Staged Training for Transformer Language Models

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, Iz Beltagy

Staged Training

Goal: Train a large language model

Now (1 stage training): [Large Model] ⇒ {Train} ⇒ [Target Model]

Proposed (multi-stage training):

[Small Model] ⇒ {Train} ⇒ {Grow} ⇒ [Larger Model] ⇒ {Train} ⇒ {Grow} .... ⇒ [Target Model]

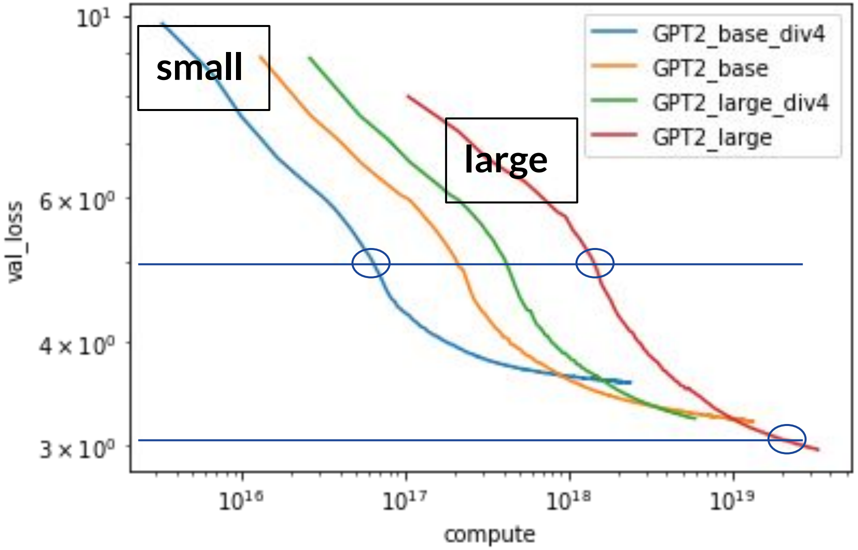
Prior work (e.g. [1]) proposed the same method but missing key ideas and intuitions to get it to work reliably and achieve max compute saving

[1] Net2Net: Accelerating Learning via Knowledge Transfer, Chen et. al., ICLR 2016

Staged Training - Facts

Smaller models are **initially faster** to train then they **plateau**

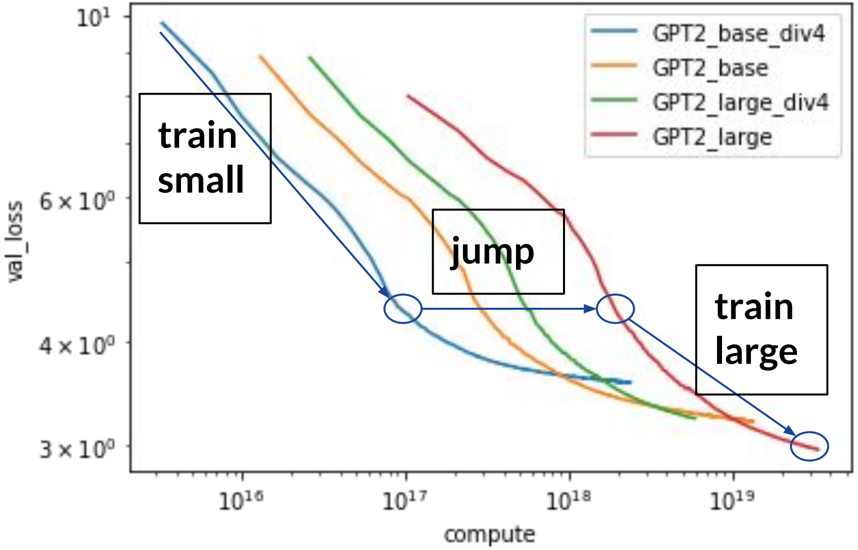
Larger models are initially **slower** than smaller models but eventually become **more efficient**



Staged Training - Intuition

Training regime:

- train small model until loss slows down
- "jump" to a larger one
- train larger one until loss slows down



Why?

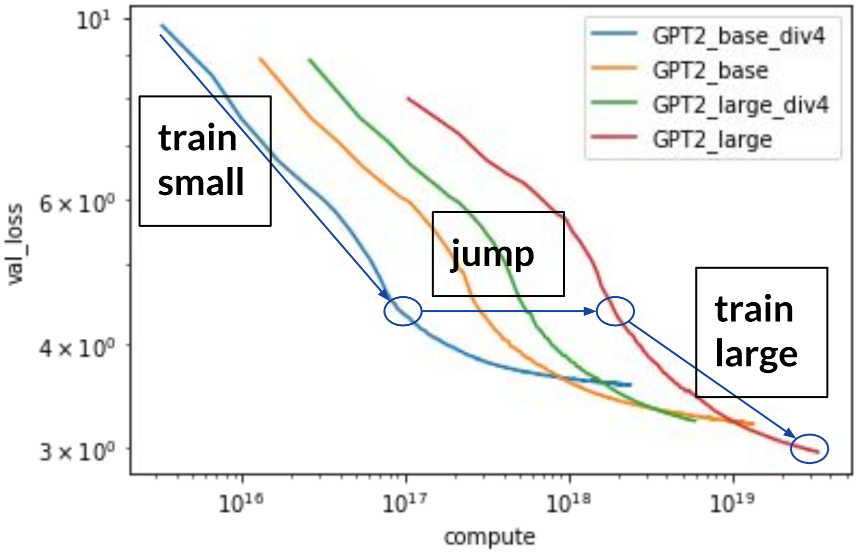
- the jump saves compute
- intermediate model sizes for free

We call the "jump" a **Growth Operator**

Staged Training - Intuition

- How to jump **effectively**

- How to identify the 3 points for **optimal** compute saving



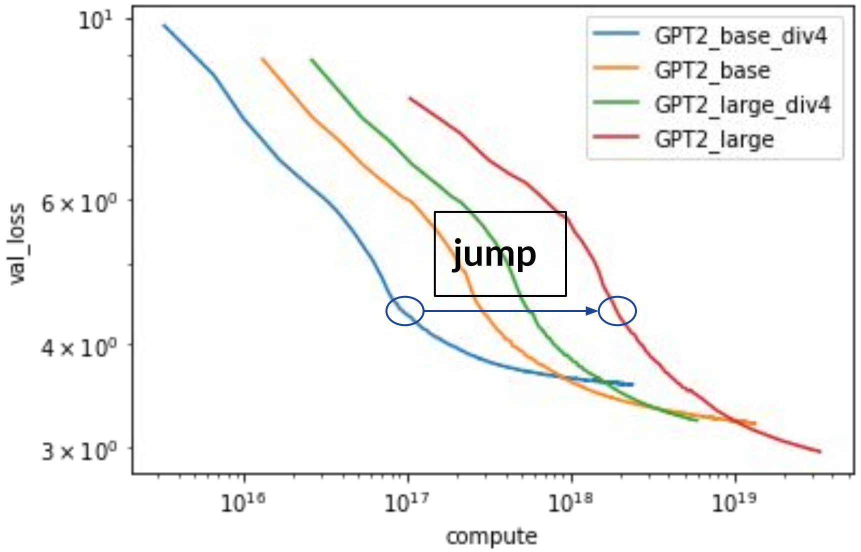
Staged Training

- **Properties of growth operators**
  - Loss preserving
    - Depth and Width operators
  - Training dynamics preserving
    - Optimizer and Learning rate
- Optimal Training Schedule
- Evaluation

Properties for Growth Operator

To effectively jump between learning curves, growth operator should have the following properties

1) **loss-preserving** (function-preserving): loss before growing model is the same as after

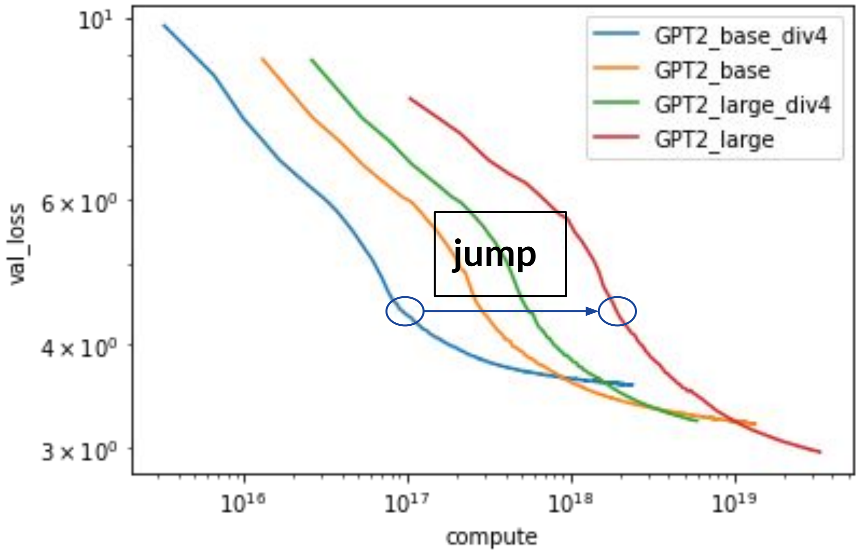


Properties for Growth Operator

To effectively jump between learning curves, growth operator should have the following properties

1) **loss-preserving** (function-preserving): loss before growing model is the same as after

2) **training-dynamics-preserving**: rate of loss change after growing the model is the same as training the model from "scratch"

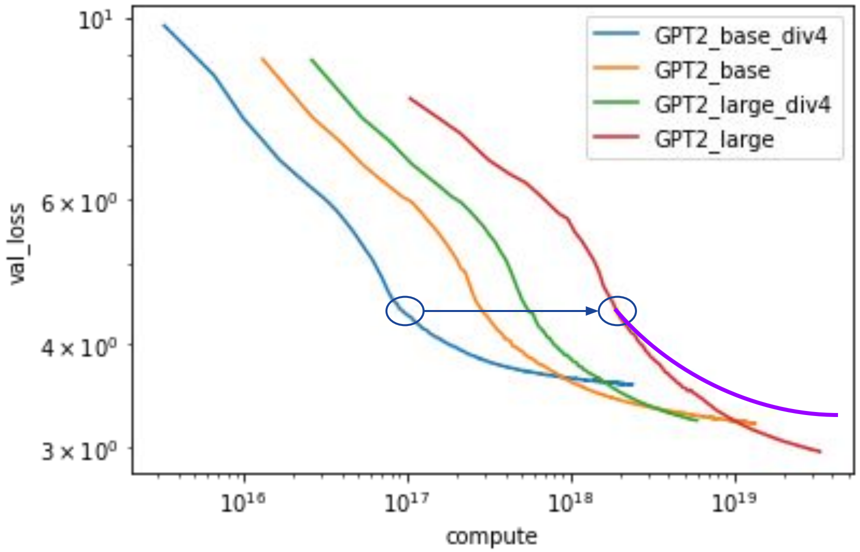


Properties for Growth Operator

training-dynamics-preserving: after growth, model trains as fast as the model trained from scratch

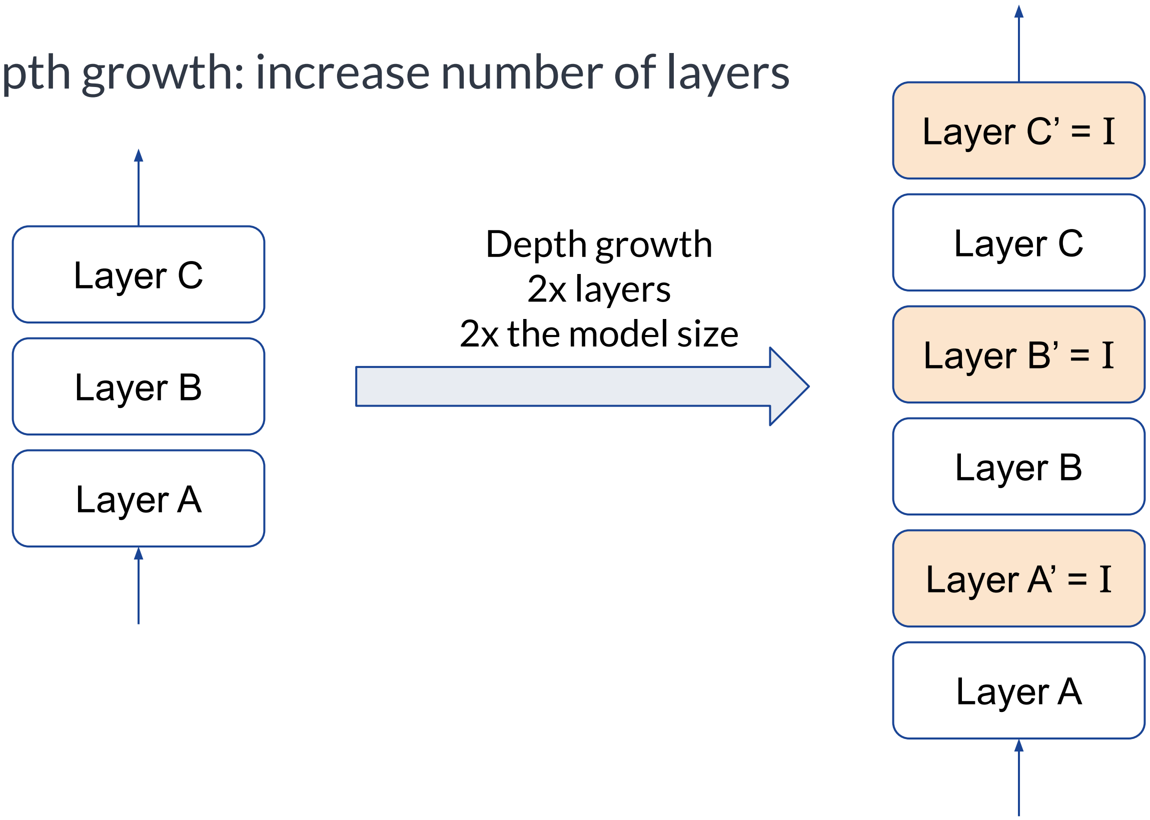
An ineffective growth operator creates a larger model but one that doesn't train fast

We are the first to recognize the importance of this property



Growth operators - Depth

Depth growth: increase number of layers

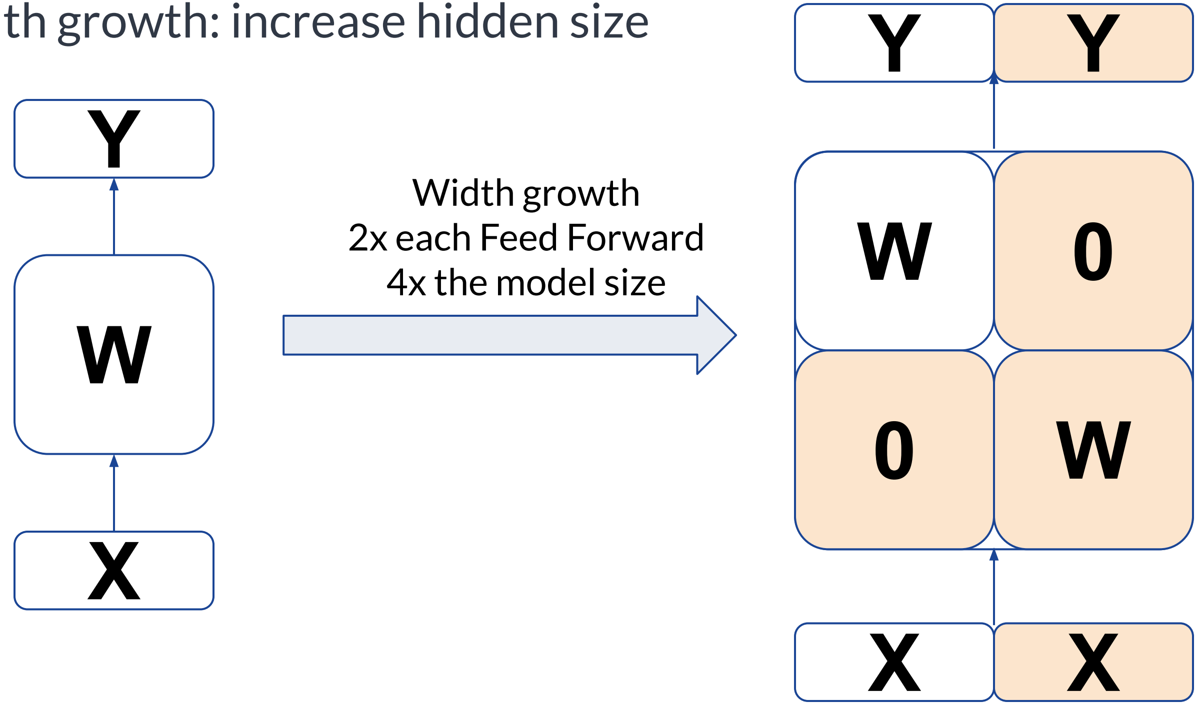


Copy layers then manipulate a few weights to convert it into an Identity

(Loss-preserving)

Growth operators - Width

Width growth: increase hidden size



Every embedding =>2x

Every FF becomes => 4x

Manipulate last hidden state to get the same logits

(Loss-preserving)

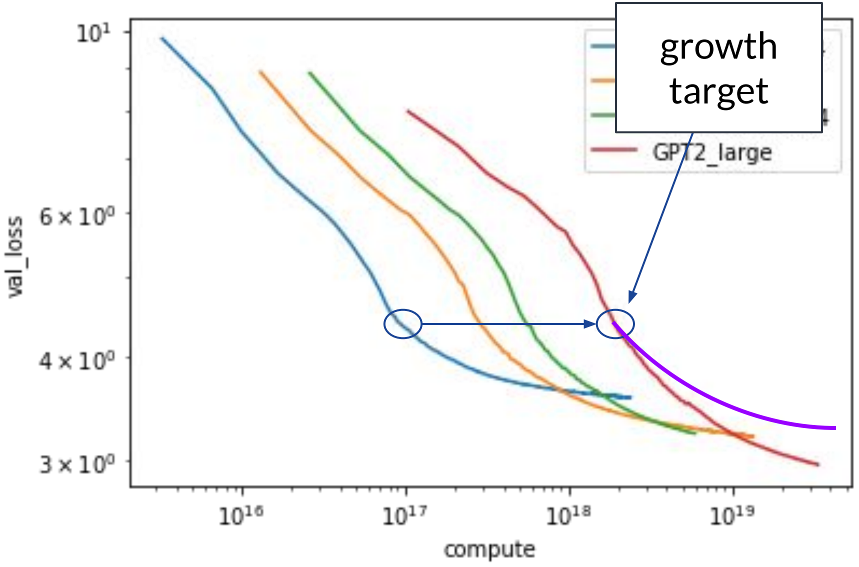
Properties for Growth Operator

To preserve training dynamics, growth operator should grow **whole training state (optimizer state and LR)** not just model

Intuition - get the whole training state to match that of one trained from scratch

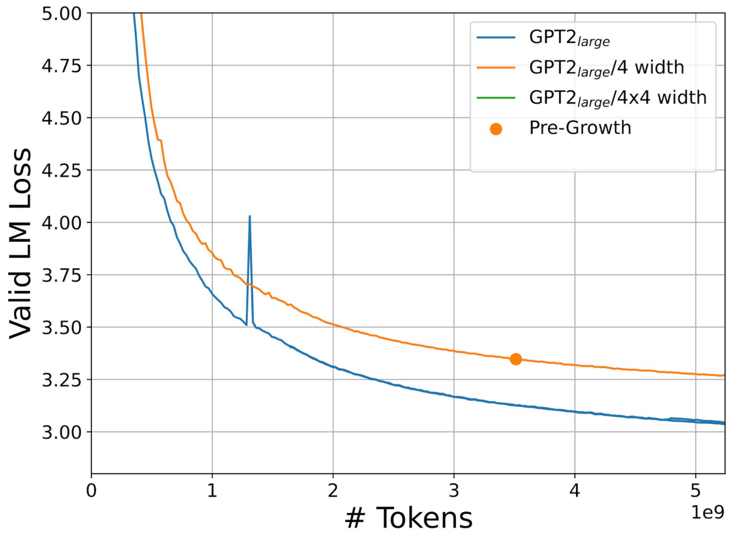
LR: use LR at growth target

Optimizer: grow optimizer state with a mostly similar growth operator to model growth (check paper for details)



Properties for Growth Operator - Evaluation

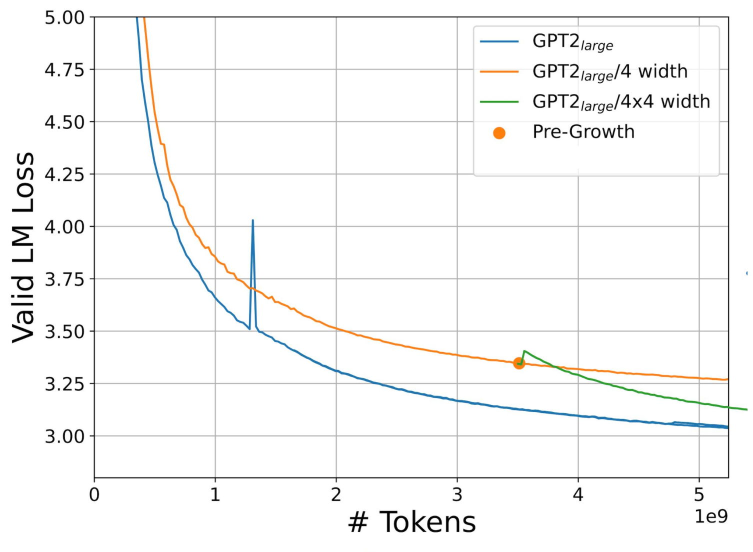
Two models trained from scratch



Properties for Growth Operator - Evaluation

Grow width of the small model. Grown model matches size of the larger model

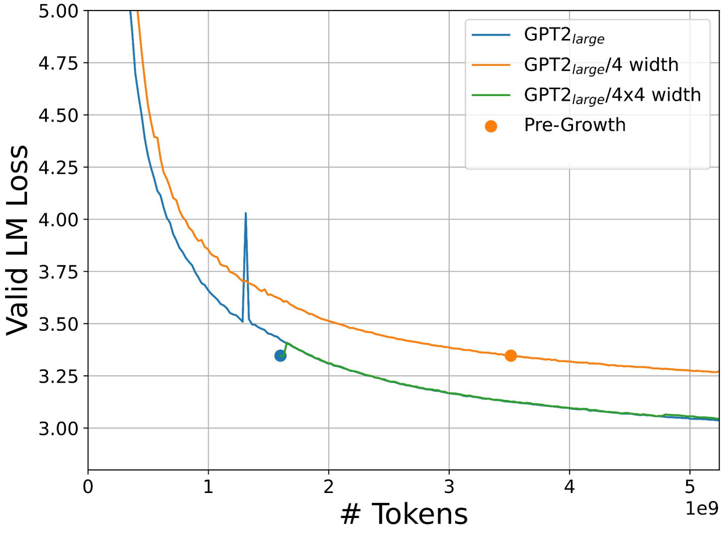
(Loss preserving)



Properties for Growth Operator - Evaluation

Overlay grown model over larger model trained from scratch

(Preserving training dynamics)



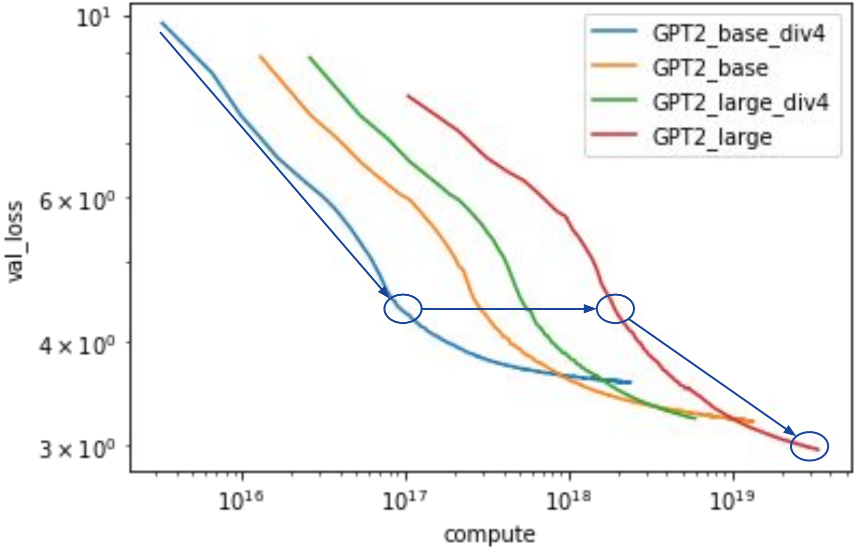
Staged Training

- Properties of growth operators
  - Loss preserving
    - Depth and Width operators
  - Training dynamics preserving
    - Optimizer and Learning rate
- **Optimal Training Schedule**
- Evaluation

Optimal Training Schedule

Prior work splits the compute heuristically between the stages.

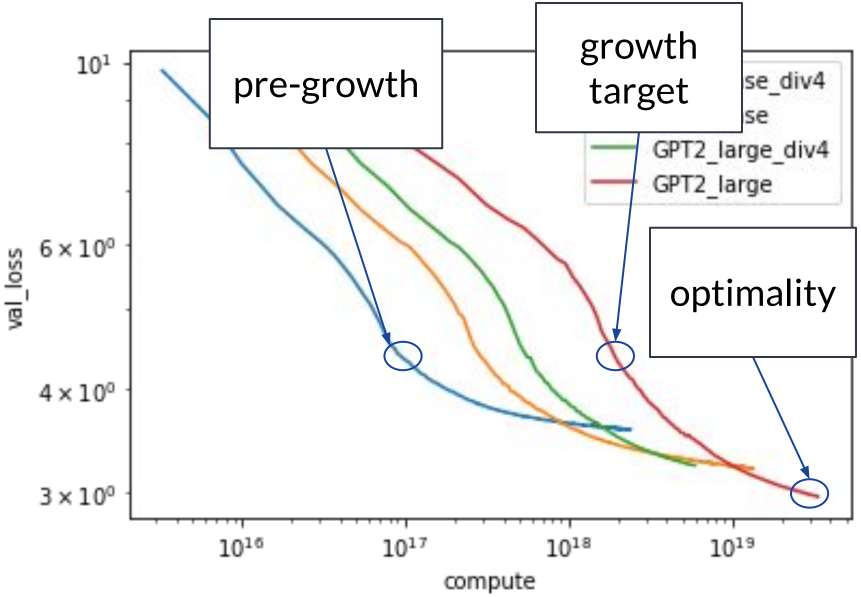
Here we see there's a **precise schedule** with the **optimal compute saving**



Optimal Training Schedule

Each stage is characterized by 3 points:

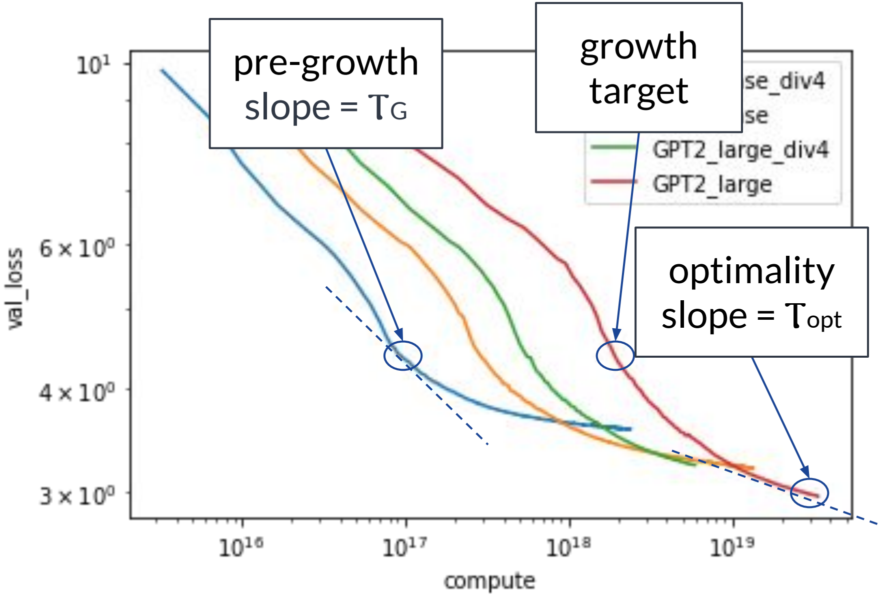
- **pre-growth**: when to grow
- **growth-target**: LR after growth
- **optimality**: stop training
  - not **convergence**
  - Read Kaplan et. al., and Hoffmann et. al.,



Optimal Training Schedule

Each stage is characterized by 3 points:

- **pre-growth**: when to grow
  - slope of learning curve, τ_{depth}, τ_{width}
- **optimality**: stop training
  - slope of learning curve, τ_{opt}
- **growth-target**: LR after growth
  - ratio $\frac{\text{steps@pre-growth}}{\text{steps@growth-target}} = \varrho$
  - function of the growth OP: ϱ_{depth}, ϱ_{width}



Check the paper for connection to Scaling Laws [Kaplan et. al.,], proof this is optimal, and how to estimate Tg, Topt, ϱg

Staged Training

- Properties of growth operators
  - Loss preserving
    - Depth and Width operators
  - Training dynamics preserving
    - Optimizer and Learning rate
- Optimal Training Schedule
- **Evaluation**

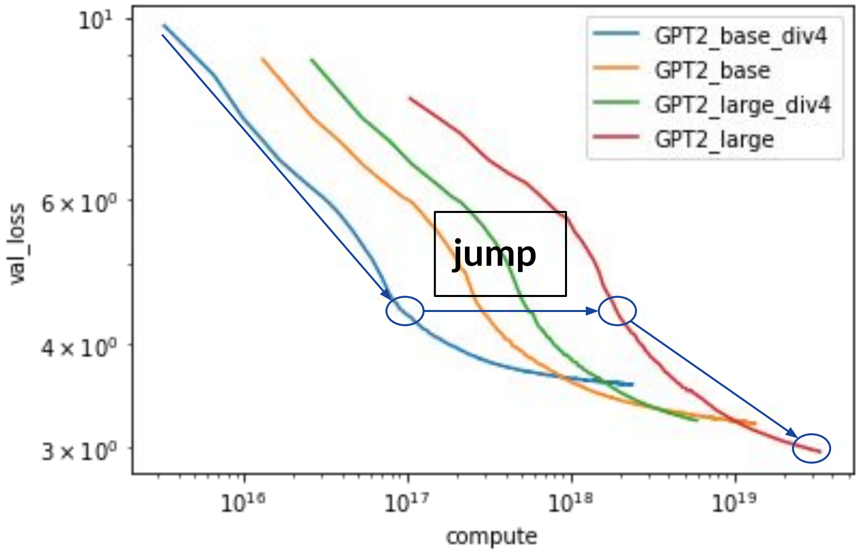
Evaluation - Pretraining loss

Percentage of compute saving

		$GPT2_{LARGE}$		$GPT2_{BASE}$	
		At OPT	After OPT	At OPT	After OPT
2 stage $_{practical}$	2xW	7.3	5.2	20.2	19.7 3
	4xW	5.3	3.8	8.6	5.5 0
	2xD	11.0	6.1	20.4	19.8 7
	4xD	7.3	5.2	10.1	6.4 8
	2xDxW	5.4	3.8	9.5	6.83 0
3 stage $_{practical}$	2x2xW	10.9	7.8	17.9	11.4 8
	2x2xD	14.5	10.4	21.4	15.9 3

Conclusion - What to remember

- Growth operator should be
    - loss-preserving
    - training-dynamics-preserving
- How to identify the 3 points for **optimal** compute saving
- Saved up to **20%** compute



paper link: https://arxiv.org/pdf/2203.06211.pdf

code link: https://github.com/allenai/staged-training