Deeper with Transformers

Nicolas.Hernandez@univ-nantes.fr

ATAL - 15 novembre 2024

Nantes Université

Laboratoire des Sciences du Numérique de Nantes

TALN

Overview

Foundation models BERT, GPT-2, T5 (TODO)

Tokenizers

Generation strategies

Scaling laws and compute-optimal models

Calculating the model size (number of parameters)

BERT, Bidirectional Encoder Representations from Transformers

BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
(Devlin et al. @Google, 2018)

Bidirectional Transformer = Transformer encoder

First Transformer model to reach human-level performance on a variety of language based tasks: question answering, sentiment classification or NSP

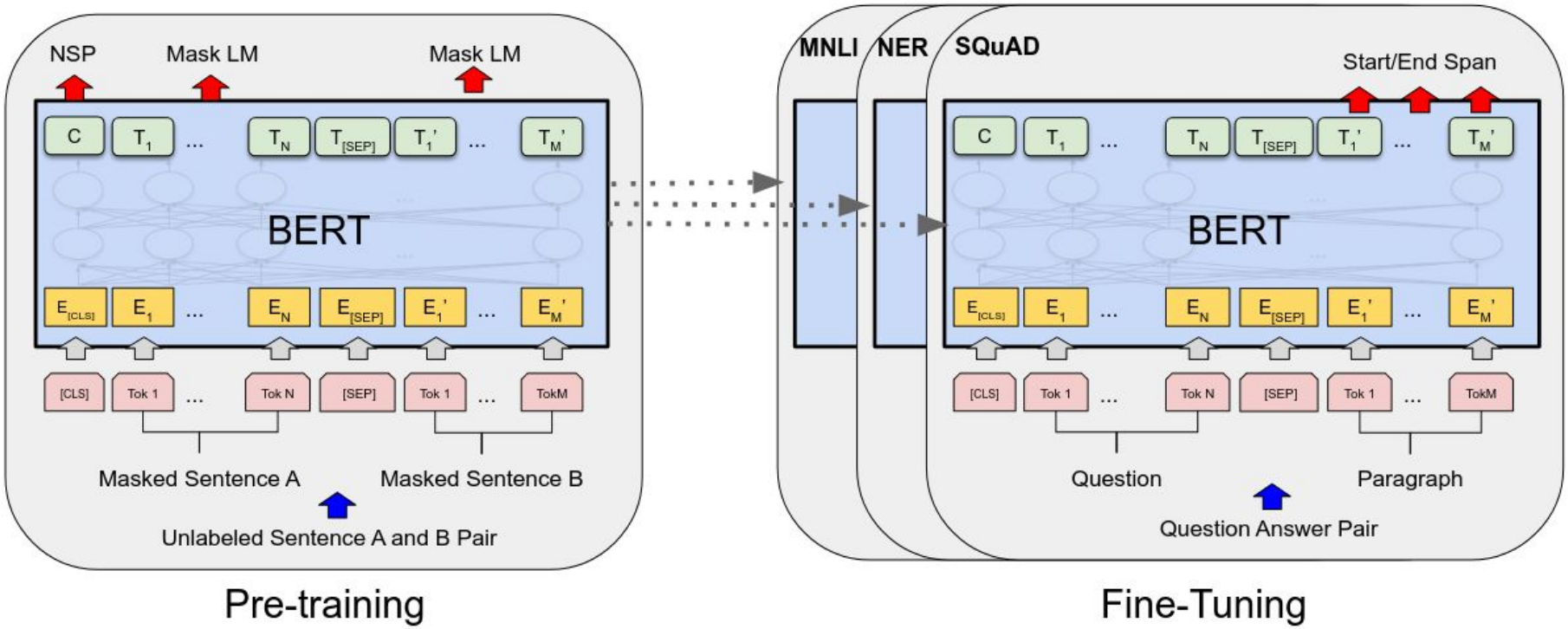
Foundation model in the paradigm pretrain and fine-tune

Two pretraining tasks: Mask Language Model and Next Sentence Prediction

2 special tokens <CLS> (prepending the input, used as a sentence representation in sequence classification tasks) and <SEP>

For classification, maps the first output token to softmax probabilities over classes or add a sequence2sequence layer for more complex tasks

BERT


NSP
Mask LM
Mask LM
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
TokM
Masked Sentence A
Masked Sentence B
Unlabeled Sentence A and B Pair
Pre-training
MNLI
NER
SQuAD
Start/End Span
Question
Paragraph
Question Answer Pair
Fine-Tuning

BERT

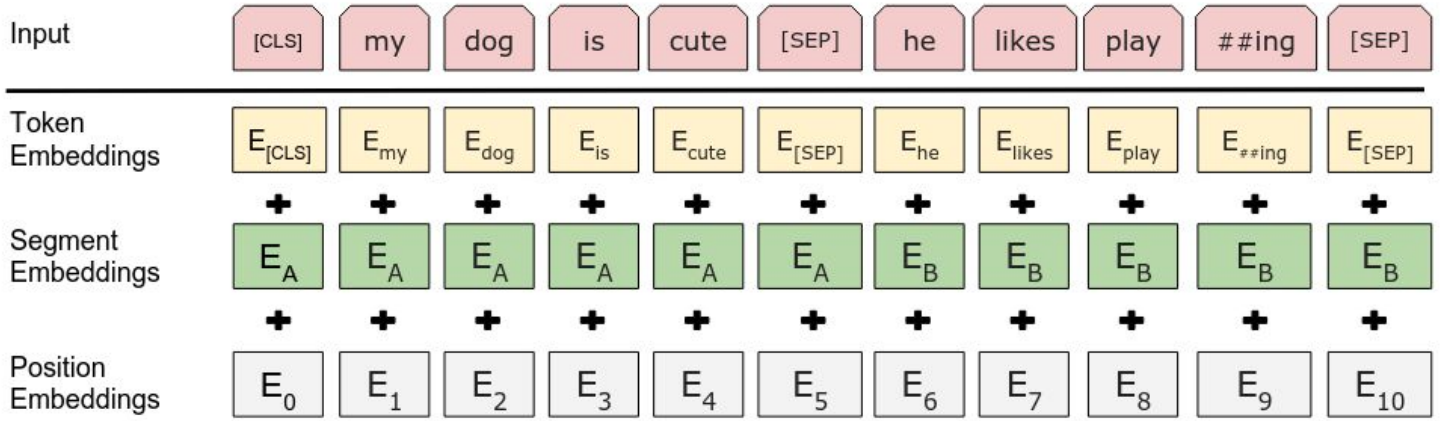


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

BERT

Pre-trained on general-domain corpus (800M words modern, unpublished authors of English books + 2.5B words English Wikipedia articles (without markup)

Use **WordPiece** embeddings (Wu et al. @Google, 2016) with a 30,000 token vocabulary

$BERT_{BASE}$ 110M parameters ; $BERT_{LARGE}$ 340M

GPT-2
Generative Pretrained Transformer

GPT-2

Language Models are Unsupervised Multitask Learners
(Radford et al. @OpenAI, 2019)

Transformer Decoder

Language generation foundation model

Pretrained on 8 M of web pages (outbound links from Reddit which received at least 3 karma)

1.8 B parameters

Use byte-pair encoding tokenizer

T5 encoder-decoder not covered in 2024-2025

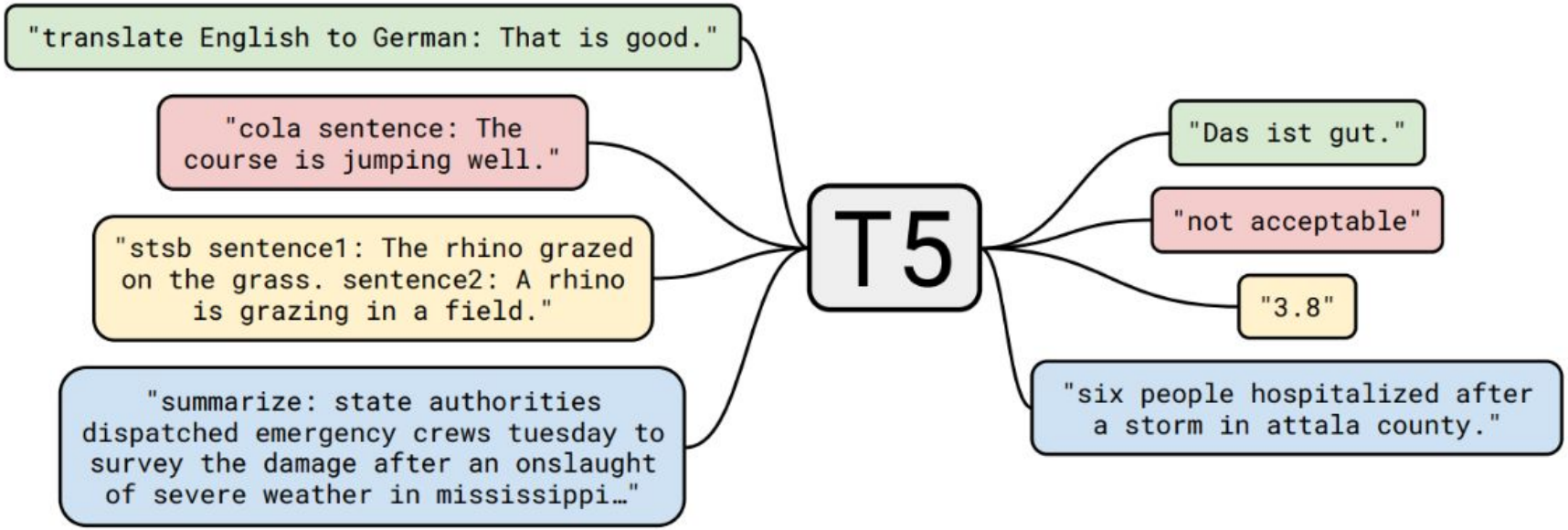


Figure 4: An illustration of T5 (Raffel et al., 2020) text-to-text text generation approach for Machine Translation, linguistic acceptability, text semantic similarity and summarizing tasks. Figure source: Raffel et al. (2020).

Combination of « text generation » and « prompt » paradigms

cola (grammatical acceptability), sts (semantic textual similarity score)

Tokenizers

- Byte-Pair Encoding (BPE)
- WordPiece

Byte Pair Encoding (BPE) tokenizer algorithm

Build a vocabulary of a desired size from a training corpus

Based on (Gage, 1994)'s algorithm for data compression:

- Find the most frequently occurring pairs of adjacent bytes in the data and replace all instances of the pair with a byte that was not in the original data
- Repeat this process until no more frequently occurring pairs or no more unused bytes to represent pairs

(Sennrich et al., 2015)'s algorithm works with unicode character instead of bytes

- Initiate the vocabulary symbols with the characters of length 1 used in the words of the training corpus
- While the desired vocabulary size is not reached,
  - search the most frequent of consecutives symbols from the vocabulary,
  - merge them as a single symbol and add it to the vocabulary

Used in GPT, GPT-2, RoBERTa, BART, DeBERTa

Byte-level BPE" might be more robust when finding unseen unicode character in test set

WordPiece Tokenizer from Google

Algorithm:

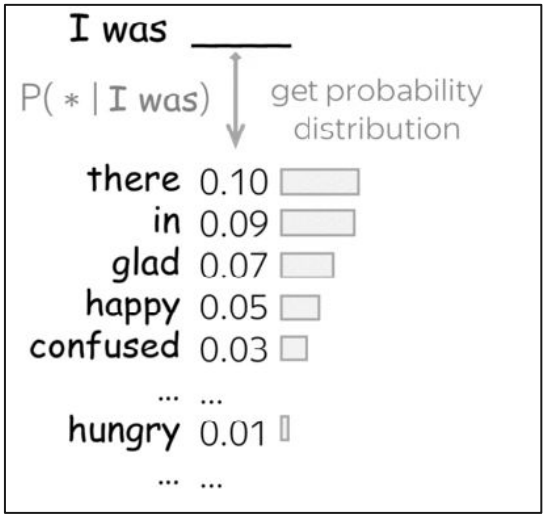
Initialize the vocabulary with all the characters occurring at the beginning of a word from the training corpus + characters within words by attaching the prefix ##

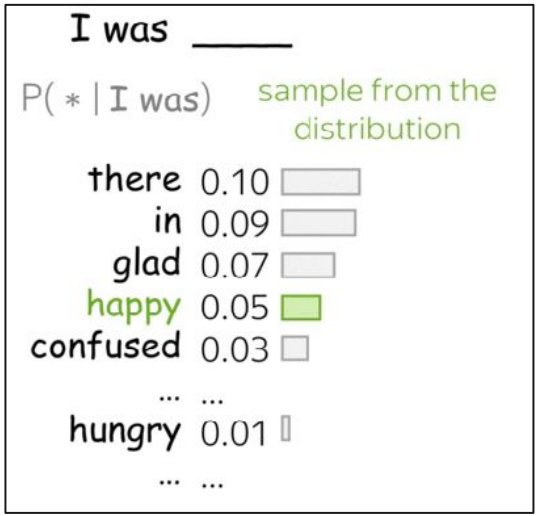
While the desired vocabulary size is not reached,

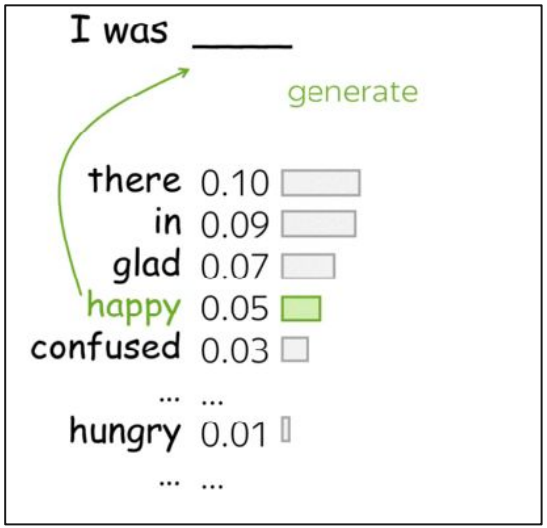
- Select pairs of characters with the highest score: `score=(freq_of_pair)/(freq_of_first_element×freq_of_second_element)`
- Merge the pair by removing ##

Used in BERT pre-trained language models

Generation/decoding strategies







Credit: lena-voita and Maxime Labonne

When a model generates texts

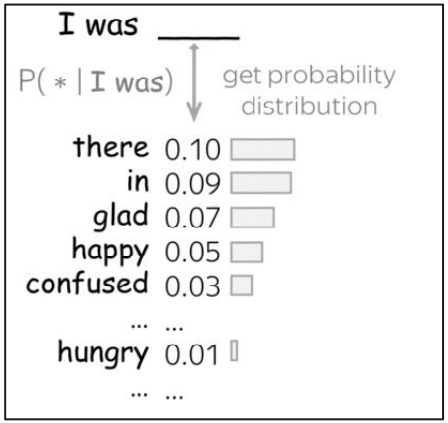
Usually you want the generated texts to be:

- coherent - the generated text has to make sense;
- diverse/creative - the model has to be able to produce very different samples

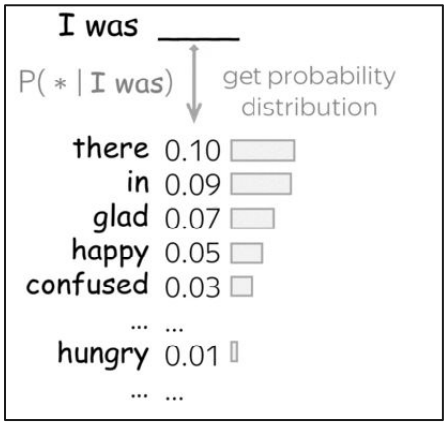
🏃 Standard sampling or greedy search (method 1)

Use the distributions predicted by the model without any modifications: **takes the most probable token at each step as the next token in the sequence**

+ Pro: intuitive, fast and efficient (doesn't keep track of multiple sequences)
- Cons: short-sighted (doesn't consider the overall effect on the sequence), can miss out on better sequences that might have appeared with slightly less probable next tokens, a bit deterministic…



The next word(s) may/will be… ?



The next word(s) may/will be… ?

« there »

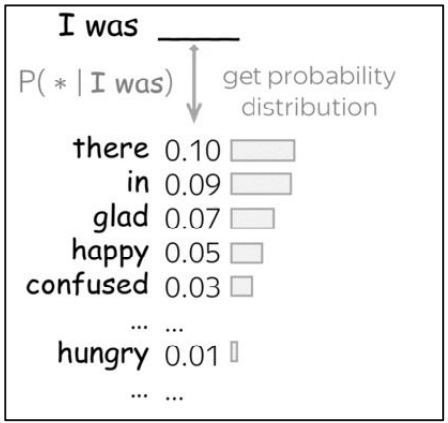
⚖️ Beam Search (method 2)

At each stage, **takes into account the `n` most likely tokens (not only one) from all the current beams (sequence) being explored**

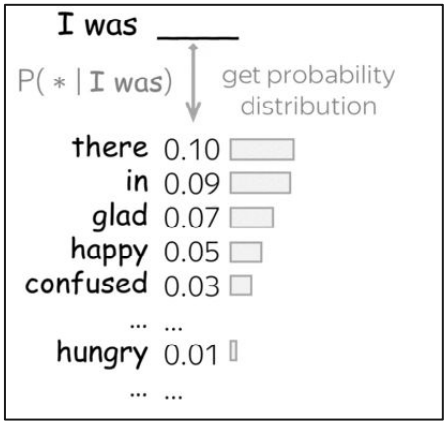
Compute and select the `n` beams with the highest overall probability

Repeat until a predefined maximum length is reached or an end-of-sequence (`eos`) token appears

- Cons: more computation than greedy
+ Pro: better results



With this context, if n = 2, the next word(s) may/will be… and the beam(s) may/will be…



With this context, if n = 2, the next word(s) may/will be…

« there » and « in »

and the beam(s) may/will be…

- « I was there »
- « I was in »

Both beams to be pursued…

Top-k sampling (method 3)

Select a token randomly from the `k` most likely options

Example:

> Given k = 3 and four tokens A, B, C and with respectives probabilities:
> P(A)= 30%, P(B)= 15%, P(C)= 5%, P(D)=1%
>
> In top-k sampling, D is disregarded
> and A will output 60% of the time (i.e. 30*100/(30+15+5)), B 30% and C 10%

Pro: prioritize the most probable tokens while introducing randomness in the selection process

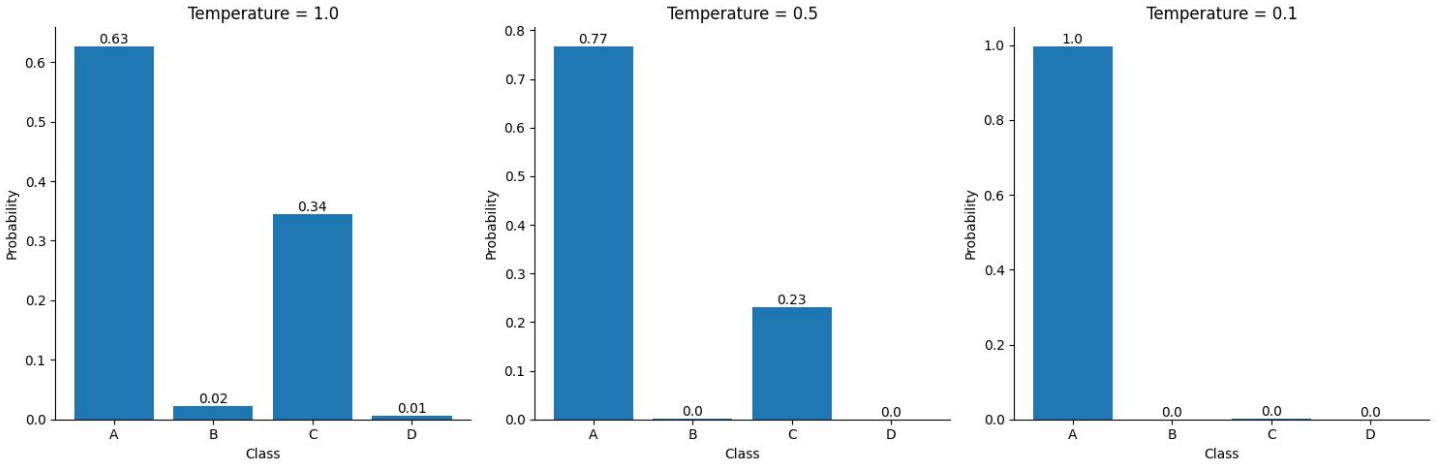
Sampling with temperature (method 4)

Affects the probabilities generated by the softmax (usually) by dividing the softmax input logits by a value `t` called the temperature

$$softmax(x_i) = \frac{e^{x_i/t}}{\sum_j e^{x_j/t}}$$

- `t=1` is the default softmax
- lower values make the most likely tokens more influential
- higher values tend to smooth the distribution

Example: given the following input logits [1.5, -1.8, 0.9, -3.2]



Credit: mlabonne

Nucleus sampling aka top-p sampling (method 5)

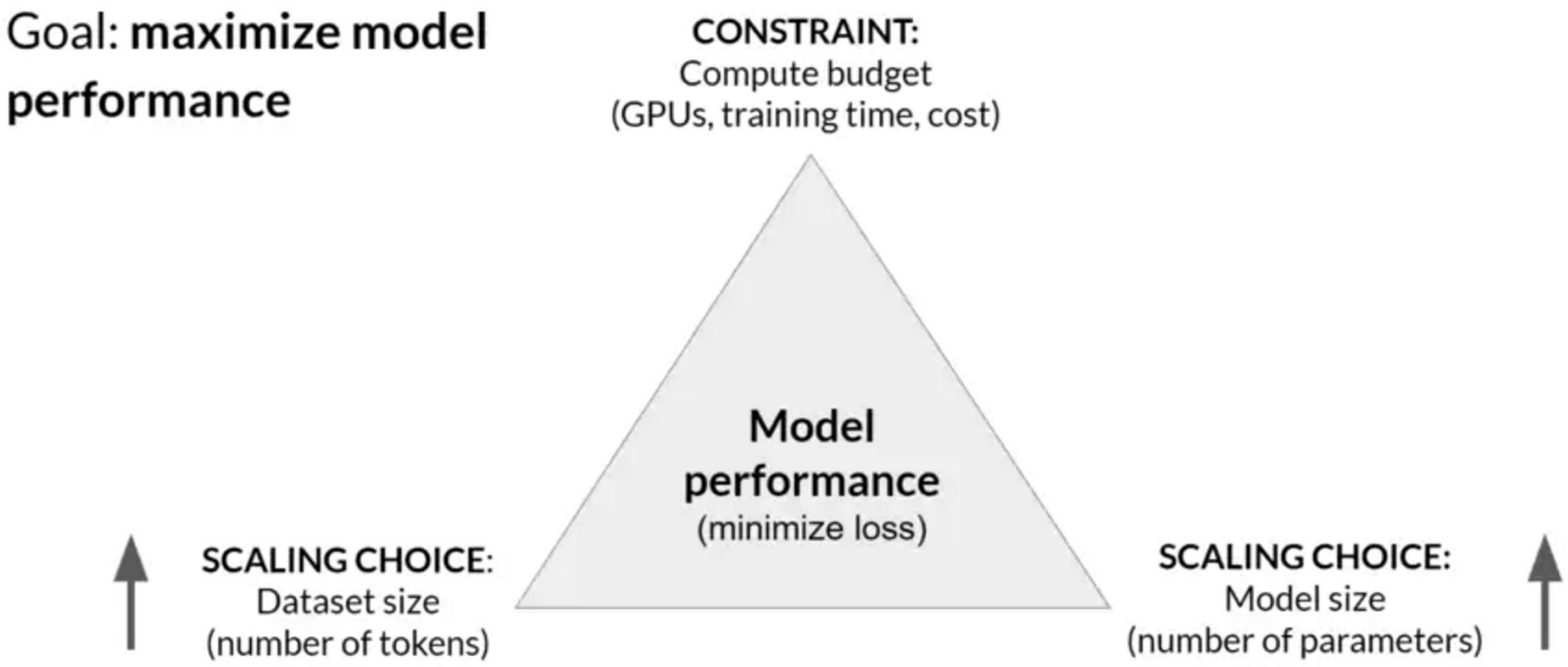
Examines the top probable tokens in descending order
Sum the probabilities of the selected tokens until the total surpasses a threshold `p`
Choose randomly the next token inside this “nucleus” of tokens

Pro: number of included tokens vary from step to step, this results in a more diverse and creative output

Play with Top K (method 3), Temperature (method 4), Top p sampling (method 5) and more on https://artefact2.github.io/llm-sampling/index.xhtml

Scaling laws and compute-optimal models

Scaling choices for pre-training



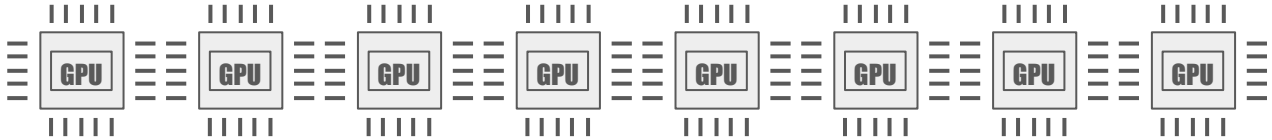
Credit: Antje Barth

Compute budget for training LLMs

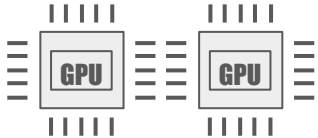
petaflops/s-day = #FLOP performed at a rate of 1 **petaFLOP per second for one day**

1 petaFLOP is equivalent to running at full efficiency for 24 hours

either **8 NVIDIA V100s**



or **2 NVIDIA A100s**



Note:

- FLOPS = floating-point operations per second ;
- mega 10^6, giga 10^9, tera 10^{12}
  et peta 10^{15} = 1,000,000,000,000,000 (1 million of billion)

For information (general comparison)

GPU model	GPU memory	Interconnect	NVIDIA RTX Virtual Workstation (vWS) support	Best used for
H100 80GB	80 GB HBM3 @ 3.35 TBps	NVLink Full Mesh @ 900 GBps	⊘	Large models with massive data tables for ML Training, Inference, HPC, BERT, DLRM
A100 80GB	80 GB HBM2e @ 1.9 TBps	NVLink Full Mesh @ 600 GBps	⊘	Large models with massive data tables for ML Training, Inference, HPC, BERT, DLRM
A100 40GB	40 GB HBM2 @ 1.6 TBps	NVLink Full Mesh @ 600 GBps	⊘	ML Training, Inference, HPC
L4	24 GB GDDR6 @ 300 GBps	N/A	✓	ML Inference, Training, Remote Visualization Workstations, Video Transcoding, HPC
T4	16 GB GDDR6 @ 320 GBps	N/A	✓	ML Inference, Training, Remote Visualization Workstations, Video Transcoding
V100	16 GB HBM2 @ 900 GBps	NVLink Ring @ 300 GBps	⊘	ML Training, Inference, HPC
P4	8 GB GDDR5 @ 192 GBps	N/A	✓	Remote Visualization Workstations, ML Inference, and Video Transcoding
P100	16 GB HBM2 @ 732 GBps	N/A	✓	ML Training, Inference, HPC, Remote Visualization Workstations

HPC = High Performance Computing

DLRM = Deep Learning Recommendation Model

Source: [google cloud documentation] (2024-12-16)

For information (performance)

Compute performance

GPU model	FP64	FP32	FP16	INT8
H100 80GB	34 TFLOPS	67 TFLOPS		
A100 80GB	9.7 TFLOPS	19.5 TFLOPS		
A100 40GB	9.7 TFLOPS	19.5 TFLOPS		
L4	0.5 TFLOPS*	30.3 TFLOPS		
T4	0.25 TFLOPS*	8.1 TFLOPS		
V100	7.8 TFLOPS	15.7 TFLOPS		
P4	0.2 TFLOPS*	5.5 TFLOPS		22 TOPS†
P100	4.7 TFLOPS	9.3 TFLOPS	18.7 TFLOPS	

Tensor core performance

GPU model	FP64	TF32	Mixed-precision FP16/FP32	INT8	INT4	FP8
H100 80GB	67 TFLOPS	989 TFLOPS†	1,979 TFLOPS*,†	3,958 TOPS†		3,958 TFLOPS†
A100 80GB	19.5 TFLOPS	156 TFLOPS	312 TFLOPS*	624 TOPS	1248 TOPS	
A100 40GB	19.5 TFLOPS	156 TFLOPS	312 TFLOPS*	624 TOPS	1248 TOPS	
L4		120 TFLOPS†	242 TFLOPS*,†	485 TOPS†		485 TFLOPS†
T4			65 TFLOPS	130 TOPS	260 TOPS	
V100			125 TFLOPS			
P4						
P100						

Source: [google cloud documentation](#) (2024-12-16)

For information (price)

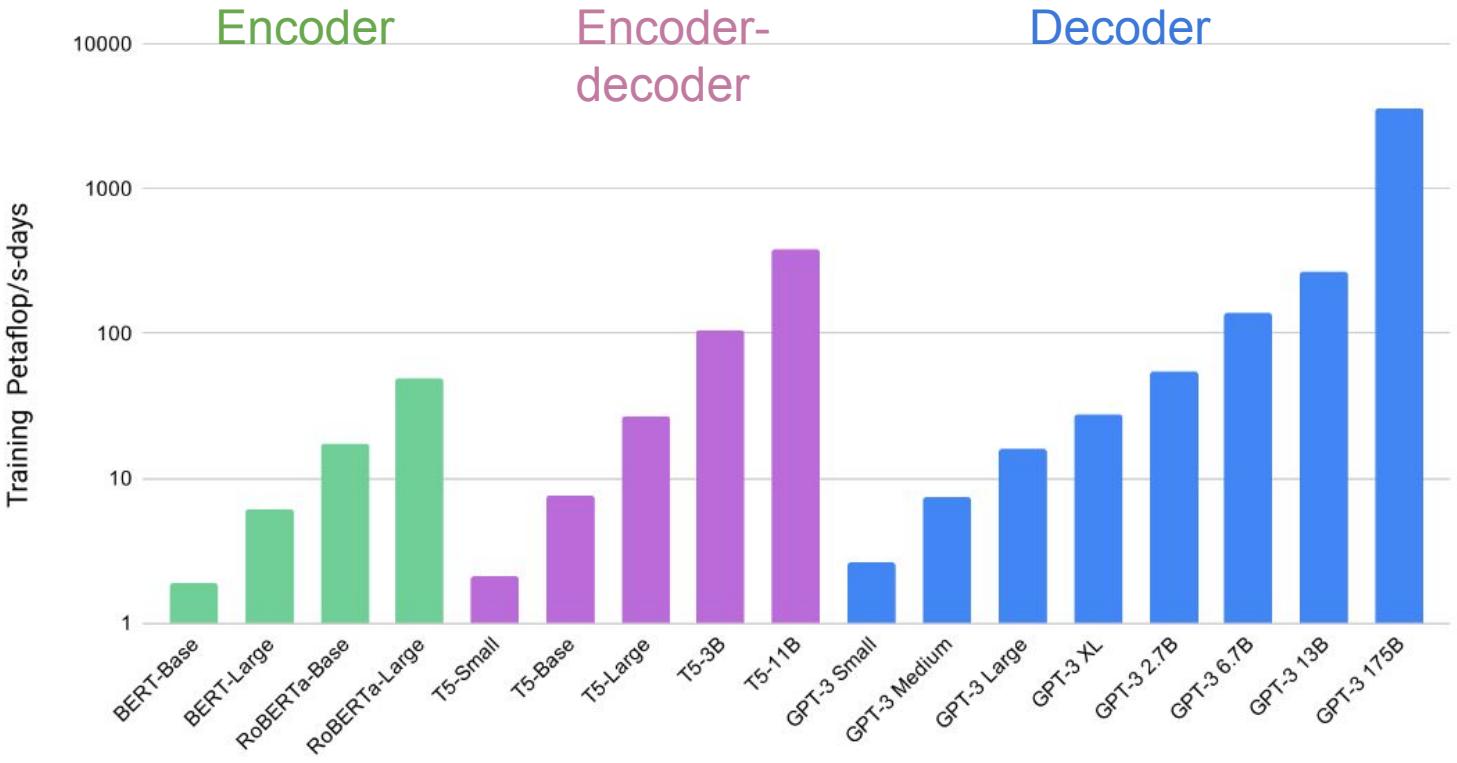
Paris (europe-west9) ▾ | ✓ Hourly | Monthly

Model	GPUs	GPU memory	GPU price (USD) per GPU	1 year commitment price** (USD) per GPU	3 year commitment price** (USD) per GPU
NVIDIA T4	1 GPU	16 GB GDDR6	$0.406 / 1 hour	$0.2552 / 1 hour	$0.1856 / 1 hour
	2 GPUs	32 GB GDDR6			
	4 GPUs	64 GB GDDR6			
NVIDIA P4	1 GPU	8 GB GDDR5	$0.696 / 1 hour	$0.43848 / 1 hour	$0.3132 / 1 hour
	2 GPUs	16 GB GDDR5			
	4 GPUs	32 GB GDDR5			
NVIDIA V100	1 GPU	16 GB HBM2	$2.8768 / 1 hour	$1.81192 / 1 hour	$1.29456 / 1 hour
	2 GPUs	32 GB HBM2			
	4 GPUs	64 GB HBM2			
	8 GPUs	128 GB HBM2			
NVIDIA P100	1 GPU	16 GB HBM2	$1.6936 / 1 hour	$1.06604 / 1 hour	$0.76212 / 1 hour
	2 GPUs	32 GB HBM2			
	4 GPUs	64 GB HBM2			

Source: google cloud documentation (2024-12-16)

Number of petaflop/s-days to pre-train various LLMs

log scale



(Brown et al.@NEURIPS 2020)

Three families of models

Difference between the models in each family is the number of parameters that were trained, **from a few hundred million for BERT base to 175 billion for the largest GPT-3**

With almost no difference between the families, **an increasing number of parameters an increasing number of petaFLOP-s/days**

I.e. Bigger models take more compute resources

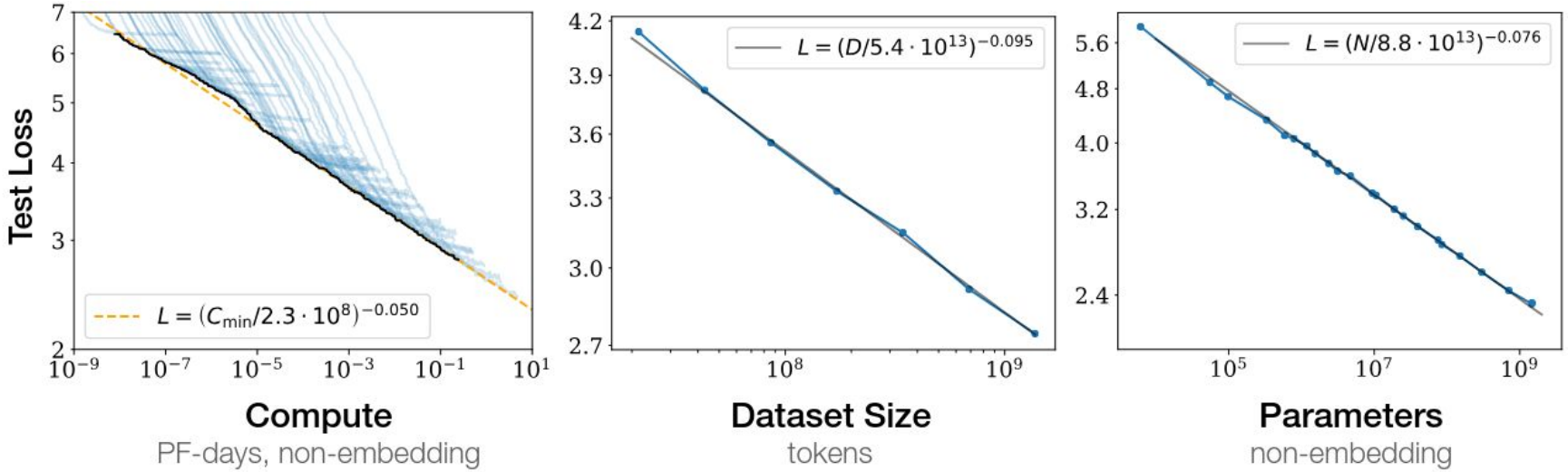
T5 XL 3B parameters required close to 100 petaFLOP-s/days

GPT-3 175B parameter model required 3,700 petaFLOP-s/days

Note: Logarithmic y-axis (each vertical increment a power of 10)

Performance improves smoothly as the model size, dataset size, or amount of compute for training increases

Loss as proxy of the model performance (the lower the better)



Scaling Laws for Neural Language Models (Kaplan et al @OpenAI, 2020)

- **Increasing one variable while freezing the other ones :**

  X and Y axis' variables appear to be related by **a power-law relationship**

- A power law is a mathematical relationship between two variables, where **one is proportional to the other** raised to some power. When **plotted on a graph where** both **axes are logarithmic**, power-law relationships appear as **straight lines**

Note : In Fig. "compute", each thin blue line represent the model loss over a single training run

What's the ideal balance between these three quantities?

Train 400 language models from 70 M to 16 B of parameters on 5 to 500 B of tokens under a given compute budget, evaluate on downstream NLP (understanding) tasks

Find that

- Language models **at this time** are **over-parameterized and significantly undertrained**: **too many parameters for the training data used**
- Smaller models trained on more data could perform as well as large models
- « for compute-optimal training, the model size and the number of training tokens should be scaled equally, **for every doubling of model size the number of training tokens should also be doubled** »

Training Compute-Optimal Large Language Models (Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch et al. @DeepMind, 2022) aka Chinchilla paper

A mathematical relationship

The authors define a loss function

$$L = L_0 + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}$$

with

- N, the number of parameters (model size)
- D the number of training tokens
- L_0 the irreductible loss, representing the best performance
- A and B, constants representing the model's underperformance compared to an ideal
- generative process
- **α** and **β** exponents that describe how loss scales with respect to model size and data size, respectively

And estimate $L(N, D) = 1.69 + 406.4/N^{0.34} + 410.7/D^{0.28}$

Chinchilla scaling laws for model and dataset size

Empirical law:

data-optimal (Chinchilla) law: 20:1 ratios of tokens trained to parameters

Model	# of parameters	Given the #parameters Ideal #tokens should be ~20x	Actual # tokens
Chinchilla	70B	~1.4T	1.4Trillion i.e. 10^{12}
LLaMA-65B	65B	~1.3T	1.4T
GPT-3	175B	~3.5T	300Billion i.e. 10^{9}
OPT-175B	175B	~3.5T	180B
BLOOM	176B	~3.5T	350B

Metrics given in short scale. In France, the long scale is official !

References

L'IA générative avec de grands modèles de langage, Chris Fregly, Antje Barth, Shelbee Eigenbrode, Mike Chambers @ DeepLearning.AI, coursera, 2023

Language Models are Few-Shot Learners, Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei, Part of Advances in Neural Information Processing Systems 33 (NeurIPS 2020)

Kaplan, Jared; McCandlish, Sam; Henighan, Tom; Brown, Tom B.; Chess, Benjamin; Child, Rewon; Gray, Scott; Radford, Alec; Wu, Jeffrey; Amodei, Dario (2020). "Scaling Laws for Neural Language Models". OpenAI, CoRR. abs/2001.08361

Training Compute-Optimal Large Language Models, 2022, Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, Laurent Sifre, DeepMind, arXiv:2203.15556

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova.

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016, Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean, arXiv:1609.08144

Gage, Philip (1994). "A New Algorithm for Data Compression". The C User Journal.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Références

- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation, 9(8):1735–1780.
- Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.
- Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 328–339, Melbourne, Australia. Association for Computational Linguistics.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8), 9.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., … Stoyanov, V. (2019). Roberta: a robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*, 5998-6008.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages