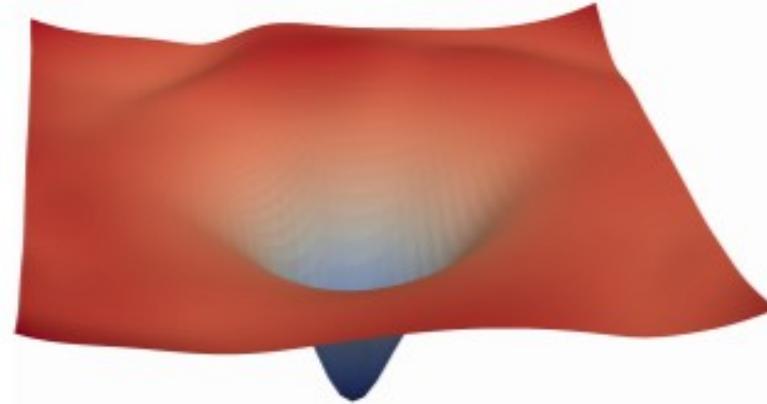
Why do we succeed in finding good local minima?

NON CONVEX COST FUNCTIONS

Recent advances such as Residual Connections simplify the optimization problem

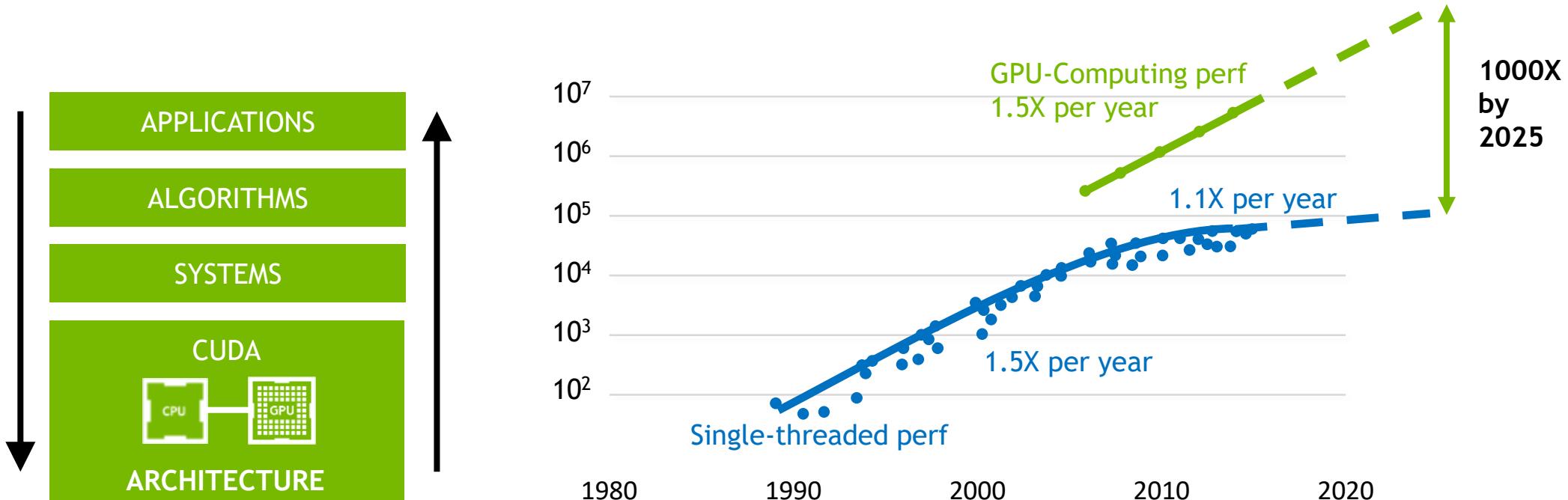


(a) without skip connections



(b) with skip connections

RISE OF GPU COMPUTING

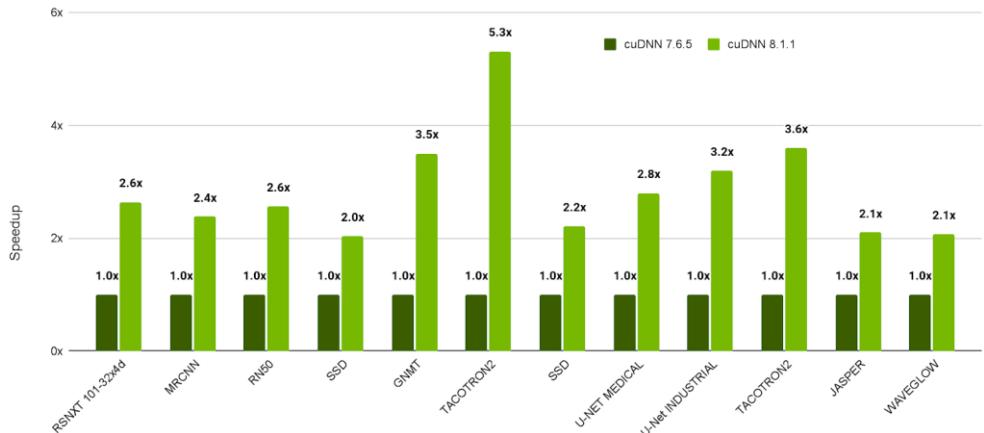


UTILIZING A SINGLE GPU EFFICIENTLY - PRELIMINARY COMMENTS

NVIDIA CUDA DEEP NEURAL NETWORK LIBRARY (cuDNN)

- GPU-accelerated library of primitives for deep neural networks.
- cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

A100 OVER 5X FASTER THAN V100 WITH CUDDNN 8.1



IMPORTANT CUDNN FLAGS

<https://pytorch.org/docs/stable/backends.html#torch-backends-cudnn>

Important aspects to consider:

- In NGC containers, the usage of TensorFloat-32 is enabled by default in order to accelerate FP32 calculations using tensor cores on Ampere or newer GPUs.
- Certain classes of CUDA functions are a potential source of non-determinism, such as atomicAdd, where the order of parallel additions to the same value is undetermined and, for floating-point variables, a source of variance in the results.
- cuDNN can automatically determine which combination of primitives is most optimal. Only use this flag when input sizes of a model are no changing!

```
# get the cuDNN version
torch.backends.cudnn.version()

# check availability
torch.backends.cudnn.is_available()

# enabling cuDNN (default = True)
torch.backends.cudnn.enabled = True

# enabling TF32 (default = True for DL)
torch.backends.cudnn.allow_tf32 = True

# enable determinism (default = False)
torch.backends.cudnn.deterministic = False

# enable auto-tuning (default = False)
torch.backends.cudnn.benchmark = True
```

WHY USING NGC CONTAINERS?

<https://catalog.ngc.nvidia.com/containers>

- Always the latest cuDNN version
- Access to all technologies we use for MLPerf (NCCL, SHARP, etc.)
- Achieve reproducibility

Catalog > Containers > PyTorch

PyTorch

Pull Tag Deploy to Vertex AI

Overview Tags Layers Security Scanning Related Collections

PyTorch

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs. Automatic differentiation is done with a tape-based system at both a functional and neural network layer level. This functionality brings a high level of flexibility and speed as a deep learning framework and provides accelerated NumPy-like functionality. NGC Containers are the easiest way to get started with PyTorch. The PyTorch NGC Container comes with all dependencies included, providing an easy place to start developing common applications, such as conversational AI, natural language processing (NLP), recommenders, and computer vision.

The PyTorch NGC Container is optimized for GPU acceleration, and contains a validated set of libraries that enable and optimize GPU performance. This container also contains software for accelerating ETL ([Dali](#), [RAPIDS](#)), Training ([cuDNN](#), [NCCL](#)), and Inference ([TensorRT](#)) workloads.

Prerequisites

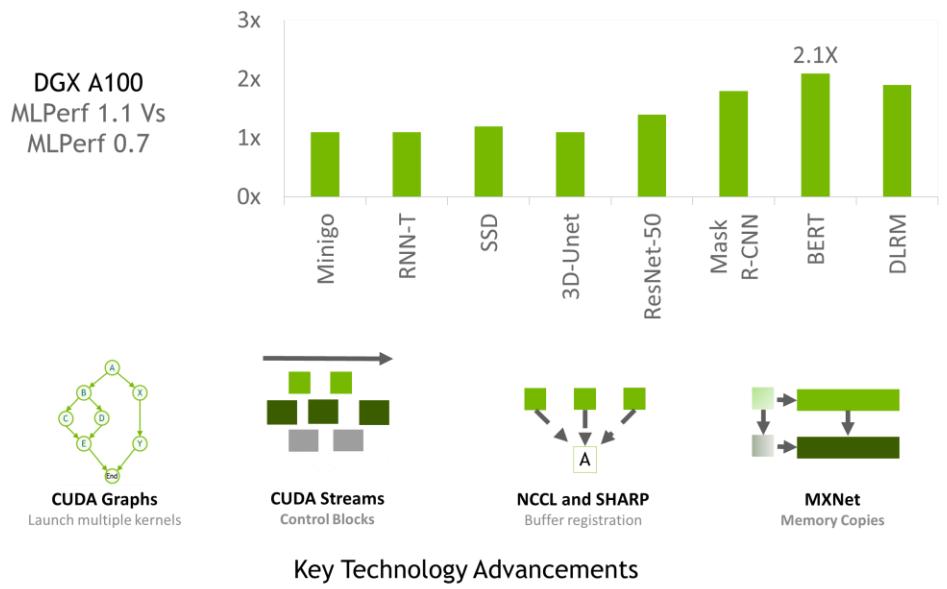
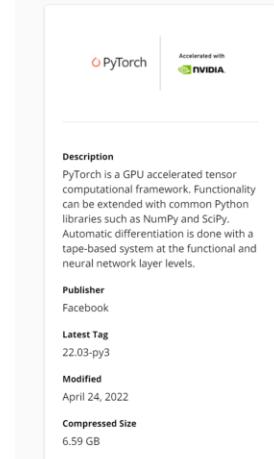
Using the PyTorch NGC Container requires the host system to have the following installed:

- Docker Engine
- NVIDIA GPU Drivers
- NVIDIA Container Toolkit

For supported versions, see the [Framework Containers Support Matrix](#) and the [NVIDIA Container Toolkit Documentation](#).

No other installation, compilation, or dependency management is required. It is not necessary to install the NVIDIA CUDA Toolkit.

Running PyTorch



A NOTE ON TIME MEASUREMENTS

<https://pytorch.org/docs/stable/backends.html#torch-backends-cudnn>

Important aspects to consider:

- Be careful with measuring time on the host.
- CUDA events are synchronization markers that can be used to monitor the device's progress, to accurately measure timing, and to synchronize CUDA streams.
- Make sure you are measuring large enough workloads.
- Always perform multiple repetitions and average the results.
- Never measure the 1st API call and perform GPU warmup.

```
start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)

start.record()
# code to be measured
...
end.record()

torch.cuda.synchronize()

elapsed_time_in_ms = start.elapsed_time(end)
```

**UTILIZING A SINGLE GPU EFFICIENTLY -
MAXIMIZING OCCUPANCY & UTILIZATION**

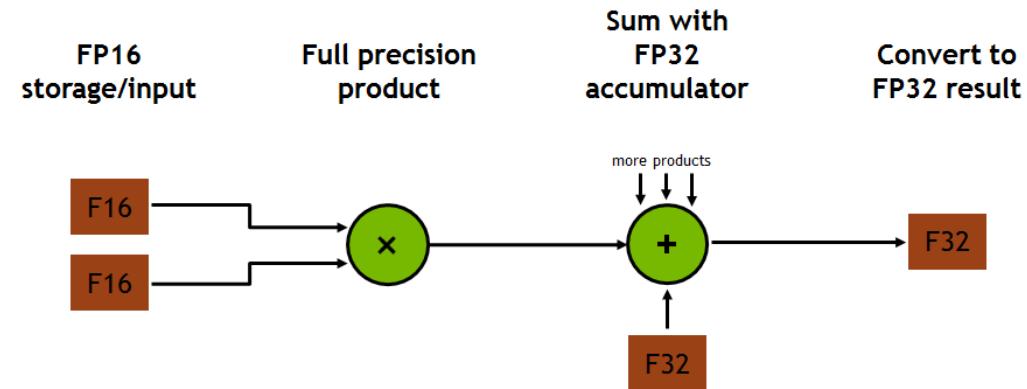
SIZING RECOMMENDATIONS

THE FUNDAMENTAL OPERATION OF DEEP LEARNING - FUSED MULTIPLY ADD

“Matrix tiles = toasts”

- Fused matrix multiply and accumulate (FMA) operations are the core operations of deep learning training and inference.
- 1st generation Tensor Cores (V100) perform 64 floating point FMA mixed-precision operations per clock (FP16 input multiply with full-precision product and FP32 accumulate), i.e., 4 4x4 matrix tiles.
- Higher generation Tensor Cores support additional precisions.

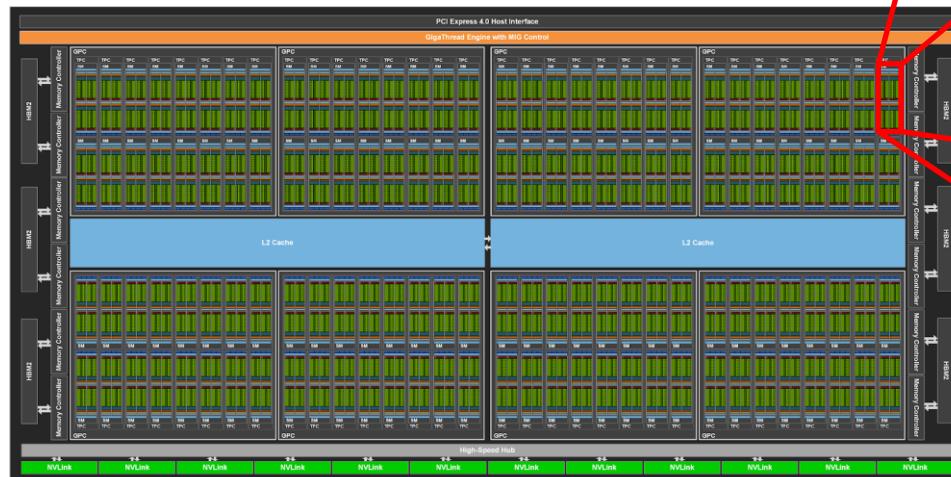
$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$



TENSOR CORES

“How many pans do it have?”

- Introduced in the Volta architecture to accelerate matrix multiply and accumulate operations.
- Specific unit of the Streaming Multiprocessors (SMs)
- For an example, an A100 has 108 SMs

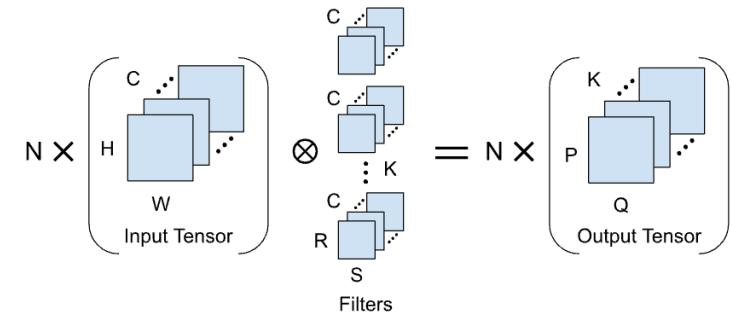


Actually, there are researchers trying to exploit tensor cores for reductions and other ops:
<https://arxiv.org/abs/1811.09736>

CHECKLIST FOR CONVOLUTIONAL LAYERS

<https://docs.nvidia.com/deeplearning/performance/dl-performance-convolutional/index.html#checklist>

- Choose the number of input and output channels to be divisible by 8 (for FP16) or 4 (for TF32). Also consider padding the input channels.
- Choose parameters (batch size, number of input and output channels) to be divisible by at least 64 and ideally 256 to enable efficient tiling and reduce overhead.
- Larger values for size-related parameters (batch size, input and output height and width, and the number of input and output channels) can improve parallelization and hence increase efficiency.
- Make sure auto-tuning is enabled, if applicable.
- Choose tensor layouts in memory to avoid transposing input and output data. We recommend using the NHWC format where possible.

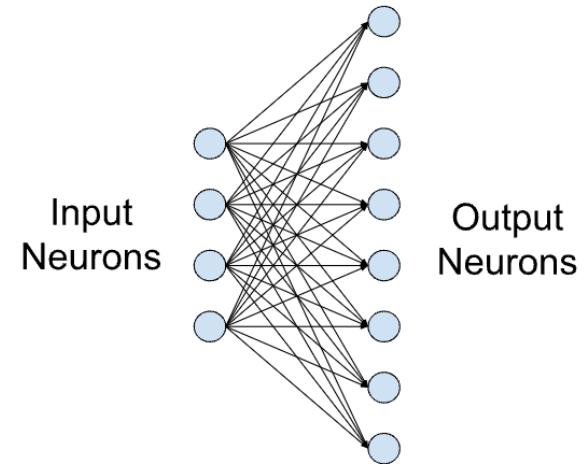


For specific guidance on 2D and particularly 3D convolutions, please also refer to
<https://docs.nvidia.com/deeplearning/cudnn/best-practices/index.html>

CHECKLIST FOR FULLY CONNECTED LAYERS

<https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#checklist>

- Choose the batch size and the number of inputs and outputs to be divisible by 4 (TF32) / 8 (FP16) / 16 (INT8) to run efficiently on Tensor Cores. For best efficiency on A100, choose these parameters to be divisible by 32 (TF32) / 64 (FP16) / 128 (INT8).
- Especially when one or more parameters are small, choosing the batch size and the number of inputs and outputs to be divisible by at least 64 and ideally 256 can streamline tiling and reduce overhead.
→ Larger values for batch size and the number of inputs and outputs improve parallelization and efficiency.
- As a rough guideline, choose batch sizes and neuron counts greater than 128 to avoid being limited by memory bandwidth (NVIDIA A100-SXM4-80GB; this threshold is similar for other A100 and V100 GPUs).



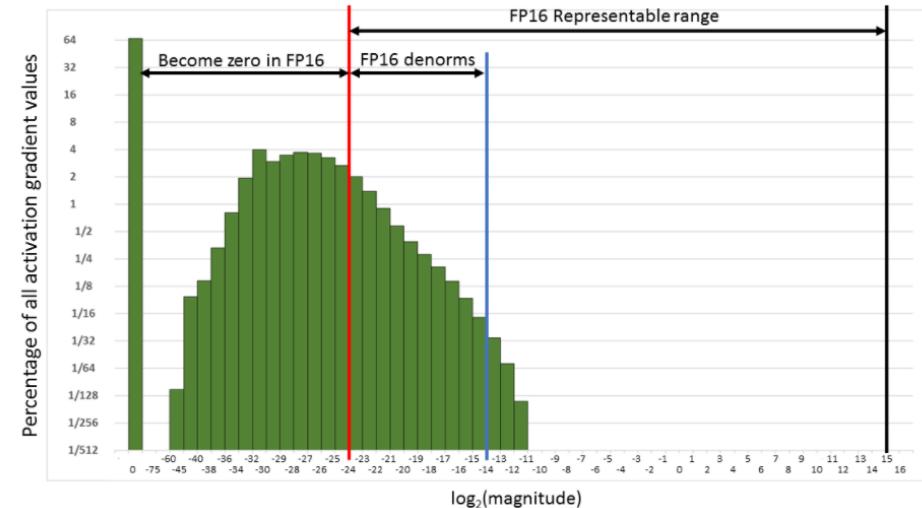
**UTILIZING A SINGLE GPU EFFICIENTLY -
MAXIMIZING OCCUPANCY & UTILIZATION**

TENSOR CORE UTILIZATION

MIXED PRECISION TRAINING - THE IDEA

Example: FP32 training of Multibox SSD network

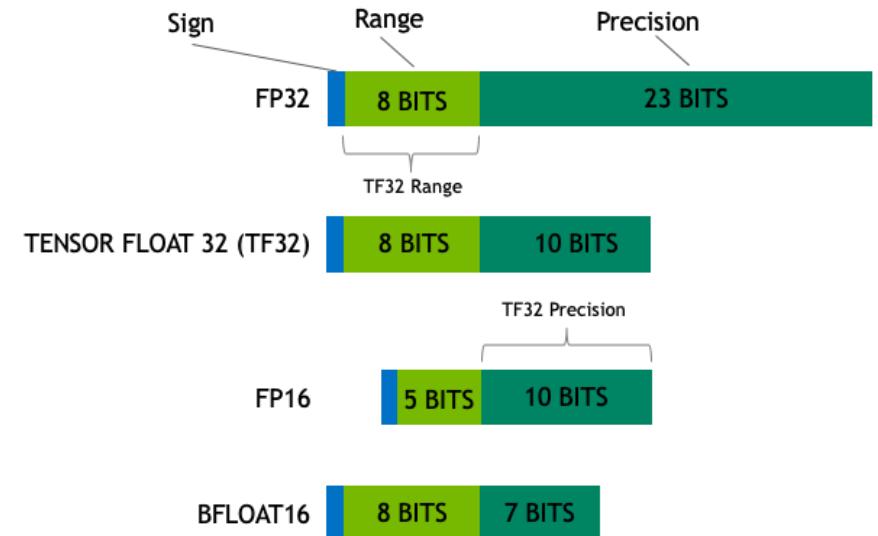
- Histogram shows activation gradient magnitudes throughout FP32 training; both axes are logarithmic.
- Observations:
 - Dynamic range of FP16 would be sufficient to cover the entire histogram. ☺
 - Without “shifting” the histogram, half of the activations would be casted to 0, however. ☹
- Idea: “shifting” = multiplication with a scale factor!
- Concern: Do I need to run a full training in order to find the scaling factor?
→ No, automatic mixed precision comes to the rescue! ☺



A NOTE ON DATA TYPES

Or why TF32 makes sense

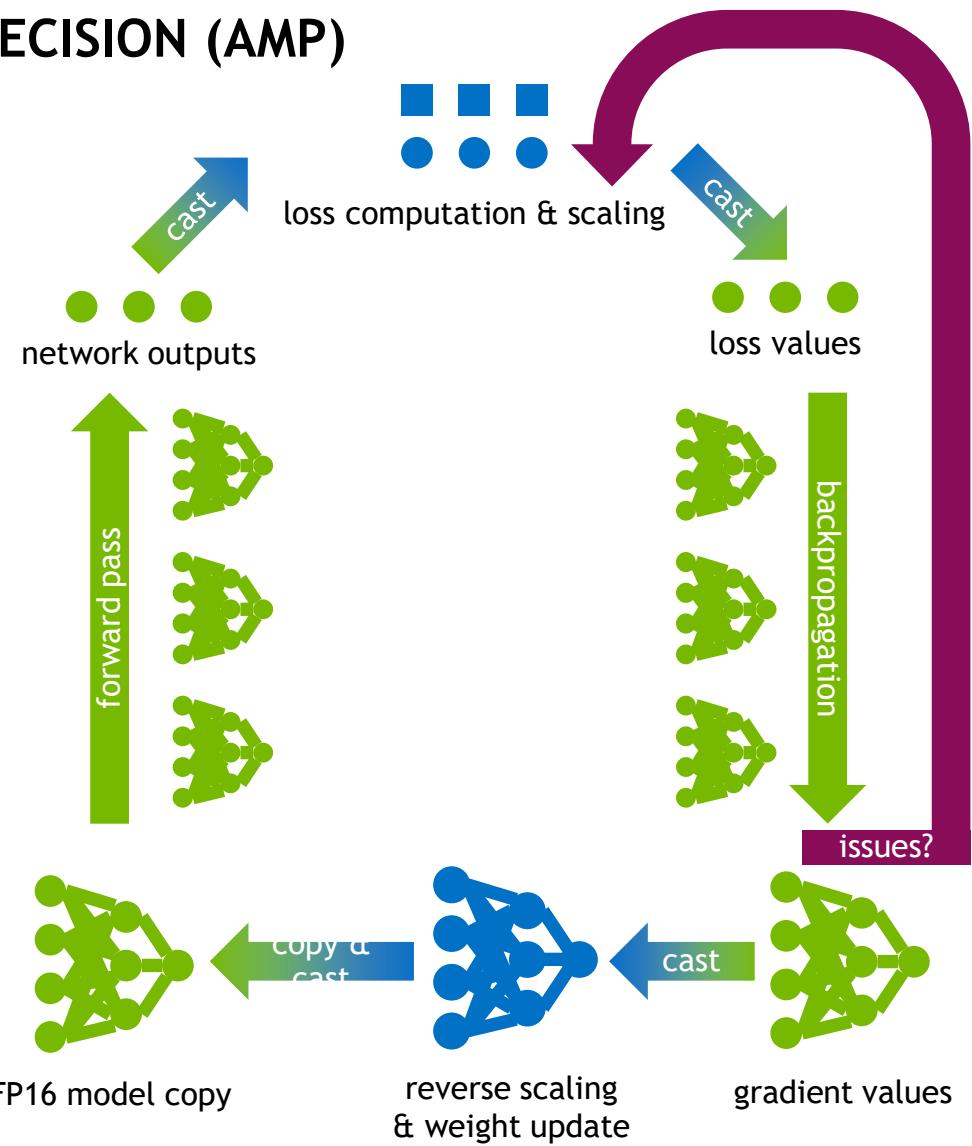
- Mixed precision training is mostly about the dynamic range and less about the precision:
 - exponent → dynamic range
 - significand field → precision
- TF32 is a great compromise between FP32 (same range) and FP16 (same precision)
- TF32 is automatically enabled in NGC containers
- No code change is necessary!



AUTOMATIC MIXED PRECISION (AMP)

Concept

- Maintain a primary copy of weights in FP32.
- Initialize scaling factor S to a large value.
- For each iteration:
 - Make an FP16 copy of the weights.
 - Forward propagation (FP16 weights and activations).
 - Multiply the resulting loss with the scaling factor S .
 - Backward propagation (FP16 weights, activations, and their gradients).
 - If there is an Inf or NaN in weight gradients:
 - Reduce S .
 - Skip the weight update and move to the next iteration.
 - Multiply the weight gradient with $1/S$.
 - Complete the weight update (including gradient clipping, etc.).
 - If there hasn't been an Inf or NaN in the last N iterations, increase S .



HOW TO USE IT?

in pytorch

- Backward passes under autocast are not recommended.
- Backward ops run in the same dtype autocast chose for corresponding forward ops.
- `scaler.step()` first unscales the gradients of the optimizer's assigned params.
- If these gradients contain infs or NaNs, `optimizer.step()` is skipped.

```
# initialize gradient scaler
scaler = GradScaler()

# training loop
for epoch in epochs:
    for input, target in data:

        # zero gradient buffers
        optimizer.zero_grad()

        # forward pass with autocasting
        with autocast():
            output = model(input)
            loss = loss_fn(output, target)

        # call backward() on scaled loss
        scaler.scale(loss).backward()

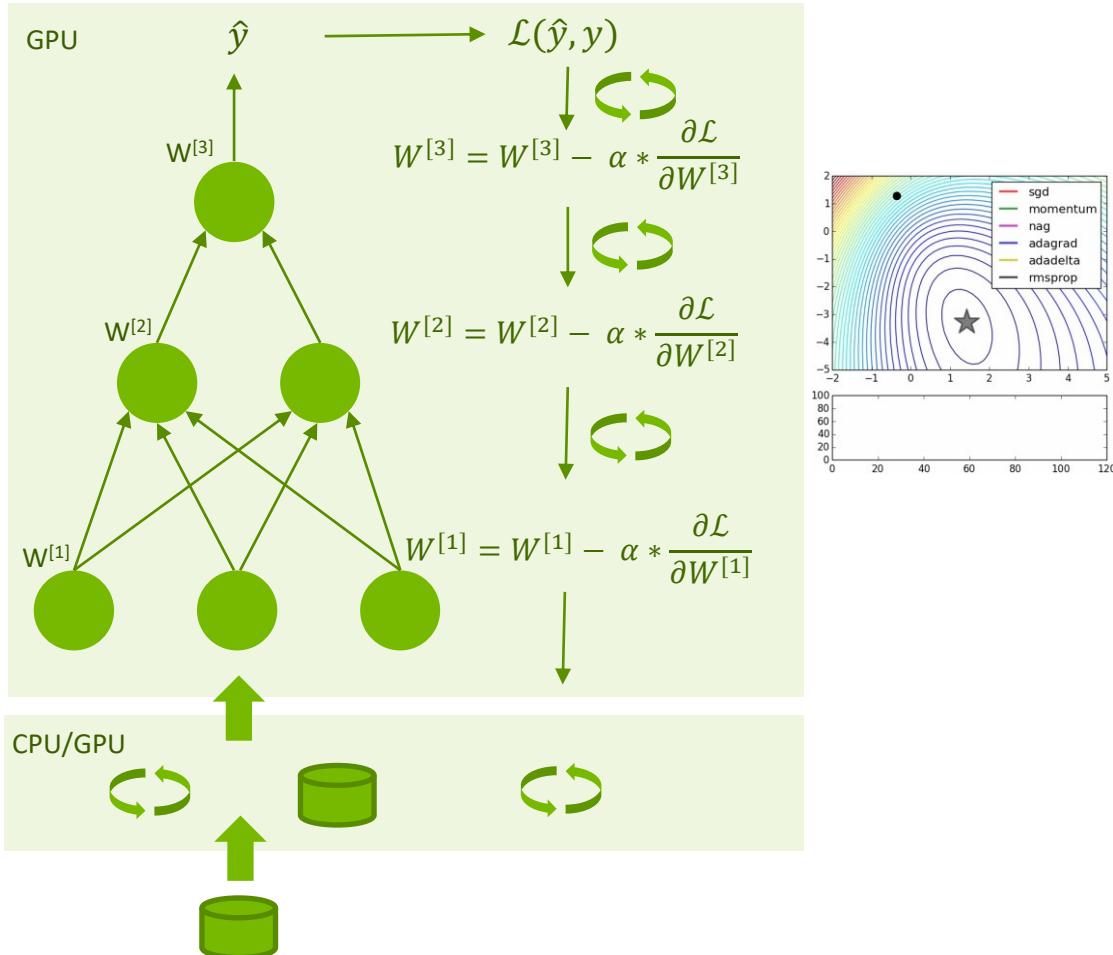
        # update if no issues
        scaler.step(optimizer)

        # updates the scale for next iteration.
        scaler.update()
```

PARALLELISM

TRAINING A NEURAL NETWORK

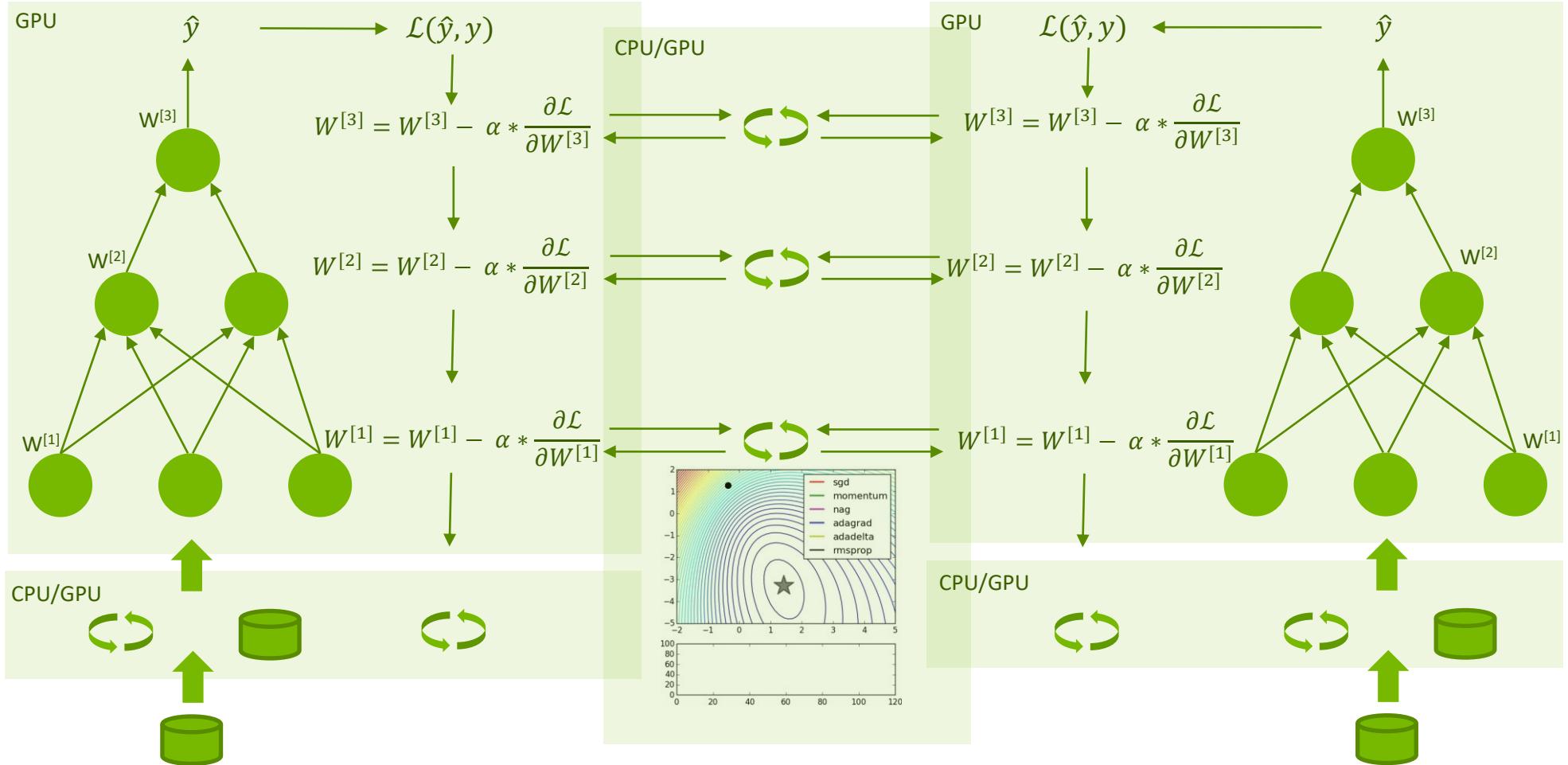
Single GPU



1. Read the data
2. Transport the data
3. Pre-process the data
4. Queue the data
5. Transport the data
6. Calculate activations for layer one*
7. Calculate activations for layer two
8. Calculate the output
9. Calculate the loss
10. Backpropagate through layer three
11. Backpropagate through layer two
12. Backpropagate through layer one
13. Execute optimisation step
14. Update the weights
15. Return control

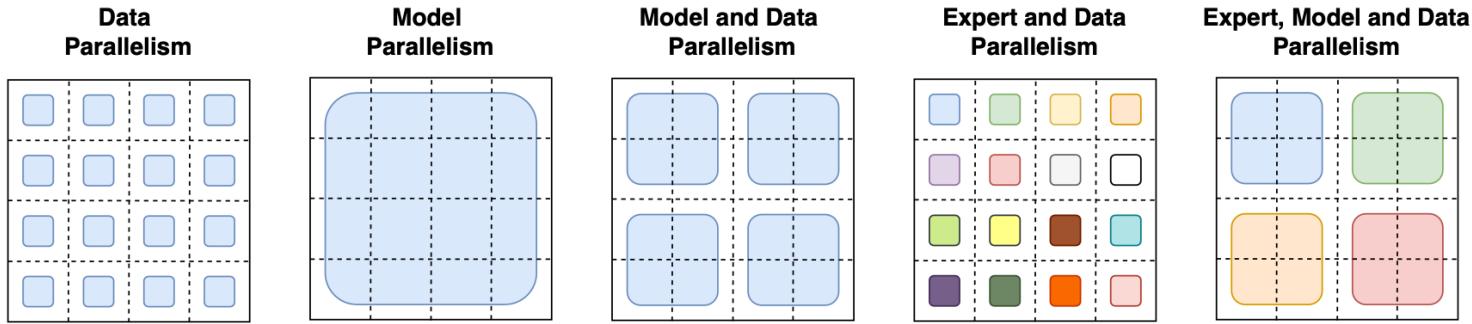
TRAINING A NEURAL NETWORK

Multiple GPUs

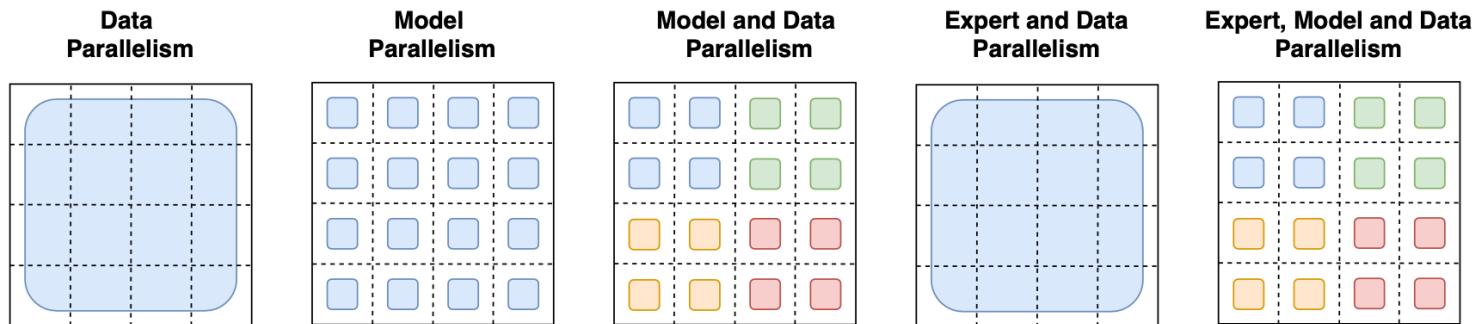


DATA AND MODEL PARALLELISM STRATEGIES

How the *model weights* are split over cores



How the *data* is split over cores



MEGATRON

Model Parallel Transformer

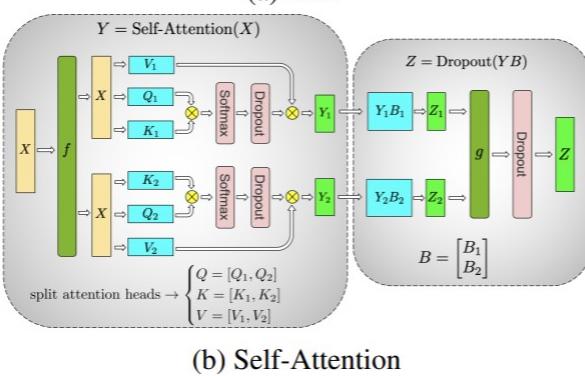
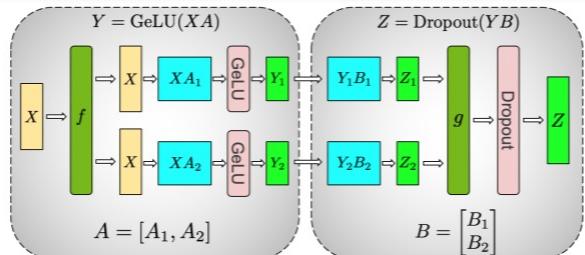


Figure 3. Blocks of Transformer with Model Parallelism. f and g are conjugate. f is an identity operator in the forward pass and all reduce in the backward pass while g is an all reduce in the forward pass and identity in the backward pass.

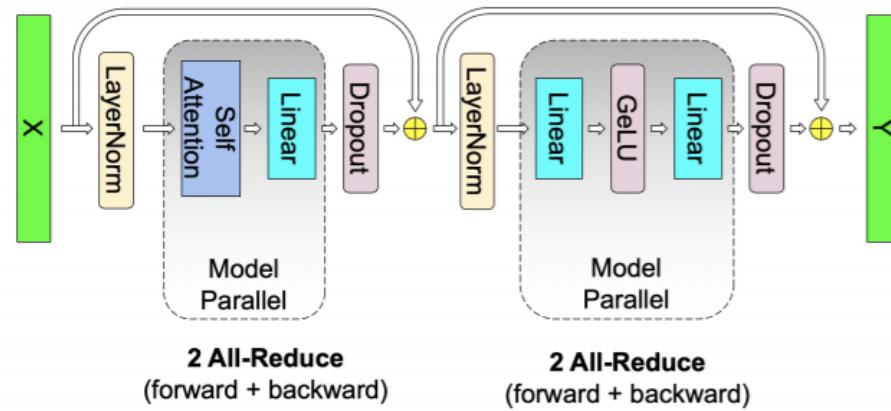


Figure 4. Communication operations in a transformer layer. There are 4 total communication operations in the forward and backward pass of a single model parallel transformer layer.



ITALIAN PERSPECTIVE

COLLABORATION FRAMEWORK

The Italian centre builds on the collaboration of three entities



- Consortium of 41 universities in the fields of Computer Engineering, Computer science, and Information technologies. More than 1300 professors involved.
 - Includes 11 labs spanning from Artificial Intelligence, Cyber Security, BigData to Digital Health.
 - The AI Lab represents the consortium within the NVAITC.
-
- Not-for-profit consortium, made up of the Italian Ministry of Education Universities and Research, 69 Italian universities, 11 Italian National Institutions.
 - One of the largest HPC facilities in Europe and hosting entity of the Leonardo EuroHPC system.

Italy Forges AI Future in Partnership with NVIDIA

Collaboration begins with a research hub at AlmageLab in Modena.
January 15, 2020 by FREDERIC PARIENTE



MAIN AREAS OF CONTRIBUTION

What we do in practice

- ▶ Enable academics at all levels to do their own research more efficiently
 - ▶ Adoption of DL/ML frameworks (NVIDIA heavily contributes to DL frameworks development)
 - ▶ Technology selection and optimization (efficient data loading, mixed-precision, inference)
 - ▶ Model architectural choices
 - ▶ Contribution to software development
 - ▶ Performance optimization and tuning through profiling
 - ▶ Workload scaling on multi GPUs/nodes
 - ▶ Discussion on research studies
 - ▶ Training
 - ▶ Support to access HPC resources

On Improving the Training of Models for the Semantic Segmentation of Benthic Communities from Orthographic Imagery

The semantic segmentation of underwater imagery is an important step in the ecological analysis of coral habitats. To date, scientists produce fine-scale area annotations manually, an exceptionally time-consuming task that could be efficiently automated by modern CNNs. This work extends previous results presented at the 3DUW'19 conference, outlining the workflow for the automated annotation of imagery from the first step of dataset preparation, to the last step of prediction reassembly. We propose an ecologically inspired strategy for an efficient dataset partition, an over-sampling methodology targeted on ortho-imagery, and a score fusion strategy. The experimental results demonstrate the effectiveness of the ecologically inspired split in improving model performance, and quantify the advantages and limitations of the proposed over-sampling strategy.

Learning to Re-Attention Dict

We present EgoACO, which learns to pool action regions by leveraging the hierarchical structure of action labels in egocentric video datasets. The core component of EgoACO is class activation pooling (CAP), a differentiable pooling operation that combines ideas from bilinear pooling for fine-grained recognition and from feature learning for discriminative localization. CAP uses self-attention with a dictionary of learnable weights to pool from the most relevant feature regions. Through CAP, EgoACO learns to decode object and scene context descriptors from video frame features. For temporal modeling in EgoACO, we design a recurrent version of class activation pooling termed Long Short-Term Attention (LSTA).



<https://www.mdpi.com/2072-4292/12/18/3106>

Towards Sustainable Action Recognition

Efficiency plays a key role in action recognition for real-time production scenarios. In this paper, we study the complexity of a variety of architectural changes to reduce the computational footprint of a network. We propose a novel architecture sharing scheme that optimizes the sequence of operations in a coordinate-descent schema, and the approach transfers knowledge from both the initial network and previous stages of the shrinking process by employing a Knowledge Distillation and an adaptive fine-tuning strategy. Experimental evaluations, conducted on two backbones and on Kinetics-400, showed the effectiveness of the approach in reducing FLOPs while maintaining accuracy. Further, the transfer capabilities of the obtained networks on UCF101 and HMDB51 was tested with success.



<http://conferences.visionbib.com/2020/pami-ego-1-20-call.html>

Novel Object Captioning

Image Captioning is the task of generating a natural language description of the visual content of an image. Most existing approaches focus on the task of generating captions for objects that are built on a number of in-domain objects, i.e., this by trying to generalize objects unseen during the training phase. In this work, we propose a novel approach to the existing model (Meshed-Memory Transformer for image Captioning) to be able to capture objects which are not present into the training set and improve its overall performance. However, this has not been possible due to the contextual publications of very large and important projects, i.e., VIVO, OSCAR, DETR, which have surpassed the initial idea of the group. The project has been useful to evaluate the new published architectures and review the research agenda.



NVIDIA Developer Ecosystem

Driving adoption with developers



NVIDIA Developer Program

- 3.5 Million Developers
 - 150+ SDKs
 - Early access programs
 - NVIDIA On-Demand videos
 - DevZone / Forums
 - Technical blogs



NVIDIA Inception

- 12,000 Startups
 - Acceleration program
 - Technical guidance
 - Cloud credits
 - NVIDIA tech discounts
 - Training credits
 - Go-to-market support



Deep Learning Institute (DLI)

- 350,000 Devs Trained
 - Hands-on, self paced courses
 - Live, instructor-led workshops
 - Educator programs and teaching kits



Higher Education & Research Programs

- 600 Projects/ Year
 - Hardware Grant Program
 - Applied Research Accelerator Program
 - DevRel outreach



NVIDIA Developer Program

The community that builds

Tools

- 450+ exclusive SDKs and models
- GPU-optimized software, model scripts, and containerized apps
- Early access programs and the NVIDIA Academic Hardware Grant Program*

Training

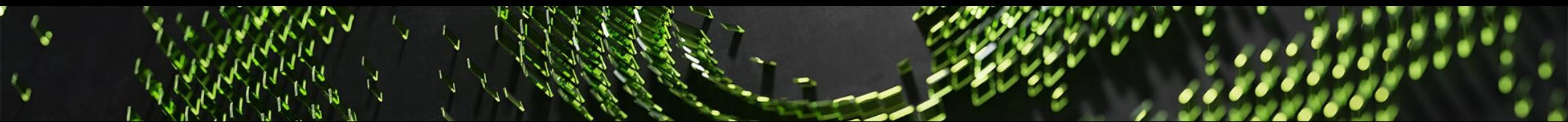
- Research papers, technical documentation, webinars, blogs, and news
- Technical training and certification
- Access to 1,000s of technical sessions on NVIDIA On-Demand

Community

- NVIDIA developer forums
- Exclusive meetups, hackathons, and events



[Join Our Community](#)



MANY THANKS!