Language Model Pretraining & Fine-Tuning

Yu Meng
University of Virginia
yumeng5@virginia.edu

Jan 22, 2025

(Recap) Course Format & Grading: Paper Presentation (30%)

- Starting from **next Monday (1/27)**, each lecture will be presented by a group of 1 or 2 students
- Every group presents one lecture (3 papers)
- Deadline: email your slides to the instructor & TA **48 hours before the lecture** (If presenting next Monday, you'll need to submit your slides by this Saturday 2pm)

(Recap) Course Format & Grading: Participation (20%)

- Starting from **next Monday (1/27)**, everyone is required to complete two mini-assignments
- **Pre-lecture question**: read the 3 papers to be introduced in the lecture, and submit a question you have when you read them
- **Post-lecture feedback**: provide feedback to the presenters after the lecture
- We'll release the Google Forms later this week (Canvas announcement)
- **Deadlines**: pre-lecture questions are due one day before the lecture (e.g., For Monday lectures, you need to submit the question by Sunday 11:59 pm); post-lecture feedback is due each Friday (both Monday & Wednesday feedback is due Friday 11:59 pm)
- The Google Forms will be closed once the deadline is passed!

(Recap) Course Format & Grading: Project (50%)

- Complete a research project, present your results, and submit a project report
- Work in a team of 1 or 2 (a larger team size requires prior approval from the instructor) – may or may not be the same team as your presentation group
- (Type 1) A comprehensive survey report: carefully examine and summarize existing literature on a topic covered in this course; provide detailed and insightful discussions on the unresolved issues, challenges, and potential future opportunities within the chosen topic
- (Type 2) A hands-on project: not constrained to the course topics but must be centered around NLP; doesn’t have to involve large language models (e.g., train or analyze smaller-scale language models for specific tasks); eligible for extra credits if publishable
- **Project proposal**: 5% (ddl: 2/5); **Mid-term report**: 10% (ddl: 3/10); **Final presentation** (ddl: 4/15) **and final report**: 35% (ddl: 5/6)

(Recap) Overview of Course Contents

- Introduction to Language Models
  - Language Model Architecture
  - Language Model Pretraining & Fine-Tuning
  - In-Context Learning
  - Scaling and Emergent Ability
- Reasoning with Language Models
  - Chain-of-Thought Generation
  - Inference-Time Scaling
- Knowledge, Factuality and Efficiency
  - Parametric Knowledge in Language Models
  - Retrieval-Augmented Language Generation (RAG)
  - Long-Context Language Models
  - Efficiency
- Language Model Post-Training
  - Instruction Tuning
  - Reinforcement Learning from Human Feedback (RLHF)
- Language Agents
  - Language Agent Basics
  - Language Models for Code
  - Multimodal Language Models
- Ethical Considerations of Language Models
  - Security and Jailbreaking
  - Bias and Calibration
  - Privacy and Legal Issues
- Looking Forward

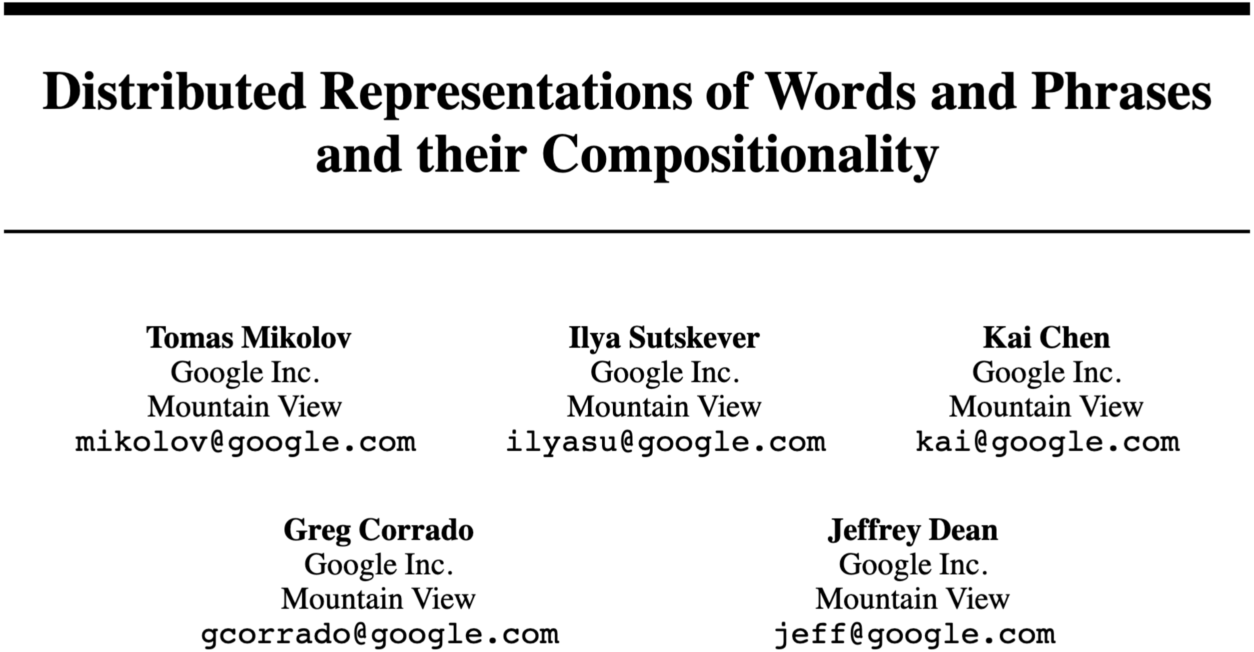
(Recap) Vector Semantics

- Represent a word as a point in a multi-dimensional semantic space
- A desirable vector semantic space: words with similar meanings are nearby in space

2D visualization of a desirable high-dimensional vector semantic space

Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

(Recap) Word2Vec Paper



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
`mikolov@google.com`

Ilya Sutskever
Google Inc.
Mountain View
`ilyasu@google.com`

Kai Chen
Google Inc.
Mountain View
`kai@google.com`

Greg Corrado
Google Inc.
Mountain View
`gcorrado@google.com`

Jeffrey Dean
Google Inc.
Mountain View
`jeff@google.com`

Paper: https://arxiv.org/pdf/1310.4546

(Recap) Distributional Hypothesis

- Words that occur in similar contexts tend to have similar meanings
- A word’s meaning is largely defined by the company it keeps (its context)
- Example: suppose we don’t know the meaning of “Ong choy” but see the following:
  - Ong choy is delicious **sautéed with garlic**
  - Ong choy is superb **over rice**
  - … ong choy **leaves** with **salty** sauces
- And we’ve seen the following contexts:
  - … spinach **sautéed with garlic over rice**
  - … chard stems and **leaves** are **delicious**
  - … collard greens and other **salty** leafy greens
- Ong choy = water spinach!

Example source: https://web.stanford.edu/~jurafsky/slp3/slides/vectorsemantics2024.pdf

(Recap) Learning Word Embeddings

- Assume a large text collection (e.g., Wikipedia)
- Hope to learn similar word embeddings for words occurring in similar contexts
- Construct a prediction task: use a center word’s embedding to predict its contexts!
- Intuition: If two words have similar embeddings, they will predict similar contexts, thus being semantically similar!

(Recap) Word2Vec Parameterized Objective

- Word2Vec objective: $$\max_{\boldsymbol{\theta}} \prod_{(w,c)\in\mathcal{D}} p_{\boldsymbol{\theta}}(c|w)$$
- Assume the log probability (i.e., logit) is proportional to vector dot product $$\log p_{\boldsymbol{\theta}}(c|w) \propto \boldsymbol{v}_c \cdot \boldsymbol{v}_w$$
- The final probability distribution is given by the softmax function: $$p_{\boldsymbol{\theta}}(c|w) = \frac{\exp(\boldsymbol{v}_c \cdot \boldsymbol{v}_w)}{\sum_{c'\in|\mathcal{V}|} \exp(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w)} \quad \Rightarrow \quad \sum_{c'\in|\mathcal{V}|} p_{\boldsymbol{\theta}}(c'|w) = 1$$
- Word2Vec objective (log-scale): $$\max_{\boldsymbol{\theta}} \sum_{(w,c)\in\mathcal{D}} \log p_{\boldsymbol{\theta}}(c|w) = \sum_{(w,c)\in\mathcal{D}} \left(\boldsymbol{v}_c \cdot \boldsymbol{v}_w - \log \sum_{c'\in|\mathcal{V}|} \exp(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w) \right)$$

(Recap) Summary: Word2Vec

- Distributional hypothesis
    - Words that occur in similar contexts tend to have similar meanings
    - Infer semantic similarity based on context similarity
- Word embeddings
    - Construct a prediction task: use a center word’s embedding to predict its contexts
    - Two words with similar embeddings will predict similar contexts => semantically similar
    - Word embedding is a form of self-supervised learningEmploy negative sampling to improve training efficiency
- Use SGD to optimize vector representations
- Word embedding applications & evaluations
    - Word similarity
    - Word analogy
    - Use as input features to downstream tasks

(Recap) Limitations: Word2Vec

- Limited Context Window:
    - only considers a fixed-size context window when generating embeddings
    - cannot effectively capture long-range dependencies (e.g. words that appear far apart)
- Static Embeddings:
    - the embeddings generated by Word2Vec are static (regardless of the context)
    - polysemy can have different meanings depending on specific context
- Not Capturing Word Order Information:
    - focuses only on co-occurrence within the context window
    - ignores the sequential structure of language

(Recap) Transformer Paper

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

Paper: https://arxiv.org/pdf/1706.03762

(Recap) Transformer Layer

Each Transformer layer contains the following important components:

- Self-attention
- Feedforward network
- Residual connections + layer norm

Transformer layer

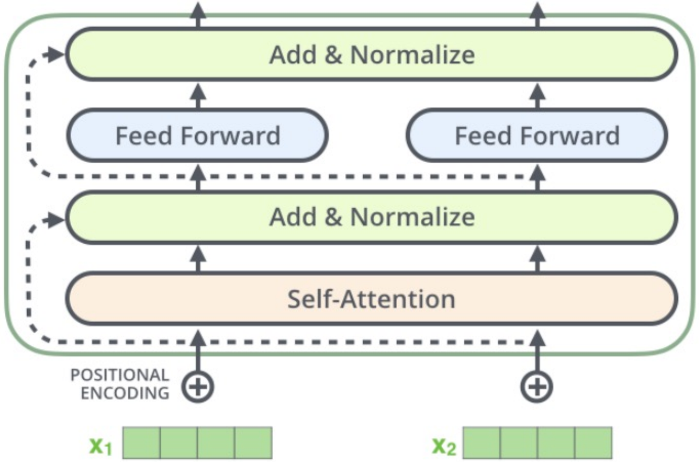


Figure source: https://jalammar.github.io/illustrated-transformer/

(Recap) Self-Attention: Intuition

- Attention: weigh the importance of different words in a sequence when processing a specific word
    - "When I'm looking at this word, which other words should I pay attention to in order to understand it better?"
- **Self-attention**: each word attends to other words in the **same** sequence
- Example: "The chicken didn't cross the road because it was too tired"
    - Suppose we are learning attention for the word "**it**"
    - With self-attention, "**it**" can decide which other words in the sentence it should focus on to better understand its meaning
    - Might assign high attention to "**chicken**" (the subject) & "**road**" (another noun)
    - Might assign less attention to words like "the" or "didn't"

Self-Attention: Example

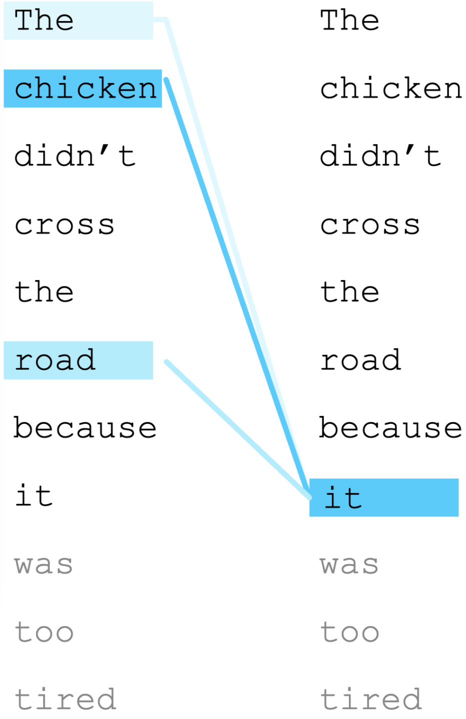
Derive the center word representation as a weighted sum of context representations!

Center word representation

Context word representation

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{x}_j, \quad \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} = 1$$

Attention score $i \rightarrow j$, summed to 1



Context word (key)	Center word (query)
The	The
chicken	chicken
didn’t	didn’t
cross	cross
the	the
road	road
because	because
it	it
was	was
too	too
tired	tired

Current word = “it”

Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Self-Attention: Attention Score Computation

- Attention score is given by the softmax function over vector dot product

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{x}_j, \quad \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} = 1$$

$$\alpha_{ij} = \mathrm{Softmax}(\boldsymbol{x}_i \cdot \boldsymbol{x}_j)$$

Center word (query) representation

Context word (key) representation

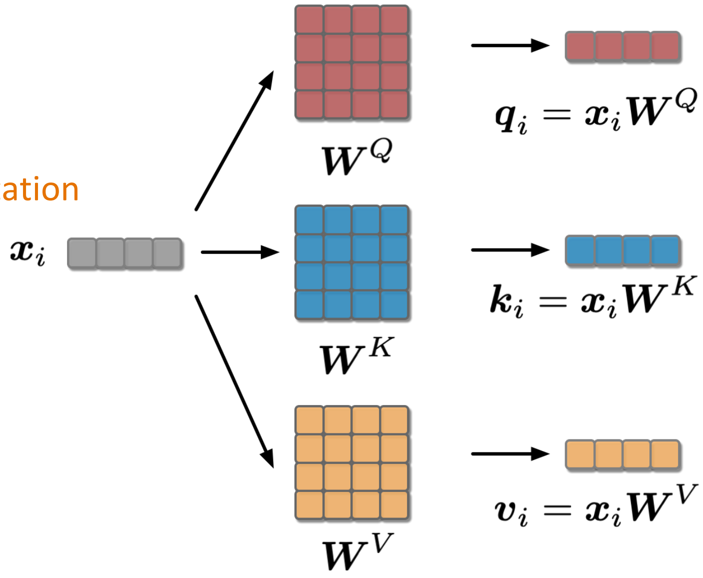
- Why use two copies of word representations for attention computation?
  - We want to reflect the different roles a word plays (as the target word being compared to others, or as the context word being compared to the target word)
  - If using the same copy of representations for attention calculation, a word will (almost) always attend to itself heavily due to high dot product with itself!

Self-Attention: Query, Key, and Value

- Each word in self-attention is represented by three different vectors
    - Allow the model to flexibly capture different types of relationships between tokens
- **Query (Q)**:
    - Represent the current word seeking information about
- **Key (K)**:
    - Represent the reference (context) against which the query is compared
- **Value (V)**:
    - Represent the actual content associated with each token to be aggregated as final output

Self-Attention: Query, Key, and Value

Each self-attention module has three weight matrices applied to the input word vector to obtain the three copies of representations



Input word representation

$$\boldsymbol{q}_i = \boldsymbol{x}_i \boldsymbol{W}^Q$$

query representation

$$\boldsymbol{k}_i = \boldsymbol{x}_i \boldsymbol{W}^K$$

key representation

$$\boldsymbol{v}_i = \boldsymbol{x}_i \boldsymbol{W}^V$$

value representation

Self-Attention: Overall Computation

- Input: single word vector of each word $\boldsymbol{x}_i$
- Compute Q, K, V representations for each word:

$$\boldsymbol{q}_i = \boldsymbol{x}_i \boldsymbol{W}^Q \quad \boldsymbol{k}_i = \boldsymbol{x}_i \boldsymbol{W}^K \quad \boldsymbol{v}_i = \boldsymbol{x}_i \boldsymbol{W}^V$$

- Compute attention scores with Q and K
  - The dot product of two vectors usually has an expected magnitude proportional to $\sqrt{d}$
  - Divide the attention score by $\sqrt{d}$ to avoid extremely large values in softmax function

$$\alpha_{ij} = \mathrm{Softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right)$$

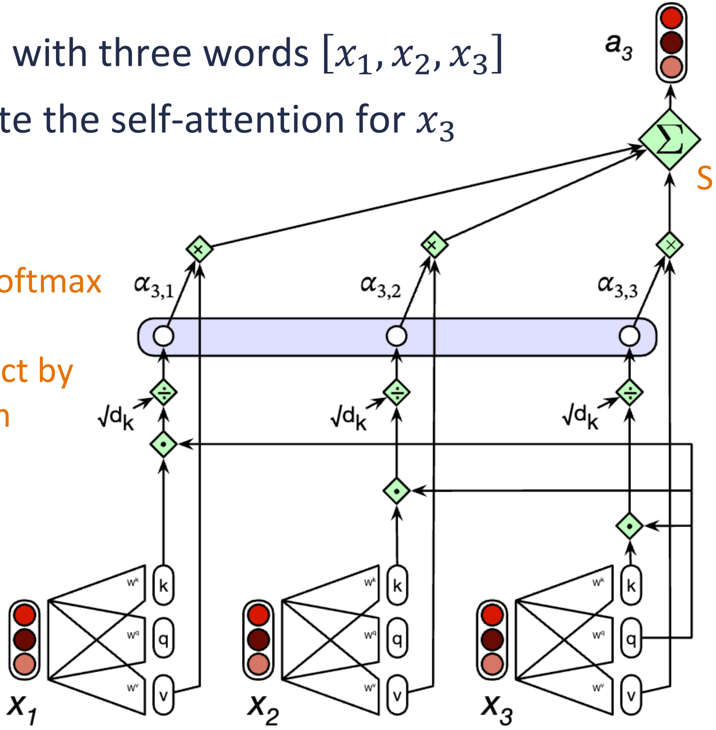
Dimensionality of q and k

- Sum the value vectors weighted by attention scores

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{v}_j$$

Self-Attention: Illustration

- Example: an input sequence with three words $[x_1, x_2, x_3]$
- Suppose we want to compute the self-attention for x_3



Output of self-attention

Sum the weighted value vectors

Obtain attention scores via softmax

Divide the dot product by vector dimension

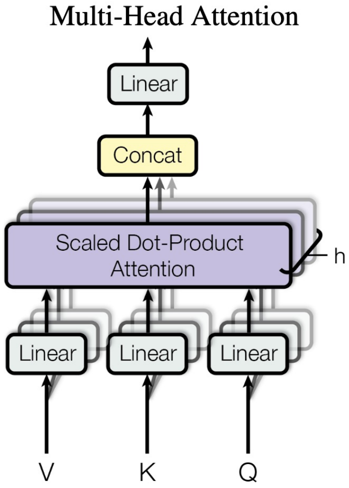
Compare x_3's query with the keys of all words

Compute query, key, value

Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Multi-Head Self-Attention

- Transformers use multiple attention heads for each self-attention module
- Intuition:
  - Each head might attend to the context for different purposes (e.g., particular kinds of patterns in the context)
  - Heads might be specialized to represent different linguistic relationships



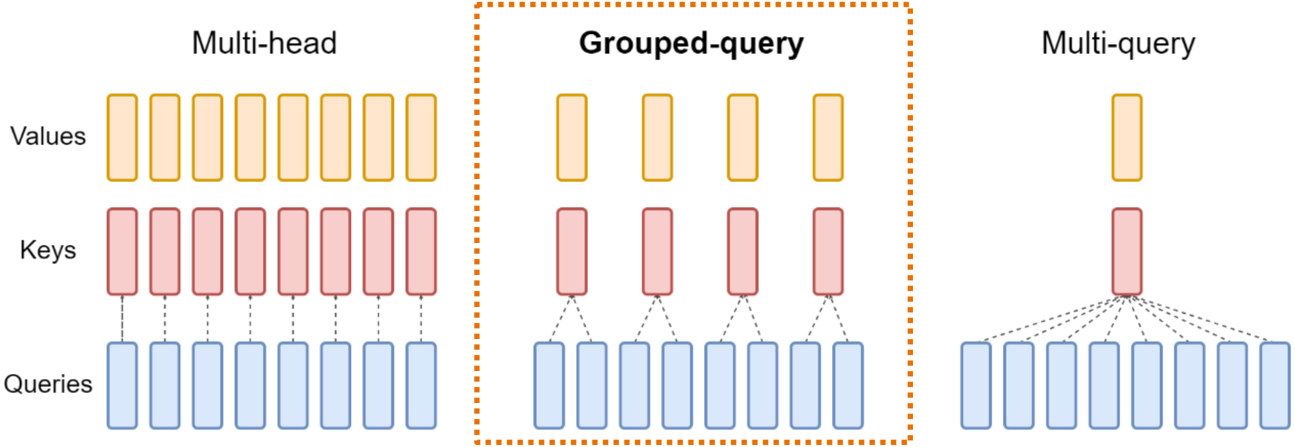
Concatenate the outputs of all heads

h attention heads are computed separately

Figure source: https://arxiv.org/pdf/1706.03762

Multi-Head Self-Attention Variants

- Multi-query attention (Fast Transformer Decoding: One Write-Head is All You Need): share keys and values across all attention heads
- Grouped-query attention (GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints): share keys and values within groups of heads



Used in latest LLMs (e.g., Llama3)

Figure source: https://arxiv.org/pdf/2305.13245

Parallel Computation of QKV

- Self-attention computation performed for each token is independent of other tokens
- Easily parallelize the entire computation, taking advantage of the efficient matrix multiplication capability of GPUs
- Process an input sequence with N words in parallel

Compute QKV for one word: $\boldsymbol{q}_i = \boldsymbol{x}_i \boldsymbol{W}^Q \quad \boldsymbol{k}_i = \boldsymbol{x}_i \boldsymbol{W}^K \quad \boldsymbol{v}_i = \boldsymbol{x}_i \boldsymbol{W}^V \in \mathbb{R}^d$

Stacking N input vectors: $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V \in \mathbb{R}^{N \times d}$

$$\boldsymbol{X} = \begin{bmatrix} \text{——} & \boldsymbol{x}_1 & \text{——} \\ \text{——} & \boldsymbol{x}_2 & \text{——} \\ \cdots & \cdots & \cdots \\ \text{——} & \boldsymbol{x}_N & \text{——} \end{bmatrix}$$

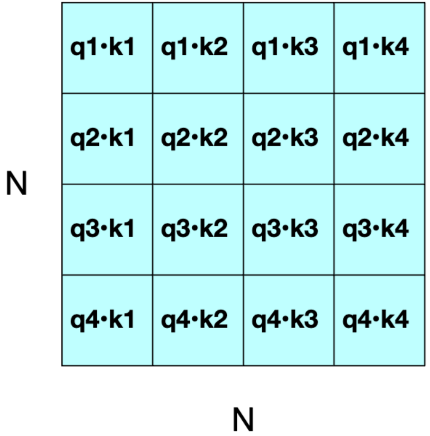
Parallel Computation of Attention

Attention computation can also be written in matrix form

Compute attention for one word: $$\boldsymbol{a}_i = \mathrm{Softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right) \cdot \boldsymbol{v}_j$$

Compute attention for one N words: $$\boldsymbol{A} = \mathrm{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}$$

Attention matrix

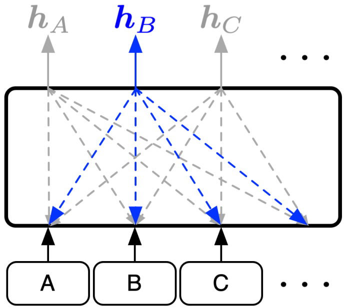


q1•k1	q1•k2	q1•k3	q1•k4
q2•k1	q2•k2	q2•k3	q2•k4
q3•k1	q3•k2	q3•k3	q3•k4
q4•k1	q4•k2	q4•k3	q4•k4

Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Bidirectional vs. Unidirectional Self-Attention

- Self-attention can capture different context dependencies
- **Bidirectional** self-attention:
  - Each position to attend to all other positions in the input sequence
  - Transformers with bidirectional self-attention are called Transformer **encoders** (e.g., BERT)
  - Use case: natural language understanding (NLU) where the entire input is available at once, such as text classification & named entity recognition

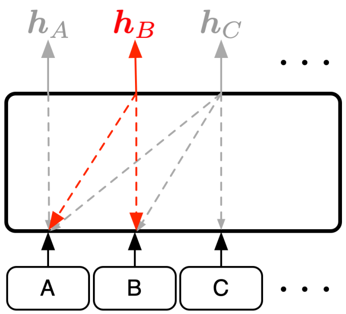


every token attends to all tokens

Bidirectional Self-Attention

Bidirectional vs. Unidirectional Self-Attention

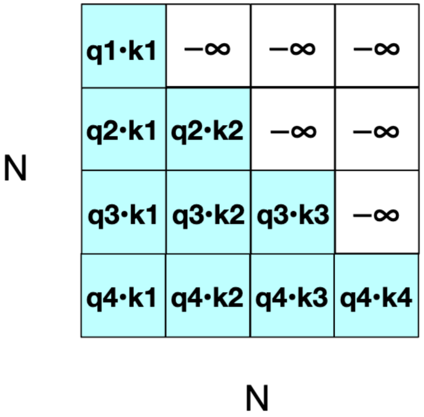
- Self-attention can capture different context dependencies
- **Unidirectional** (or **causal**) self-attention:
  - Each position can only attend to earlier positions in the sequence (including itself).
  - Transformers with unidirectional self-attention are called Transformer **decoders** (e.g., GPT)
  - Use case: natural language generation (NLG) where the model generates output sequentially



every token attends to its previous tokens

Unidirectional Self-Attention

upper-triangle portion set to -inf



q1•k1	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞
q4•k1	q4•k2	q4•k3	q4•k4

Position Encoding

- Motivation: inject positional information to input vectors

$$\boldsymbol{q}_i = \boldsymbol{x}_i \boldsymbol{W}^Q \quad \boldsymbol{k}_i = \boldsymbol{x}_i \boldsymbol{W}^K \quad \boldsymbol{v}_i = \boldsymbol{x}_i \boldsymbol{W}^V \in \mathbb{R}^d$$

$$\boldsymbol{a}_i = \text{Softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right) \cdot \boldsymbol{v}_j$$

When $\boldsymbol{x}$ is word embedding, $\boldsymbol{q}$ and $\boldsymbol{k}$ do not have positional information!

- How to know the word positions in the sequence? Use position encoding!

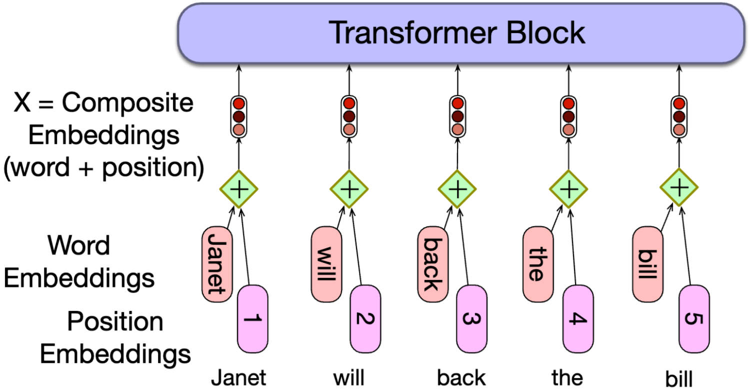


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Position Encoding Methods

- Absolute position encoding (the original Transformer paper)
  - Learn position embeddings for each position
  - Not generalize well to sequences longer than those seen in training
- Relative position encoding (Self-Attention with Relative Position Representations)
  - Encode the relative distance between words rather than their absolute positions
  - Generalize better to sequences of different lengths
- Rotary position embedding (RoFormer: Enhanced Transformer with Rotary Position Embedding)
  - Apply a rotation matrix to the word embeddings based on their positions
  - Incorporate both absolute and relative positions
  - Generalize effectively to longer sequences
  - Widely-used in latest LLMs

Summary: Transformer

- Motivation: weigh the importance of different words in a sequence when processing a specific word
- Implementation: represent each word with three vectors:
    - Query: the current word that seeks information
    - Key: context word to be retrieved information from
    - Value: semantic content to be aggregated as the new word representation
- Allow parallel computation of all input words
- Usually deployed with multiple heads to capture various linguistic relationships
- Can be either unidirectional (only attend to previous words) or bidirectional (attend to all words)
- Need to use position encodings to inject positional information

Limitations: Transformer

- Quadratic Complexity wrt Sequence Length:
  - self-attention has a quadratically complexity with the sequence length
  - processing long sequences is extremely compute & memory expensive
- Interpretability & Explainability:
  - complex architecture with many layers and attention heads (totaling billions of parameters)
  - difficult to understand how they arrive at their predictions & debug
- Positional Encoding:
  - the original Transformer paper adopts manually-defined position encodings – likely suboptimal
  - follow-up works propose advance position encoding methods to enhance expressiveness

Agenda: Language Model Pretraining & Fine-Tuning

- Background: Pretraining & Fine-Tuning
- Decoder Pretraining
- Encoder Pretraining
- Encoder-Decoder Pretraining

Pretraining: Motivation

- There are abundant text data on the web, with rich information of linguistic features and knowledge about the world
- Learning from these easy-to-obtain data greatly benefits various downstream tasks

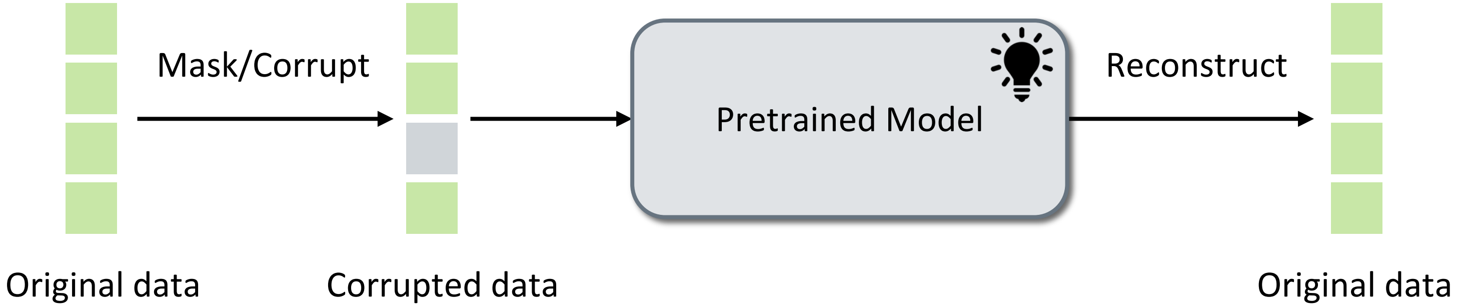
Pretraining: Multi-Task Learning

- In my free time, I like to **{run, banana}** (*Grammar*)
- I went to the zoo to see giraffes, lions, and **{zebras, spoon}** (*Lexical semantics*)
- The capital of Denmark is **{Copenhagen, London}** (*World knowledge*)
- I was engaged and on the edge of my seat the whole time. The movie was **{good, bad}** (*Sentiment analysis*)
- The word for "pretty" in Spanish is **{bonita, hola}** (*Translation*)
- 3 + 8 + 4 = **{15, 11}** (*Math*)
- …

Examples from: https://docs.google.com/presentation/d/1hQUd3pF8_2Gr2Obc89LKjmHL0DlH-uof9M0yFVd3FA4/edit#slide=id.g28e2e9aa709_0_1

Pretraining: Self-Supervised Learning

- Pretraining is a form of **self-supervised** learning
- Make a part of the input unknown to the model
- Use other parts of the input to reconstruct/predict the unknown part



No Human Supervision Needed!

Pretraining + Fine-Tuning

- Pretraining: trained with pretext tasks on large-scale text corpora
- Fine-tuning (also called post-training): adjust the pretrained model's parameters with fine-tuning data
- Fine-tuning data can have different forms:
  - Task-specific labeled data (e.g., sentiment classification, named entity recognition)
  - (Multi-turn) dialogue data (i.e., instruction tuning)

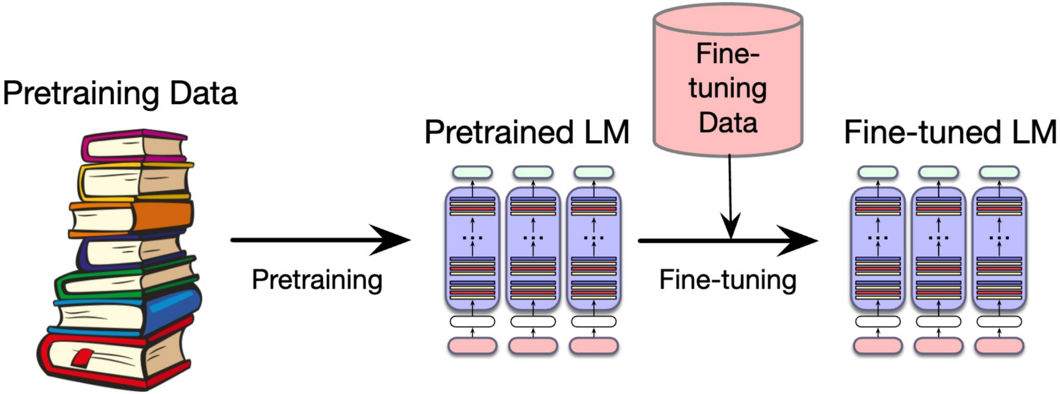


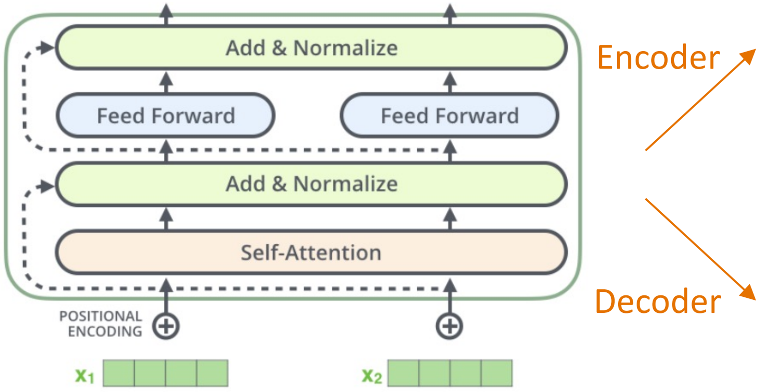
Figure source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

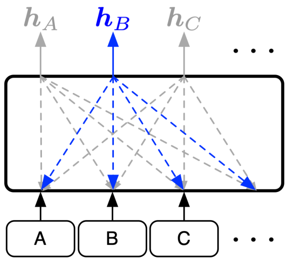
Transformer for Pretraining

- Transformer is the common backbone architecture for language model pretraining
- **Efficiency**: Transformer processes all tokens in a sequence simultaneously – fast and efficient to train, especially on large datasets
- **Scalability**: Transformer architectures have shown impressive scaling properties, with performance improving as model size and training data increase (more on this later!)
- **Versatility**: Transformer can be adapted for various tasks and modalities beyond just text, including vision, audio, and other multimodal applications

Transformer Architectures

- Based on the type of self-attention, Transformer can be instantiated as
    - Encoder: Bidirectional self-attention
    - Decoder: Unidirectional self-attention
    - Encoder-decoder: Use both encoder and decoder





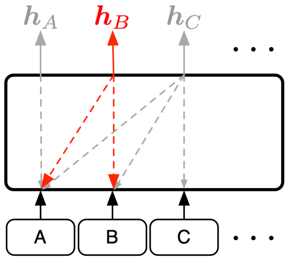
every token attends to all tokens

Bidirectional Self-Attention

N

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

N



every token attends to its previous tokens

Unidirectional Self-Attention

N

q1·k1	$-\infty$	$-\infty$	$-\infty$
q2·k1	q2·k2	$-\infty$	$-\infty$
q3·k1	q3·k2	q3·k3	$-\infty$
q4·k1	q4·k2	q4·k3	q4·k4

N

Applications of Transformer Architectures

- Encoder (e.g., BERT):
  - Capture bidirectional context to learn each token representations
  - Suitable for natural language understanding (NLU) tasks
- Decoder (modern large language models, e.g., GPT):
  - Use prior context to predict the next token (conventional language modeling)
  - Suitable for natural language generation (NLG) tasks
  - Can also be used for NLU tasks by generating the class labels as tokens
- Encoder-decoder (e.g., BART, T5):
  - Use the encoder to process input, and use the decoder to generate outputs
  - Can conduct all tasks that encoders/decoders can do

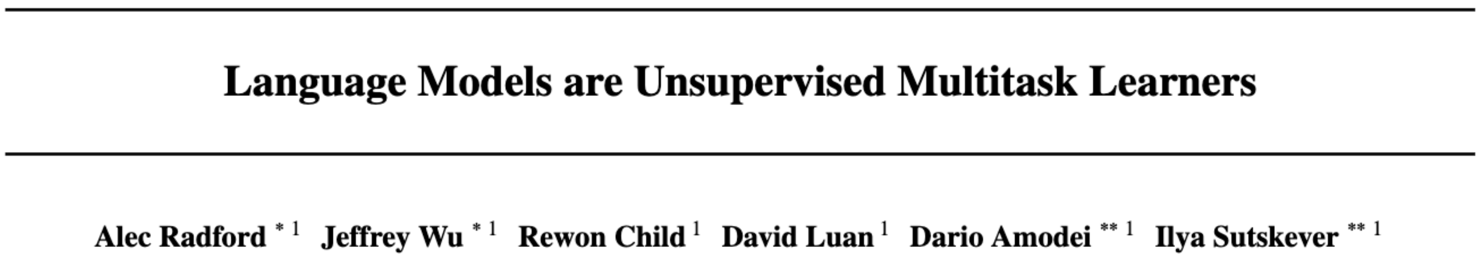
NLU:
Text classification
Named entity recognition
Relation extraction
Sentiment analysis
…

NLG:
Text summarization
Machine translation
Dialogue system
Question answering
…

Agenda: Language Model Pretraining & Fine-Tuning

- Background: Pretraining & Fine-Tuning
- Decoder Pretraining
- Encoder Pretraining
- Encoder-Decoder Pretraining

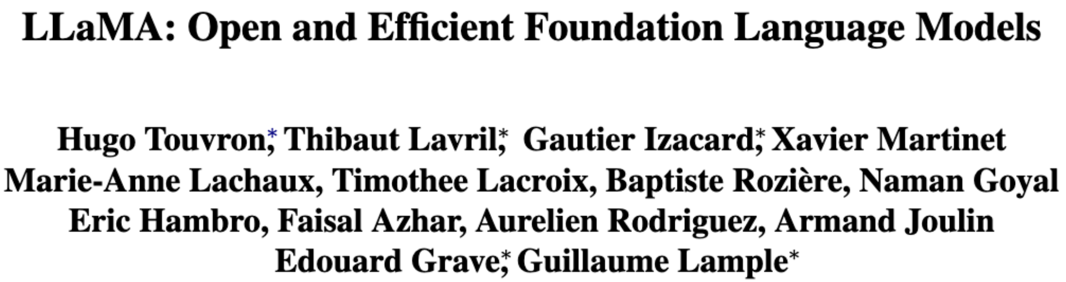
GPT & Llama



Language Models are Unsupervised Multitask Learners

Alec Radford * 1 **Jeffrey Wu** * 1 **Rewon Child** 1 **David Luan** 1 **Dario Amodei** ** 1 **Ilya Sutskever** ** 1

Paper: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf



LLaMA: Open and Efficient Foundation Language Models

Hugo Touvron*, Thibaut Lavril*, Gautier Izacard*, Xavier Martinet
Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal
Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin
Edouard Grave*, Guillaume Lample*

Paper: https://arxiv.org/pdf/2302.13971

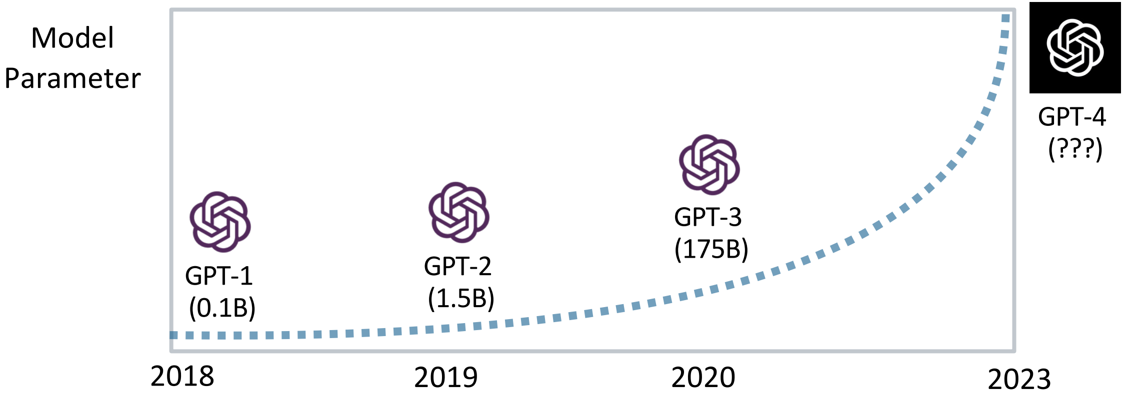
Decoder Pretraining

- Decoder architecture is the prominent choice in large language models
- Pretraining decoders is first introduced in GPT (generative pretraining) models
- Follow the standard language modeling (cross-entropy) objective

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\log p_{\boldsymbol{\theta}}(x_i|x_1, x_2, \ldots, x_{i-1})$$

GPT Series

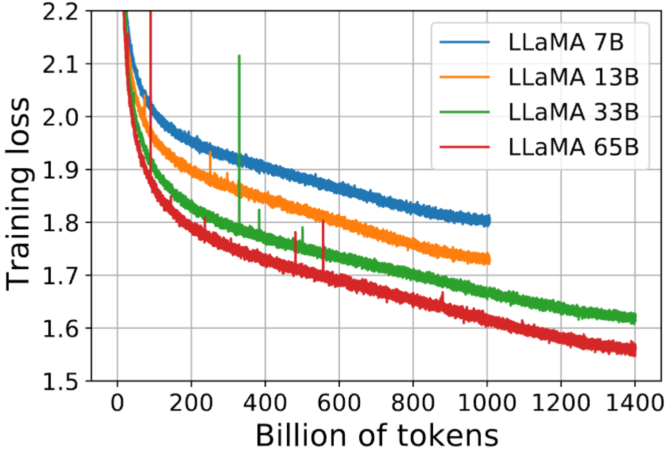
- GPT-1 (2018): 12 layers, 117M parameters, trained in ~1 week
- GPT-2 (2019): 48 layers, 1.5B parameters, trained in ~1 month
- GPT-3 (2020): 96 layers, 175B parameters, trained in several months



Papers: (GPT-1) https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
(GPT-2) https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
(GPT-3) https://arxiv.org/pdf/2005.14165.pdf

Llama Series

- Llama-1 (2023/02): 7B/13B/33B/65B
- Llama-2 (2023/07): 7B/13B/70B
- Llama-3 (3.1 & 3.2) (2024/07): 1B/3B/8B/70B/405B w/ multi-modality



Larger models learn pretraining data better

Papers: (Llama-1) https://arxiv.org/pdf/2302.13971
(Llama-2) https://arxiv.org/pdf/2307.09288
(Llama-3) https://arxiv.org/pdf/2407.21783

Agenda: Language Model Pretraining & Fine-Tuning

- Background: Pretraining & Fine-Tuning
- Decoder Pretraining
- Encoder Pretraining
- Encoder-Decoder Pretraining

BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Paper: https://arxiv.org/pdf/1810.04805.pdf

Encoder Pretraining: BERT

- BERT pretrains encoder models with bidirectionality
- **Masked language modeling** (MLM): With 15% words randomly masked, the model learns bidirectional contextual information to predict the masked words

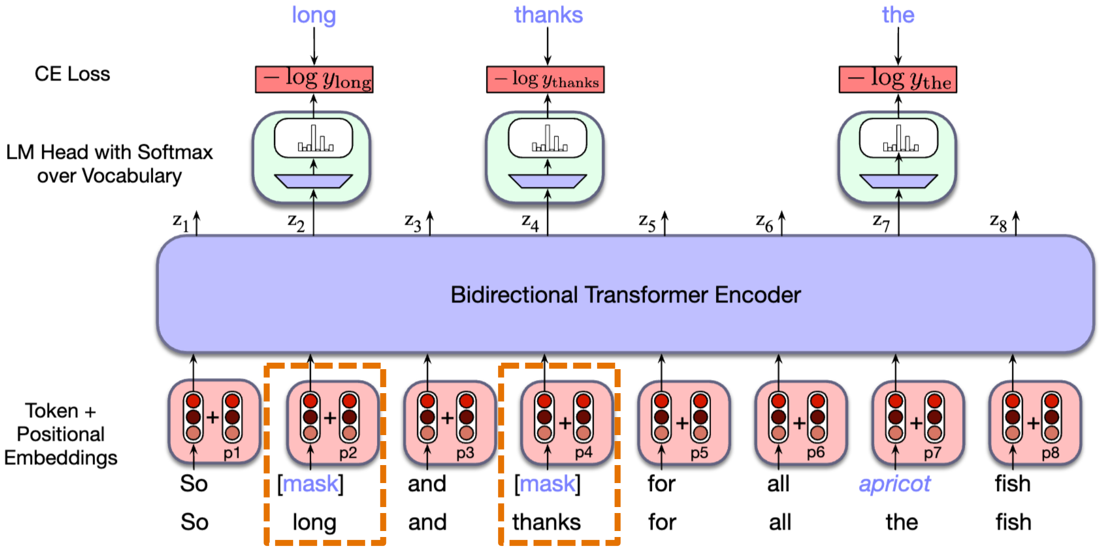


Figure source: https://web.stanford.edu/~jurafsky/slp3/11.pdf

Encoder Pretraining: BERT

- **Next sentence prediction** (NSP): the model is presented with pairs of sentences
- The model is trained to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentence

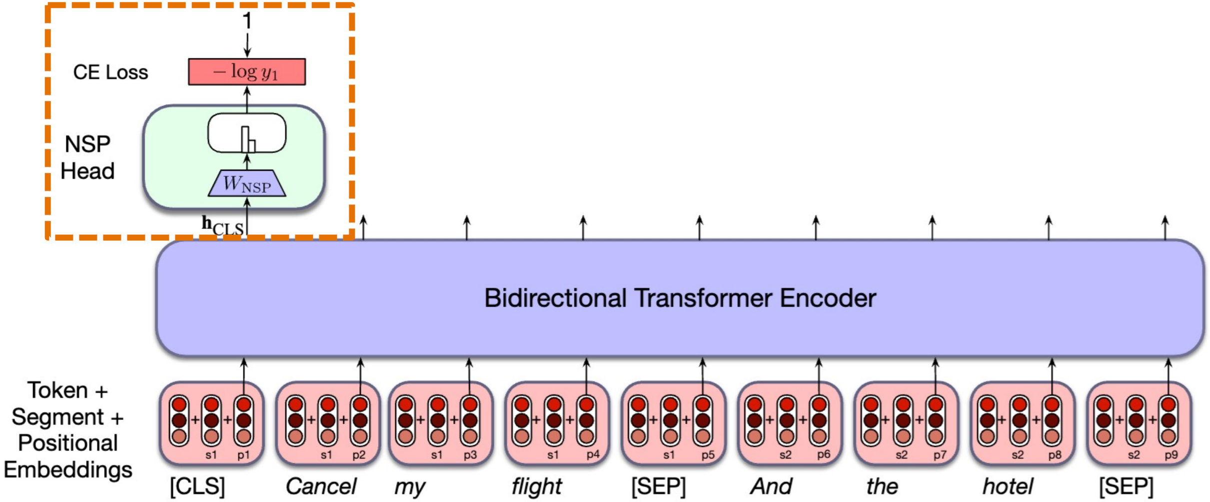
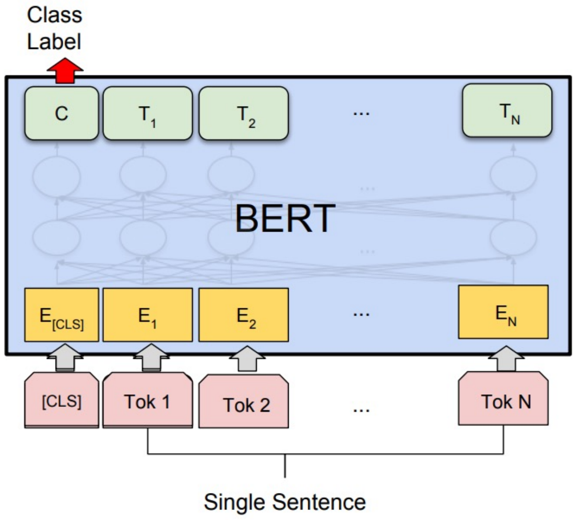


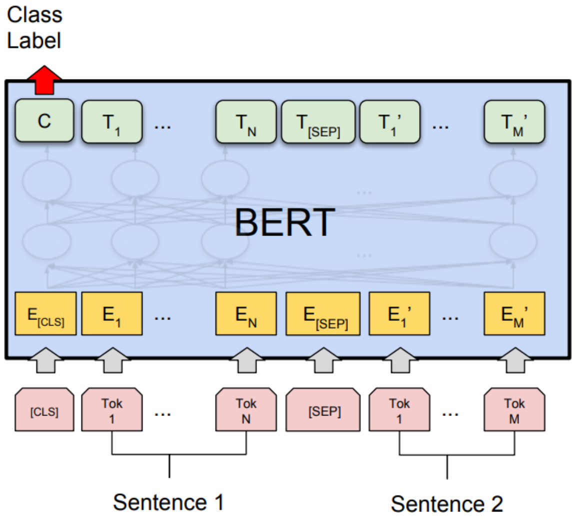
Figure source: https://web.stanford.edu/~jurafsky/slp3/11.pdf

BERT Fine-Tuning

- Fine-tuning pretrained BERT models takes different forms depending on task types
- Usually replace the LM head with a linear layer fine-tuned on task-specific data



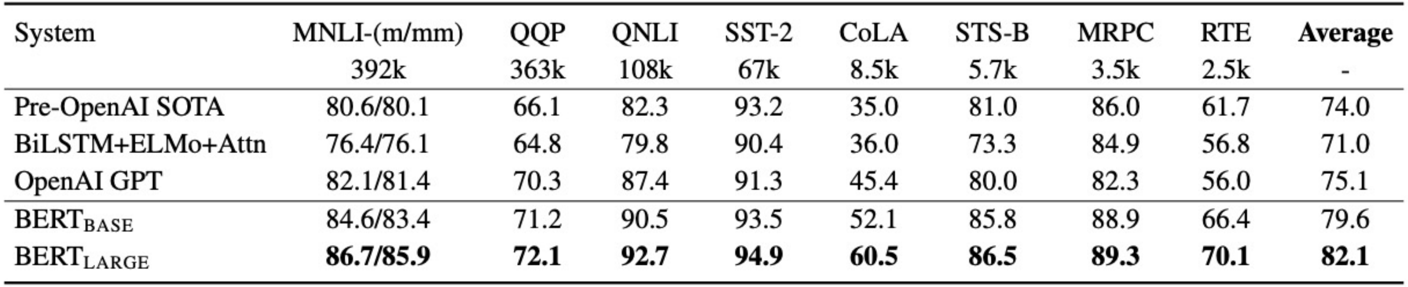
Single sequence classification



Sequence-pair classification

BERT vs. GPT on NLU tasks

- BERT outperforms GPT-1 on a set of NLU tasks
- Encoders capture **bidirectional** contexts – build a richer understanding of the text by looking at both preceding and following words
- Are encoder models still better than state-of-the-art (large) decoder models?
    - LLMs can be as good as (if not better than) encoders model on NLU: Can ChatGPT Understand Too?
    - The sheer model size + massive amount of pretraining data compensate for LLMs' unidirectional processing

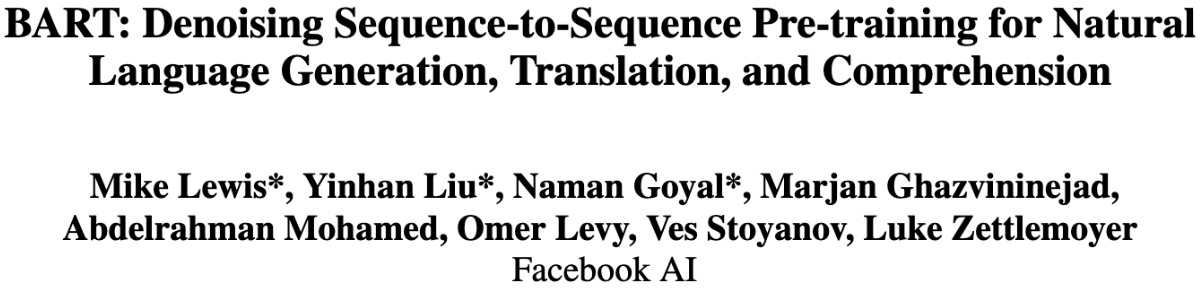


System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1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

Agenda: Language Model Pretraining & Fine-Tuning

- Background: Pretraining & Fine-Tuning
- Decoder Pretraining
- Encoder Pretraining
- Encoder-Decoder Pretraining

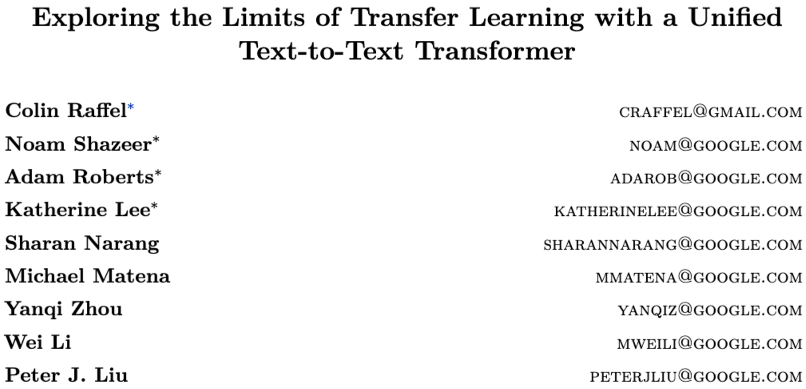
BART & T5



BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

Mike Lewis*, Yinhan Liu*, Naman Goyal*, Marjan Ghazvininejad,
Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer
Facebook AI

Paper: https://arxiv.org/pdf/1910.13461



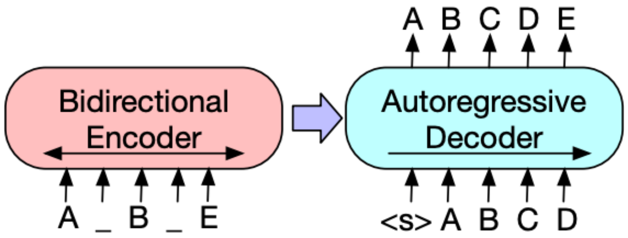
Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

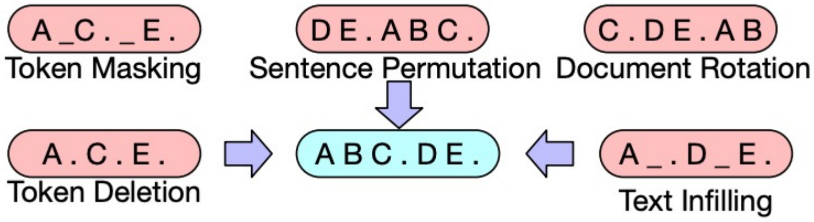
Colin Raffel* CRAFFEL@GMAIL.COM
Noam Shazeer* NOAM@GOOGLE.COM
Adam Roberts* ADAROB@GOOGLE.COM
Katherine Lee* KATHERINELEE@GOOGLE.COM
Sharan Narang SHARANNARANG@GOOGLE.COM
Michael Matena MMATENA@GOOGLE.COM
Yanqi Zhou YANQIZ@GOOGLE.COM
Wei Li MWEILI@GOOGLE.COM
Peter J. Liu PETERJLIU@GOOGLE.COM

Paper: https://arxiv.org/pdf/1910.10683

Encoder-Decoder Pretraining: BART

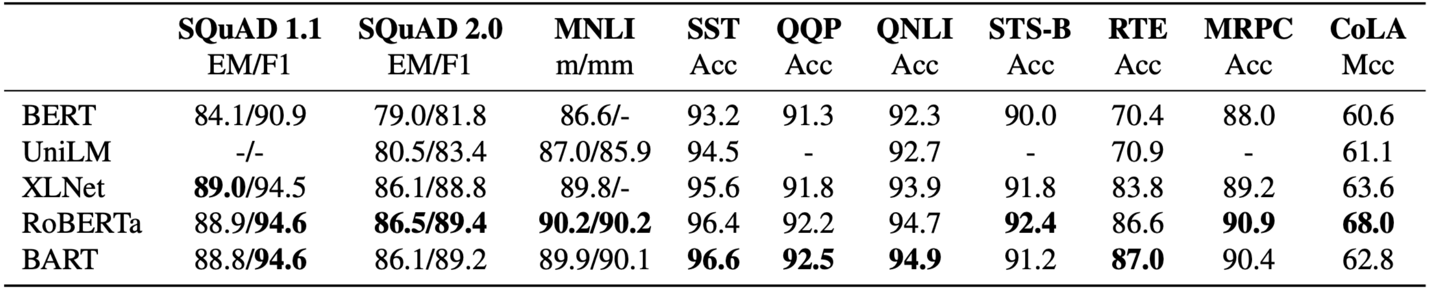
- Pretraining: Apply a series of noising schemes (e.g., masks, deletions, permutations…) to input sequences and train the model to recover the original sequences
- Fine-Tuning:
  - For NLU tasks: Feed the same input into the encoder and decoder, and use the final decoder token for classification
  - For NLG tasks: The encoder takes the input sequence, and the decoder generates outputs autoregressively



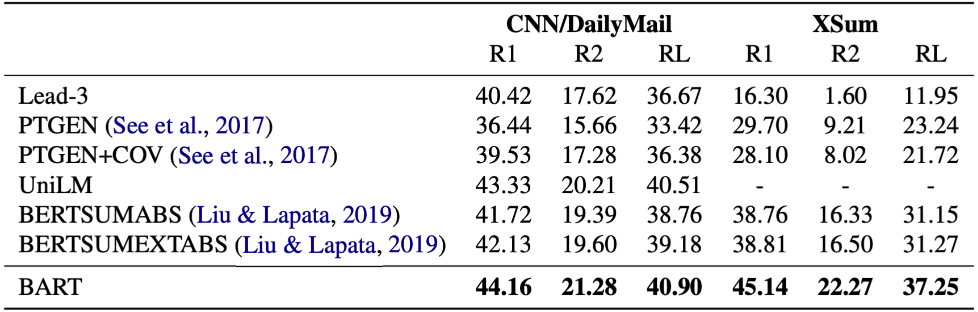


BART Performance

- Comparable to encoders on NLU tasks
- Good performance on NLG tasks



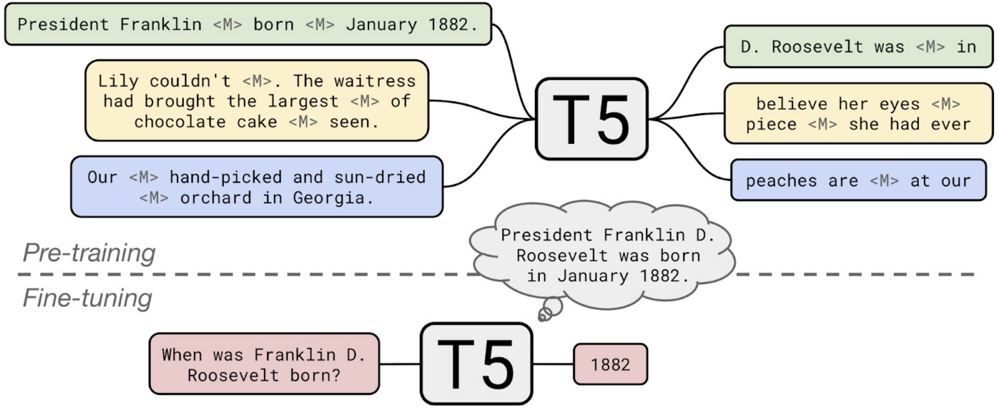
	SQuAD 1.1 EM/F1	SQuAD 2.0 EM/F1	MNLI m/mm	SST Acc	QQP Acc	QNLI Acc	STS-B Acc	RTE Acc	MRPC Acc	CoLA Mcc
BERT	84.1/90.9	79.0/81.8	86.6/-	93.2	91.3	92.3	90.0	70.4	88.0	60.6
UniLM	-/-	80.5/83.4	87.0/85.9	94.5	-	92.7	-	70.9	-	61.1
XLNet	**89.0**/94.5	86.1/88.8	89.8/-	95.6	91.8	93.9	91.8	83.8	89.2	63.6
RoBERTa	88.9/**94.6**	**86.5/89.4**	**90.2/90.2**	96.4	92.2	94.7	**92.4**	86.6	**90.9**	**68.0**
BART	88.8/**94.6**	86.1/89.2	89.9/90.1	**96.6**	**92.5**	**94.9**	91.2	**87.0**	90.4	62.8



	CNN/DailyMail			**XSum**		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**

Encoder-Decoder Pretraining: T5

- T5: **T**ext-**to**-**T**ext **T**ransfer **T**ransformer
- Pretraining: Mask out spans of texts; generate the original spans
- Fine-Tuning: Convert every task into a sequence-to-sequence generation problem
- We'll see this model again in the instruction tuning lectures



President Franklin <M> born <M> January 1882.

Lily couldn't <M>. The waitress had brought the largest <M> of chocolate cake <M> seen.

Our <M> hand-picked and sun-dried <M> orchard in Georgia.

D. Roosevelt was <M> in

believe her eyes <M> piece <M> she had ever

peaches are <M> at our

President Franklin D. Roosevelt was born in January 1882.

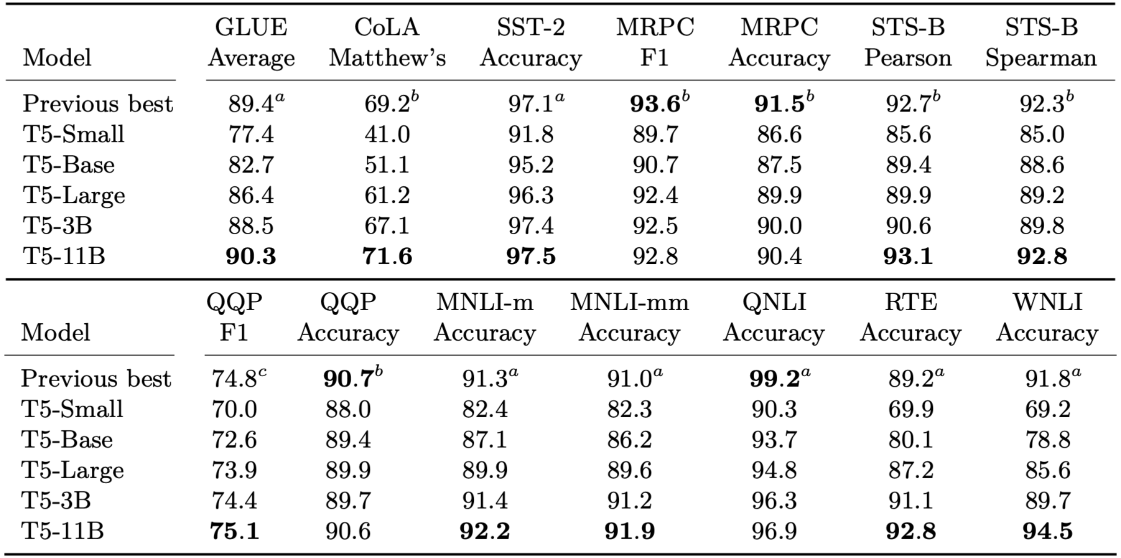
When was Franklin D. Roosevelt born?

1882

T5: https://arxiv.org/pdf/1910.10683

T5 Performance

- Good performance across various tasks
- T5 vs. BART performance: unclear comparison due to difference in model sizes & training setups



Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**90.3**	**71.6**	**97.5**	92.8	90.4	**93.1**	**92.8**

Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	74.8[c]	**90.7**[b]	91.3[a]	91.0[a]	**99.2**[a]	89.2[a]	91.8[a]
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2
T5-Base	72.6	89.4	87.1	86.2	93.7	80.1	78.8
T5-Large	73.9	89.9	89.9	89.6	94.8	87.2	85.6
T5-3B	74.4	89.7	91.4	91.2	96.3	91.1	89.7
T5-11B	**75.1**	90.6	**92.2**	**91.9**	96.9	**92.8**	**94.5**

Encoder-Decoder vs. Decoder-Only

- Modern LLMs are mostly based on the decoder-only Transformer architecture
- Simplicity:
  - Decoder-only models are simpler in structure (one Transformer model)
  - Encoder-decoder models require two Transformer models
- Efficiency:
  - Decoder-only models are more parameter-efficient for text generation
  - Encoder-decoder models' encoder part does not contribute to generation
- Scalability:
  - Decoder-only models scale very well with increased model size and data
  - Encoder-decoder models do not outperform decoder-only models at large model sizes

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu