AI @ Scale

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

ABOUT ME

Giuseppe Fiameni – gfiameni@nvidia.com

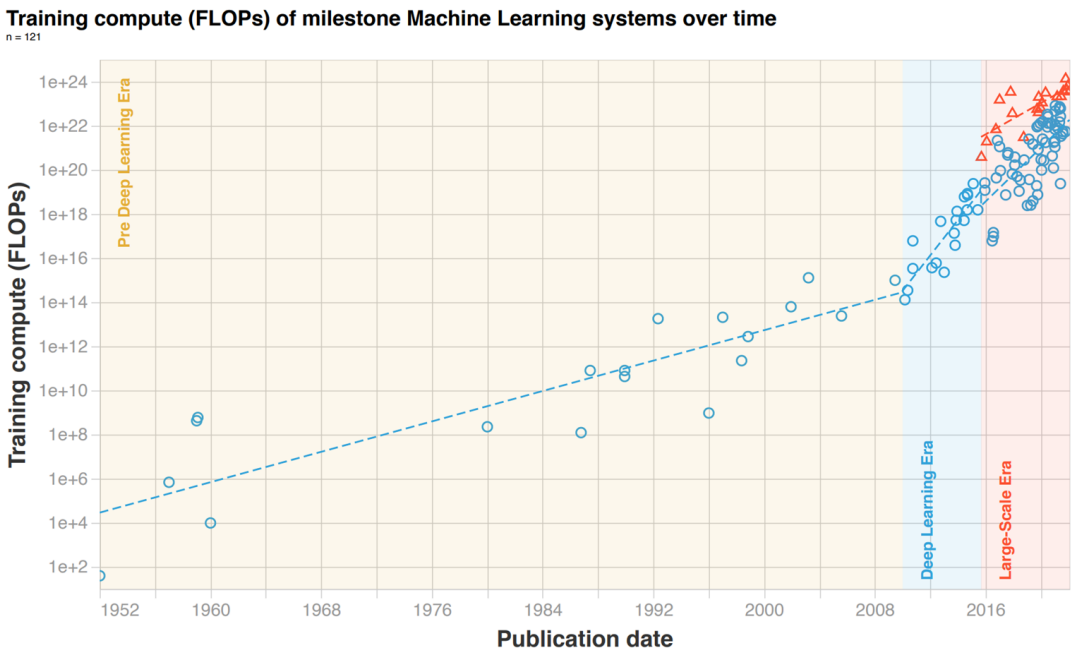
- Data Scientist & AI Expert @ NVIDIA
  - *Support Higher Education and Research through collaboration projects*
- Lead engineer of the NVIDIA AI Technology Center, Italy
  - *Agreement among CINI, CINECA and NVIDIA to accelerate scientific discovery*
- 10+ years experience delivering HPC applications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




MODULUS
CLARA MONAI
RIVA
MAXINE
NEMO
MERLIN
AVATAR
DRIVE
ISAAC
METROPOLIS
HOLOSCAN
APPLICATION FRAMEWORKS
PLATFORM
NVIDIA HPC
NVIDIA AI
NVIDIA OMNIVERSE
SYSTEM SOFTWARE
RTX
CUDA-X
PHYSX
UCF
DOCA
MAG
BASE CMD
FLEET CMD
AERIAL
HARDWARE
RTX
DGX
HGX
EGX
OVX
SUPER POD
AGX
GPU
CPU
DPU
NIC
SWITCH
SOC

NVIDIA H100

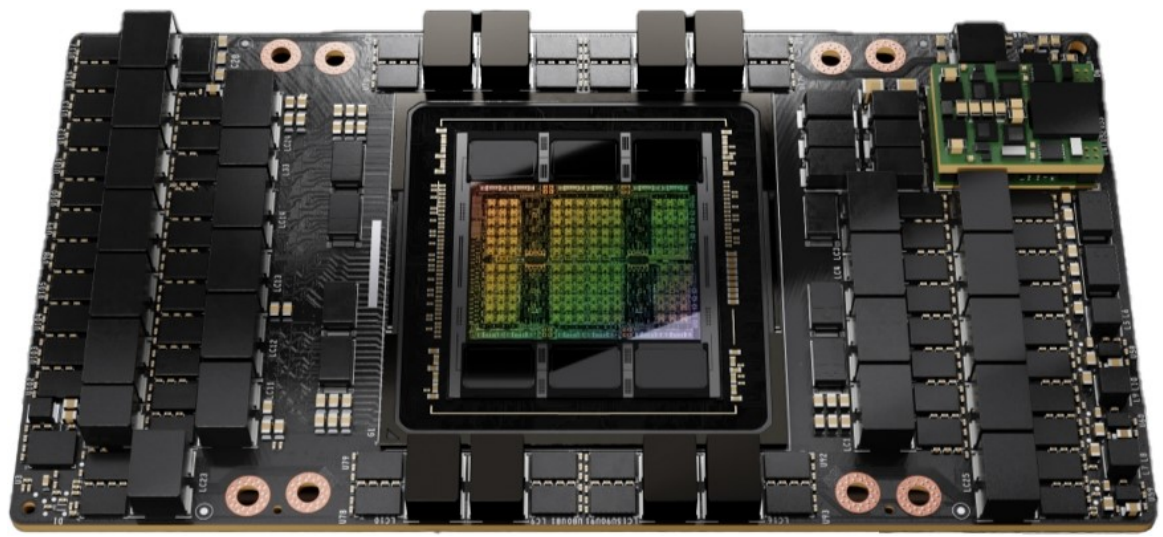
Unprecedented Performance, Scalability, and Security for Every Data Center

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

TRANFORMER 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



FP8, FP16, TF32 performance include sparsity. X-factor compared to A100

Grace Hopper

CPU+GPU Designed for Giant Scale AI and HPC

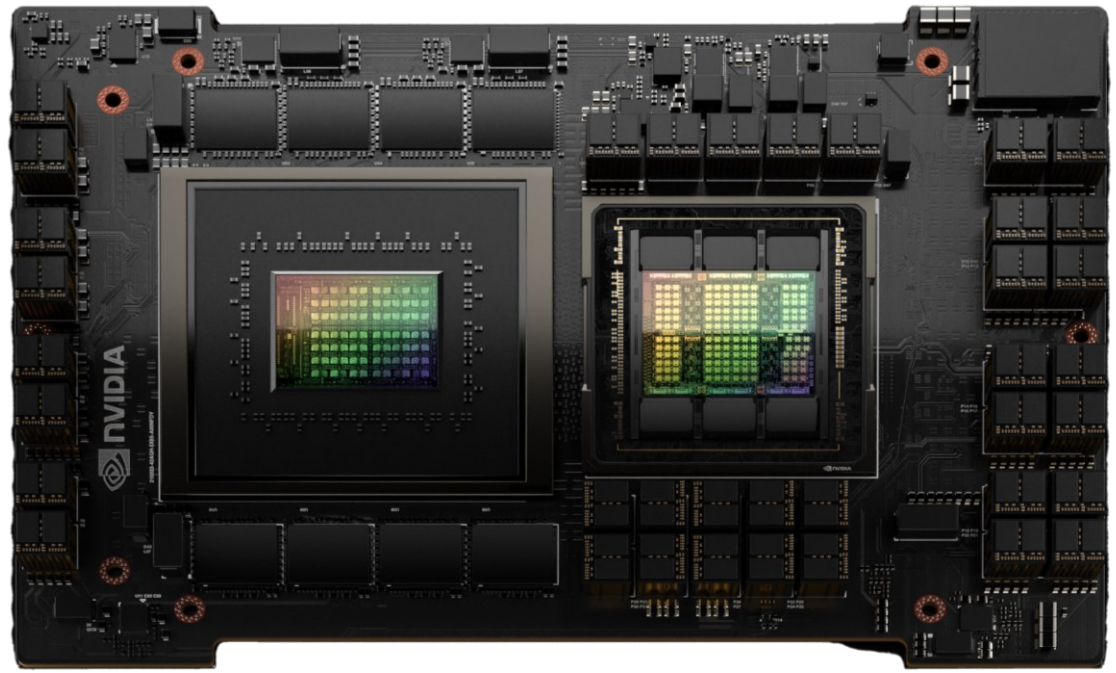
600GB Memory GPU for Giant Models

New 900 GB/s Coherent Interface

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

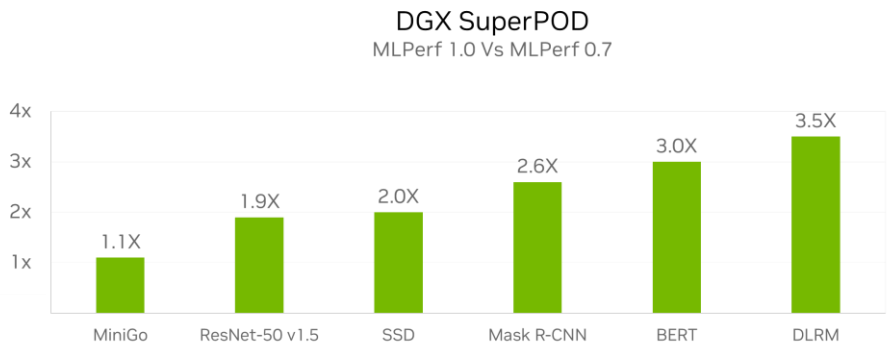
Runs Nvidia Computing Stacks

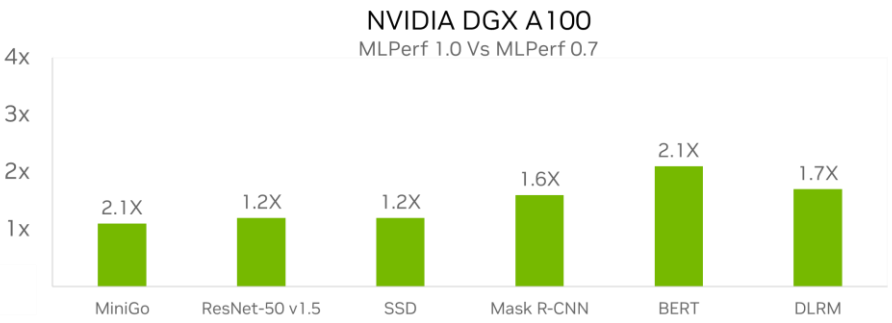
Available 1H 2023

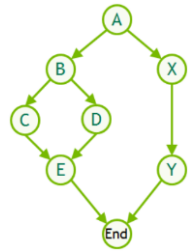


NVIDIA AI DELIVERS 3.5X HIGHER PERFORMANCE IN 1 YEAR

Fueled by continuous platform innovation



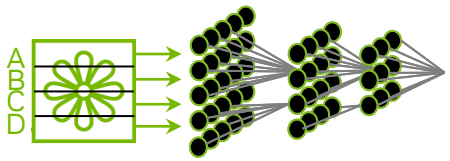




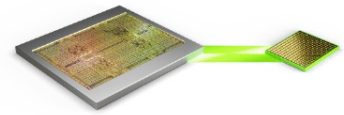
CUDA Graphs
Up to 1.7x

SHARP
Up to 1.2x

Larger Scale
Up to 4096 GPUs



Spatial Data Parallelism
Up to 1.6x



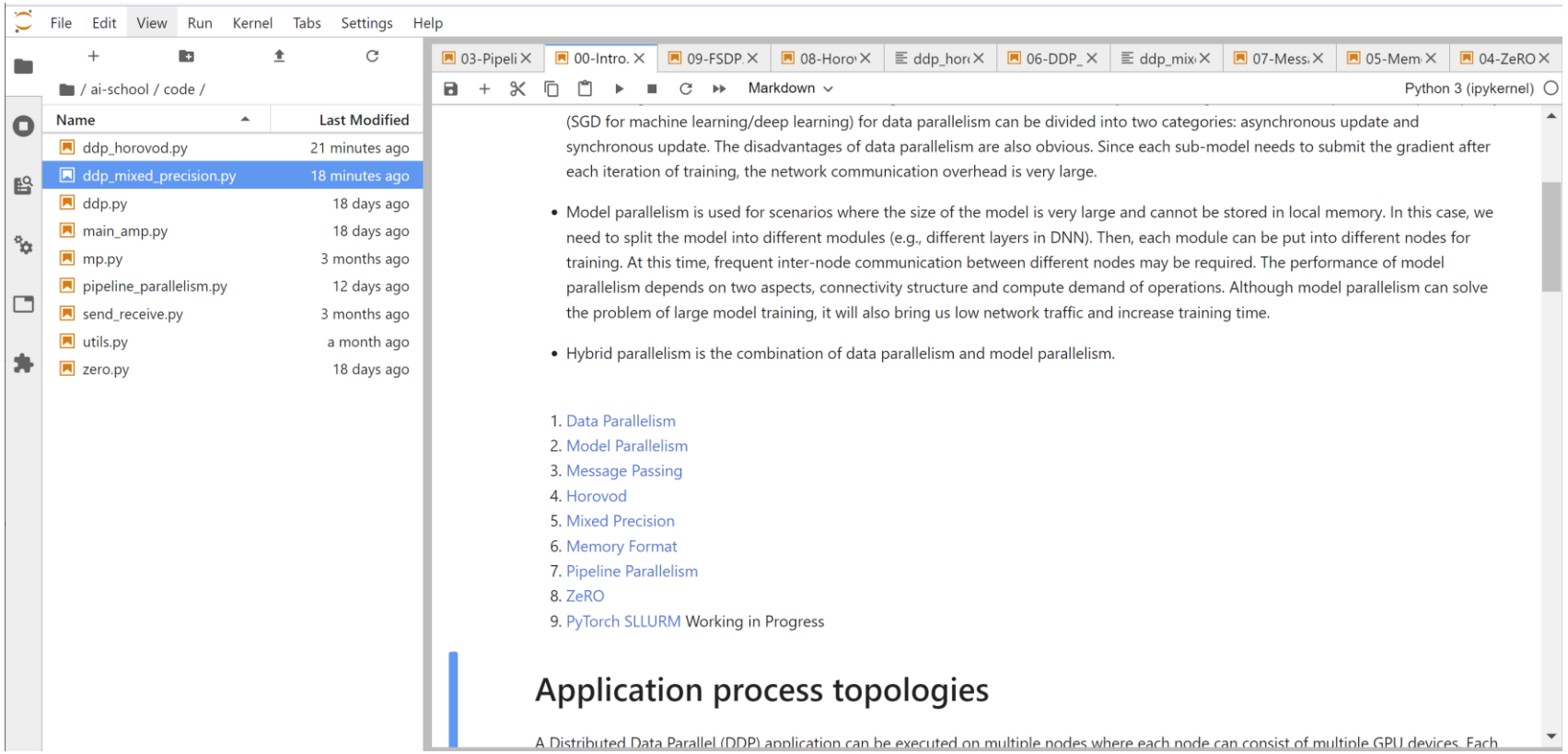
HBM2e
Memory Bandwidth >2TB/s

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 | MLPef 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

NVIDIA

NOTEBOOKS & CODE

https://github.com/gfiameni/hpdl/



AGENDA

Why Large datasets and Large Neural Networks?
Why NN models size keep increasing

Challenges of Supervised training

Scaling law
What governs model scaling

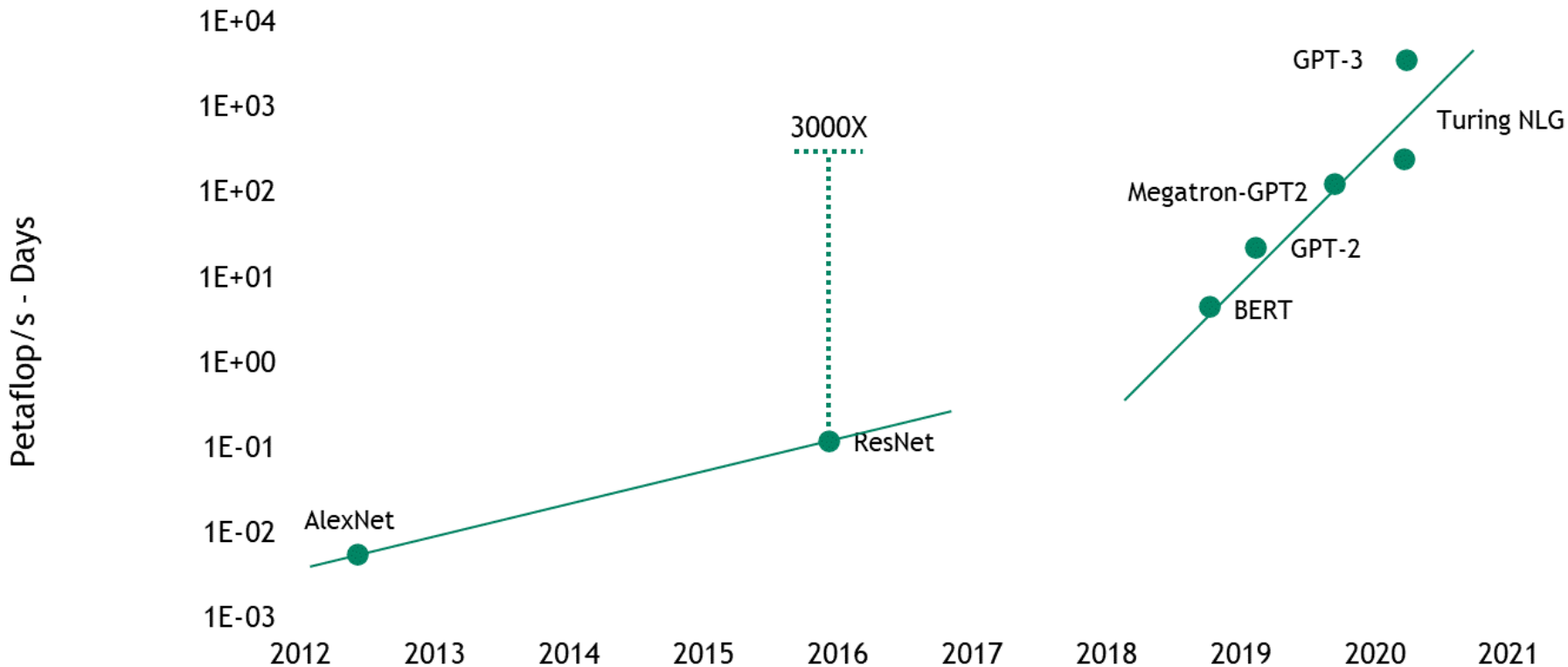
Training large neural networks

Data Parallelism vs Model Parallelism

Code samples

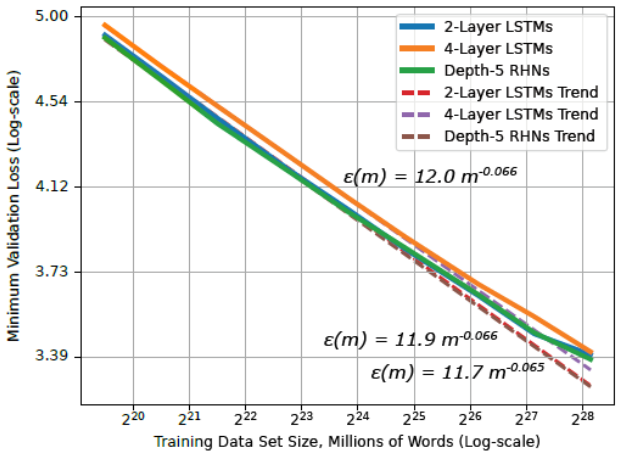
WHY LARGE DATASETS AND MODELS

New trend in model growth

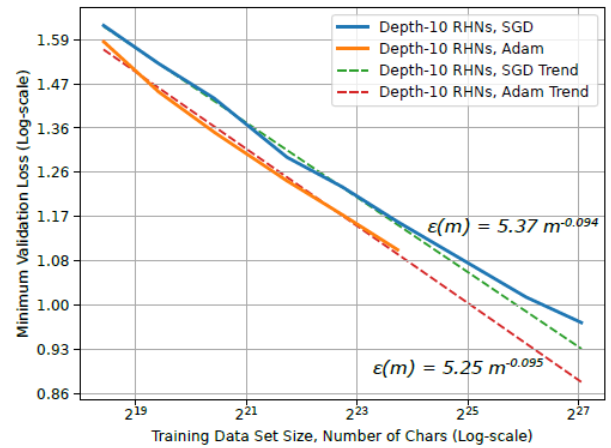


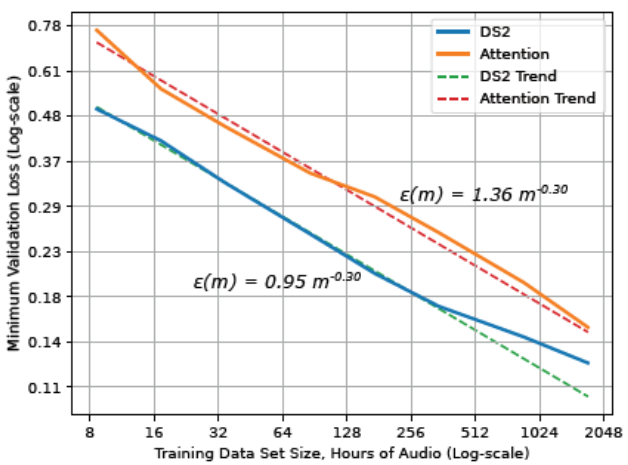
RELIABLE WAY FOR IMPROVING MODEL PERFORMANCE

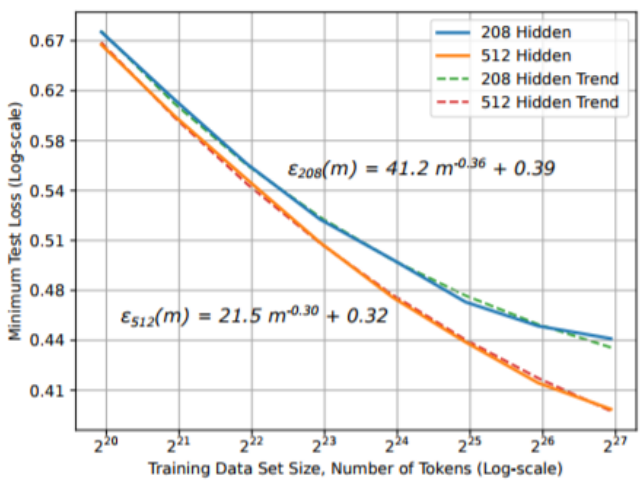
Logarithmic relationship between the dataset size and accuracy

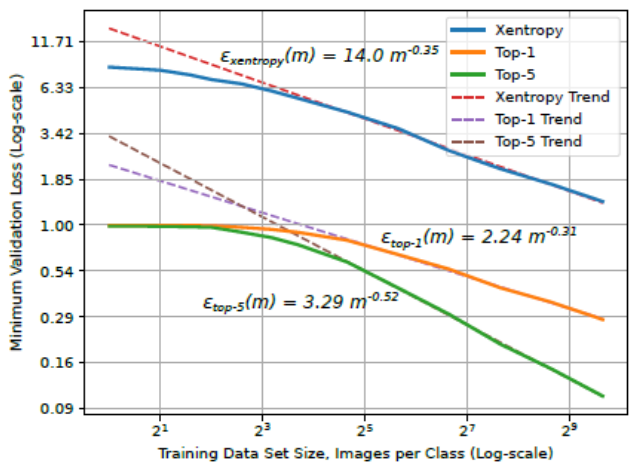


- Translation
- Language Models
- Character Language Models
- Image Classification
- Attention Speech Models





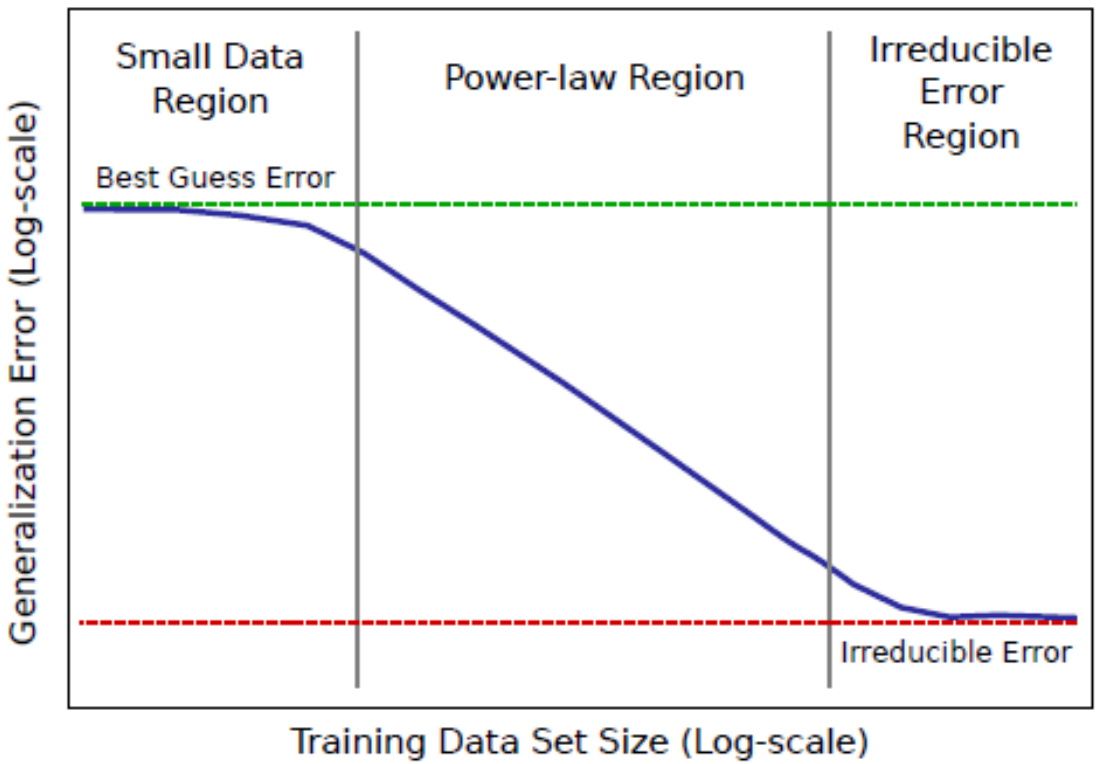




Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., ... & Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically. arXiv preprint arXiv:1712.00409.

NEW PROMISE

Logarithmic relationship between the dataset size and accuracy



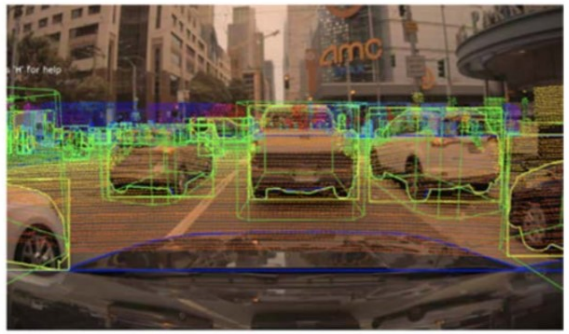
Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., ... & Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically. arXiv preprint arXiv:1712.00409.

CHALLENGES OF SUPERVISED TRAINING

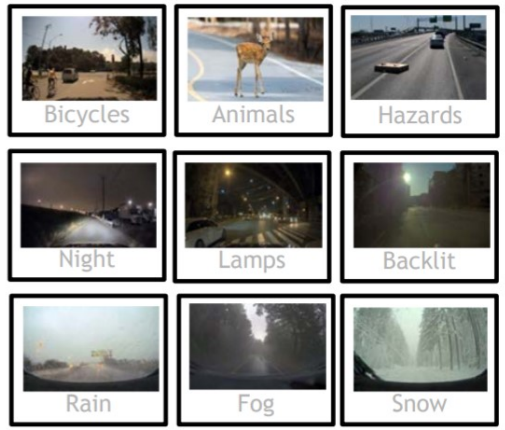
THE COMPLEXITY

Direct impact on data collection and labelling effort

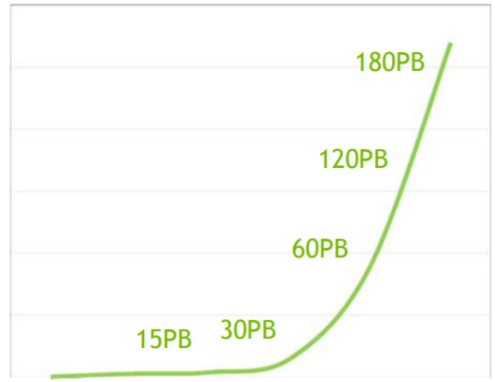
More Functionality × More Conditions = MASSIVE Data



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



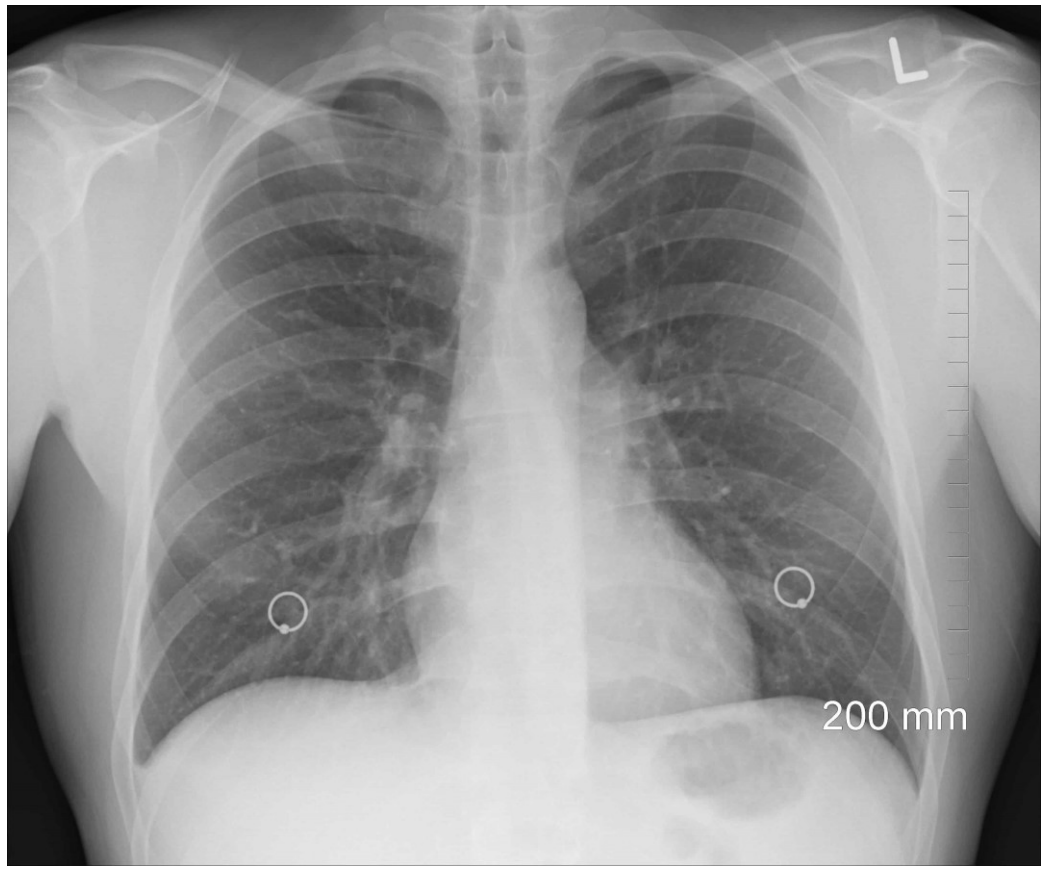
…and require more real examples to meet safety targets…



…resulting in exponential data and compute needs

THE COMPLEXITY

What about problems that require much higher level of expertise



THE COMPLEXITY

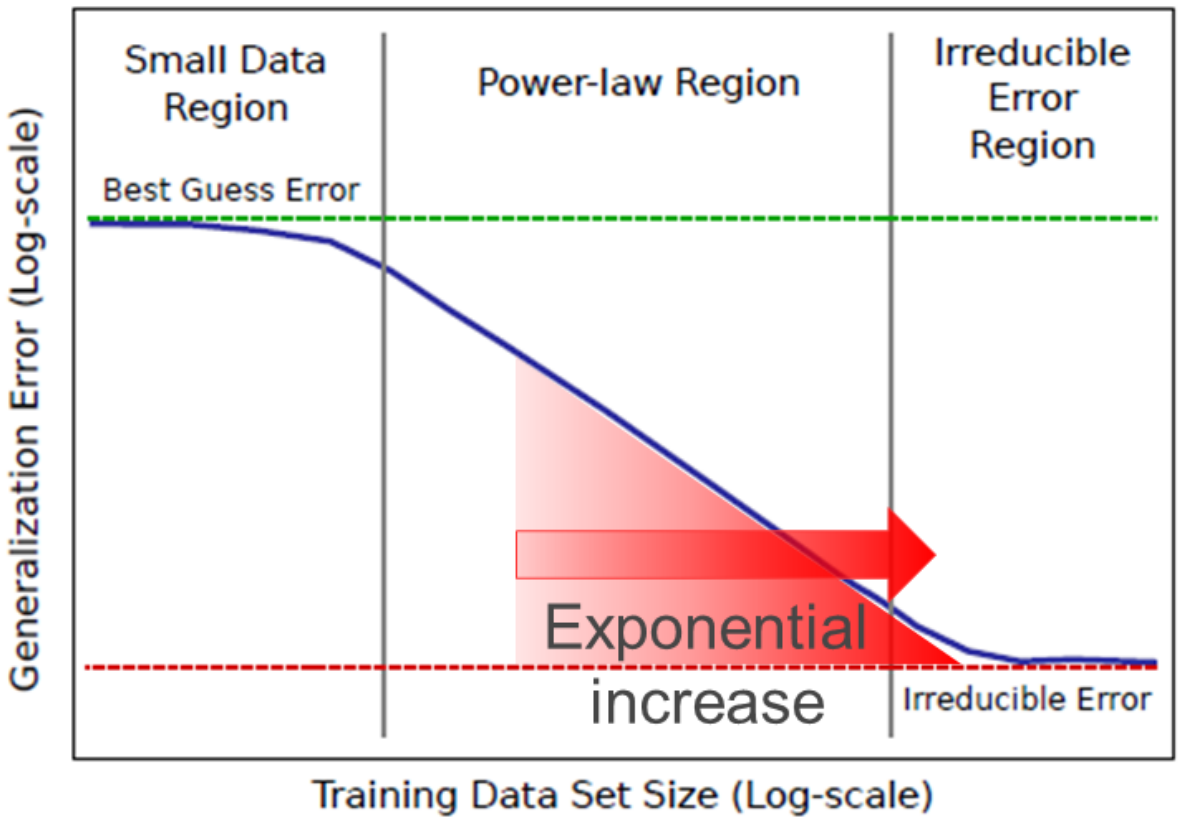
What about problems that require much higher level of expertise

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

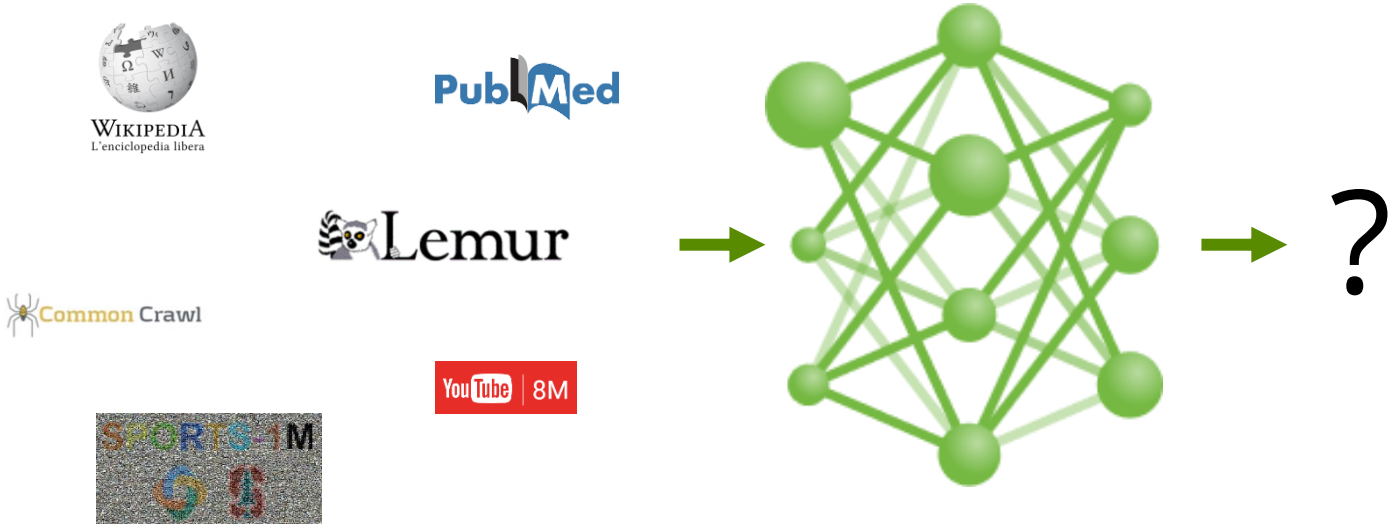


Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., ... & Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically. arXiv preprint arXiv:1712.00409.

SELF AND UN-SUPERVISED LEARNING

SELF AND UN-SUPERVISED LEARNING

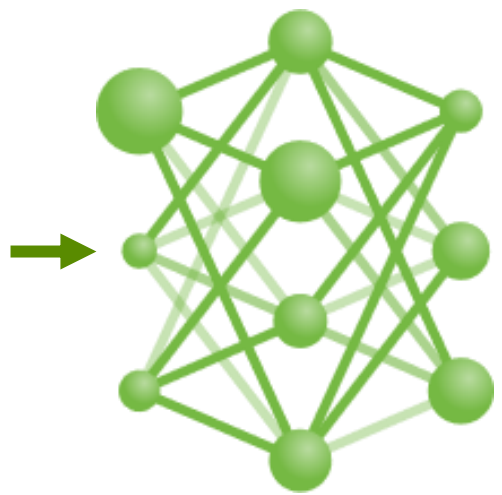
What to do in the absence of labels



SELF-SUPERVISED AND UNSUPERVISED LEARNING

Natural Language Processing – Masked Language Models

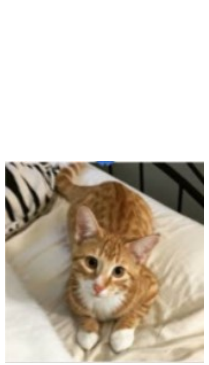
Lorem ipsum [MASK] sit amet, consectetur adipiscing elit.

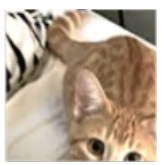


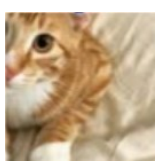
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

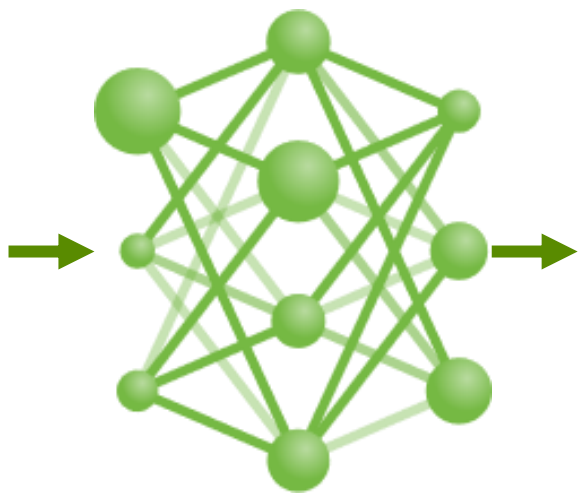
SELF-SUPERVISED AND UNSUPERVISED LEARNING

Computer Vision – Contrastive Learning





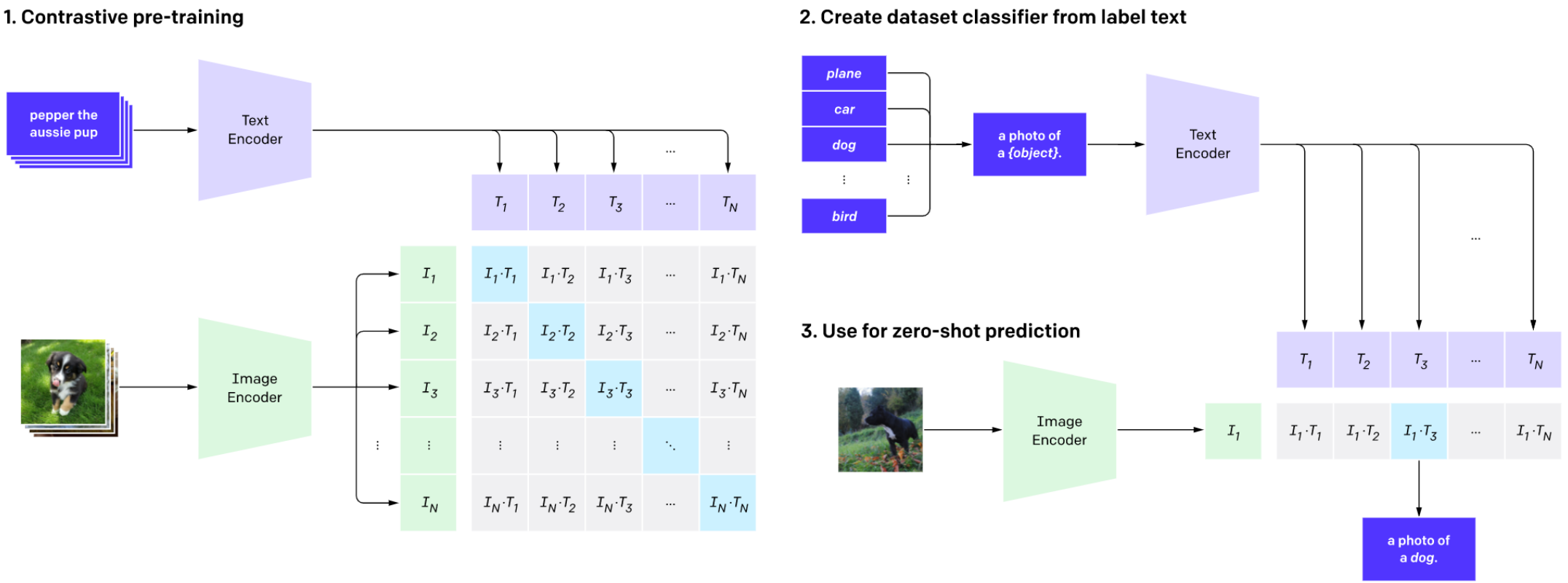




Do the pictures have the same origin: Yes/No

SELF SUPERVISED AND UNSUPERVISED LEARNING

Computer Vision – More complex contrastive learning

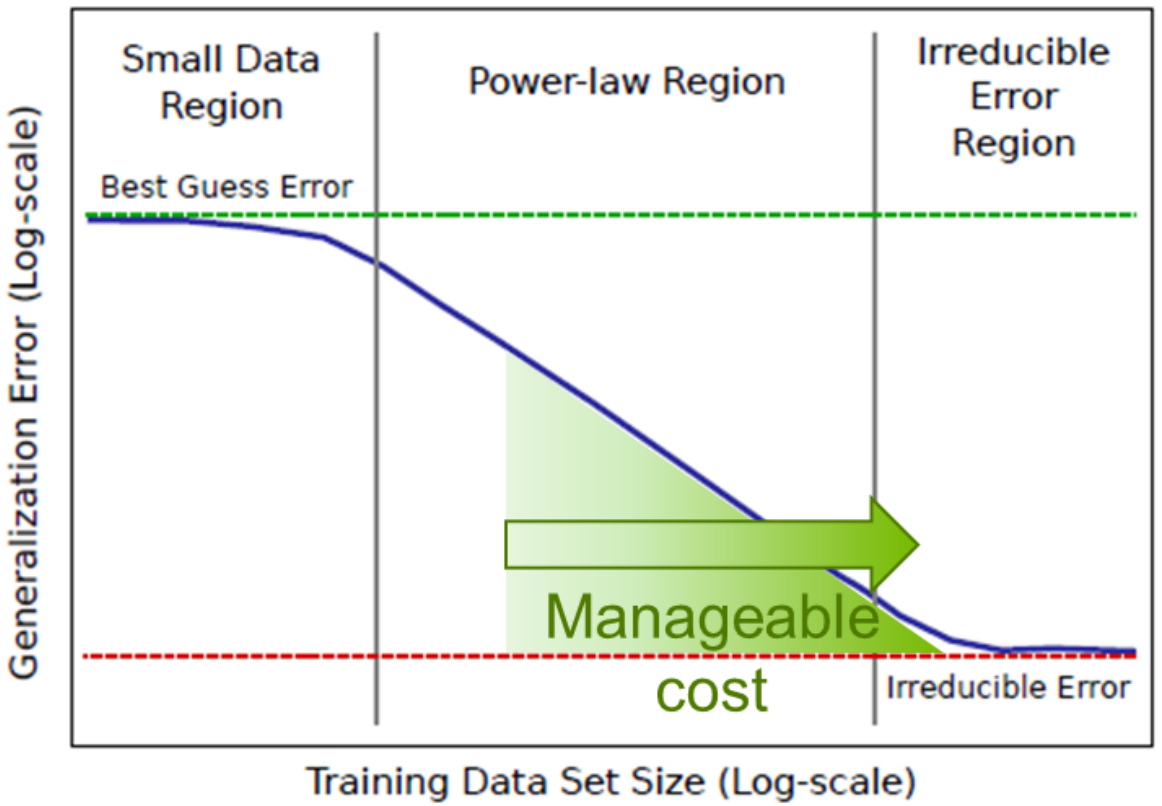


https://openai.com/blog/clip/

NVIDIA

THE COST OF LABELLING

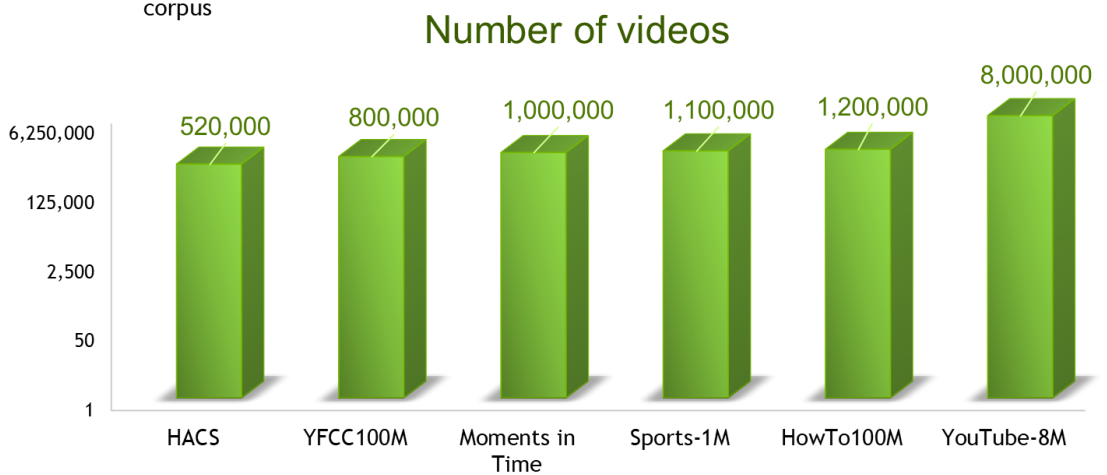
Semi supervised models



Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., ... & Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically. arXiv preprint arXiv:1712.00409.

SELF SUPERVISED AND UNSUPERVISED LEARNING

Abundance of unlabelled data in Text



SCALING LAWS

SCALING LAWS APPLY TO NLP

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

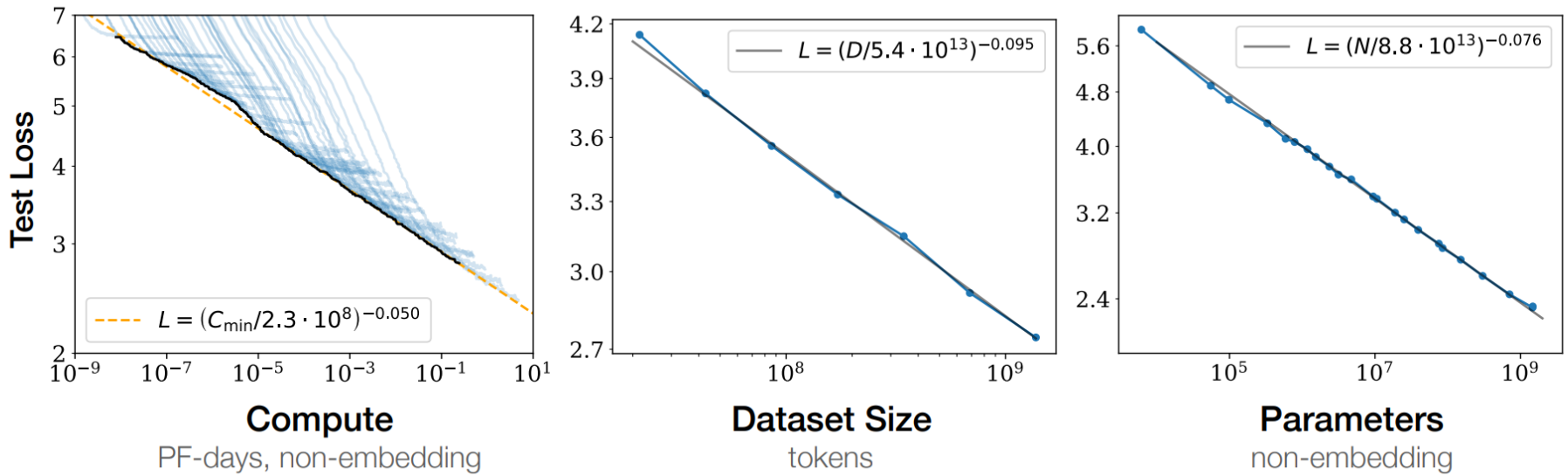


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*.

SCALING LAWS APPLY TO COMPUTER VISION TOO

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

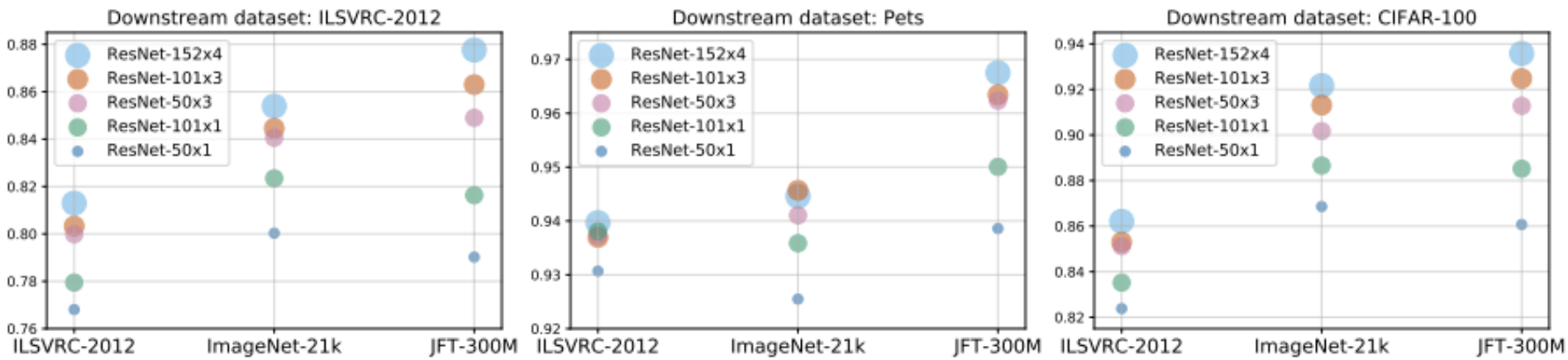


Fig. 5: Effect of upstream data (shown on the x-axis) and model size on downstream performance. Note that exclusively using more data or larger models may hurt performance; instead, both need to be increased in tandem.

Big Transfer (BiT): General Visual Representation Learning

IT IS MORE THAN JUST ACCURACY

IMPORTANCE OF DATASET SIZE

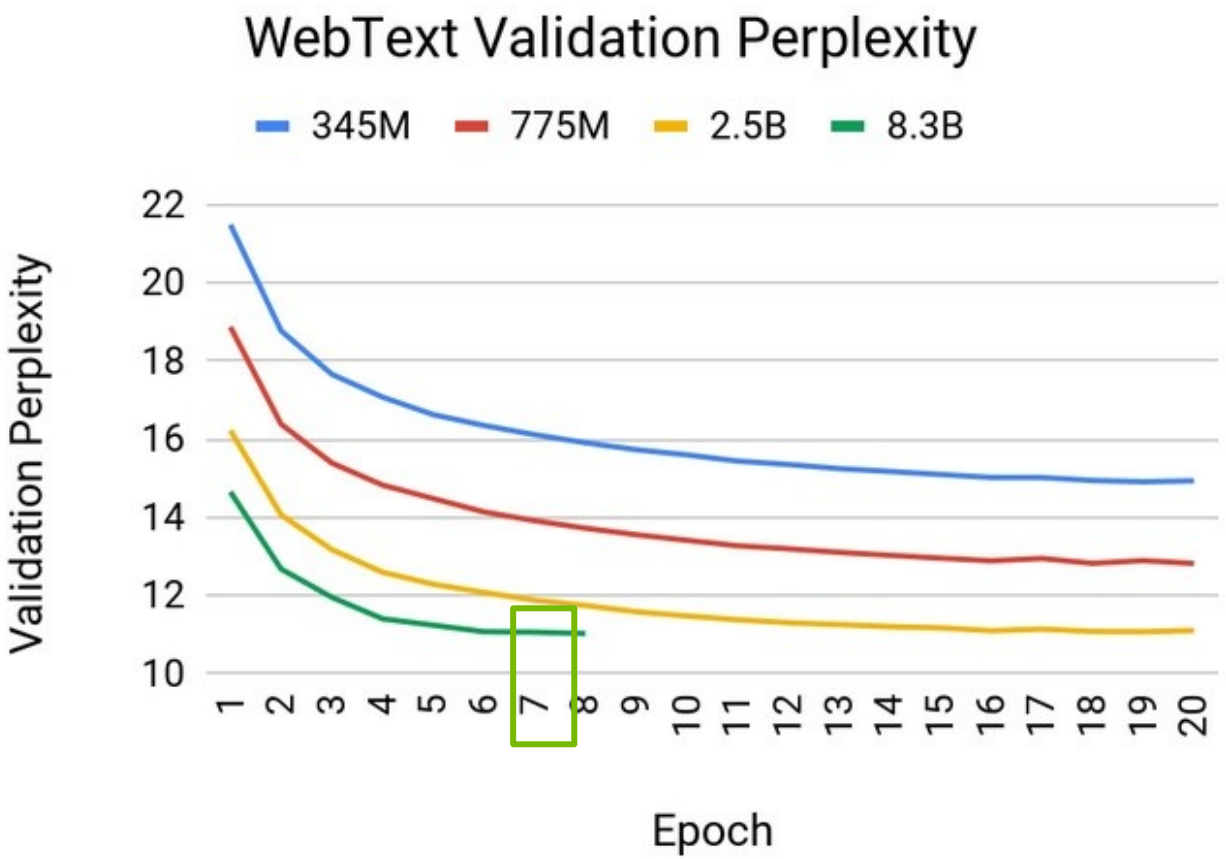
Dataset size more important than neural network design

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

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*.

EVEN MORE IMPORTANTLY

Large Neural Networks use data more efficiently



Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. arXiv preprint arXiv:1909.08053
Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Agarwal, S. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*..

PERSPECTIVE

WHAT DO I MEAN BY BIG

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

Not a linear scale

3.14*10^8 PFLOPS

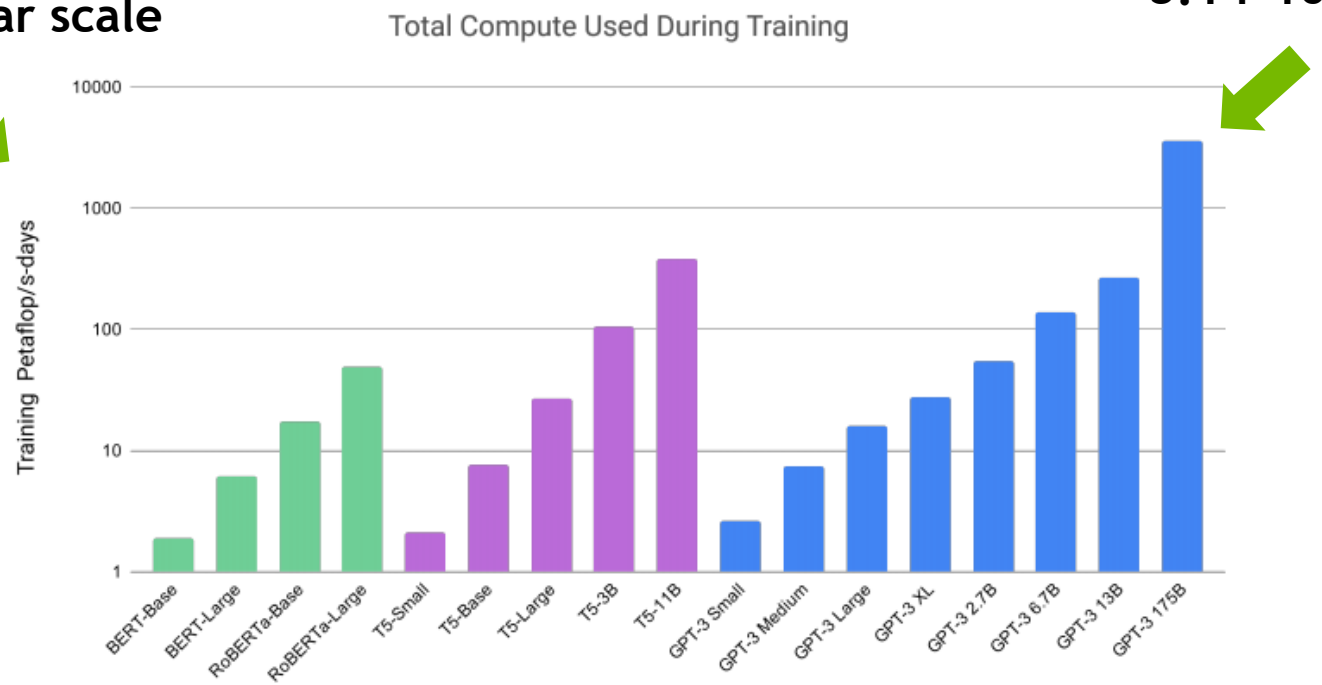


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 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

~31 years on a single A100

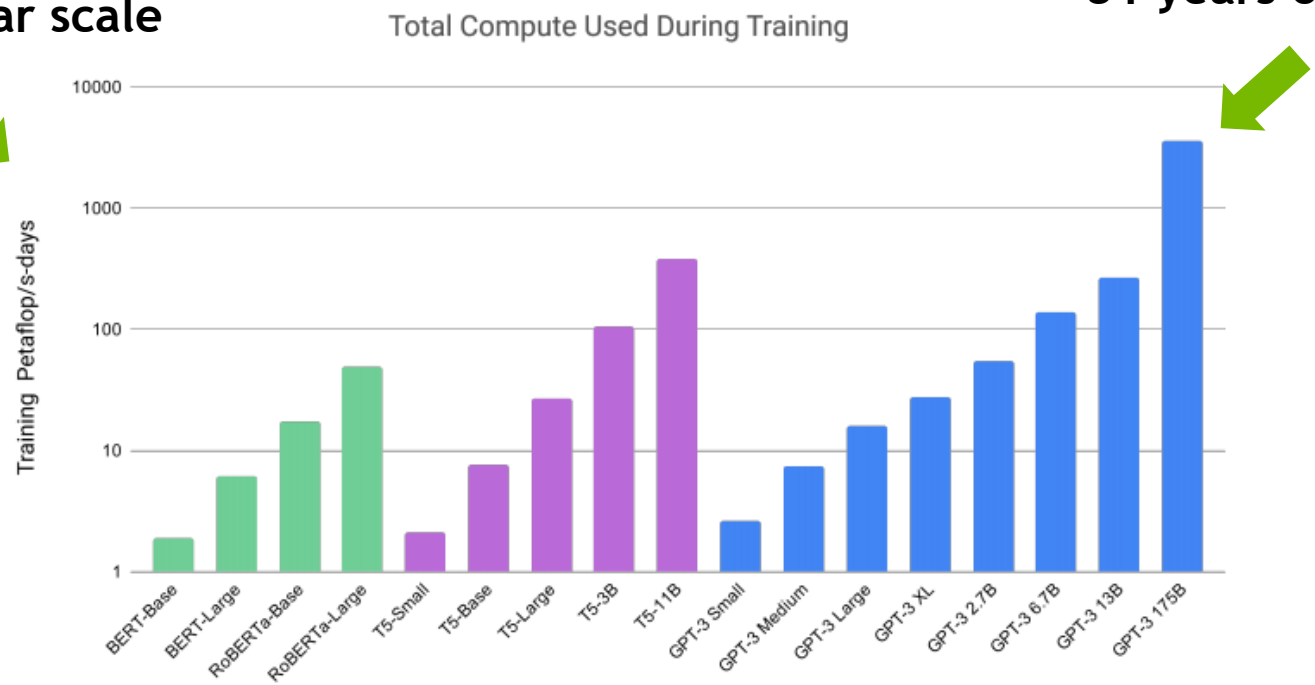


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 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=8*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,G
    days_needed=calculate_days_needed(T,gpt3_params,n,x)
    print(" ----------------------------------------------------------------------------
    print(" language model :{} with {} number of parameters , it will need {} days to compu
```

```
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
```

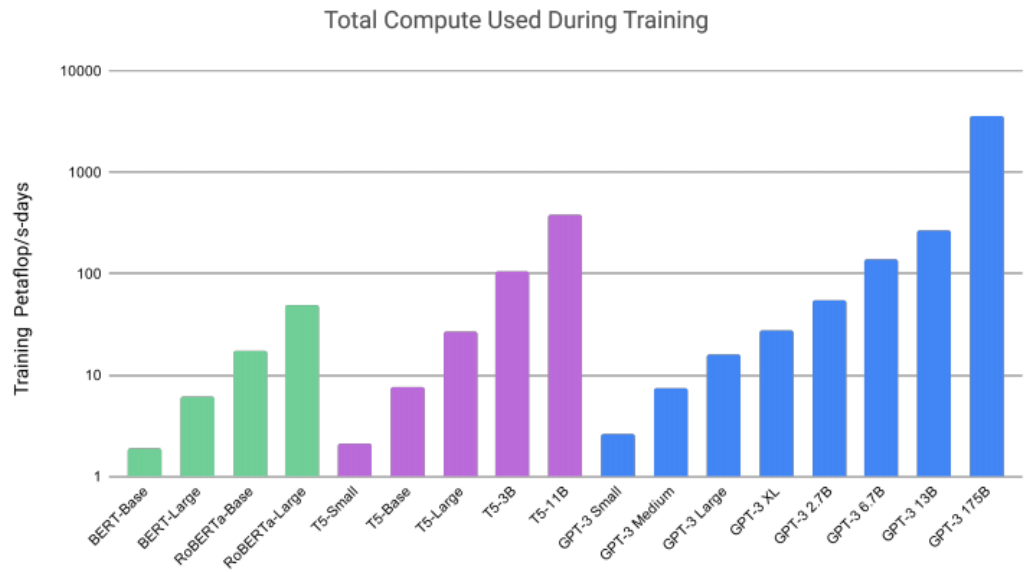


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 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

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

1 billion parameters

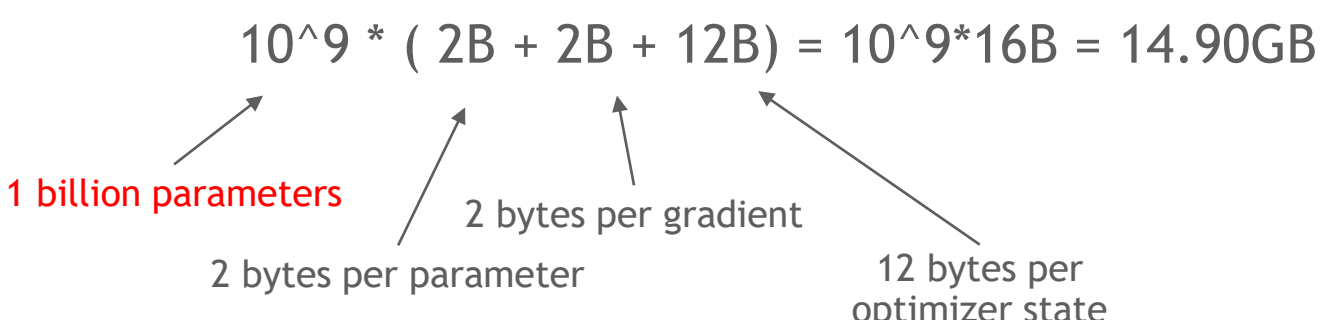
2 bytes per parameter

2 bytes per gradient

12 bytes per optimizer state

GOING BIGGER

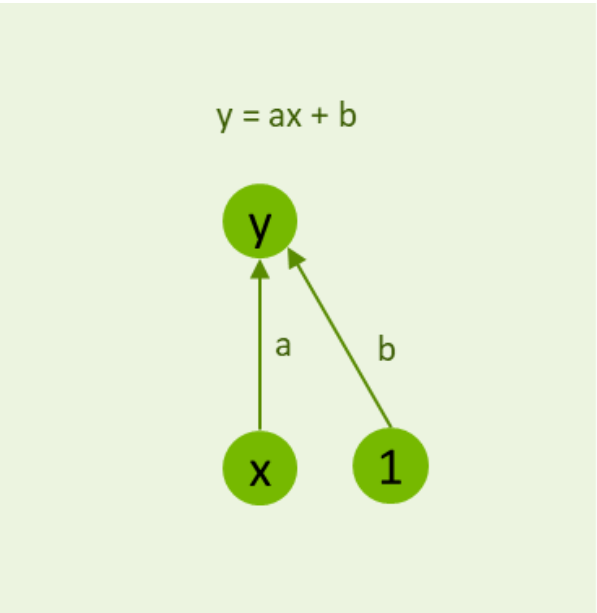
- What about activations?
- What about **2 or 3** billion parameter models?

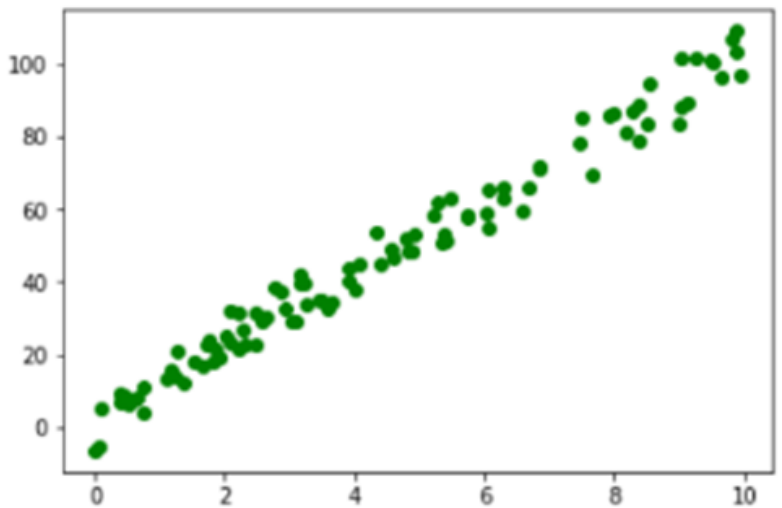


WHAT MAKES THEM POSSIBLE

TRIVIAL EXAMPLE

Lets start with the simplest of problems – linear neuron

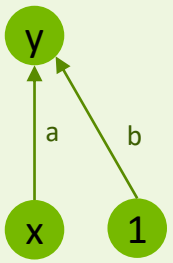




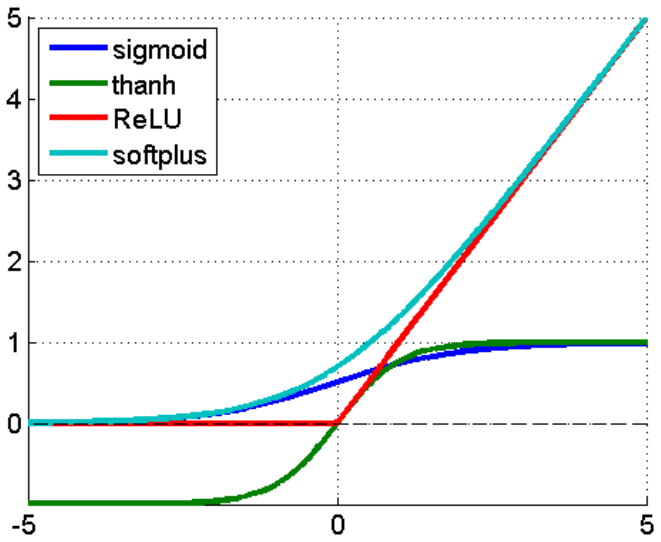
MODERN NEURAL NETWORKS

How do they differ from our trivial example?

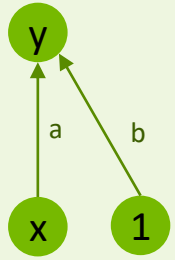
y = ax + b



Nonlinearity

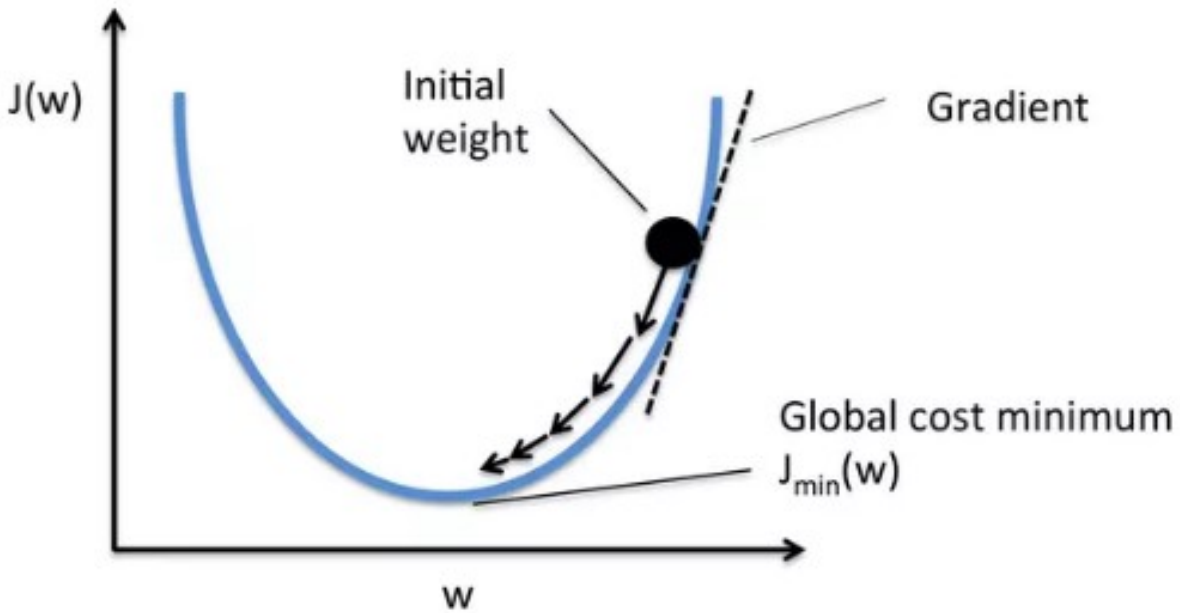


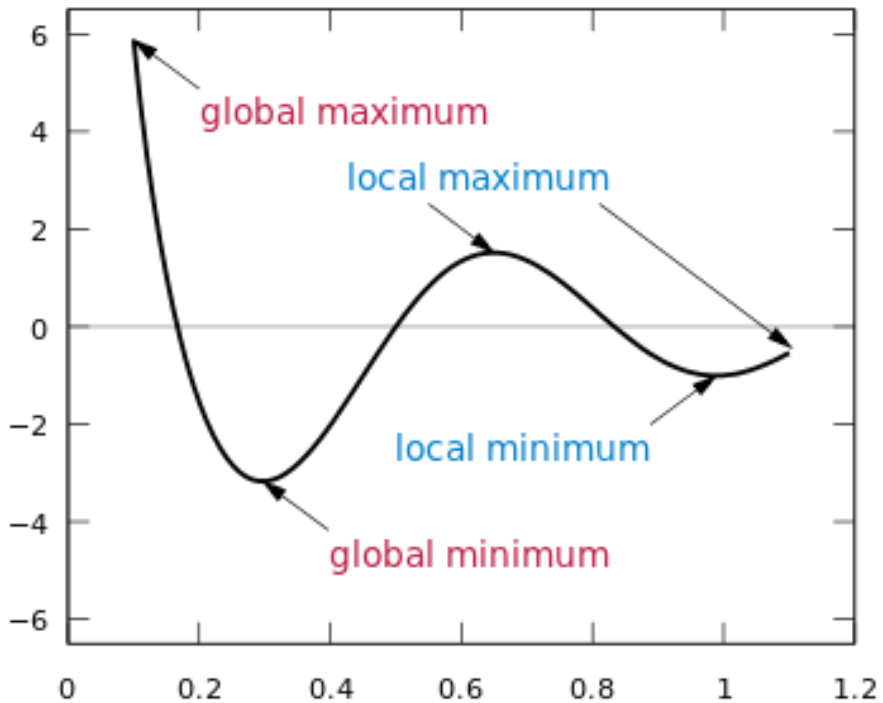
y = f(ax + b)



NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

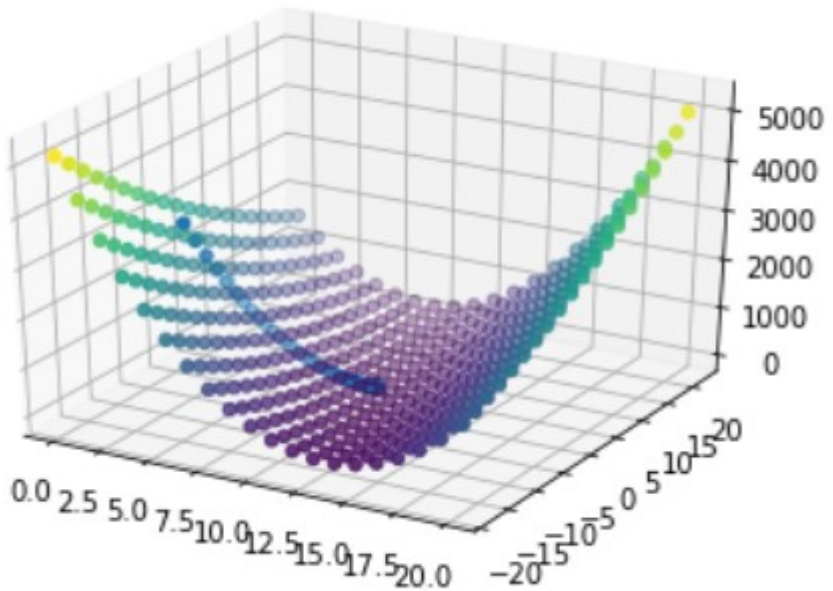




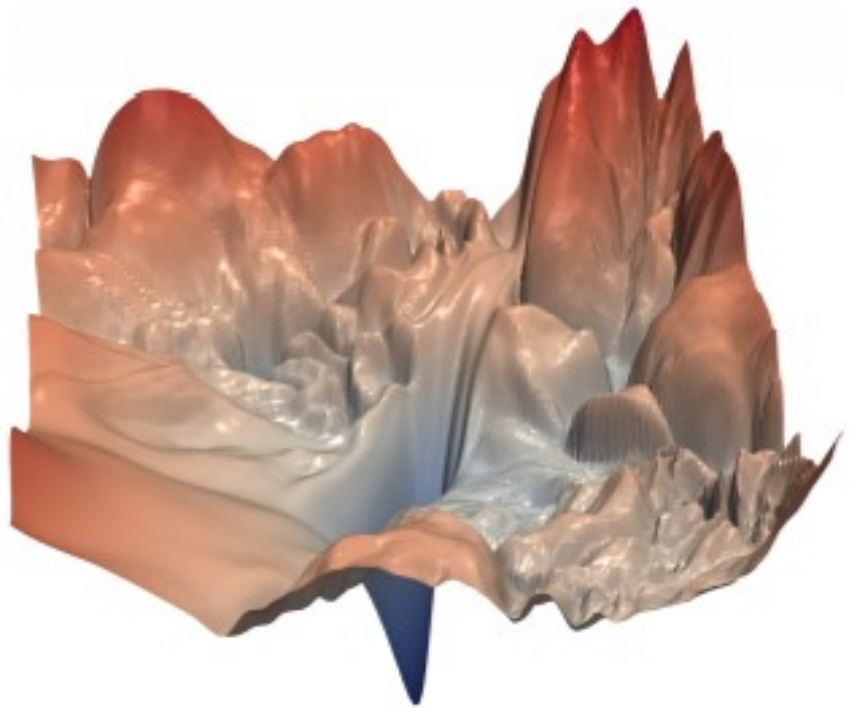
NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

Linear Neuron cost function



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



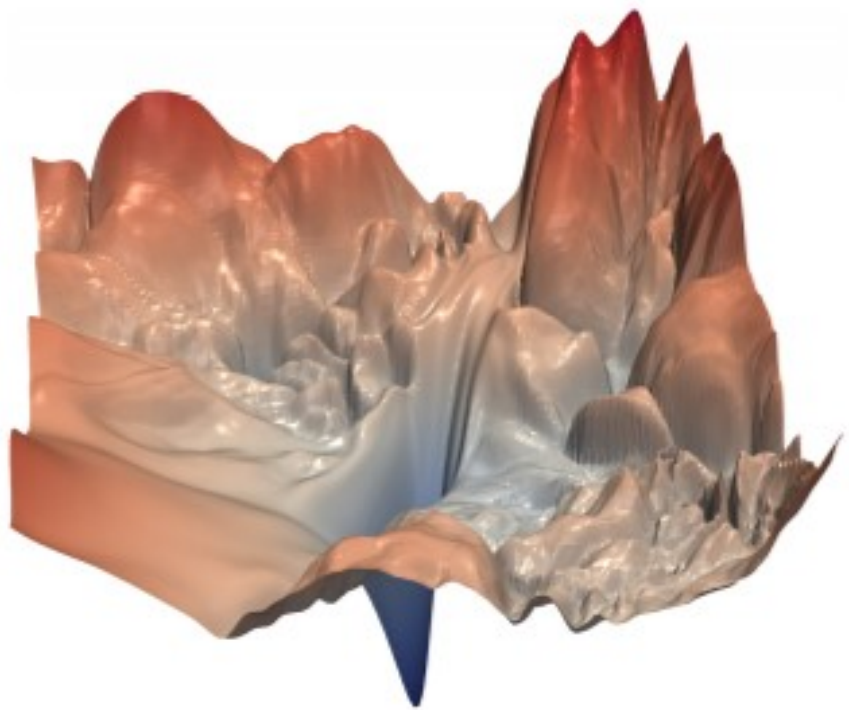
Li, H., Xu, Z., Taylor, G., & Goldstein, T. (2017). Visualizing the Loss Landscape of Neural Nets. *arXiv preprint arXiv:1712.09913*.

NON CONVEX COST FUNCTIONS

Those differences make the optimization problem much more difficult

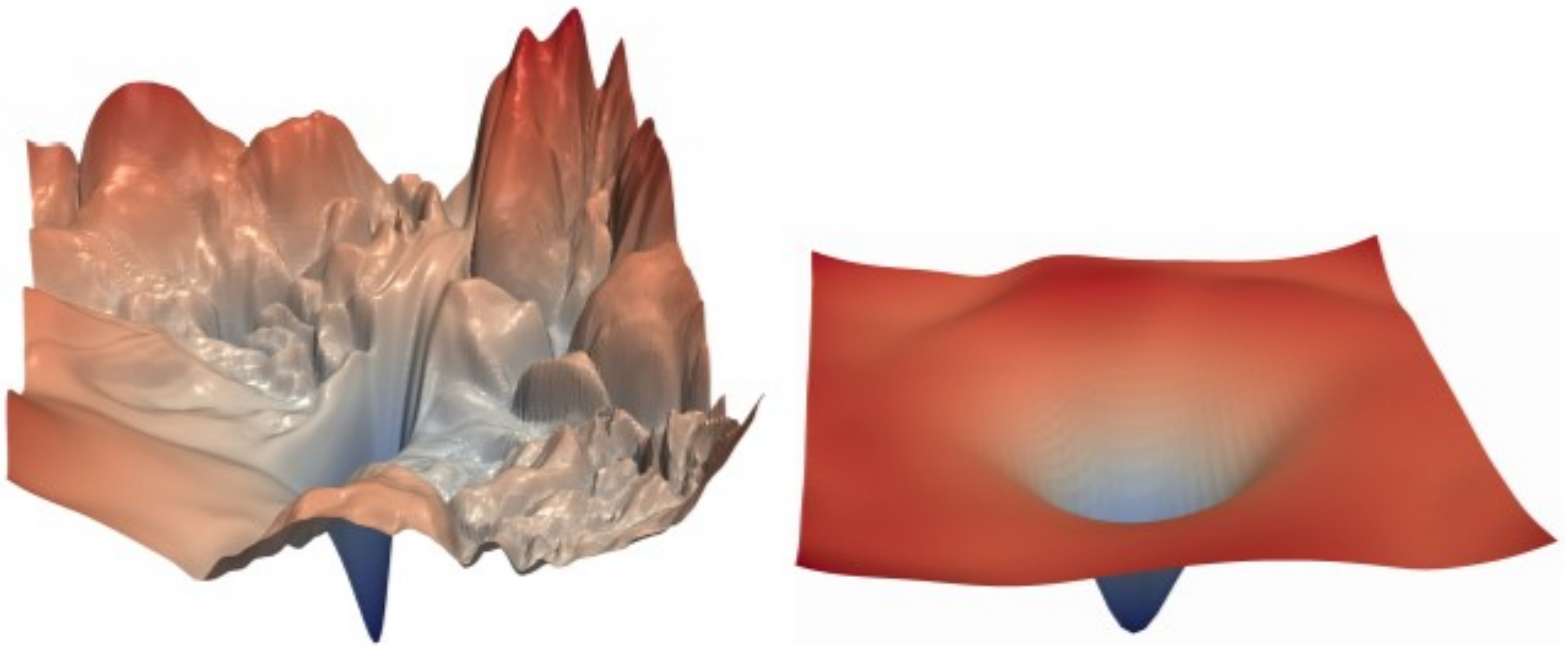
Why do we succeed in finding good local minima?

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



NON CONVEX COST FUNCTIONS

Recent advances such as Residual Connections simplify the optimization problem

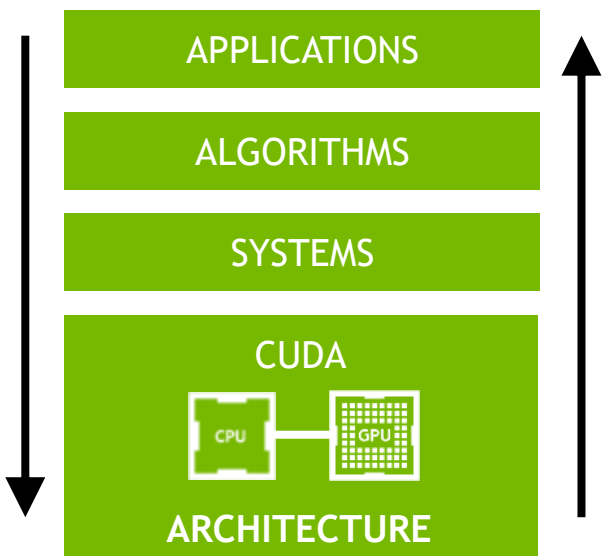


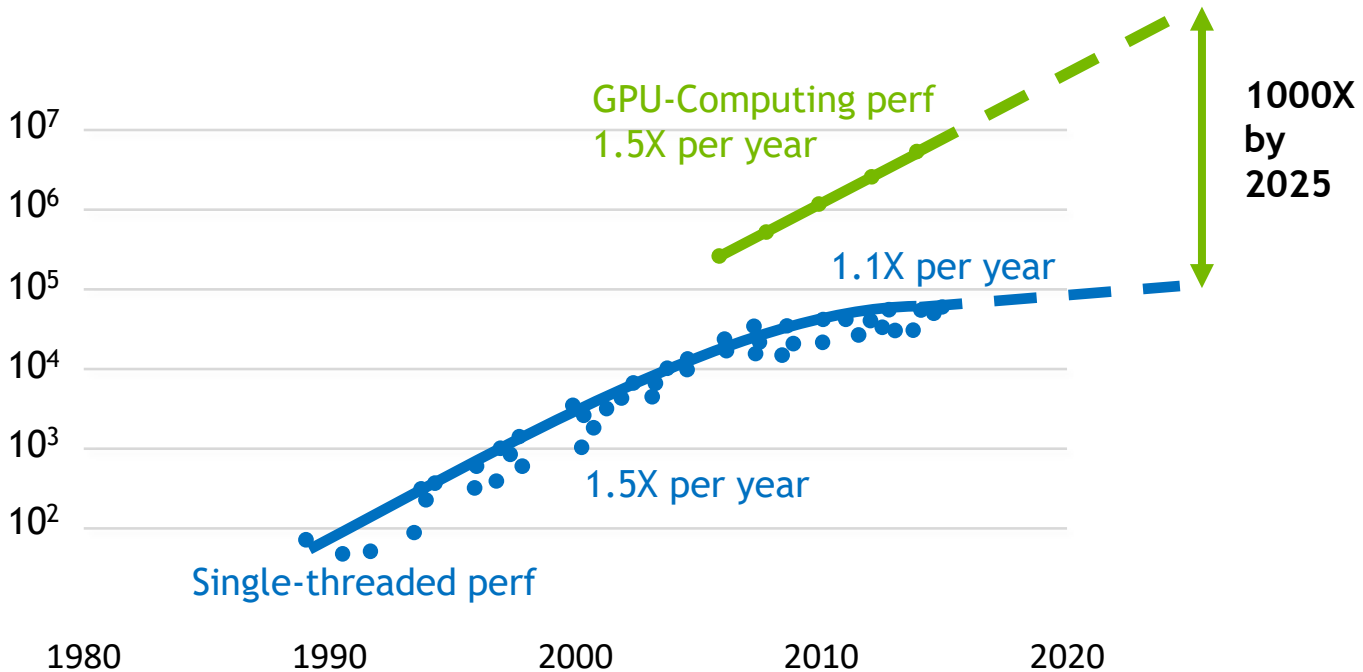
(a) without skip connections

(b) with skip connections

Li, H., Xu, Z., Taylor, G., & Goldstein, T. (2017). Visualizing the Loss Landscape of Neural Nets. *arXiv preprint arXiv:1712.09913*.

RISE OF GPU COMPUTING





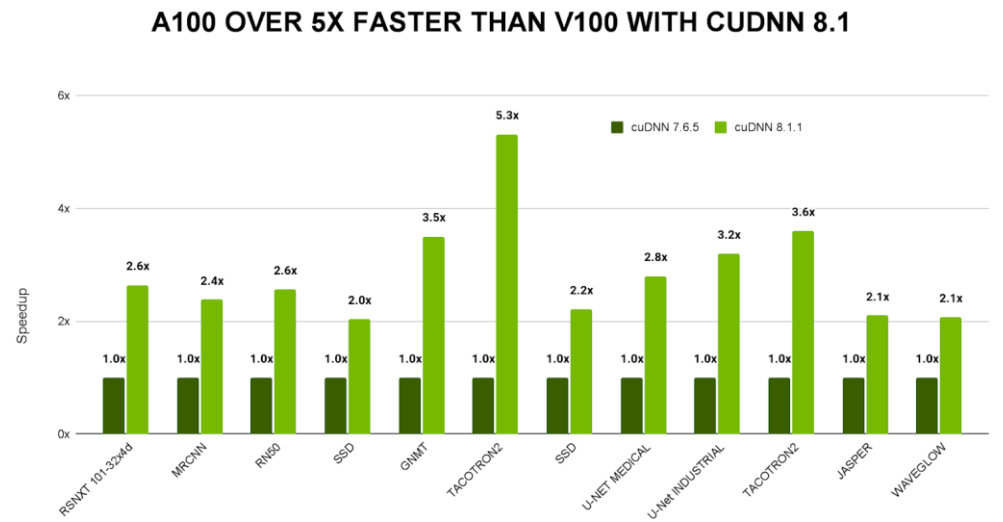
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

NVIDIA

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.



Caffe · Caffe2 · Chainer · Microsoft Cognitive Toolkit · MATLAB · mxnet

PaddlePaddle · PyTorch · TensorFlow · torch · Wolfram Language

IMPORTANT CUDNN FLAGS

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

Important aspects to consider:

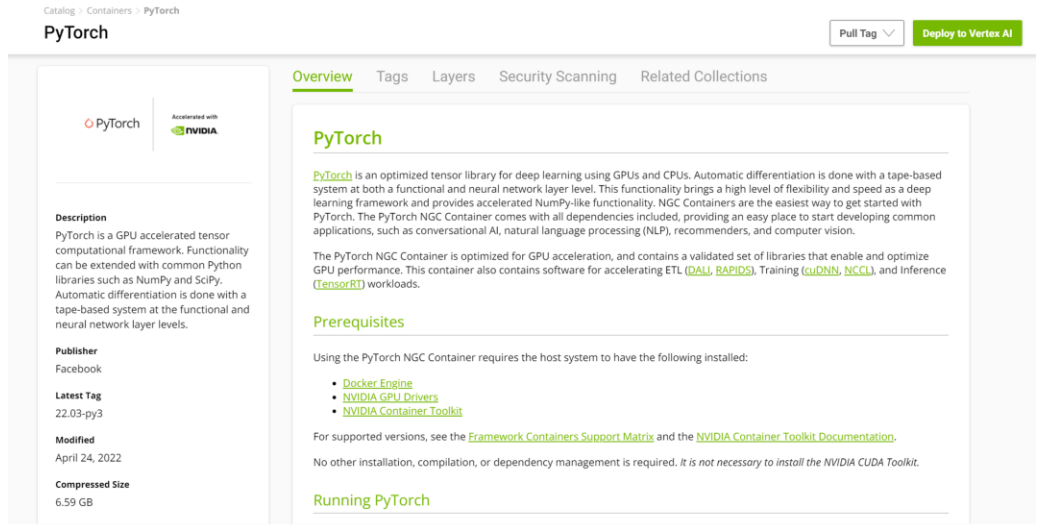
- In NGC containers, the usage of TensorFloat-32 is enabled by default in order to accelerate FP32 calculations using tensor cores on Ampere or newer GPUs.
- Certain classes of CUDA functions are a potential source of non-determinism, such as atomicAdd, where the order of parallel additions to the same 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!

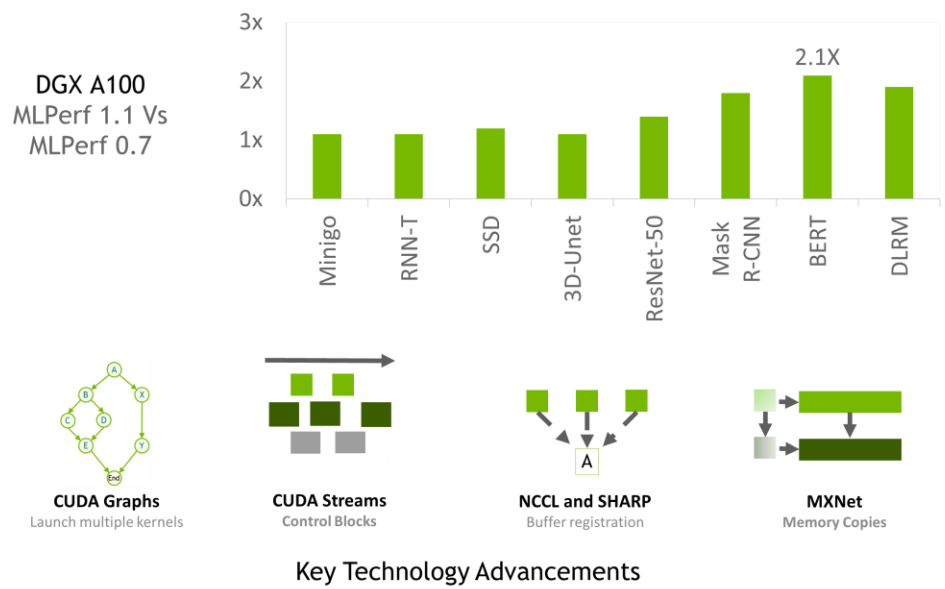
```
# get the cuDNN version
torch.backends.cudnn.version()

# 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





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)
```

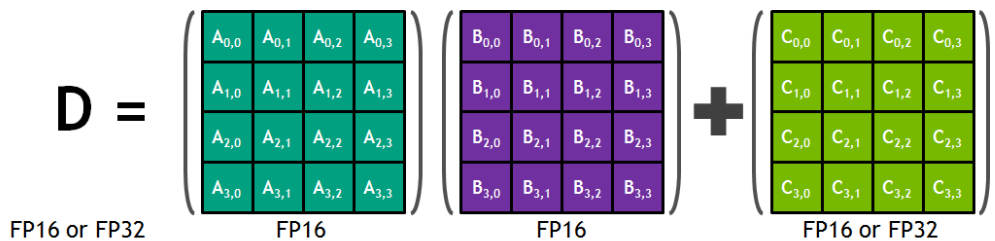
UTILIZING A SINGLE GPU EFFICIENTLY – MAXIMIZING OCCUPANCY & UTILIZATION

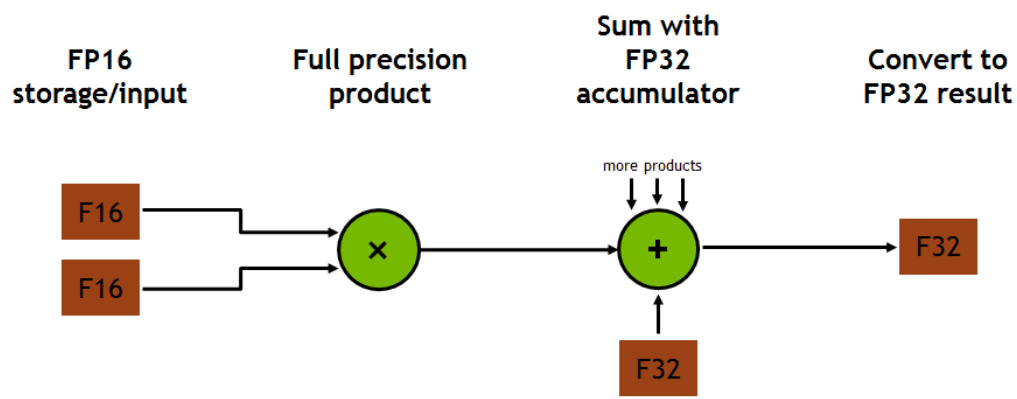
SIZING RECOMMENDATIONS

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.

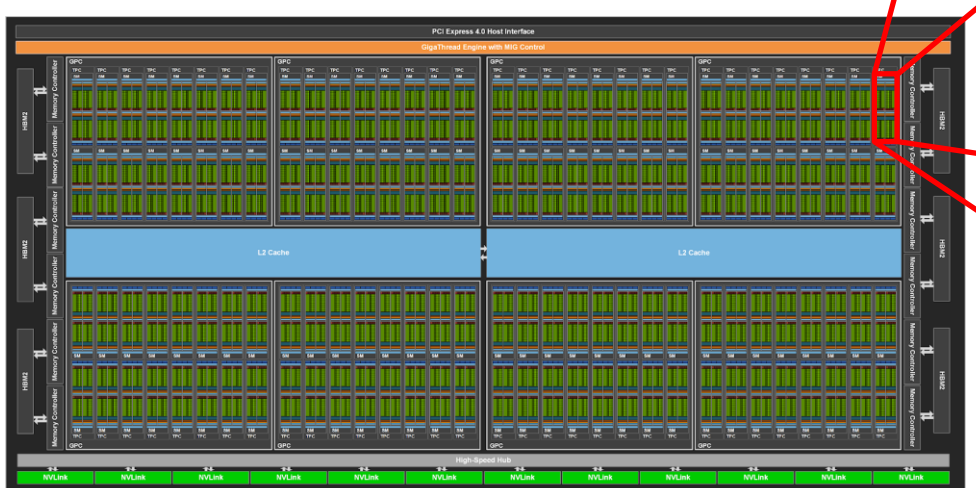




TENSOR CORES

"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



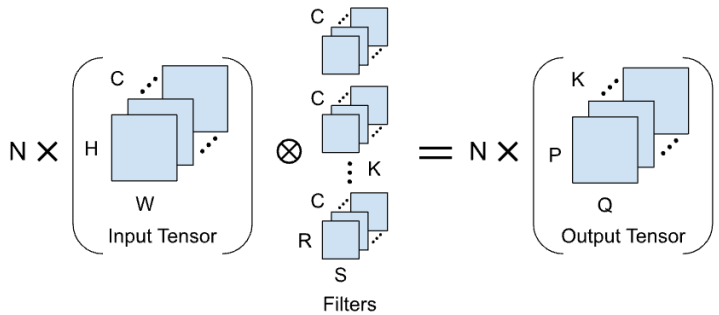
Actually, there are researchers trying to exploit tensor cores for reductions and other ops:
https://arxiv.org/abs/1811.09736

NVIDIA

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

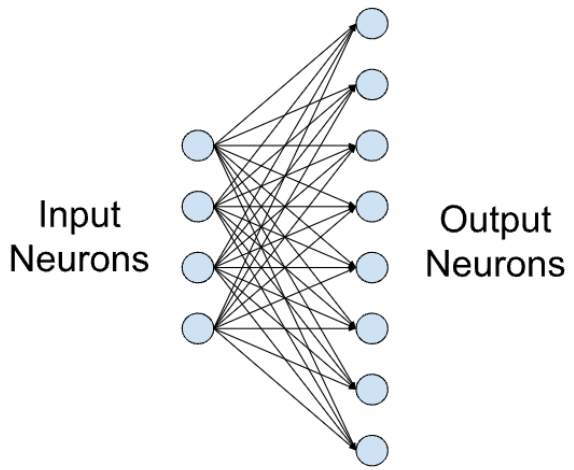
CHECKLIST FOR FULLY CONNECTED LAYERS

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).



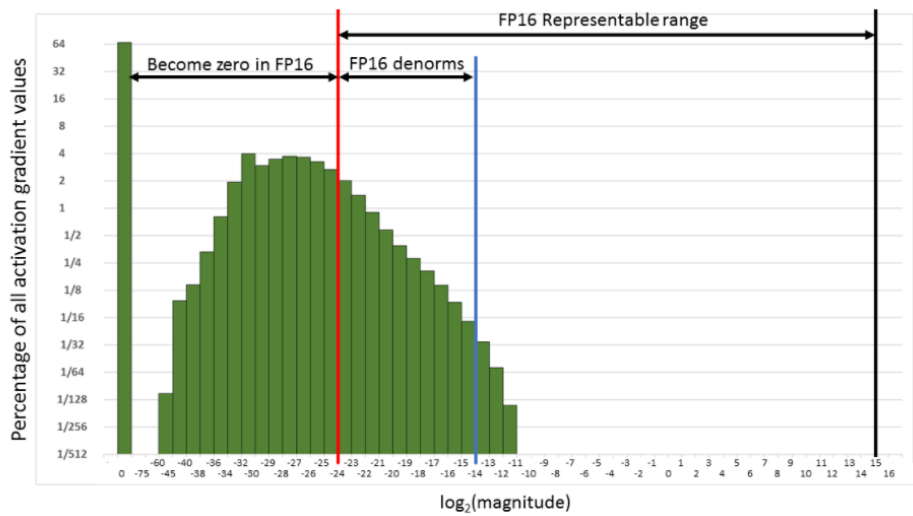
UTILIZING A SINGLE GPU EFFICIENTLY – MAXIMIZING OCCUPANCY & UTILIZATION

TENSOR CORE UTILIZATION

MIXED PRECISION TRAINING - THE IDEA

Example: FP32 training of Multibox SSD network

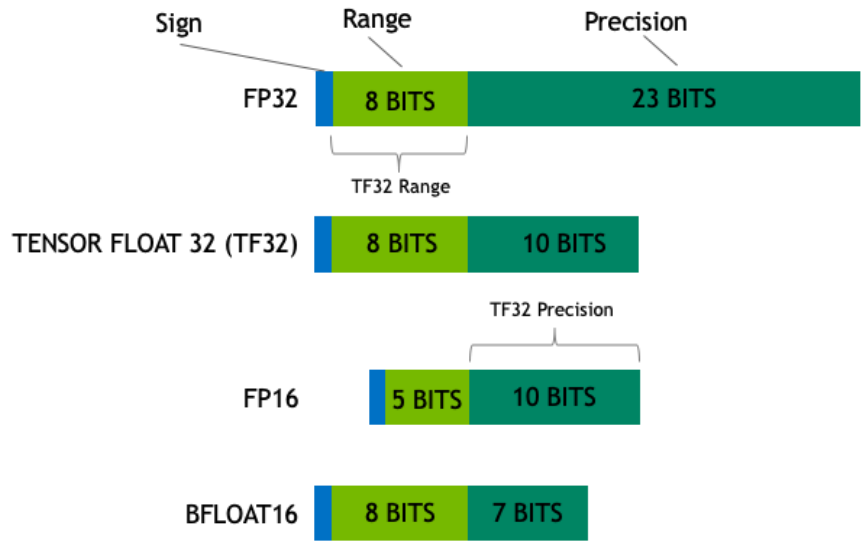
- Histogram shows activation gradient magnitudes throughout FP32 training; both axes are logarithmic.
- Observations:
  - Dynamic range of FP16 would be sufficient to cover the entire histogram. ☺
  - Without “shifting” the histogram, half of the activations would be casted to 0, however. ☹
- Idea: “shifting” = 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

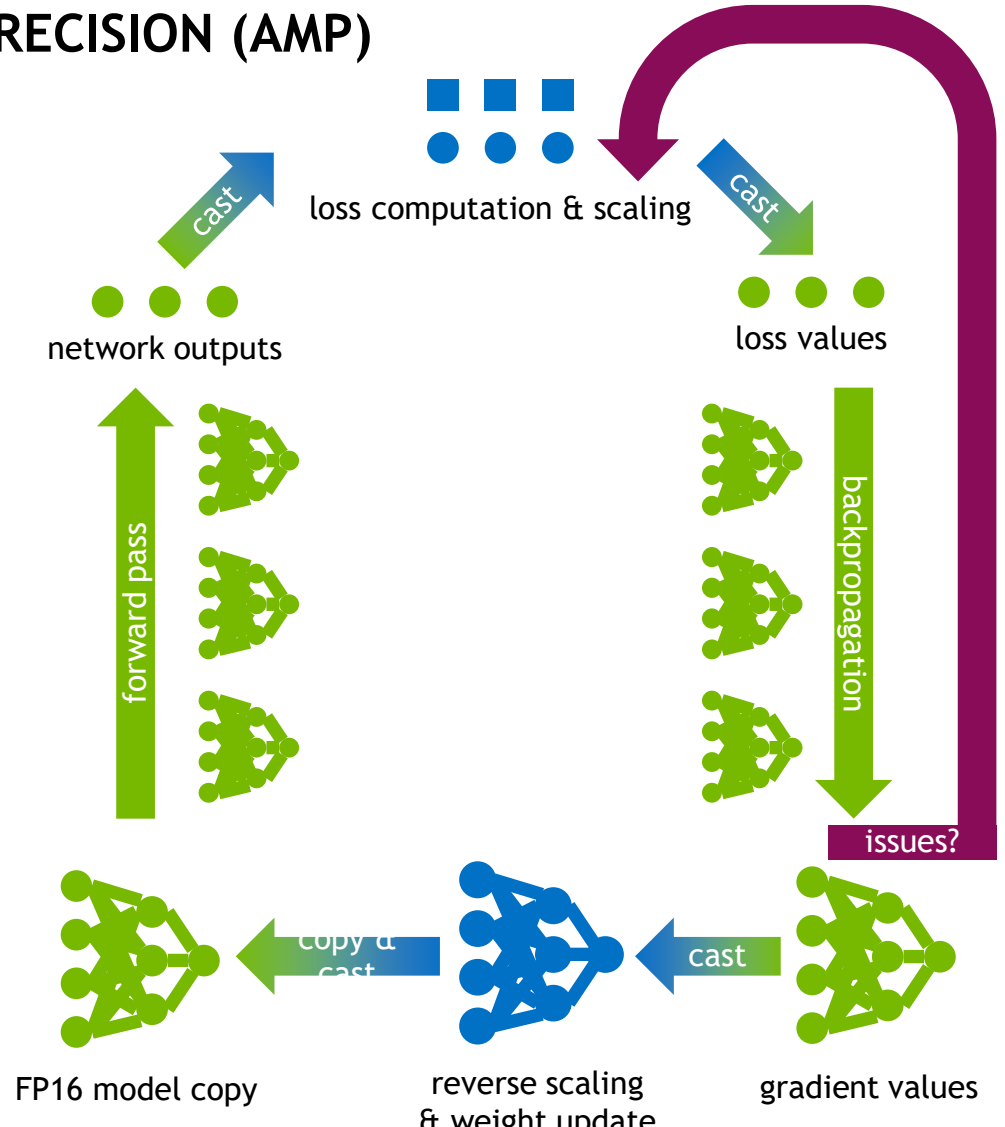
- Mixed precision training is mostly about the dynamic range and less about the precision:
  - exponent → dynamic range
  - significand field → precision
- TF32 is a great compromise between FP32 (same range) and FP16 (same precision)
- TF32 is automatically enabled in NGC containers
- No code change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

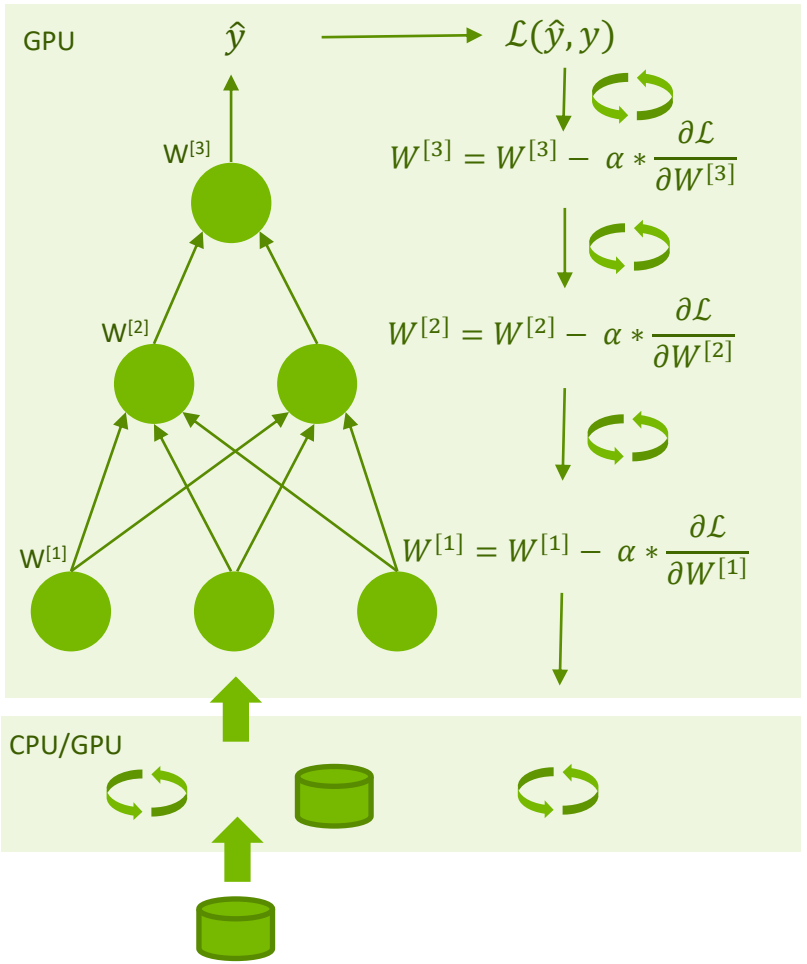
- Backward passes under autocast are not recommended.
- Backward ops run in the same dtype autocast chose for corresponding forward ops.
- scaler.step() first unscales the gradients of the optimizer's assigned params.
- If these gradients contain infs or NaNs, optimizer.step() is skipped.

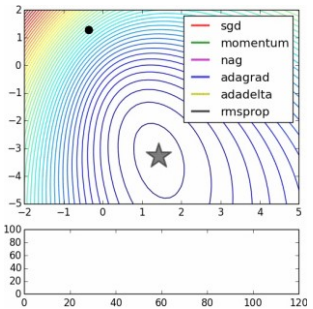
```
# initialize gradient scaler
scaler = GradScaler()

# 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

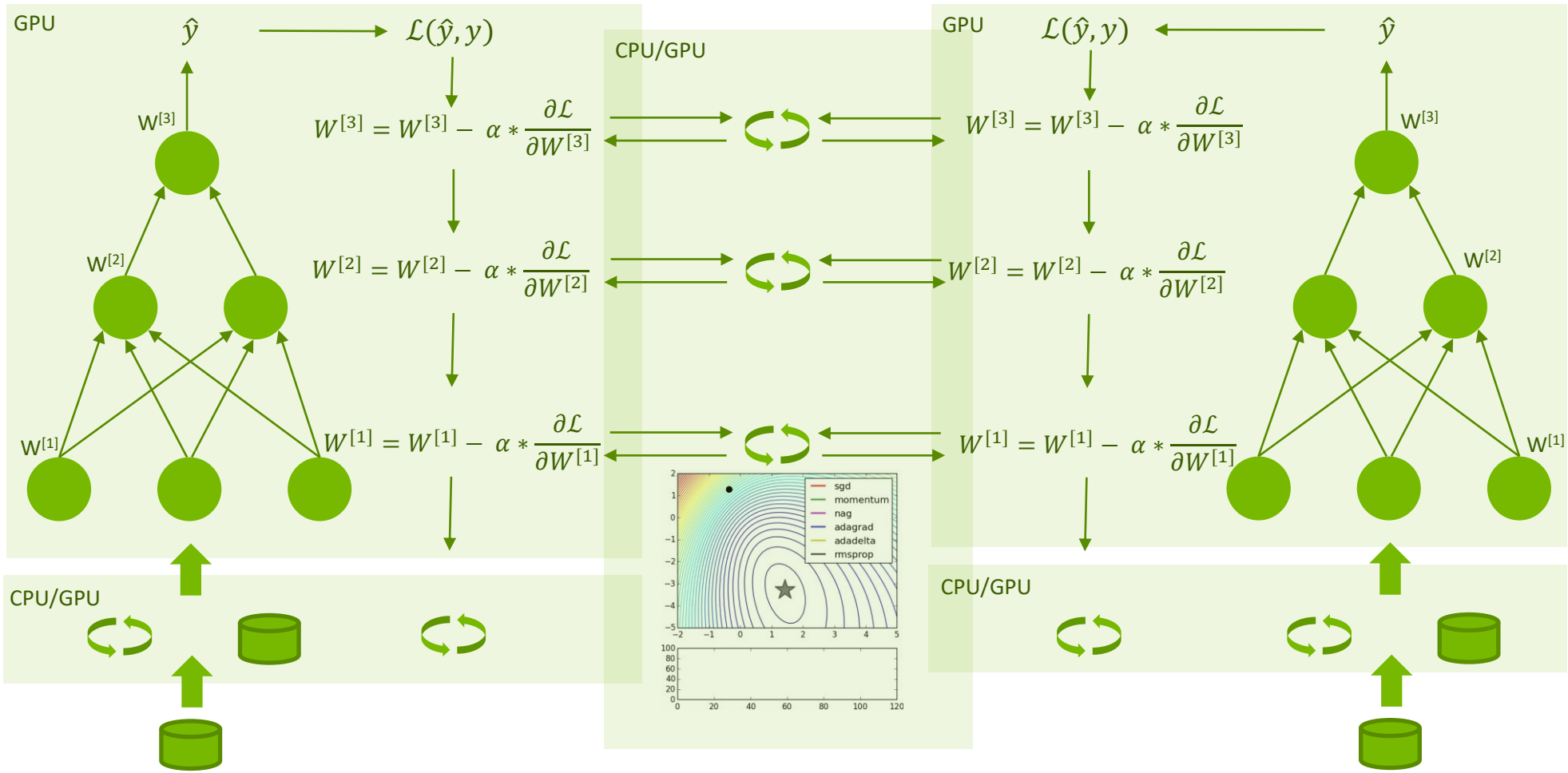




1. Read the data
2. Transport the data
3. Pre-process the data
4. Queue the data
5. Transport the data
6. Calculate activations for layer one*
7. Calculate activations for layer two
8. Calculate the output
9. Calculate the loss
10. Backpropagate through layer three
11. Backpropagate through layer two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

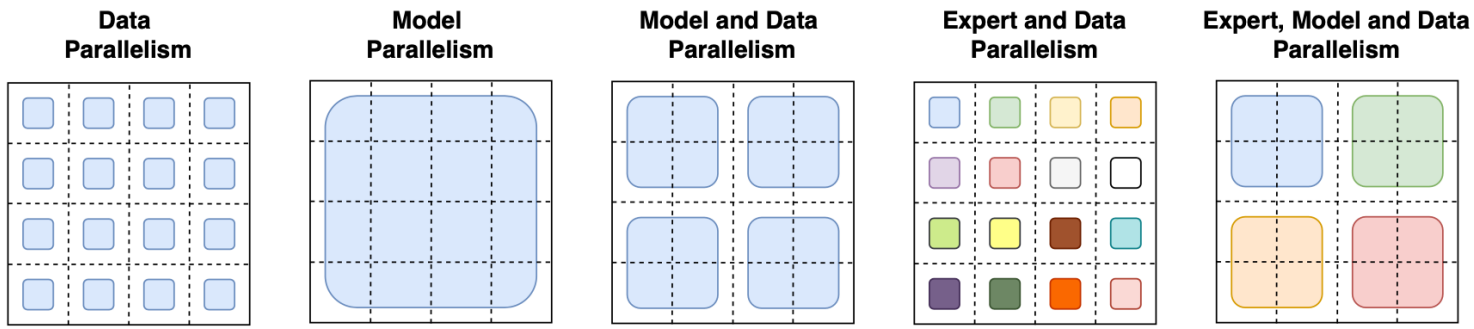
TRAINING A NEURAL NETWORK

Multiple GPUs

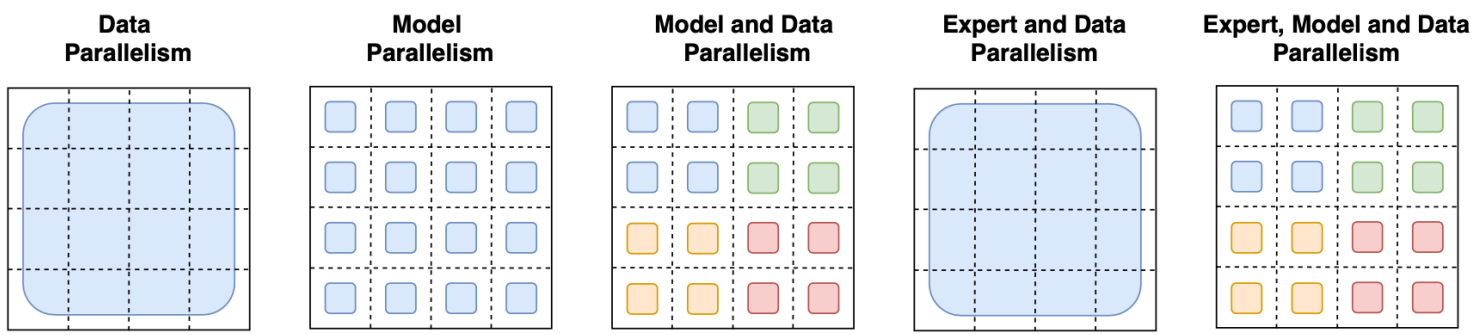


DATA AND MODEL PARALLELISM STRATEGIES

How the *model weights* are split over cores



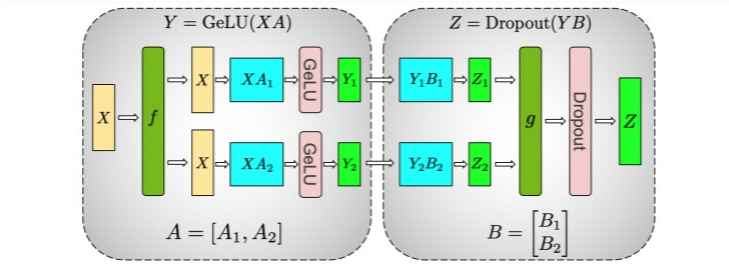
How the *data* is split over cores



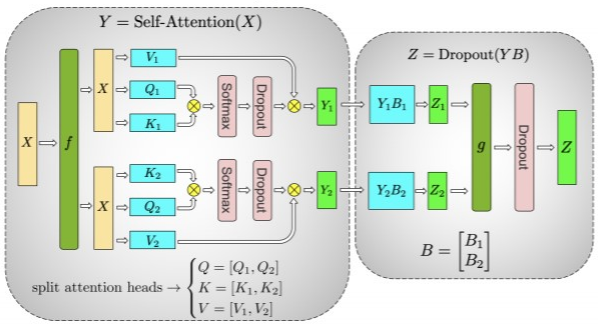
Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity, William Fedus and Barret Zoph and Noam Shazeer, 2021, https://arxiv.org/abs/2101.03961

MEGATRON

Model Parallel Transformer



(a) MLP



(b) Self-Attention

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.

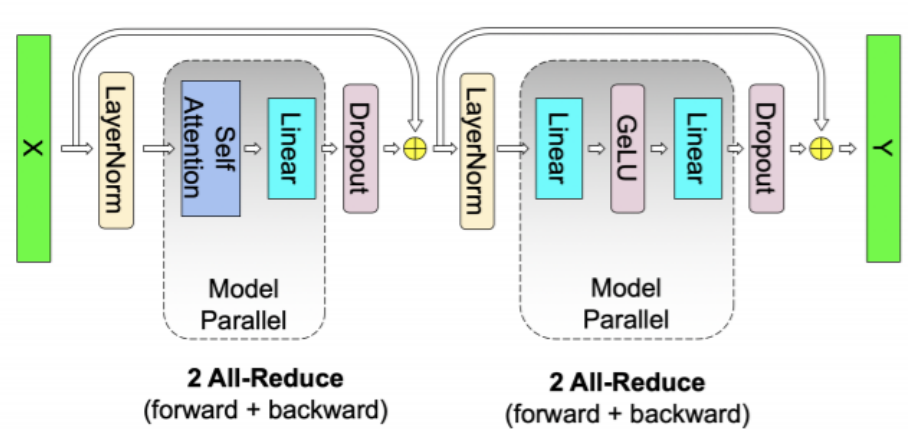
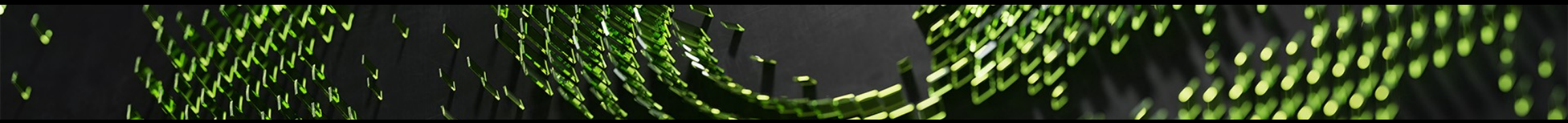


Figure 4. Communication operations in a transformer layer. There are 4 total communication operations in the forward and backward pass of a single model parallel transformer layer.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. arXiv preprint arXiv:1909.08053.



ITALIAN PERSPECTIVE

COLLABORATION FRAMEWORK

The Italian centre builds on the collaboration of three entities

consorzio interuniversitario nazionale per l'informatica

- Consortium of 41 universities in the fields of Computer Engineering, Computer science, and Information technologies. More that 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.

CINECA

- 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 AImageLab in Modena.

January 15, 2020 by FRÉDÉRIC PARIENTÉ

NVIDIA

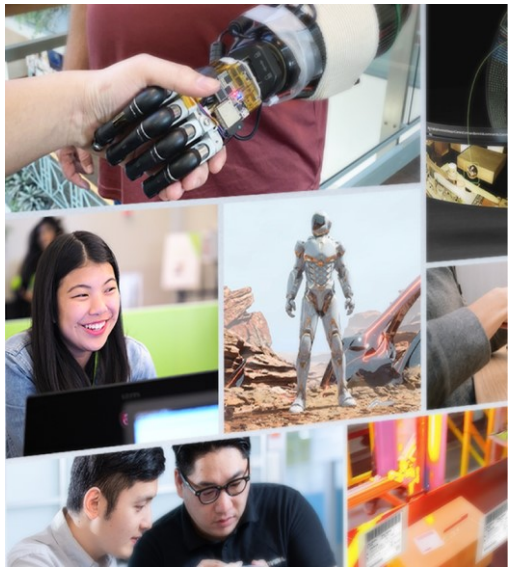
MAIN AREAS OF CONTRIBUTION

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**

NVIDIA Developer Ecosystem

Driving adoption with developers



NVIDIA Developer Program

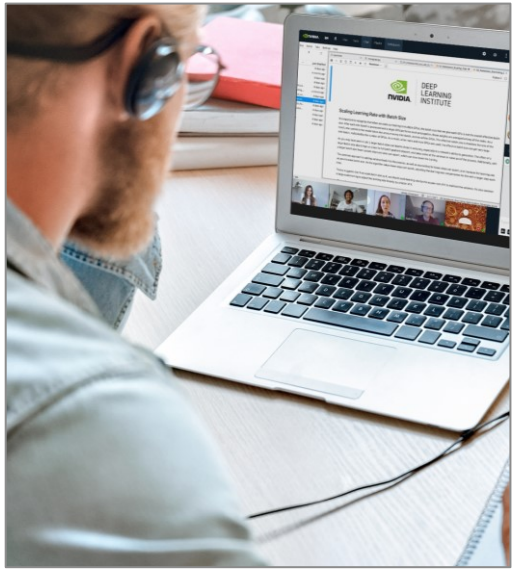
3.5 Million Developers

- 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*

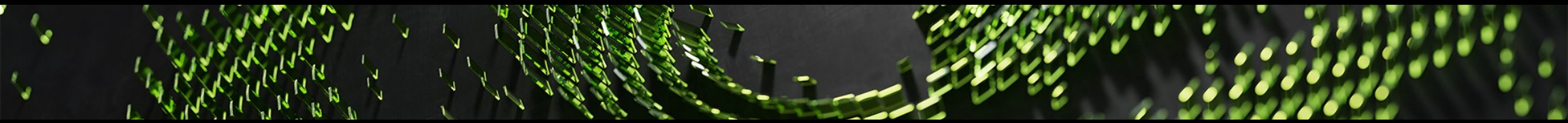
Training

- 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!