Probabilistic Graphical Models & Probabilistic AI

Ben Lengerich

Lecture 20: Implementing a GPT from Scratch

April 15, 2025

Reading: See course homepage

Today

- From Transformer to GPT
- Implementing a GPT in Python
- From "GPT" to GPT-4

From Transformer to GPT

Recall the "Transformer"

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to $-\infty$) …

☆ Save 99 Cite Cited by 174852 Related articles All 73 versions »

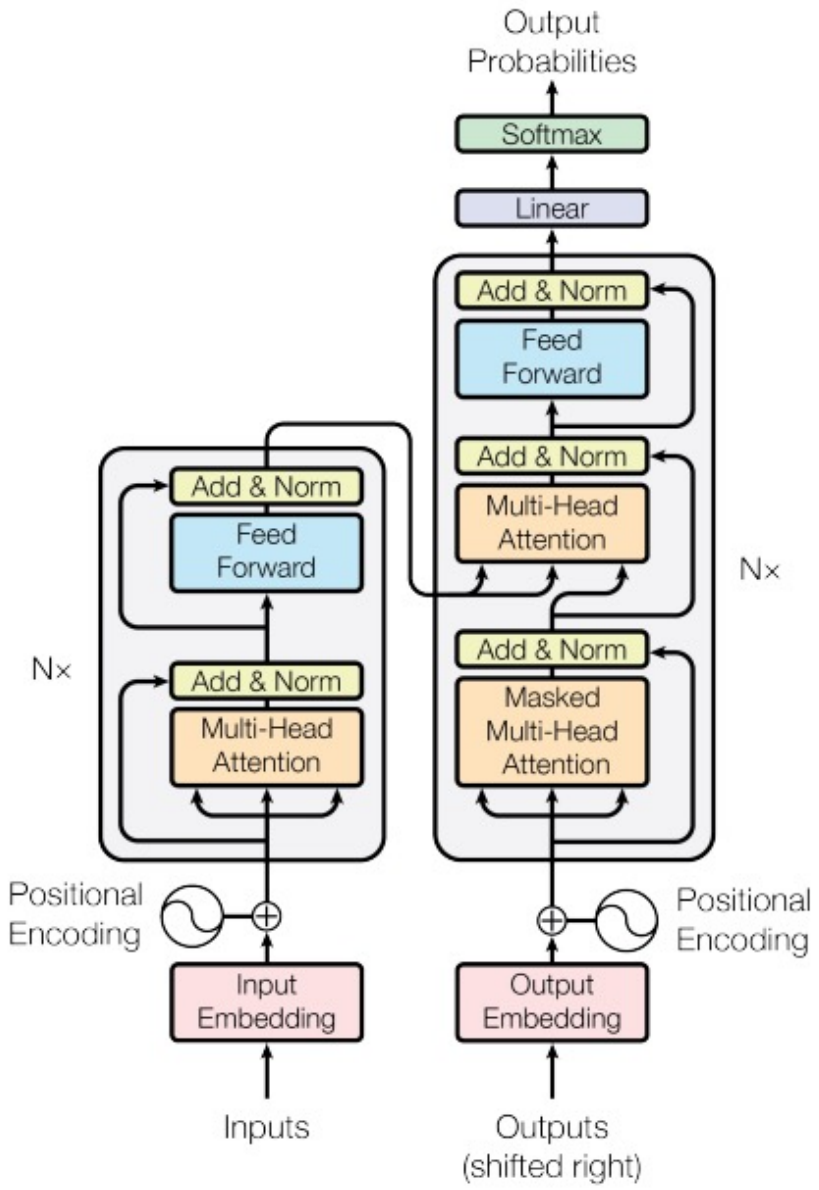


Figure 1: The Transformer - model architecture.

Recall the "Transformer"

- **Original Transformer** (Vaswani et al., 2017):
  - Encoder-decoder architecture for sequence-to-sequence tasks
  - Parallelizable self-attention instead of recurrence
  - Positional encodings enable order sensitivity
- **Encoder**: Processes input sequence
- **Decoder**: Generates output sequence using masked attention + encoder output
- Inspired by machine translation (observe full input sequence, predict full output sequence)

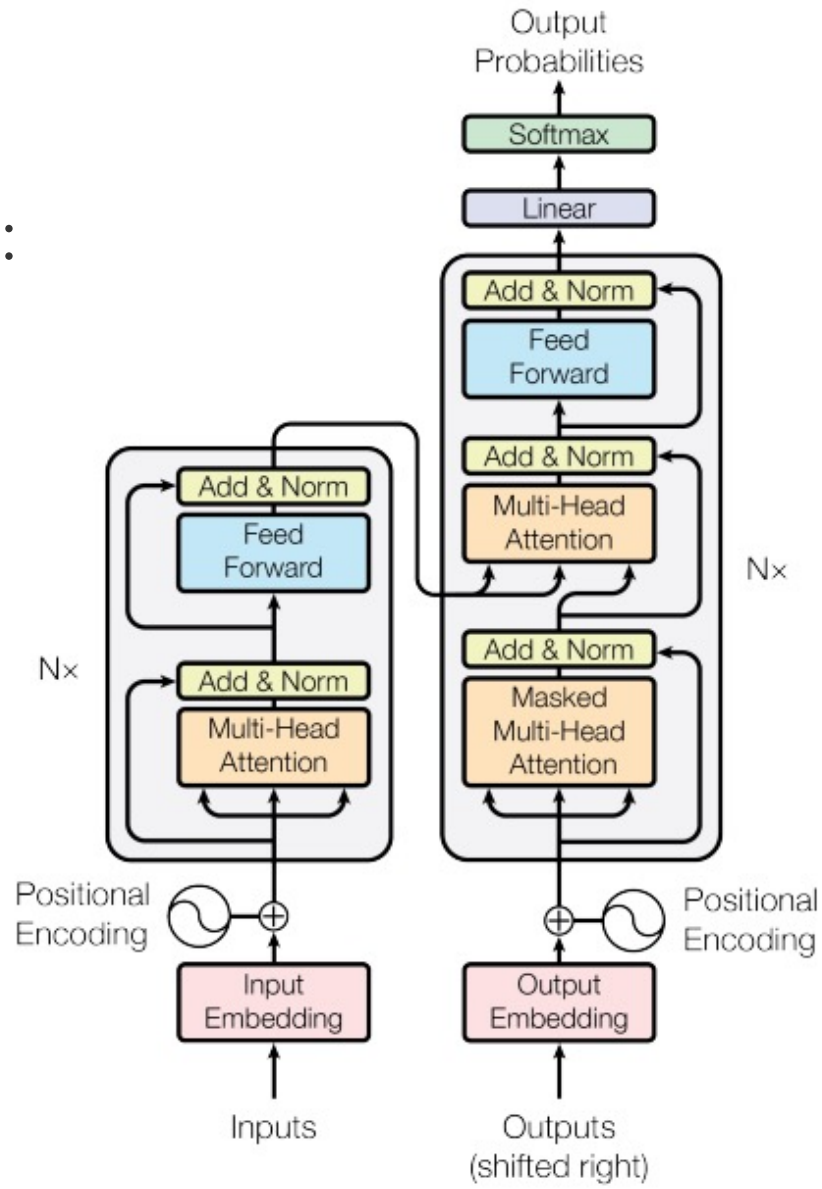


Figure 1: The Transformer - model architecture.

From Sequence Transduction to Sequence Modeling

- **Original Transformer** (Vaswani et al., 2017):

$$P(Y \mid X) = \prod_t P(Y_t \mid Y_{<t}, X)$$

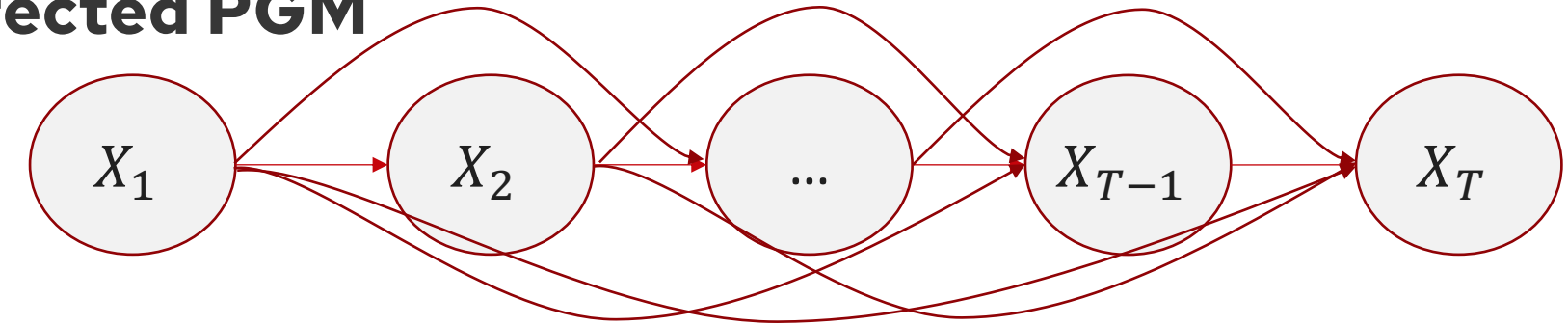
  - **Conditional** sequence model for tasks like translation (input → output)
- **Generative Pretrained Transformer (GPT)** Models:

$$P(X) = \prod_t P(X_t \mid X_{<t})$$

  - **Unconditional** generative model over raw text
- Architectural consequence: **no encoder**, only a decoder with causal structure

GPT = Probabilistic Model + Transformer Decoder

- **Directed PGM**



$$P_\theta(X) = \prod_i \prod_t P_\theta(X_{i,t} \mid X_{i,<t})$$

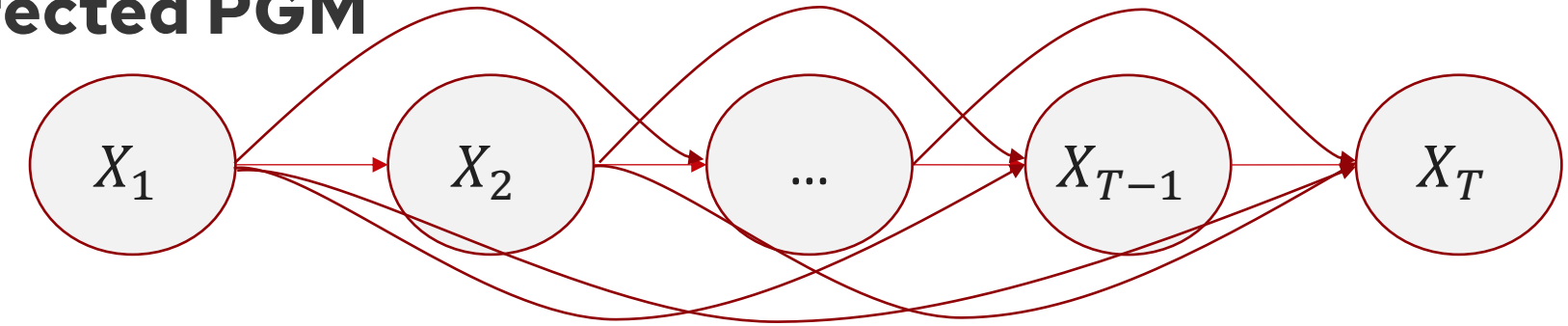
- **Probabilistic objective:** Max log-likelihood of observed seqs

$$\max_\theta \sum_i \sum_t \log P_\theta\left(X_{i,t} \mid X_{i,<t}\right)$$

[Radford et al., Improving Language Understanding by Generative Pre-Training]

GPT = Probabilistic Model + Transformer Decoder

- **Directed PGM**



$$P_\theta(X) = \prod_i \prod_t P_\theta(X_{i,t} \mid X_{i,<t})$$

- **Model structure:**
  - Input: token embeddings + positional encodings
  - Masked multi-head attention: Enforces "causality"
  - Decoder stack: Learns $P(X_t \mid X_{<t})$
  - Output: softmax over vocabulary

[Radford et al., Improving Language Understanding by Generative Pre-Training]

Let’s write a GPT

Let's write a GPT

- EasyGPT repo
  - Adapted from Andrej Karpathy's nanoGPT

From our “GPT” to GPT-4

From our “GPT” to GPT-1

- Architecture:
  - Tokenizer: Characters → “Byte-Pair Encoder” tokenizer
  - Activation: ReLU → GELU
  - Weight sharing for embedding / output
  - Scale (117M params):
    - Layers: 4 → 12
    - Attention Heads: 4 → 12
    - Block size (max context): 32 → 512
    - Vocab: 65 → 40000 BPE tokens
    - Embedding dim: 64 → 768
- Training:
  - Dataset: TinyShakespeare (1MB) → BookCorpus (5GB)
  - Initialization & normalization: Default → Carefully tuned
  - Optimizer: Vanilla Adam → Adam + learning rate warmup + weight decay
- Inference:
  - Sampling: Greedy → Top-k

From GPT-1 to GPT-2

GPT-2: [Radford et al., Language Models are Unsupervised Multitask Learners]

- Architecture:
  - Scale: Variety of options, with biggest (1.5B params):
    - Layers: 12 → 48
    - Attention Heads: 12 → 25
    - Embedding Dim: 768 → 1600
    - Block size (max context): 512 → 1024
    - Vocab: 40k → 50k tokens
- Training:
  - Dataset: BookCorpus (5GB) → WebText (40GB)

You can reproduce GPT-2 yourself:
https://github.com/karpathy/nanoGPT
(takes 4 days to train on an 8xA100 machine)

From GPT-2 to GPT-3

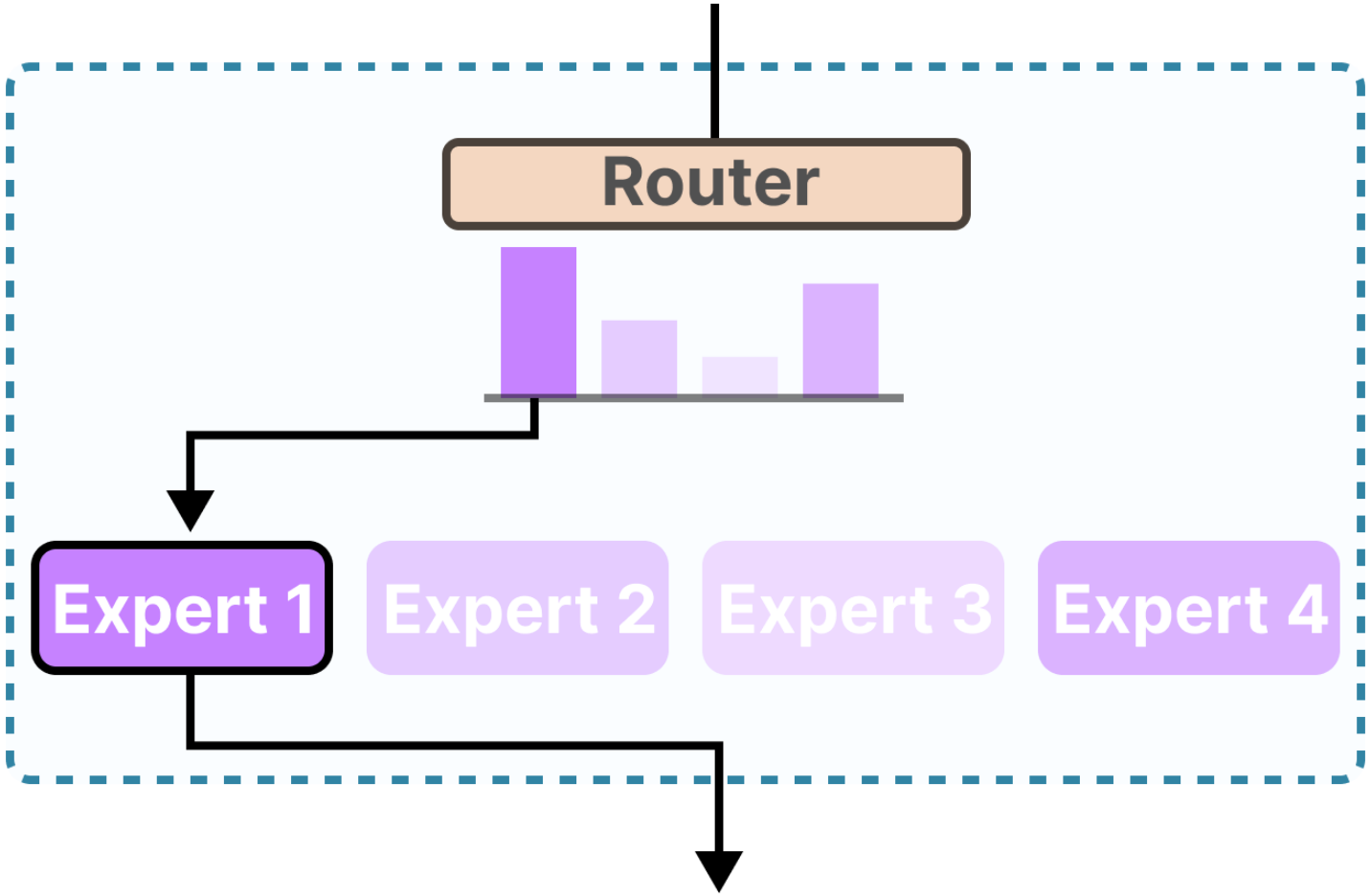
- Architecture:
  - Scale (1.5B → 175B params):
    - Block size (max context): 1024 → 2048
    - Layers: 48 → 96
    - Embedding Dim: 1600 → 12,288
    - Attention Heads: 25 → 96
- Training:
  - Dataset: WebText (40GB) → Common Crawl + books, Wikipedia, code, etc. (~570GB)

GPT-3: [Brown et al., Language Models are Few-Shot Learners]

From GPT-3 to GPT-4?

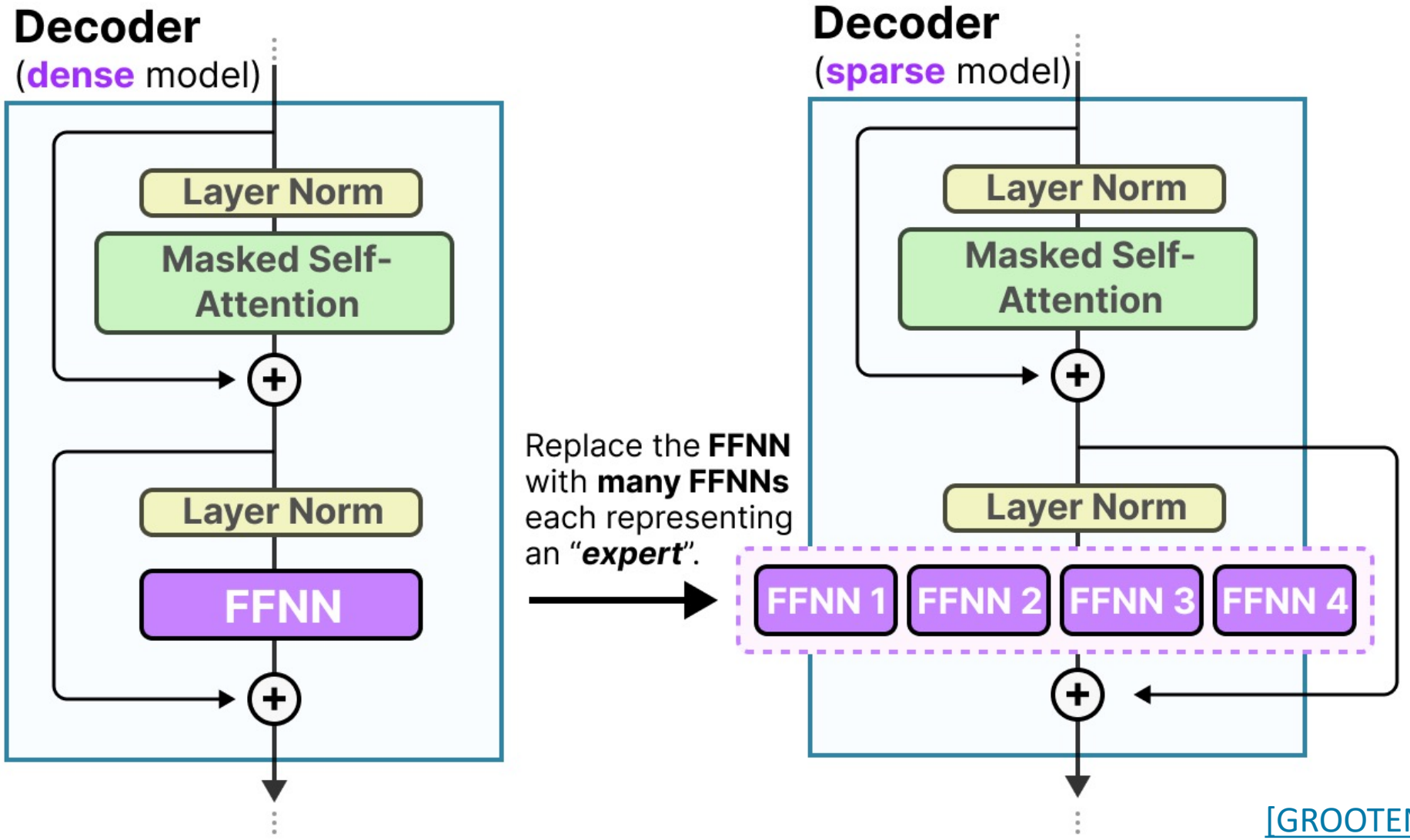
- Architecture:
  - Likely includes MoE
  - Tokenizer: Includes image patches for multi-modal
  - Scale:
    - Total parameters: 175B → Likely >1T
    - Block size (max context): 2048 → 128k
- Training:
  - Dataset: Common Crawl + books, Wikipedia, code, etc. (~570GB) → Larger, undisclosed data training. Reported 13T tokens (~50TB)
  - Alignment: Reinforcement learning + human feedback + system-level "safety"

Mixture of Experts



[GROOTENDORST]

Mixture of Experts inside Transformer Decoder



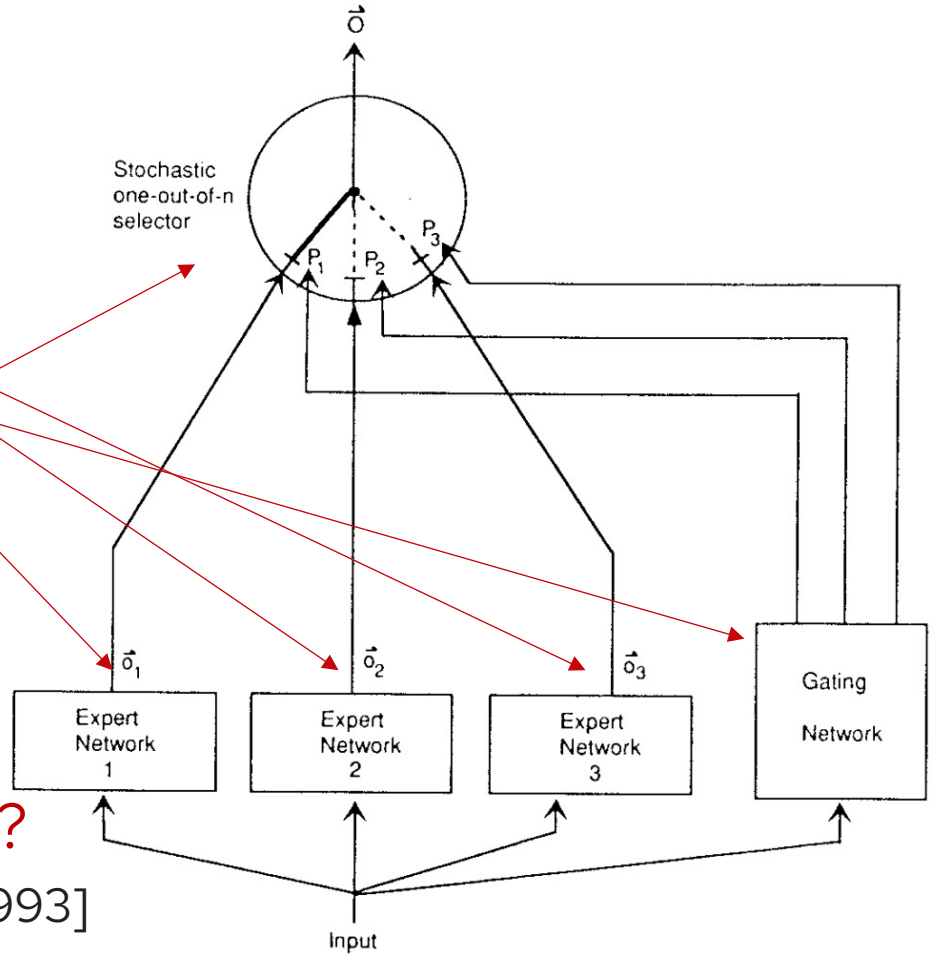
[GROOTENDORST]

Mixture of Experts: A probabilistic idea

- Let

$$P(Y \mid X) = \sum_m g_m(X) \cdot P_m(Y \mid X)$$

- Constrain $\sum_m g_m(X) = 1$ and $g_m(X) \geq 0 \; \forall \; m, \mathrm{X}.$



How would you estimate these parameters?

[Hierarchical mixtures of experts and the EM algorithm, 1993]

"Adaptive Mixtures of Local Experts"
Jacobs et al 1991

Mixture of Experts: Unifies Several Approaches

Let

$$P(Y \mid X) = \sum_m g_m(X) \cdot P_m(Y \mid X)$$

- **Mixture of Experts** [Jacobs et al 1991]: $g_m(X)$ is a learned gating function.
- **Bagging** [Breiman 1996]: $g_m(X) = \frac{1}{M}$ is a constant, uniform weighting.
- **Boosting** [Freund & Schapire 1997]: $g_m(X) = \alpha_m$ is a constant per-expert weight.

Mixture of Experts: Some analysis

Let

$$P(Y \mid X) = \sum_m g_m(X) \cdot P_m(Y \mid X)$$

- Let's examine the mean functions. Define:

$$\bar{f}(x) \coloneqq \mathrm{E}[\, Y \mid X \,] = \sum_m g_m(X) E_m[\, Y \mid X \,] = \sum_m g_m(X)\, f_m(x)$$

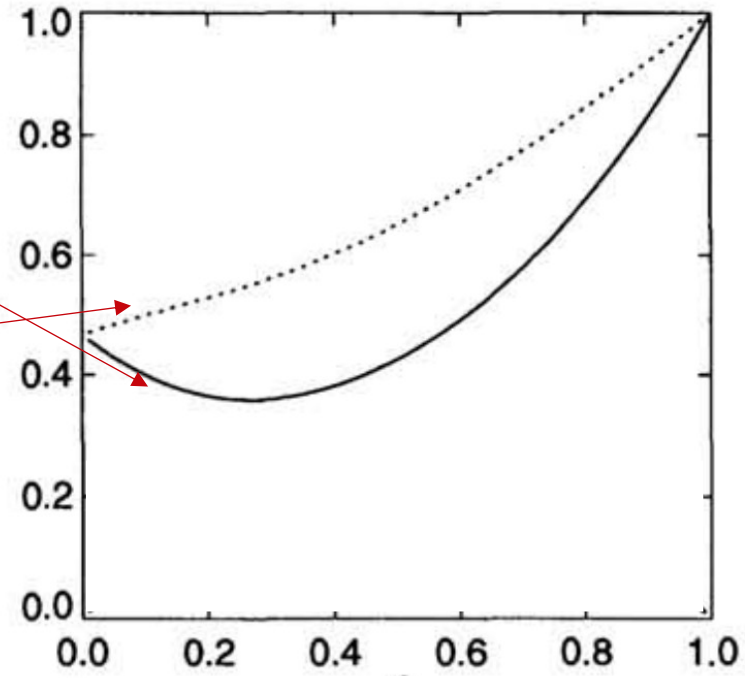
- Let's compare:
  - $\epsilon(x) \coloneqq \left(Y - \bar{f}(x)\right)^2$ ← "Ensemble error"
  - $\bar{\epsilon}(x) \coloneqq \frac{1}{M}\sum_m \left(Y - f_m(x)\right)^2$ ← "Average expert error"

Will these be minimized for the same ensemble?

Mixture of Experts: Some analysis

- $\epsilon(x) := \left(Y - \bar{f}(x)\right)^2$ ← "Ensemble error"
- $\bar{\epsilon}(x) := \frac{1}{M}\sum_m \left(Y - f_m(x)\right)^2$ ← "Average expert error"

Linear f_m



Diversity in Experts
(% Unique Training Data)

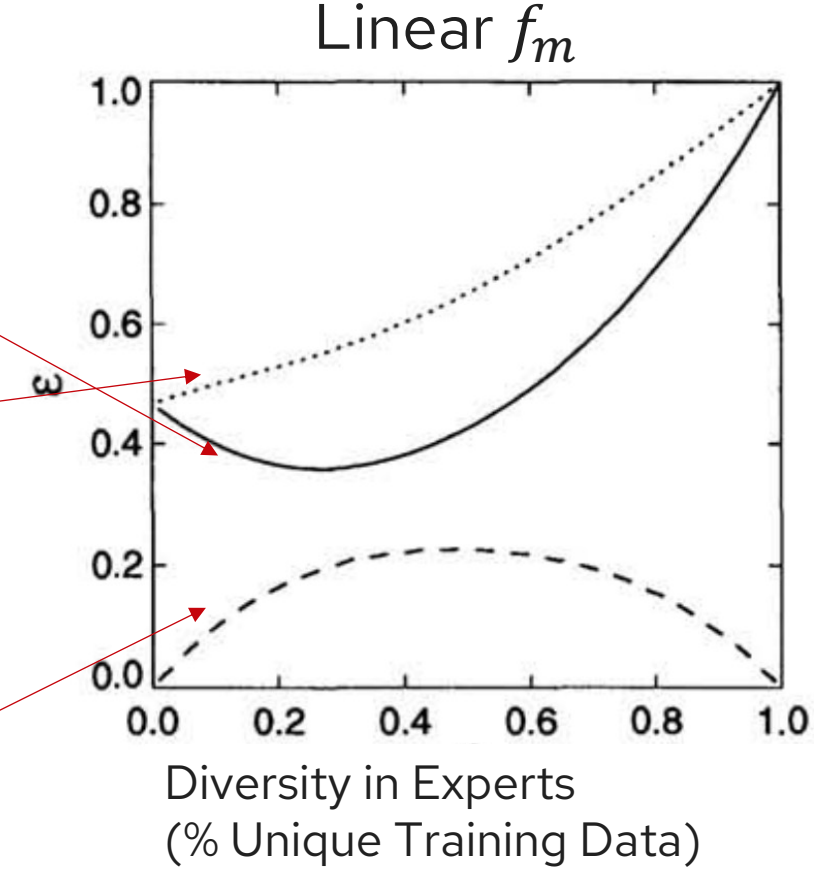
[Sollich & Krogh 1995]

Mixture of Experts: Some analysis

- $\epsilon(x) := \left(Y - \bar{f}(x)\right)^2$ ← "Ensemble error"
- $\bar{\epsilon}(x) := \frac{1}{M}\sum_m \left(Y - f_m(x)\right)^2$ ← "Average expert error"

Intuition: Expert errors are wrong in individualized ways and cancel out through consensus.

→ Slight "overfitting" of experts helps!



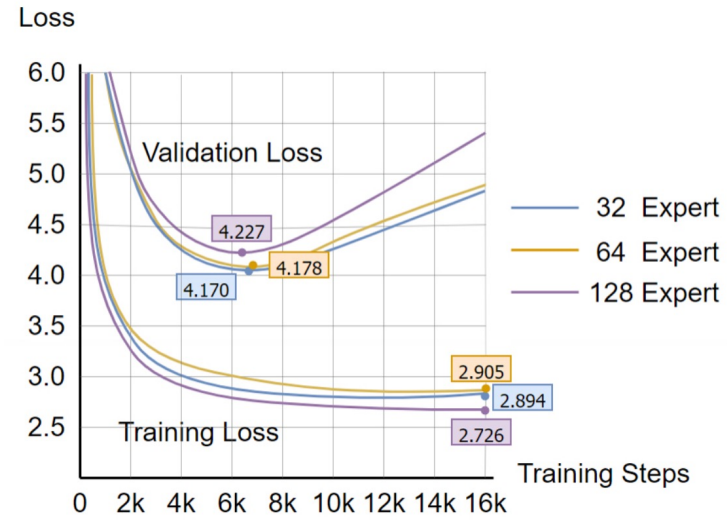
[Sollich & Krogh 1995]

Mixture of Experts: In LLMs

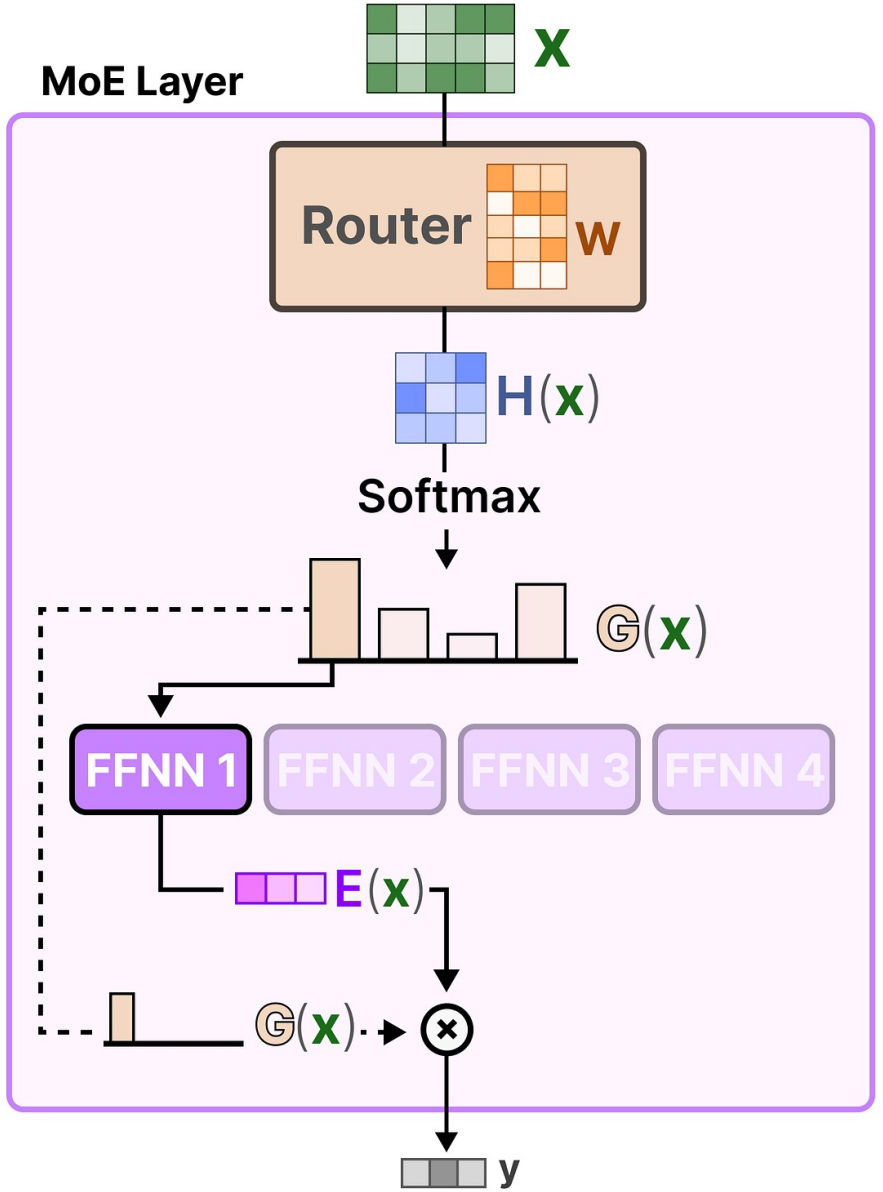
- Implications for serving?

- Implications for training?

Easy to have experts over-specialize



[XIE ET AL 2022]



[GROOTENDORST]

Summary: From Transformer to GPT

Component	Transformer	GPT
Architecture	Encoder-decoder (full)	Decoder-only
Attention	Full self-attention	Masked (causal) self-attention
Positional encoding	Sinusoidal (original)	Learned positional embeddings
Output	Task-specific	Next-token prediction
Training objective	Flexible (e.g., translation)	Language modeling (autoregressive)
Inference	Depends on task	Greedy / sampling for text gen

Summary: From GPT-1 to GPT-4

- **Architecture:**
  - **Scale:** Variety of options, with biggest (1.5B params → >1T params):
    - Block size (max context): 512 → 128k
    - Layers: 12 → >96
    - Attention Heads: 12 → >96
    - Embedding Dim: 768 → >12,288
    - Vocab: 40k → >50k tokens
  - Tokenizer: Includes image patches for multimodal
  - **Mixture-of-Experts**
- **Training:**
  - Dataset: BookCorpus (5GB) → Private 13T tokens (~50TB)
  - Reinforcement learning for alignment

Questions?