Transformer and Large-Language Models

Tianxiang (Adam) Gao

November 4, 2024

Outline

Recap: Seq2Seq Models

- **Word Analogy Task**: Given an analogy "a is to b as c is to ?", find the word d by maximizing cosine similarity.
- **GloVe**: Uses a co-occurrence matrix to capture word meaning based on surrounding context.
- **Bias in Word Embeddings**: Identify bias directions using definitional pairs, then neutralize non-definitional words.
- **Seq2Seq**: Use an RNN Encoder-Decoder architecture to handle tasks where both input and output are sequences.
- **Conditional Language Model**: The output sequence is generated sequentially based on the context vector that summarizes the input sequence
- **Beam Search**: Keeps multiple high-probability sequences to improve output quality.
- **BLEU Score**: A metric using modified precision to assess the accuracy of generated sequences.
- **Distinct Convex Vector**: A distinct context word $\boldsymbol{e}_t$ is used to generate each target word $\hat{\boldsymbol{y}}_t$

$$\mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_1, \cdots, \boldsymbol{y}_{t-1}) = \mathbb{P}_{\boldsymbol{\phi}}(\boldsymbol{y}_t \mid \boldsymbol{s}_t), \quad \text{where} \quad \boldsymbol{s}_t = g_{\boldsymbol{\phi}}(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t)$$

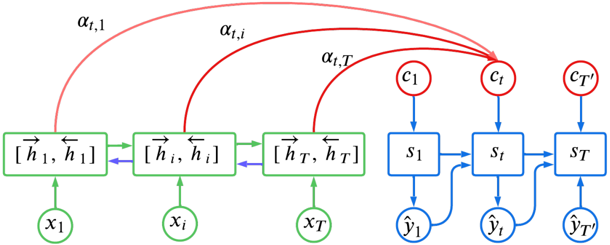
- **Attention Weights**: The distinct context word $\boldsymbol{e}_t$ is a weighted sum of encoder hidden states:

$$\boldsymbol{c}_t = \sum_i \alpha_{t,i} \boldsymbol{h}_i,$$

  where $\alpha_{t,i} = \text{softmax}(\boldsymbol{e}_{t,i})$ are **attention weights**
- **Alignment Scores**: Alignment scores $\boldsymbol{e}_{t,i} = a(\boldsymbol{s}_{\cdot t-1}, \boldsymbol{h}_i)$ indicate relevance between encoder hidden states $\boldsymbol{h}_i$ and decoder states $\boldsymbol{s}_{t-1}$.

Outline

Recall: Attention Mechanism in Seq2Seq



- In Seq2Seq models, the conditional probability of the target word $\boldsymbol{y}_t$ is given by:

$$\mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}) = g_{\phi}(\boldsymbol{s}_t),$$

where the decoder hidden state $\boldsymbol{s}_t$ is updated with the *distinct* context vector $\boldsymbol{c}_t$ as follows:

$$\boldsymbol{s}_t = \mathsf{GRU}(\boldsymbol{s}_{t-1}, [\boldsymbol{y}_{t-1}; \boldsymbol{c}_t]), \quad \text{with} \quad \boldsymbol{c}_t = \sum_{i=1}^{T_x} \alpha_{t,i} \boldsymbol{h}_i,$$

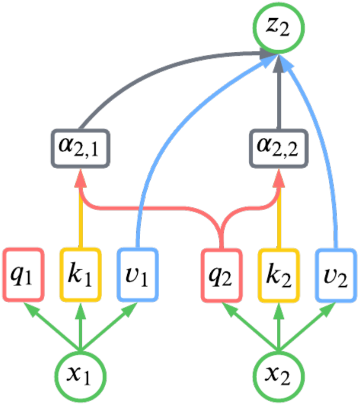
where $\alpha_t \in \mathbb{R}^{T_x}$ are the attention weights based on alignment scores $\boldsymbol{e}_t \in \mathbb{R}^{T_x}$:

$$\alpha_t = \mathsf{softmax}(\boldsymbol{e}_t), \quad \text{with} \quad \boldsymbol{e}_{t,i} = \boldsymbol{a}^\top \tanh(\boldsymbol{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i]).$$

Bahdanau et al., "Neural Machine Translation by Jointly Learning to Align and Translate," ICLR 2015

Self-Attention

Define: Self-attention creates a contextually enriched representation of each token by learning its relevance to all other tokens in the sequence.



- For each token, three vectors are computed:

$$\boldsymbol{q}_t = \boldsymbol{W}^q \boldsymbol{x}_t, \quad \boldsymbol{k}_t = \boldsymbol{W}^k \boldsymbol{x}_t, \quad \boldsymbol{v}_t = \boldsymbol{W}^v \boldsymbol{x}_t$$

where the query $\boldsymbol{q}_t$ interacts with keys $\boldsymbol{k}_i$ to measure relevance:

$$\boldsymbol{\alpha}_{t,i} \propto \boldsymbol{q}_t^\top \boldsymbol{k}_i, \quad \Rightarrow \quad \boldsymbol{\alpha}_t = \text{softmax}\left(\frac{\boldsymbol{K}\boldsymbol{q}_t}{\sqrt{d_k}}\right),$$

where

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1 & \cdots & \boldsymbol{k}_T \end{bmatrix}^\top$$

- The new representation $\boldsymbol{z}_t$ is a weighted sum of value vectors $\boldsymbol{v}_i$:

$$\boldsymbol{z}_t = \sum_{i=1}^{T} \boldsymbol{\alpha}_{t,i} \boldsymbol{v}_i$$

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Self-Attention: Matrix Form

- Define matrices:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{q\top}, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{k\top}, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{v\top}$$

  where

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_T \end{bmatrix}^\top, \quad \boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1 & \cdots & \boldsymbol{k}_T \end{bmatrix}^\top, \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_T \end{bmatrix}^\top$$

- The attention weights are computed as:

$$\begin{bmatrix} \boldsymbol{\alpha}_1 & \cdots & \boldsymbol{\alpha}_T \end{bmatrix} = \text{softmax}\left(\frac{\boldsymbol{K}\boldsymbol{Q}^\top}{\sqrt{d_k}}\right)$$

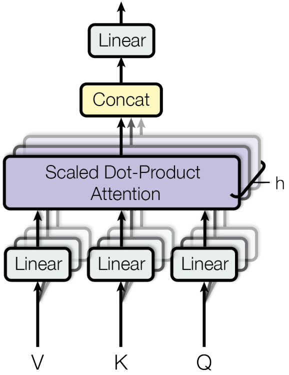
- The new representation $\boldsymbol{Z}$ is then:

$$\boldsymbol{Z} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}$$

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Multi-Head Attention

Definition: Multi-head attention extends self-attention by allowing multiple heads to focus on **different aspects** of each token, capturing diverse patterns and dependencies across the sequence.



- Each head produces an independent attention output $\boldsymbol{Z}_h$:

$$\boldsymbol{Z}_h = \text{Attention}(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h),$$

  where $\boldsymbol{Q}_h = \boldsymbol{Q}\boldsymbol{W}_h^q$, $\boldsymbol{K}_h = \boldsymbol{K}\boldsymbol{W}_h^k$, and $\boldsymbol{V}_h = \boldsymbol{V}\boldsymbol{W}_h^v$.
- Head outputs are concatenated and linearly transformed to form the final representation:

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{Z}_1 & \cdots & \boldsymbol{Z}_H \end{bmatrix} \boldsymbol{W}_o^\top$$

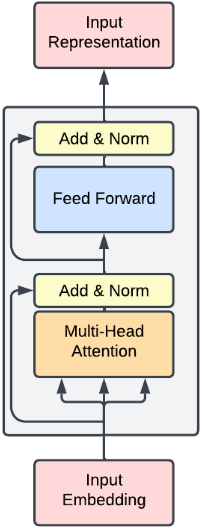
  where $\boldsymbol{W}_o$ is the output projection and H is the number of heads.
- **In Matrix Form**: With $\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h$ for each head,

$$\boldsymbol{Z} = \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \in \mathbb{R}^{T \times d_{\text{model}}}$$

Note: The multi-head can be computed in **parallel**, each with complexity $\mathcal{O}(T^2 d)$.

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Multi-Head Attention Layer: LayerNorm and FNN



Layer Normalization computes statistics across different **hidden units**:

- In an RNN or MLP, a hidden state update is given by:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}, \quad \boldsymbol{h} = \tanh(\boldsymbol{z})$$

  where the pre-activation vector $\boldsymbol{z} \in \mathbb{R}^m$.

- The statistics are computed across the **hidden units**:

$$\boldsymbol{z}_i = \boldsymbol{w}_i^\top \boldsymbol{x}, \quad \mu = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{z}_i, \quad \sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{z}_i - \mu)^2}$$

  where $\boldsymbol{w}_i$ is the ith row of $\boldsymbol{W}$.

- Re-scale and shift the normalized pre-activation:

$$\boldsymbol{z}_{\text{norm}} = \frac{\boldsymbol{z} - \mu}{\sigma}, \quad \tilde{\boldsymbol{z}} = \boldsymbol{\alpha} \odot \boldsymbol{z}_{\text{norm}} + \boldsymbol{\beta}$$

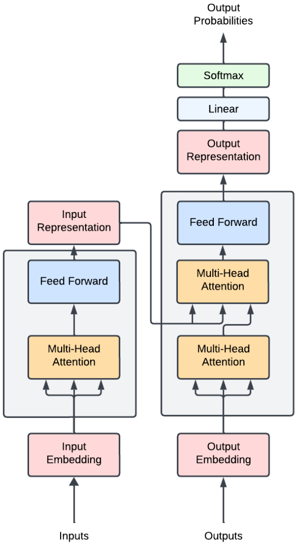
  where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are trainable

Feed-Forward Layer captures non-linear relationships between tokens:

$$\text{FFN}(\boldsymbol{x}_t) = \text{ReLU}(\boldsymbol{x}_t \boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

Ba et al. "Layer normalization." NeurIPS 2016 in Deep Learning Symposium

Multi-Head Attention Encoder and Decoder Stacks



Encoder:

- **Input Embedding**: Converts input to dense word embeddings.
- **Multi-Head Attention**: Enhances token representations by attending to various parts of the sequence.
- **FFN**: Applies non-linear transformations to capture complex relationships between words.

Decoder:

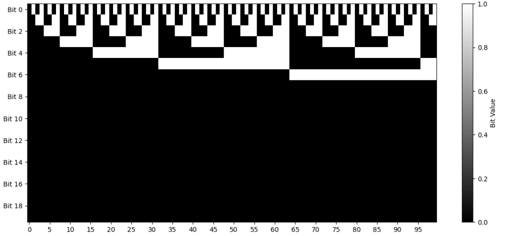
- **Output Embedding**: Converts the previously generated output (one-hot encoded) into dense word embeddings.
- **First Multi-Head Attention**: Refines the output embeddings or hidden states using self-attention.
- **Second Multi-Head Attention**: Uses the output of the first attention as the query Q, with K and V from the encoder's output, allowing the decoder to attend to the input sequence.
- **Final Output**: Computes the conditional probability of the next token through a linear layer and softmax.

Vaswani et al., "Attention is All You Need," NeurIPS 2017

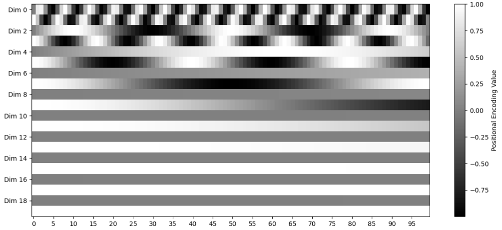
Positional Encoding

- Unlike RNNs, transformers do **not** inherently process tokens in sequence order.
- Positional Encoding is defined as:

$$\mathrm{PE}_{\mathrm{pos},2i} = \sin\left(\mathrm{pos}/10000^{\frac{2i}{d_{\mathrm{model}}}}\right), \quad \text{and} \quad \mathrm{PE}_{\mathrm{pos},2i+1} = \cos\left(\mathrm{pos}/10000^{\frac{2i}{d_{\mathrm{model}}}}\right)$$



Binary Representation



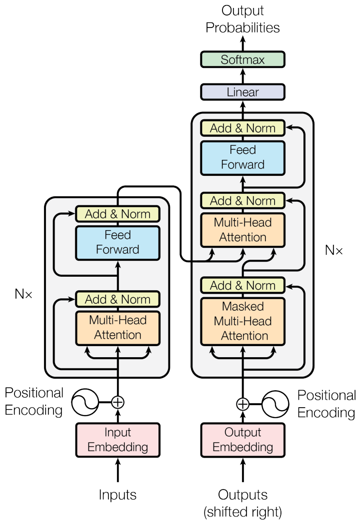
Position Encoding

Key Properties

- Provides unique and consistent representation for each position.
- Represents positions in a low-dimensional subspace.
- Enables linear transformations for relative positioning, *i.e.*, $\exists$ a linear M_k s.t. $M_k \mathrm{PE}_{\mathrm{pos}+k} = \mathrm{PE}_{\mathrm{pos}}$.

Amirhossein Kazemnejad's Blog: Positional Encoding in Transformers

Transformer



Training Process: Teacher Forcing

- During training, the true output sequence is fed into the decoder.
- Masking ensures only past and current tokens are visible, preserving autoregressive properties.
- Cross-entropy loss is used to compare the predicted probability distribution with the true token.

Loss Function: Cross-Entropy

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} y_t \log \mathbb{P}(\hat{y}_t)$$

- y_t: True one-hot encoded token.
- $\mathbb{P}(\hat{y}_t)$: Predicted probability for the token at time t.

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Summary

- **Self-attention** refines the representation of each token by learning its relevance to other tokens using **query-key** pairs.
- **Multi-head self-attention** captures **different aspects** of each token, enhancing the overall representation.
- **Layer normalization** computes statistics **across hidden units** to stabilize information propagation.
- **Positional encoding** adds **order** information to word embeddings, enabling the model to learn relative positioning through a linear transformation.
- **Encoder-decoder attention** refines the output representation by attending to the input sequence representation.
- The **Transformer** uses **teacher forcing** with cross-entropy loss to facilitate effective training.

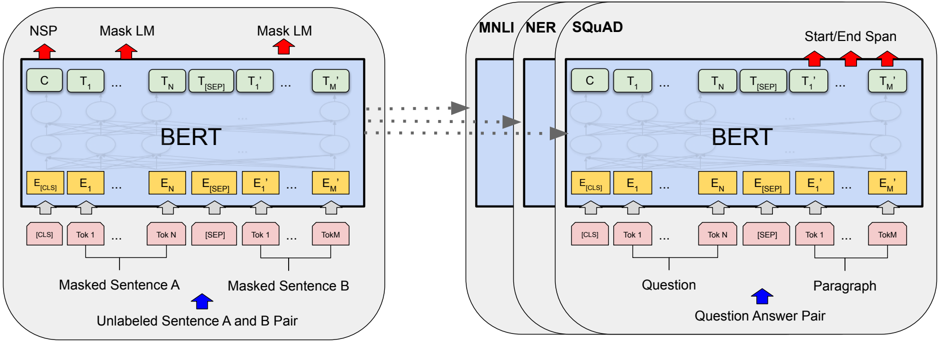
Outline

BERT

BERT: Encoder-Only

Definition: BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.

- Utilizes only the **encoder** part of the Transformer architecture.
- Designed for pre-training on large corpora and fine-tuning on downstream NLP tasks.



Pre-training　　　　Fine-Tuning

Model Details: BERT_Large, with 340 million parameters, was trained on TPU v3 pods over 4 days using the BooksCorpus (800 million words) and English Wikipedia (2.5 billion words) datasets.

Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL 2019.

BERT: Pre-training

Masked Language Modeling (MLM): Randomly masks 15% of tokens in the input sequence and predicts masked tokens based on context.

- **Problem**: `[MASK]` token never used in finite-tuning
- **Solution**: Do not always replace selected words with `[MASK]`, e.g., `my dog is hairy`
  - 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is [MASK]`
  - 10% of the time: Replace the word with a random word, e.g., `my dog is apple`
  - 10% of the time: Keep the word unchanged, e.g., `my dog is hairy`.

Next Sentence Prediction (NSP): Predicts if the second sentence follows the first, using the hidden state of the **[CLS]** token for binary classification (*IsNext* or *NotNext*).

- Input=`[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]`, Label=`IsNext`
- Input=`[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]`, Label=`NotNext`

Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL 2019.

BERT: Performance on SQuAD 1.1

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

GPT

OpenAI
ChatGPT

GPT-1: Decoder-Only Generative Pretrained Transformer

Unsupervised Pre-training: Given a sequence of tokens $\{\boldsymbol{x}_t\} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$:

- Maximize the likelihood in the standard language model:

$$\mathcal{L}_1 = \sum_t \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-k}, \cdots, \boldsymbol{x}_{t-1})$$

- Transformer decoder structure:

$$\boldsymbol{h}^{(0)} = \boldsymbol{x}\boldsymbol{W}_e + \boldsymbol{W}_p, \quad \boldsymbol{h}^{(\ell)} = \text{transformer\_layer}(\boldsymbol{h}^{(\ell-1)}), \ \forall \ell \in [L], \quad \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}_t) = \text{softmax}(\boldsymbol{h}^L \boldsymbol{W}_e^\top),$$

  where $\boldsymbol{x} = (\boldsymbol{x}_{-k}, \cdots, \boldsymbol{x}_{-1})$ is the context vector, L is the number of layers, $\boldsymbol{W}_e$ is the embedding matrix, and $\boldsymbol{W}_p$ is the positional embedding matrix.

Supervised Fine-tuning: Given a label $\boldsymbol{y}$:

- Pass the inputs $\{\boldsymbol{x}_t\}$ through the pre-trained model to obtain $\boldsymbol{h}^{(L)}$, which is used to predict $\boldsymbol{y}$:

$$\mathbb{P}_{\boldsymbol{\phi}}(\boldsymbol{y} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) = \text{softmax}(\boldsymbol{h}_t^{(L)} \boldsymbol{W}_y)$$

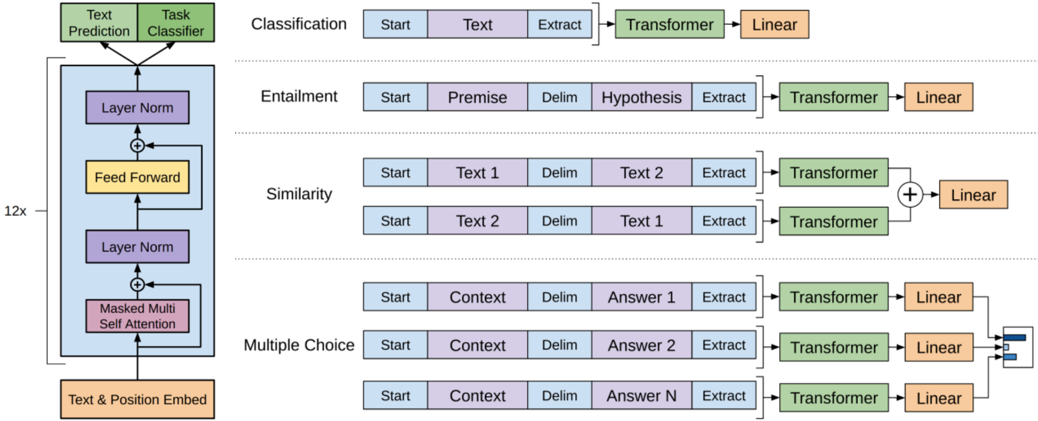
- Maximize the supervised likelihood:

$$\mathcal{L}_2 = \sum_{(\boldsymbol{x}, \boldsymbol{y})} \log \mathbb{P}_{\boldsymbol{\phi}}(\boldsymbol{y} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t)$$

- Improved performance is achieved by fine-tuning with a combined objective: $\mathcal{L}_3 = \mathcal{L}_1 + \lambda \mathcal{L}_2$

Radford et al., "Improving Language Understanding by Generative Pre-Training," arXiv 2018.

GPT-1: Task-Specific Input Transformations



Model Details: GPT-1, with 117 million parameters, are trained on the BooksCorpus dataset (800 million words).

Radford et al., "Improving Language Understanding by Generative Pre-Training," arXiv 2018.

GPT-2: WebText Dataset

WebText Dataset:

- Curated using Reddit as a **filter** to include diverse, high-quality content.
- Comprised of approximately 8 million documents and 40GB of text.

Training:

- Trained as a **standard language model** on WebText.
- Objective: Maximize the likelihood of predicting the next token in a sequence.

Zero-Shot Inference:

- Inputs during inference are formatted as **prompts** that specify tasks.
- Generates responses based on pre-trained language understanding, without fine-tuning.
- Example: **Input**: "Q: What is the capital of France? A:" and **Response**: "Paris"

Model Details:

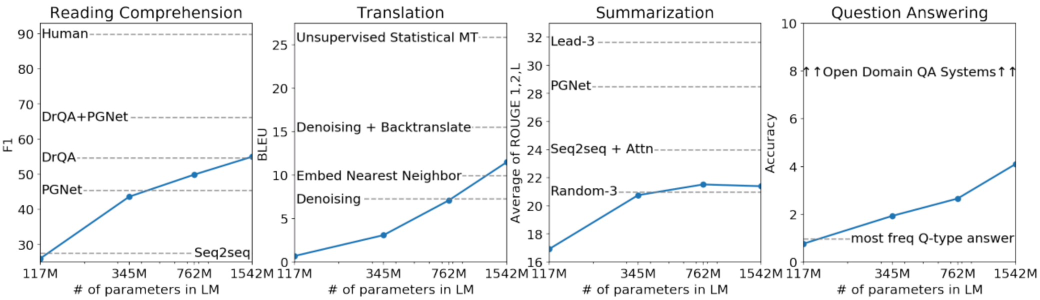
- 12-layer Transformer decoder with 1.5 billion parameters.
- Hidden size of 1600, with 25 attention heads.

Radford et al., "GPT-2: Language Models are Unsupervised Multitask Learners," arXiv 2019.

GPT-2: Zero-Shot Inference

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

Model configurations



Zero-shot performance of WebText GPT-2 on many NLP tasks.

Radford et al., "GPT-2: Language Models are Unsupervised Multitask Learners," arXiv 2019.

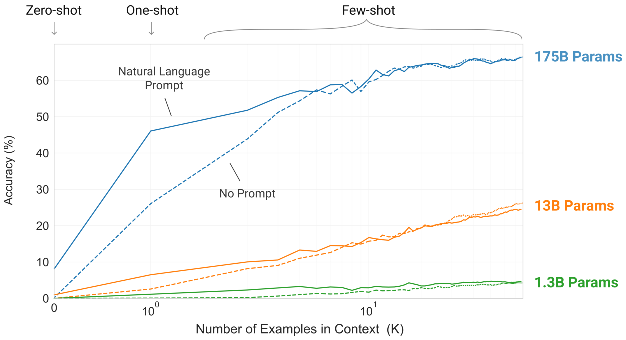
GPT-3: Limitation of Pretraining and Finite-Tune

Limitation of Pretrained and Finite-Tune in NLP:

- **Large Data Limitation**: Language models need extensive **labeled** datasets for new tasks, limiting their practical use.
- **Overfitting and Generalization Issues**: Powerful models risk overfitting, especially when fine-tuned on small data with limited diversity, leading to poor real-world generalization.
- **Human Adaptability**: Humans learn language tasks with minimal examples and switch tasks seamlessly; achieving this in NLP systems would improve their versatility.

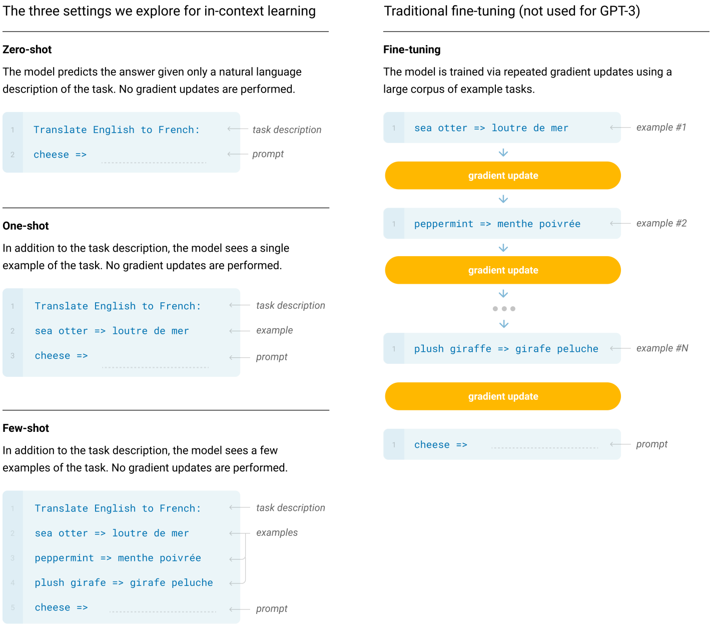
Solution: In-context learning and Transformer

- **In-Context Learning**: the model is conditioned on a natural language instruction and a few **demonstrations** of the task and is then expected to complete further instances of the task simply by predicting what comes next.



Brown, Tom B., et al. "Language Models are Few-Shot Learners." NeurIPS 2020.

GPT-3: In-Context Learning

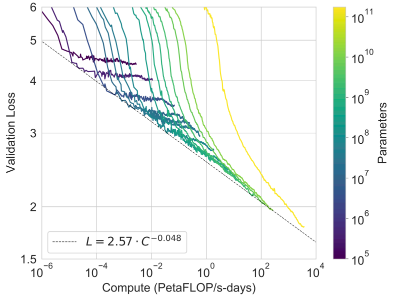


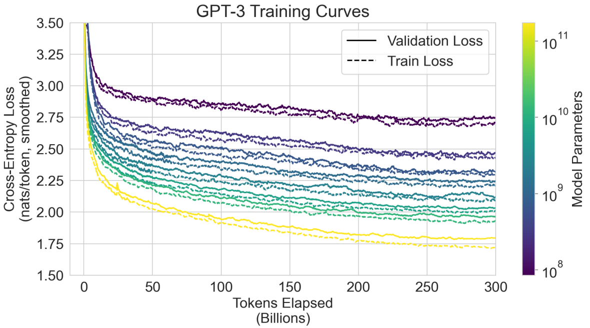
GPT-3: Training

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

GPT-3: Neural Scaling Laws





Kaplan, et al. "Scaling laws for neural language models." arXiv 2020.