Statistical Natural Language Processing

Part IX: NLP using Transformers

Henning Wachsmuth

https://ai.uni-hannover.de

Learning Objectives

Concepts

- Multi-head self-attention
- Building blocks of transformers
- Contextual embeddings
- Masked language modeling

Methods

- Left-to-right transformers for text generation
- Bidirectional transformers for text classification and sequence labeling
- Encoder-decoder transformers for sequence-to-sequence generation
- Instruction fine-tuning and alignment of large language models
- Prompting and sampling strategies for large language models

Tasks

- Language modeling and text summarization
- Sentiment analysis and part-of-speech tagging

Outline of the Course

I. Overview

II. Basics of Data Science

III. Basics of Natural Language Processing

IV. Representation Learning

V. NLP using Clustering

VI. NLP using Classification and Regression

VII. NLP using Sequence Labeling

VIII. NLP using Neural Networks

IX. NLP using Transformers

- Introduction
- Left-to-Right Transformers
- Bidirectional Transformers
- Encoder-Decoder Transformers
- Large Language Models
- Conclusion

X. Practical Issues

Introduction

Transformers

Limitations of long-short term memories (LSTMs)

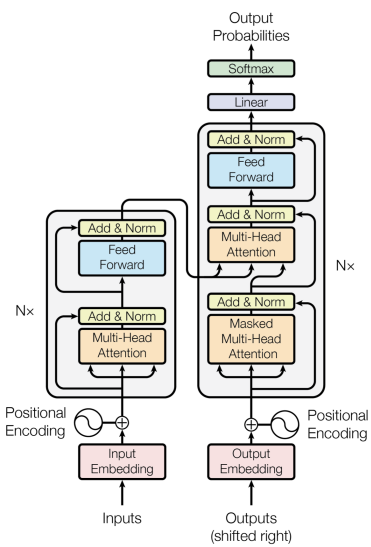
- LSTMs may still struggle with modeling long-term dependencies, since their memory is limited by the size of the hidden layer.
- Moreover, their sequential nature leaves few room for parallelization.

Idea of transformers

- A technique for sequence processing without recurrent connections
- *Self-attention* models relations of words over long distances by using information from large contexts.

Transformer neural network

- A block-wise architecture of multilayer networks that combines self-attention layers with other network architecture concepts
- Input. An embedding sequence $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
- Output. An embedding sequence $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_k)$



Vaswani et al. (2017)

Transformers

Basic Concepts

Transfer learning

- Transformers acquire knowledge from unsupervised tasks and apply it to more easily solve other (supervised) tasks
- Pretraining. Learn a general transformer model from huge data
- Fine-tuning. Adjust the model to perform some downstream task

Contextual embeddings

- Transformers embed a word w based on the context of the given text.
- Different contexts lead to different embeddings of w.

Transformer architectures

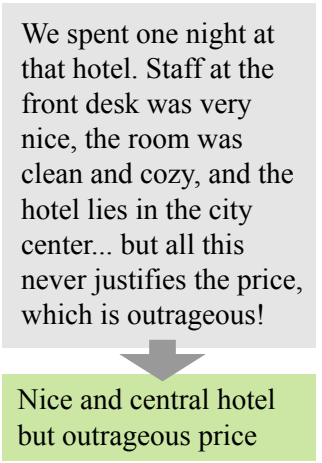
- Different transformers are used for encoding and/or decoding text:
- Left-to-right. Mimics sequential text processing, usually for decoding
- Bidirectional. Allows processing a full text at a time, only for encoding
- Encoder-decoder. Combines both architectures

Transformers

Transformers in NLP

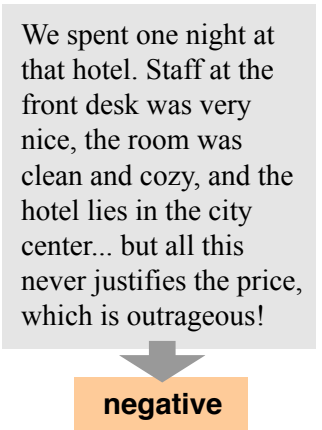
Transformers for generation tasks

- Transformers are designed for decoding/generation.
- A left-to-right or encoder-decoder transformer can be pretrained to work as a language model.
- Task-specific outputs are modeled as input endings.
- Examples. Language modeling, text summarization



Transformers for analysis tasks

- Transformers can also be used for text classification, sequence labeling, and similar tasks.
- A pretrained encoder can be combined with an FNN, a CRF, or similar, and then fine-tuned on the task.
- This may drastically reduce the need for labeled data.
- Examples. Sentiment analysis, part-of-speech tagging



Transformers

From Transformers to Large Language Models

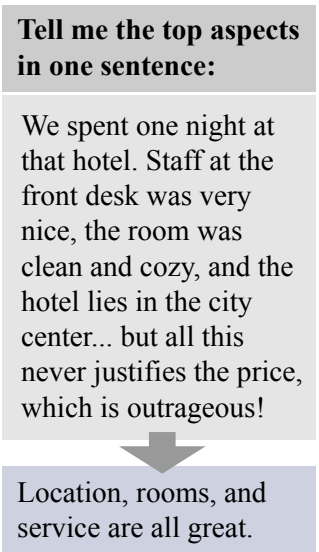
Impact of transformers

- The transformer architecture is state of the art in most NLP tasks.
- So far, all common *large language models* are transformer-based.
- Still, task-specific adaptations are often helpful.

Large language model (LLM)

- A language model, usually with billions of parameters
- Special training steps make LLMs instruction-following.
- Leading LLMs answer reasonably to most prompts.

  But they may create *hallucinations* and *bias* (see Lecture Part X).



Additional concepts of LLMs

- Training process. Instruction fine-tuning, alignment, and more
- Input handling. Prompting strategies and methods
- Ouput handling. Sampling strategies during output generation

Left-to-Right Transformers

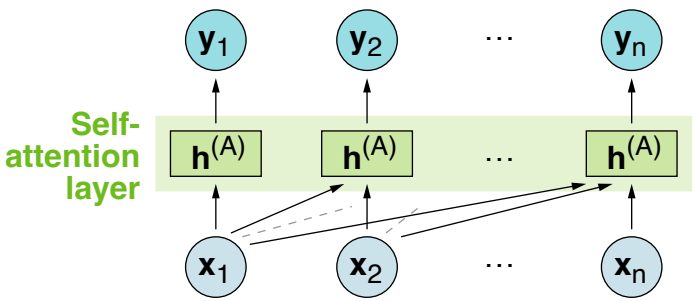
Left-to-Right Transformers

Self-attention layer $\mathbf{h}^{(A)}$

- A specific type of hidden layer that maps a sequence $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to a sequence $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$
- The idea is to model each $\mathbf{x}_j$ based on the context of the other inputs.
- This allows learning which inputs are relevant to which others.

Left-to-right self-attention

- When processing $\mathbf{x}_j$, $\mathbf{h}^{(A)}$ has access to all $\mathbf{x}_k$ with $k \leq j$, but not with $k > j$.
- This enables the transformers to do autoregressive generation.



Computational efficiency

- The computation performed for $\mathbf{x}_j$ is independent of those for other $\mathbf{x}_k$.
- Thus, training and inference of self-attention layers can be parallelized.

Left-to-Right Transformers

Self-Attention Layers: Input Representation

Modeling relevance in context

- The core of (left-to-right) self-attention is to score the relevance of all inputs $\mathbf{x}_k$, $k \leq j$, for a given input $\mathbf{x}_j$.
- The scores are used to weight the influence of $\mathbf{x}_k$ for the output $\mathbf{y}_j$ of $\mathbf{x}_j$.
- In the processing of X, each $\mathbf{x}_j$ takes on three different *roles*.

Roles of inputs

- Query ($\mathbf{q}_j$). $\mathbf{x}_j$ is in the focus; all $\mathbf{x}_k$ with $k \leq j$ are compared to it
- Key ($\mathbf{k}_j$). $\mathbf{x}_j$ is compared to any query $\mathbf{q}_l = \mathbf{x}_l$, $l \geq j$
- Value ($\mathbf{v}_j$). $\mathbf{x}_j$ is a value used to compute the output $\mathbf{y}_l$, $l \geq j$

Representation of roles

- To represent $\mathbf{x}_j$ in each of its roles, weight matrices are learned:

  For vectors of length m (say, $m = 1024$), a matrix is of size $m \times m$.

$$\mathbf{q}_j := \mathbf{W}^{(Q)} \cdot \mathbf{x}_j \qquad \mathbf{k}_j := \mathbf{W}^{(K)} \cdot \mathbf{x}_j \qquad \mathbf{v}_j := \mathbf{W}^{(V)} \cdot \mathbf{x}_j$$

Left-to-Right Transformers

Self-Attention Layers: Output Computation

Score computation

- The relevance of $\mathbf{x}_k$ for a focus $\mathbf{x}_j$ is modeled as a similarity.
- Similarity is computed as the dot product between key $\mathbf{k}_k$ and query $\mathbf{q}_j$:

$$score(\mathbf{x}_j, \mathbf{x}_k) := \mathbf{q}_j \cdot \mathbf{k}_k$$

Weight computation

- Each score may range from $-\infty$ to ∞, the larger the more similar.
- For normalization, it is often scaled based on the embedding length m.
- The scores for $\mathbf{x}_j$ are then mapped to a vector of j weights α_{ij}:

$$\forall 1 \leq k \leq j : \; \alpha_{jk} := \text{softmax}\left(\frac{score(\mathbf{x}_j, \mathbf{x}_k)}{\sqrt{m}}\right)$$

Output computation

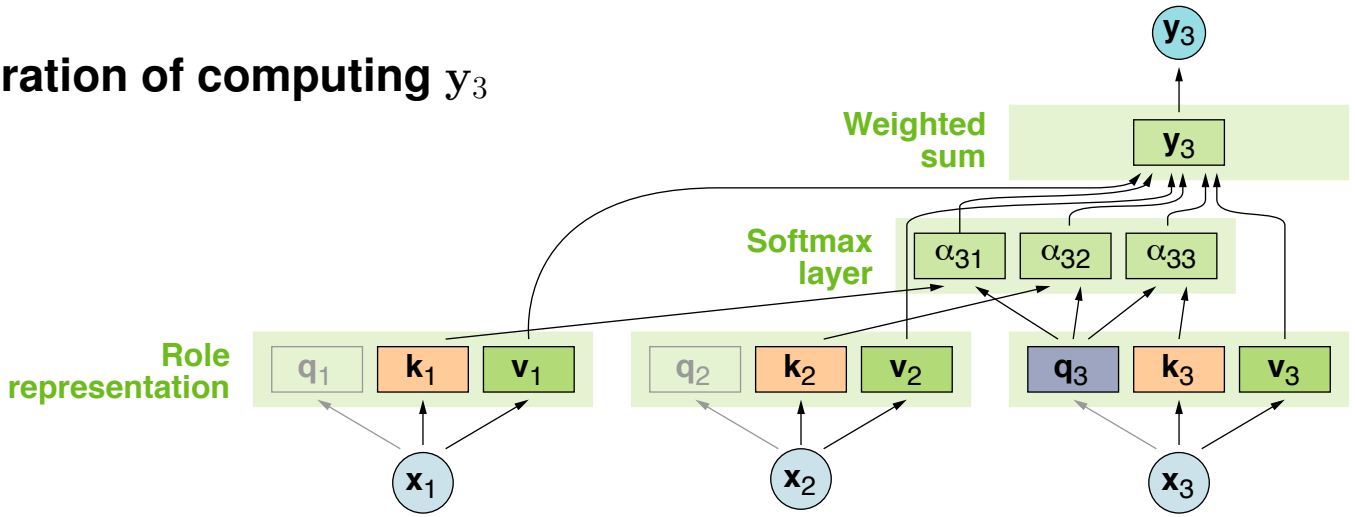
- Finally, the output $\mathbf{y}_j$ of $\mathbf{x}_j$ is the weighted sum of the values $\mathbf{v}_k$:

$$\mathbf{y}_j := \sum_{k=1}^{j} \alpha_{jk} \cdot \mathbf{v}_k$$

Left-to-Right Transformers

Self-Attention Layers: Efficiency

Illustration of computing $\mathbf{y}_3$



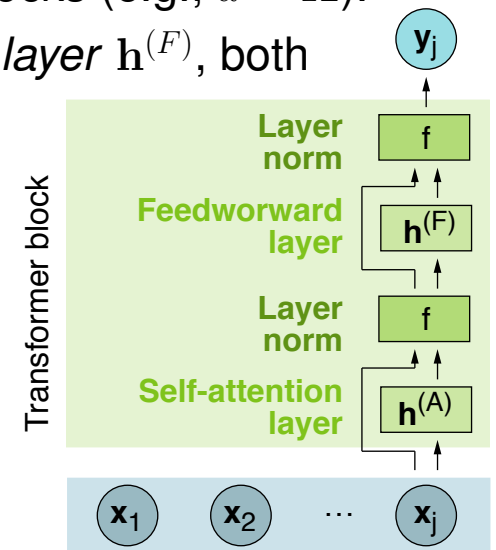
Computational efficiency

- Each output $\mathbf{y}_j \in Y$ can be computed in parallel.
- Still, each self-attention layer compares all input pairs $(\mathbf{x}_j, \mathbf{x}_k)$, which is quadratic in the length of X.
- This makes self-attention very expensive for longer inputs.
- Standard transformer libraries limit the input length (e.g., to 512 tokens).

Transformer Architecture

Transformer block

- A transformer consists of $d \geq 1$ stacked layer blocks (e.g., $d = 12$).
- Block. A self-attention layer $\mathbf{h}^{(A)}$, a *feedforward layer* $\mathbf{h}^{(F)}$, both with *layer norm* f and *residual connections*



Feedforward layer

- $\mathbf{h}^{(F)}$ transforms each token alone non-linearly.

Layer norm

- f normalizes the output of a layer $\mathbf{h}$ in a way that is optimal for gradient-based training.
- It rescales $\mathbf{h}$ using mean μ and standard deviation σ, and adds weights:

  Notice: More recent transfomers put the layer norms *before* the layers.

$$f(\mathbf{h}) \;:=\; \gamma \cdot \frac{\mathbf{h} - \mu(\mathbf{h})}{\sigma(\mathbf{h})} + \beta$$

Residual connection

- Passes information between two layers, skipping an intermediate layer:

$$\mathbf{z} \;:=\; f(\mathbf{x} + \mathbf{h}^{(A)}) \qquad\qquad \mathbf{y} \;:=\; f(\mathbf{z} + \mathbf{h}^{(F)})$$

Transformer Architecture

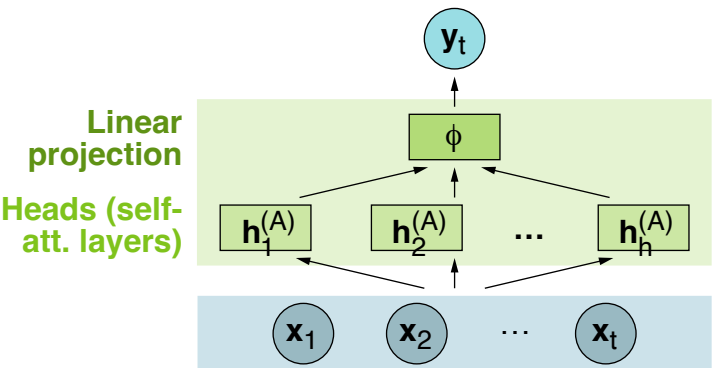
Multi-Head Self-Attention

Multi-head self-attention layer $\mathbf{h}^{(A)}$

- Words may relate to each other in various ways simultaneously.
- Transformers tackle this issue with *multi-head* self-attention layers: sets of $h \geq 1$ self-attention layers at the same depth.

Head

- Each single self-attention layer $\mathbf{h}_l^{(A)}$ is called a *head*.
- Each $\mathbf{h}_l^{(A)}$ has its own matrices $\mathbf{W}_l^{(K)}$, $\mathbf{W}_l^{(Q)}$, and $\mathbf{W}_l^{(V)}$.



Output computation

- The outputs of all heads are concatenated and reduced to the input dimensionality using a linear projection ϕ with weights $\mathbf{W}^{(\phi)}$:

$$\mathbf{h}^{(A)} := \phi(\mathbf{h}_1^{(A)}, \ldots, \mathbf{h}_h^{(A)}) = (\mathbf{h}_1^{(A)} \oplus \ldots \oplus \mathbf{h}_h^{(A)}) \cdot \mathbf{W}^{(\phi)}$$

Transformer Architecture

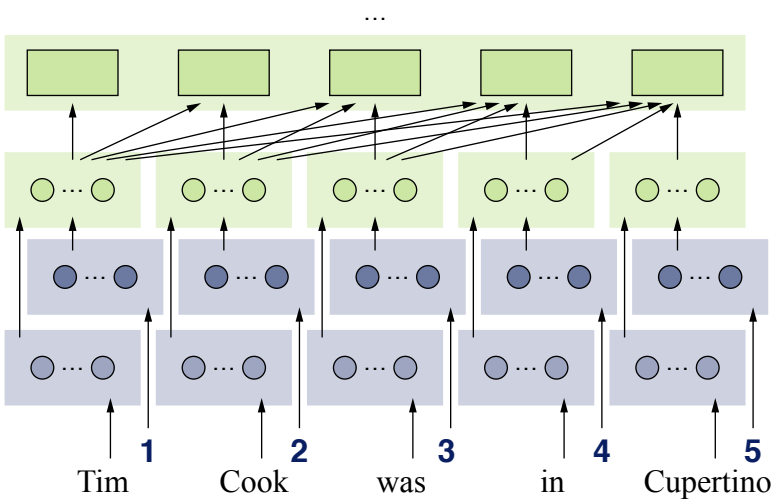
Positional Embeddings

Modeling word order

- The order of words is modeled by combining each embedding $\mathbf{x}_j$ with a *positional embedding* $\mathbf{x}_j^{(P)}$:

$$\mathbf{x}_j + \mathbf{x}_j^{(P)}$$

- $\mathbf{x}_j^{(P)}$ is specific to the position j of $\mathbf{x}_j$ in the sequence X.



Ways to embed positions

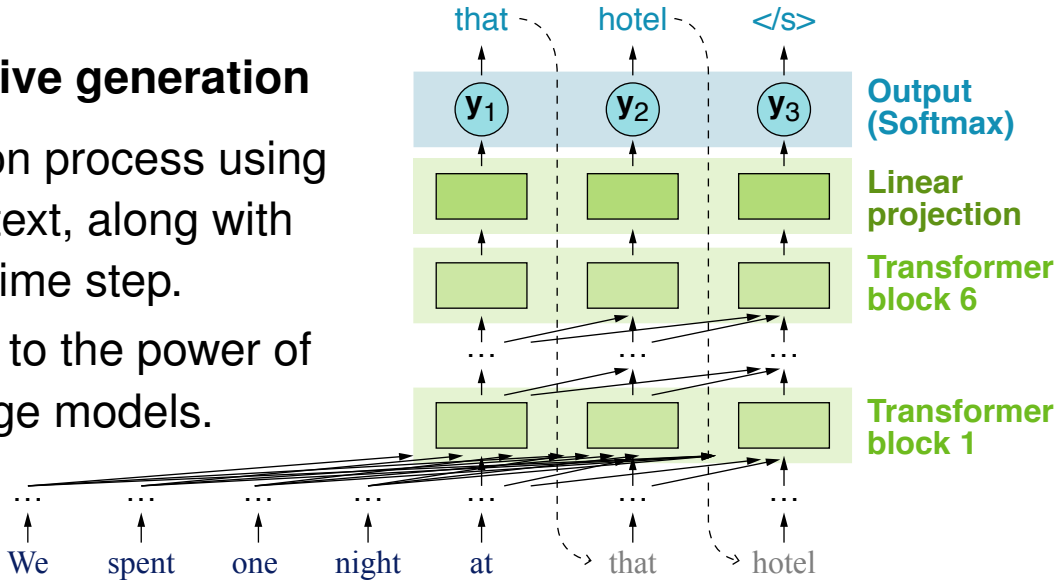
- Learning. Learn to embed j on data up to some maximum position; fewer training examples exist for later positions, though
- Static function. Map positions to embeddings in a way that captures their inherent relationship.

  For example, positions 1 and 2 should be more similar than position 1 and 10.

Left-to-Right Transformers in NLP

Contextual autoregressive generation

- Prime the generation process using the entire prior context, along with the output at each time step.
- This idea is the key to the power of transformer language models.



Transformers as language models

- Left-to-right transformers are trained to predict the next word using teacher forcing.
- This way, *each* instance can be processed in parallel.

Example: Text summarization

- Input. A long(er) text, such as an article or review
- Output. A short(er) text, summarizing the main points

We spent one night at that hotel. Staff at the front desk was very nice, the room was clean and cozy, and the hotel lies in the city center... but all this never justifies the price, which is outrageous!

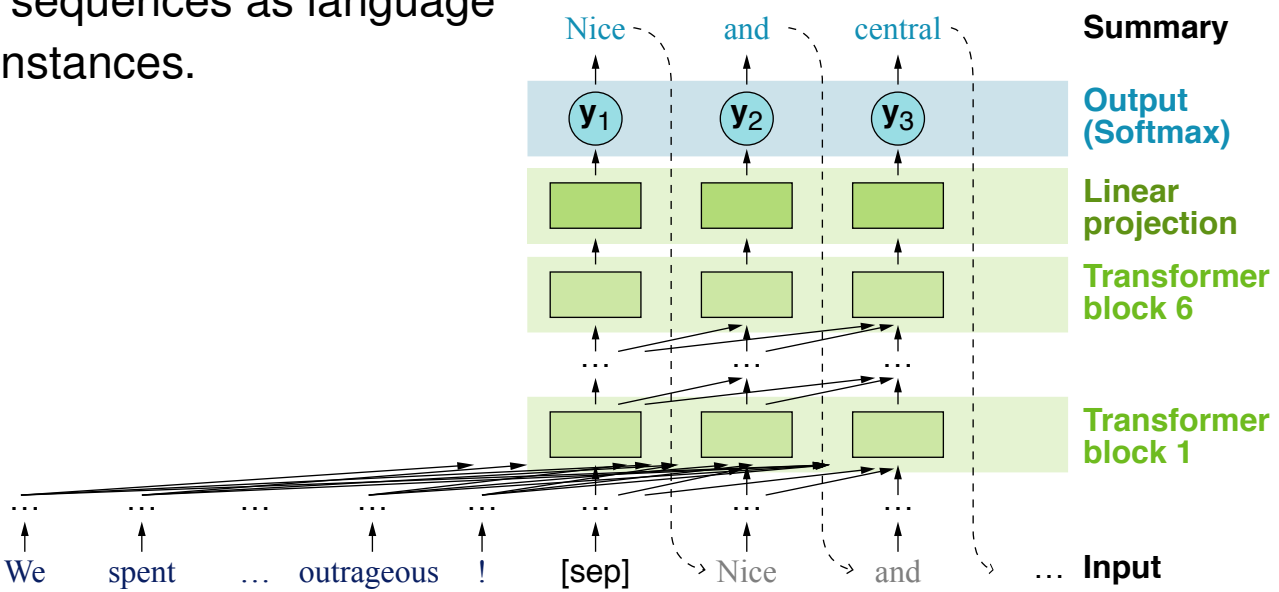
Nice and central hotel but outrageous price

Left-to-Right Transformers in NLP

Text Summarization

Training

- Input. Training pairs of text $(w_1, \ldots, w_n)$ and summary $(w'_1, \ldots, w'_k)$
- Append each pair with a separator tag: $(w_1, \ldots, w_n, \texttt{[sep]}, w'_1, \ldots, w'_k)$
- Use these sequences as language modeling instances.



Inference

- Prime the model with an input text, followed by the tag `[sep]`.
- In each step, the model has access to the text and previous outputs.

Bidirectional Transformers

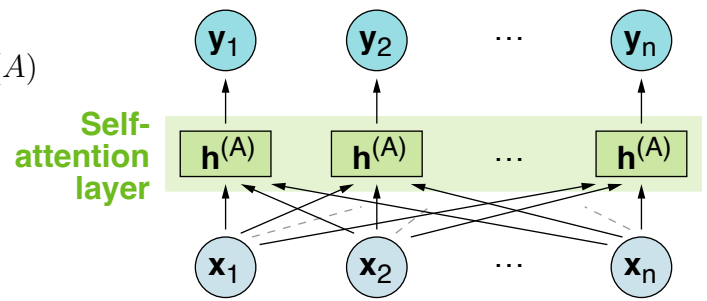
Bidirectional Transformers

Limitations of left-to-right transformers

- By concept, left-to-right transformers can model *prior* context only.
- This is suboptimal for tasks such as classification or sequence labeling.

Bidirectional transformer (encoder)

- A transformer that maps from n input embeddings $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to n output embeddings $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$.
- For any $\mathbf{x}_j$, each self-attention layer $\mathbf{h}^{(A)}$ can access the whole X.
- Each $\mathbf{y}_j$ defines a representation of $\mathbf{x}_j$ *contextualized* by the sequence X.



Usage for downstream tasks

- Bidirectional encoders can be pretrained *self-supervised*.
- For downstream tasks, extra layers are added and trained supervised.
- The encoder may be frozen or fine-tuned towards the task.

Bidirectional Transformers

Contextualization

Self-attention layers

- Mostly, input representation and output computation of self-attention are exactly as in left-to-right transformers.
- The key difference lies in the access to the whole input sequence X.

Subword tokenization

- Most implementations split tokens into subwords for further processing.
- This avoids unknown/rare word problems and reduces vocabulary size.
- Subwords range from a single character to a whole word.
- For tasks that need words, subwords are merged again.

  Different methods exist for both splitting and merging, but are outside the scope here.

Computational efficiency

- As before, each output $\mathbf{y}_j \in Y$ can be computed in parallel.
- Still, both time and memory grow quadratically with the length of X.

Bidirectional Transformers

Self-Supervised Training

Transformers as task solvers

- Due to the free access to X, bidirectional transformers are not trained to predict next words (i.e., as a language model).
- Instead, they learn to solve tasks that can be trained self-supervised.

Self-supervised learning

- Self supervision refers to idea of creating training data automatically.
- A common method is to corrupt an input text and let a model recover it.
- Corruptions. Masking, reordering, substitution, deletion, ...

Common training tasks

- Masked language modeling. Predict missing words in a text.
  For tasks such as coreference resolution, longer spans may be useful.
- Next sentence prediction. Decide if two sentences follow each other.

Bidirectional Transformers

Masked Language Modeling

Masked language modeling (aka the cloze task)

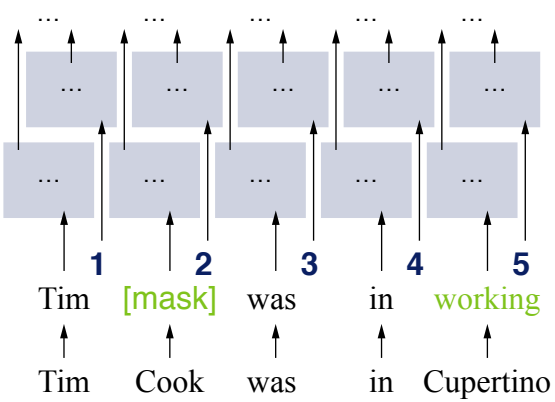
- Given a sequence of words where one or more are missing (masked), predict each missing word $\tilde{w}$.

_____ Cook is the _____ of Apple.

- For each $\tilde{w}$, the probability distribution over the vocabulary is learned.

Training process

- Sample tokens from training sequences for learning.
- Prepare each token in one of three ways:
  1. Mask it with a unique tag `[mask]`.
  2. Replace it with some vocabulary token, chosen based on token probabilities
  3. Leave it unchanged
- Learn to predict the original tokens.



Bidirectional Transformers

Example: BERT (Devlin et al., 2019)

Bidirectional Encoder Representations from Transformers (BERT)

- First bidirectional transformer model, published by Google
- 12 blocks, multi-head self-attention with 12 heads, 768 units per layer
- Subword vocabulary with 30,000 tokens

  This all results in over 100M parameters (recent models are much larger).

Data basis of BERT

- Books Corpus. 0.8 billion words from book texts
- English Wikipedia. 2.5 billion words from articles

  GPT-3 is trained on 470x as many words.

https://flickr.com

Training of BERT

- Masked language modeling. 15% of all tokens in training sequences for learning: 80% masked, 10% replaced, 10% left
- Next sentence prediction. 50% of training pairs positive, 50% negative

  About 40 epochs on both tasks simultaneously until model convergence

Bidirectional Transformers

Contextual Embeddings

Result of training

- Embedding model. A mapping from (sub)words to their embeddings
- Bidirectional encoder. A network that predicts *contextual embeddings* for any input sequence

Contextual(ized) embedding

- A vector representation $\mathbf{v}_j$ of some aspect of the meaning of a word w_j in the context of a sequence $(w_1, \ldots, w_j, \ldots, w_n)$
- As static embeddings, such embeddings can be used for any task.

”tim cook is a ceo”	$\rightarrow$	$\mathbf{v}_{cook} = (0.43, 0.52, 0.21, 0.19, \ldots, 0.33)$
”tim is a cook”	$\rightarrow$	$\mathbf{v}_{cook} = (0.55, 0.01, 0.88, 0.18, \ldots, 0.33)$

What output to use?

- Final. Use the $\mathbf{y}_j$ from the last transformer block
- Mixed. Average, or concatenate, the $\mathbf{y}_j$ from multiple blocks (e.g., last 4)

Birectional Transformers in NLP

Text classification

- A unique tag `[cls]` is prepended to all sequences $(w_1, \ldots, w_n)$ as w_0, both during pretraining and encoding.
- For w_0, $\mathbf{x}_0$ thereby represents the entire sequence $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.
- The output $\mathbf{y}_0$ of the final block is then input to an added *classifier head*.

  Classifier head: A feedforward layer/network that predicts the class label.

Sequence labeling

- Each final output $\mathbf{y}_j$ is mapped to its label probabilities using Softmax.
- Greedy. Simply use the most likely tag as the prediction.
- CRF. Input the label probabilities to a *conditional random field head*.

  CRF head: A normal CRF that can take global label transitions into account.

Training

- The added heads are trained supervised on training data.
- The training loss can also be used to fine-tune the pretrained encoder.

  Often, updating only the last few transformer blocks works best in practice.

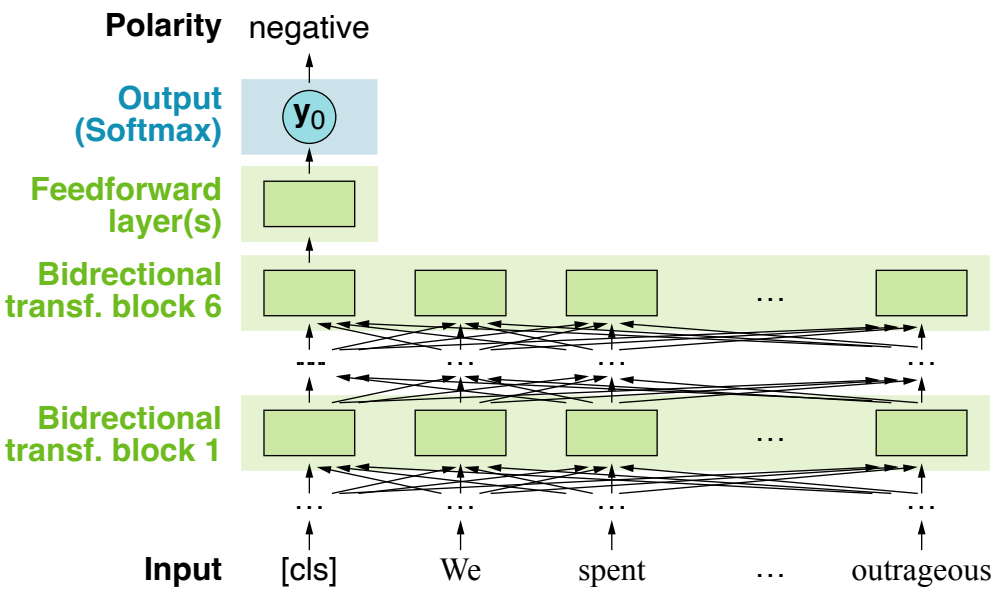
Birectional Transformers in NLP

Sentiment Analysis

Example: Sentiment analysis

- Input. A text sequence $(w_1, \ldots, w_n)$ with prepended tag $w_0 = \texttt{[cls]}$
- Output. The probability distribution over all sentiment polarities

  Shown here: Polarity with highest probability

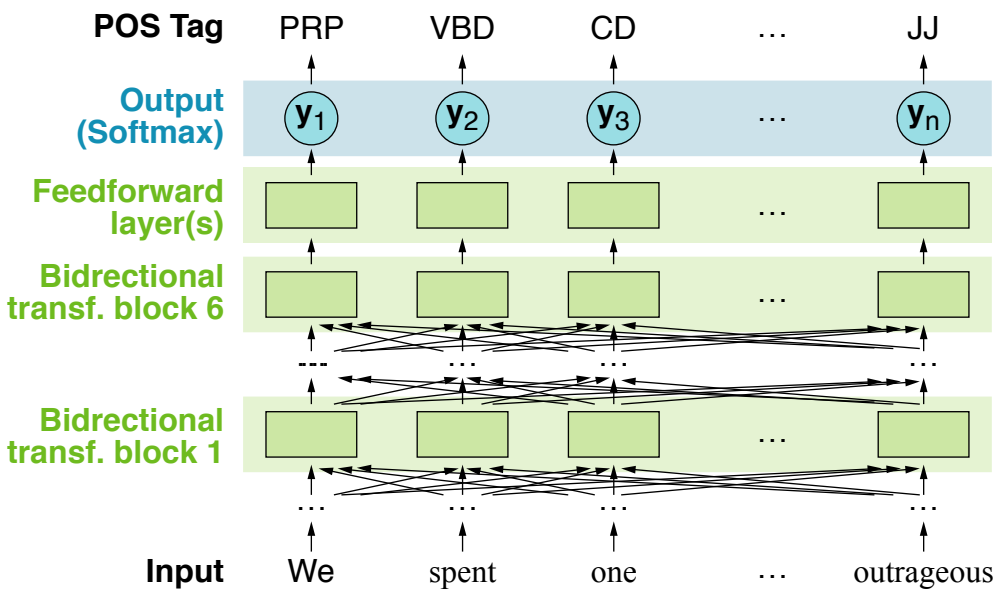


Birectional Transformers in NLP

Part-of-speech tagging

Example: Part-of-speech tagging

- Input. A text sequence $(w_1, \ldots, w_n)$
- Output. The probability distribution over all tags for each word w_j

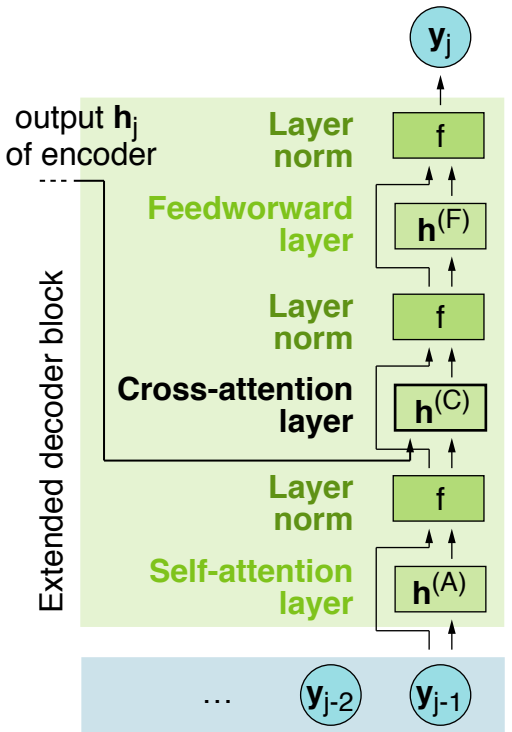


Encoder-Decoder Transformers

Encoder-Decoder Transformers

Encoder-decoder transformer (Vaswani et al., 2017)

- A transformer that combines a bidirectional encoder with a left-to-right decoder
- It maps an input sequence $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to an output sequence $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_k)$.



Extended decoder blocks

- The encoder's output is one representation $\mathbf{h}_j$ for each $\mathbf{x}_j$ of X.
- To attend to $\mathbf{h}_j$, each decoder transformer block has an extra *cross-attention* layer $\mathbf{h}^{(C)}$.

Cross-attention (aka source attention)

- The query $\mathbf{q}_j$ is the previous output $\mathbf{y}_{j-1}$ (or its earlier representation).
- The key $\mathbf{k}_j$ and value $\mathbf{v}_j$ come from the output $\mathbf{h}_j$ of the encoder:

  The rest is identical to the multi-head self-attention from above.

$$\mathbf{q}_j := \mathbf{W}^{(Q)} \cdot \mathbf{y}_{j-1} \qquad \mathbf{k}_j := \mathbf{W}^{(K)} \cdot \mathbf{h}_j \qquad \mathbf{v}_j := \mathbf{W}^{(V)} \cdot \mathbf{h}_j$$

Encoder-Decoder Transformers

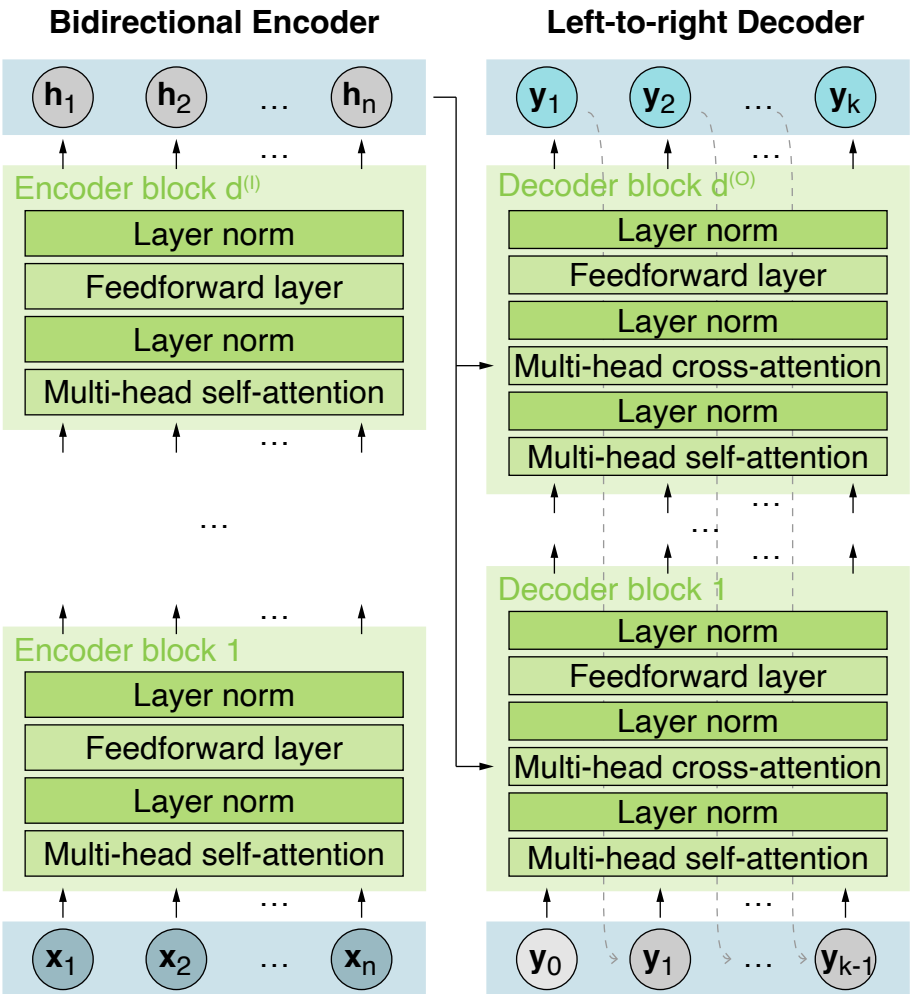
Architecture

Bidirectional encoder

- Input. $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
- $d^{(I)}$ transformer blocks compute a contextual representation
- Output. $H_e = (\mathbf{h}_1, \ldots, \mathbf{h}_n)$

Left-to-right decoder

- Input. H_e and $\mathbf{y}_0$ for a unique start tag `[sep]`
- $d^{(O)}$ transformer blocks create the output, primed on H_e, autoregressively
- Output. $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_k)$



Encoder-Decoder Transformers in NLP

Sequence-to-sequence generation

- Task. Given an input text $D^{(I)}$, write an output text $D^{(O)}$ of a certain kind
- For open-ended outputs, left-to-right transformers prove best so far.
- If $D^{(O)}$ must fulfill defined constraints, encoder-decoders may be better.

Selected sequence-to-sequence tasks

- Text summarization. As defined above
- Machine translation. Convert a text from one language to another.
- Style transfer. Change the style of a text while preserving its content.
- Conclusion generation. Infer an argument's claim from its reasons.
- Debiasing. Rewrite a text into a version free of bias.

Available transformer models

- Left-to-right. GPT-x, LLaMA, Gemma, Mixtral, ...
- Bidirectional. BERT, XLNet, RoBERTa, DeBERTA, ...
- Encoder-decoder. BART, Pegasus, T5, Flan-T5, ...

https://flickr.com

https://deviantart.com

Large Language Models

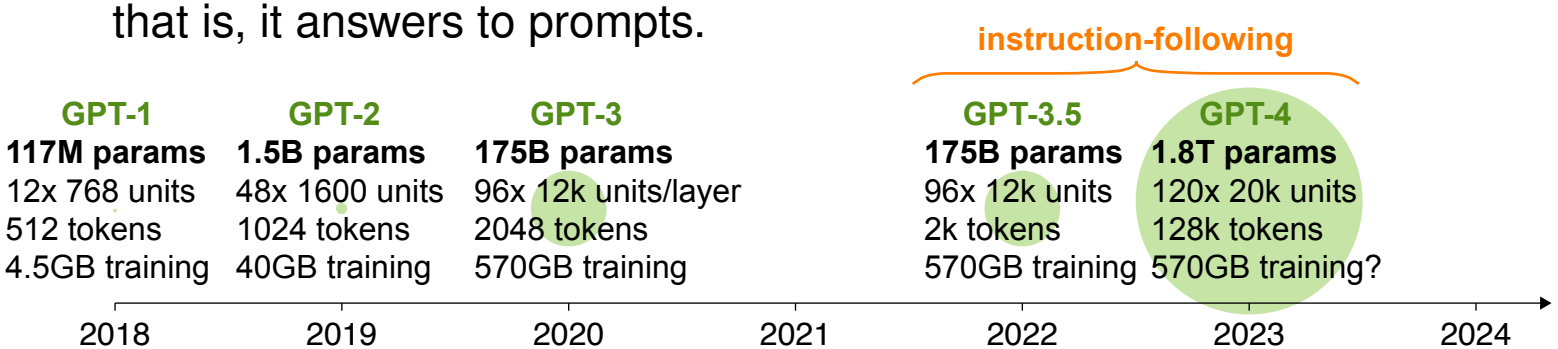
Large Language Models

Recap: Language model (LM)

- A representation of a probability distribution over token sequences
- LMs assign a probability $P(S)$ to any sequence $S = (w_1, \ldots, w_m)$, $m \geq 1$.
- This can be used to predict the most likely next word w_{m+1}.

Large language model (LLM)

- *Large* is not exactly defined, but most LLMs have billions of parameters.
- Mostly, a pretrained transformer LM is meant that *follows instructions*, that is, it answers to prompts.



Key aspects of LLMs

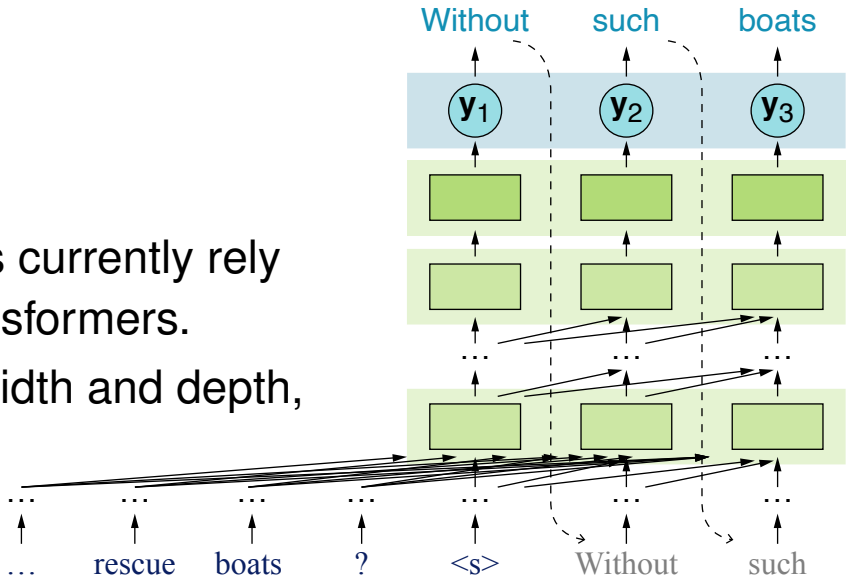
- Architecture, training process, input handling, and output handling

Large Language Models

Architecture

Architecture of LLMs

- At their core, all leading LLMs currently rely on variants of left-to-right transformers.
- Main differences arise from width and depth, context size, and training.



Size of LLMs

- *Size* refers to the number of parameters of the LLM (i.e., its weights).
- The more parameters, the more can be learned and memorized.
- "Overparameterization" is assumed to be critical for the power of LLMs.

Architecture variants

- Modified transformers, such as a *mixture of experts*

  Multiple feedforward blocks per layer, each seeing different tokens (Jiang et al., 2024)
- Advanced architectures, such as a *backpack language model*

  Mixes static and contextual vectors for explainability and control (Hewitt et al., 2023)

Training of LLMs

Main training phases of LLMs

1. Pretraining. Train LLM on language modeling
2. Instruction fine-tuning. Teach LLM to follow instructions (see below)
3. Alignment. Optimize LLM behavior towards specific values (see below)
4. (Task-specific) Fine-tuning. Optimize LLM for a specific task

Instruction vs. task-specific fine-tuning

- Instruction. Aims to generalize task-solving capabilities across tasks
- Task-specific. Aims to specialize an LLM towards solving a specific task

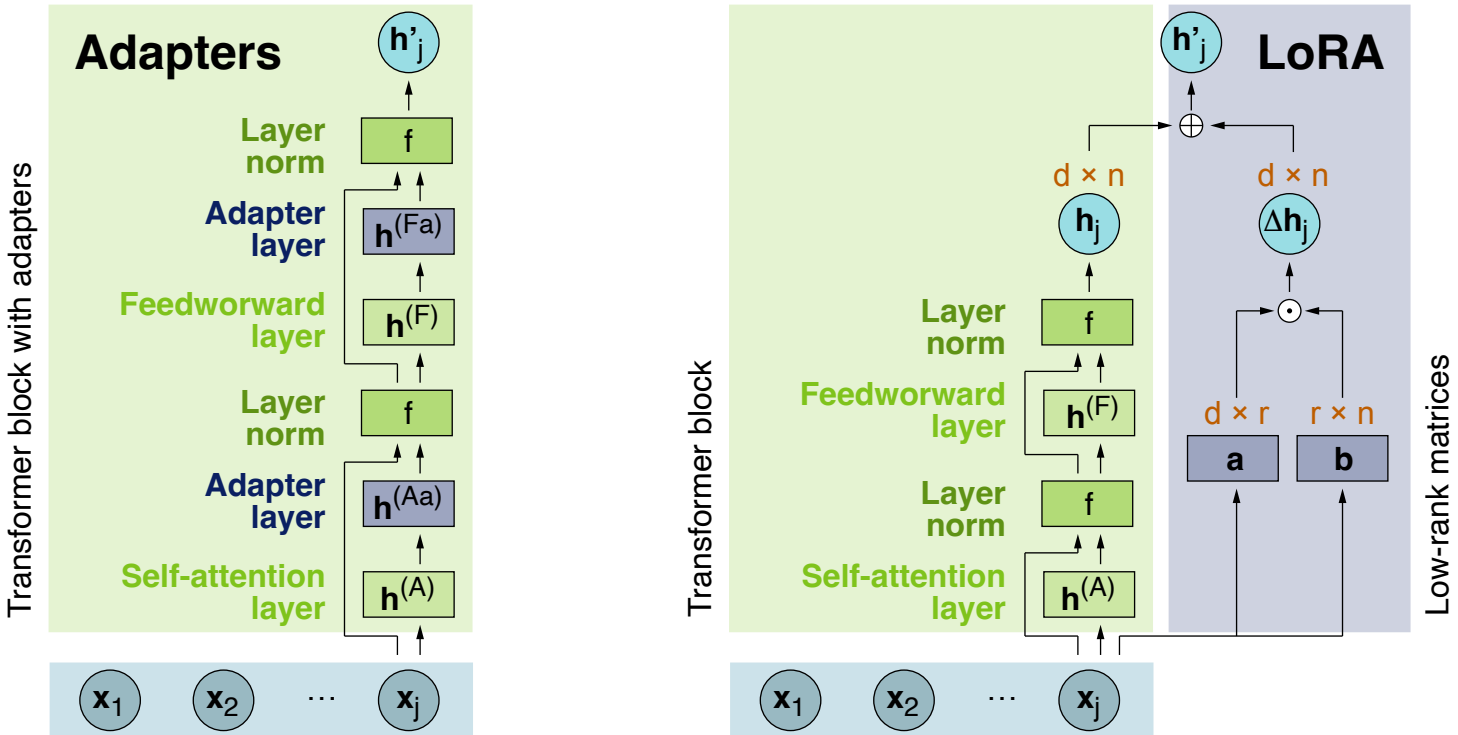
  An LLM may run through both, but the latter may have negative effects on the former.

Types of LLM fine-tuning

- Full fine-tuning. Adjust all transformer weights (may take long).
- Adapters. Train extra layers added to each transformer block.
- Low-Rank Adaptation (LoRA). Train two extra low-rank matrices in each layer, whose product is added to the original weights.

Training of LLMs

Parameter-Efficient Fine-Tuning (Houlsby et al., 2019; Hu et al., 2021)



Parameter-efficient fine-tuning

- Adapters. Useful when an LLM shall be fine-tuned for multiple tasks
- LoRA. Useful for general LLM fine-tuning and even more efficient

Instruction Fine-Tuning

Tasks and instructions (for LLMs)

- Task. A tuple (I, X, Y) of instruction I, input X, and correct output Y
- Instruction. A token sequence I describing how to obtain Y from X.

  For brevity, no distinction here between tokens and their embedding representations

Instruction: Label the input as having positive or negative sentiment towards its subject.

Input: Staff at the front desk was very nice

Output: positive

Instruction fine-tuning (IFT)

- A supervised training technique that teaches LLMs to follow instructions
- It fine-tunes LLMs on tasks (I, X, Y) with input (I, X) and output Y.

Why instruction fine-tuning?

- IFT enables prompting, i.e., making LLMs answer to any instruction.
- *Instruction-following LLMs* can even tackle unseen tasks without any more fine-tuning, often effectively.

Instruction Fine-Tuning

Training Data and Process

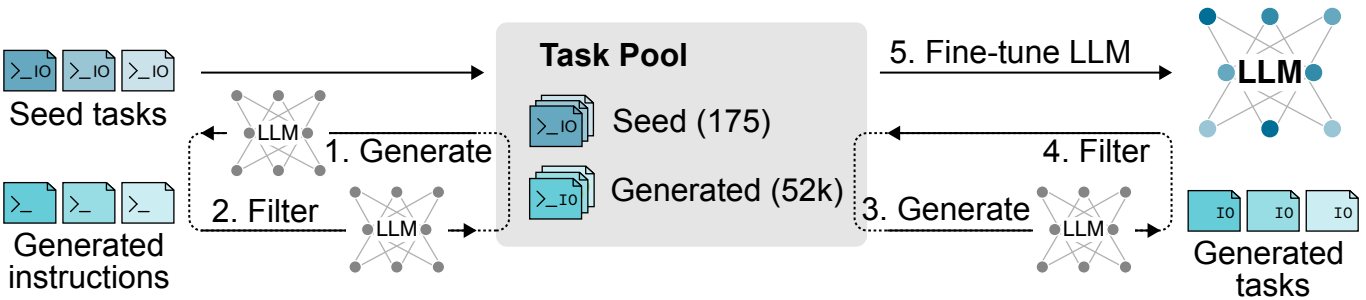
IFT training data

- The set of instructions $\{I_1, \ldots, I_m\}$ should be large and diverse.
- The set of tasks per I_j should be small but diverse (e.g., one per label).
- This way, the LLM learns to generalize across instructions rather than to address specific instructions.

How to obtain IFT training data?

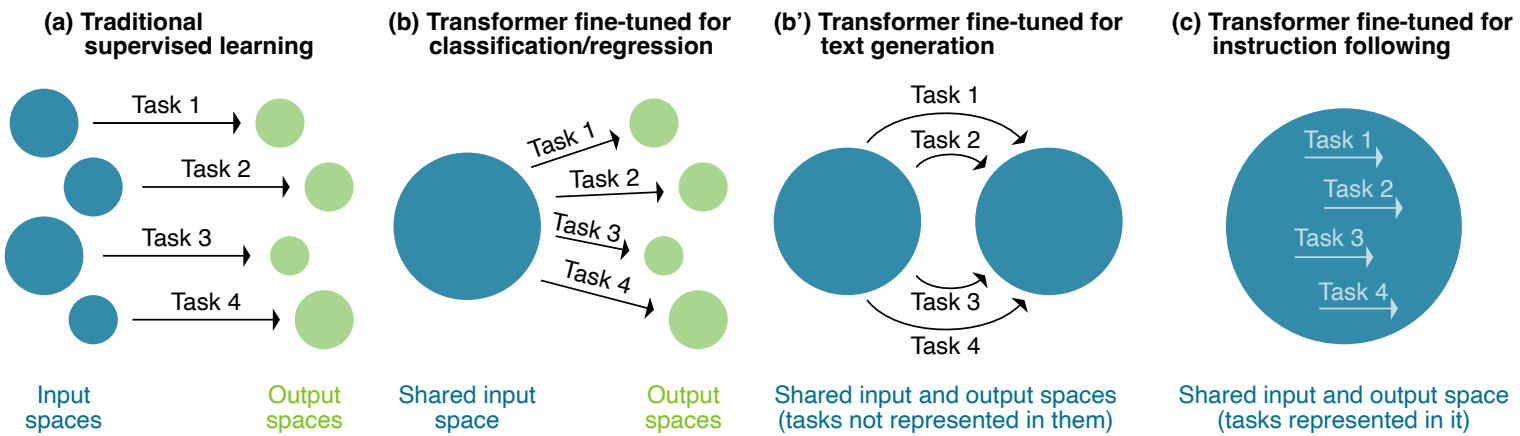
- At least, a set of seed tasks is usually created manually.
- Further tasks may be generated by other LLMs, e.g., via *self-instruct*.

Example: Self-Instruct (Wang et al., 2023)



Instruction Fine-Tuning

Impact (Wachsmuth et al., 2024)



Representations not shared across contexts or tasks	**Representations** shared across contexts only	**Representations** shared across contexts and tasks	**Representations** shared across contexts and tasks
Knowledge not shared across tasks	**Knowledge** not shared across tasks	**Knowledge** not shared across tasks	**Knowledge** shared across tasks

Alignment

Limitation of instruction fine-tuning

- IFT focuses only on optimizing language modeling towards answering.
- It does not prevent LLMs from generating false or harmful output.

Alignment

- A technique that optimizes LLM answers towards human preferences
- Preferences come from human or machine feedback on specific values.

  For general LLMs, common values are *helpfulness, honesty, and harmlessness* (3H).

Prompt: Tell me why we need rescue boats.	**Output 1:** Without, innocent refugees are killed. **Output 2:** Without, innocent refugees may die.	→ worse → better

Selected alignment techniques

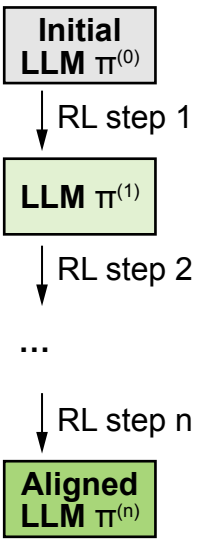
- Proximal policy optimization. Train a reward model on preference data; use reinforcement learning to maximize the reward of the LLM's output.
- Direct preference optimization. Fine-tune LLM on preference data.

  Un-alignment exists, too. Alternatives to alignment include *activation-based steering*.

Alignment

Proximal Policy Optimization (Schulman et al., 2017)

Reinforcement Learning (RL) for LLMs



- RL optimizes a *policy* π based on *rewards* given to *actions* performed in *states* in $n \gg 0$ steps.
- Policy. The LLM for the given task (e.g., generating answers)
  The LLM's parameters imply its token output probabilities for any input.
- State. The text generated up to some point in time
- Action. Appending a specific token to the text
- Reward. A score reflecting the quality of the text

Proximal policy optimization (PPO) in a nutshell

- PPO iteratively learns a *value model* based on a *reward model*.
- In each iteration i, PPO adjusts $\pi^{(i-1)}$ based on the value model.
  Changes are regularized to limit the *KL-divergence* of the token probabilities.
- Reward model. Computes scores for states, trained on preference data
- Value model. Predicts score gains of performing an action a' in a state compared to the most likely action a

Alignment

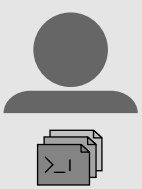
Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022)

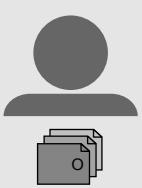
- A technique for both IFT and alignment with three main stages:

1. Instruction fine-tuning

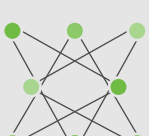
Humans write input instructions



Humans write task outputs

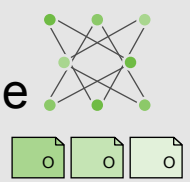


LLM is fine-tuned on tasks

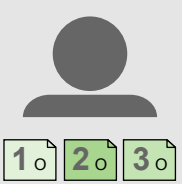


2. Reward model training

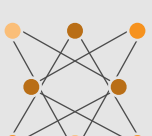
LLM writes multiple task outputs



Humans rank them by preference

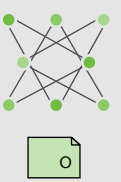


Rew. model is trained on preferences



3. PPO-based alignment

LLM writes output for instruction



Rew. model computes reward

LLM is updated using PPO

Input Handling of LLMs

Input handling

- Prompt. Text segment given to an LLM as context for output generation
  That is, the combination of instruction and (potential) input instance
- Prompting. The act of phrasing a prompt to tackle a given task
  Due to language modeling, LLMs may be sensitive to small phrasing variations.

Prompt engineering

- The tuning of prompts to maximize an LLM's effectiveness on a task
- Often, a manual process based on domain, task, or LLM knowledge.
- Common *prompting strategies* and *prompting methods* exist.

Advanced prompting methods

- Few-shot prompting. Give examples for how to solve the task.
- Retrieval-augmented generation. Add retrieved external information.
- Learning to prompt. Automatically optimize prompt on training data.
  Techniques for the latter include *prompt expansion* and *soft prompting* (left out below)

Input Handling of LLMs

General Prompting Strategies

Selected prompting strategies

- Persona. Instruct LLM to answer from the view of a specific person type.
- Task description. Specify the task to be tackled in sufficient detail.
- Definitions. Add definitions of concepts relevant to the task.
- Directional stimuli. Give hints on how the output should look like.
- Reasoning steps. Add step-by-step instructions on what to do.

  A widely used reasoning-step strategy is *chain-of-thought* (Wei et al., 2022).

Persona	Imagine you are doing the customer relationship management of a hotel, analyzing what pasts guests think about your hotel.
Task description	You should classify the sentiment polarity of this opinion: <INPUT>
Definition	An opinion is a statement that evaluates a specific aspect of the hotel.
Directional stimulus	You should output one of two label as the polarity: “positive” or “negative”.
Reasoning steps	To do so, first identify the aspect being talked about in the statement. Then, identify what sentiment is expressed towards the aspect and decide whether this is positive or negative for the hotel. The resulting label is

- Demonstrations. Give explicit examples of outputs for other inputs.

  This is known as *few-shot prompting* (see next slide).

Input Handling of LLMs

Few-Shot Prompting

Few-shot prompting

- The inclusion of $k \geq 1$ demonstrations (*shots*) of the task in the prompt
- This affects the LLM's behavior and how the output looks like.

  For contrast, using $k = 0$ demonstrations is called *zero-shot prompting*.

Shot 1
Shot 2

Opinion: the room was clean and cozy. Polarity: positive
Opinion: this alone never justifies the price. Polarity: negative
Opinion: <INPUT>. Polarity:

How many shots to use?

- The primary benefit is to demonstrate the task and output format.
- Thus, a small k is often enough (e.g., one shot per classification label).

  Too many shots may cause the LLM to overfit to details of the examples.

How to select shots?

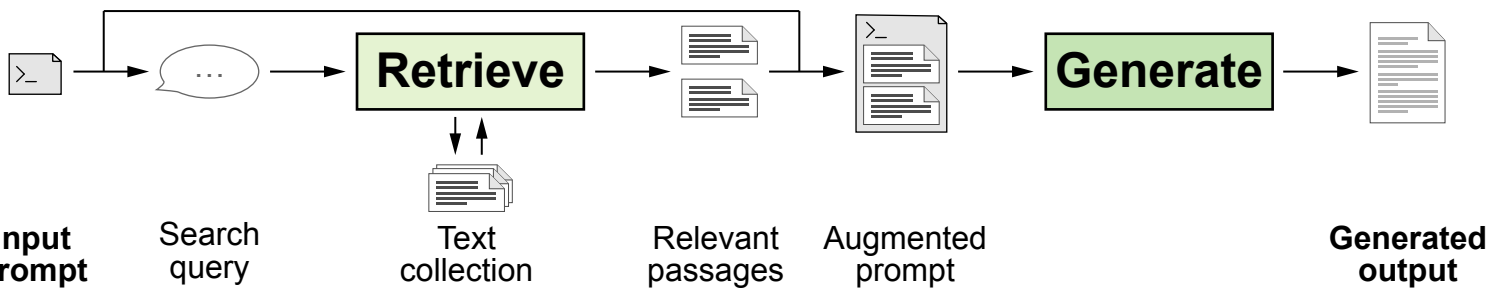
- Static. For all inputs, use the same hand-crafted or random shots.
- Dynamic. Select shots based on input (e.g., the most similar ones).

Input Handling of LLMs

Retrieval-Augmented Generation

Retrieval-augmented generation (RAG)

- Retrieve. Query a text collection for passages relevant to the prompt.
- Generate. Input the prompt to the LLM, augmented by the passages.



- The queries and prompts need to be engineered accordingly.
- The goal is to make the output source-based and more factual.

  More on factuality (and hallucinations) in Lecture Part X.

In-context learning

- An LLM's behavior to improve based on information given in the prompt
- Somewhat a misnomer: The LLM's weights are not updated thereby.

Output Handling of LLMs

Modeling tasks for LLMs

- LLMs generate token by token in the usual language modeling way.
- Generation tasks can thus be directly addressed with LLMs.
- Analysis tasks need to be cast as generation tasks via specific prompts and their interpretation.

Prompt: The sentiment of “Staff at the front desk was very nice” is: $\rightarrow$ $P(\text{“positive”} \mid \text{Prompt})$ vs. $P(\text{“negative”} \mid \text{Prompt})$

Output generation with LLMs

- What token to output next, depends on the tokens’ probabilities:

Prompt: Can you tell me an argument in favor of having rescue boats?

LLM: Without such boats, many innocent refugees will <?>

$\rightarrow$

$P(\text{“die”} \mid \text{dialogue}) = .004$
$P(\text{“drown”} \mid \text{dialogue}) = .003$
$P(\text{“suffer”} \mid \text{dialogue}) = .001$

- Diversity (or “creativity”) can be adjusted via *sampling strategies*.
  Sampling is complementary to beam search, both can be applied.

Output Handling of LLMs

Sampling Strategies

Greedy decoding

- Generate the most likely word next: $w_{m+1} = \text{argmax}_{w \in V} P(w|\ w_1, \ldots, w_m)$
- Greedy decoding behaves fully deterministically.

Top-k and top-p sampling

- Top-k. Randomly sample from the $k \geq 1$ most likely next words.
- Top-p (nucleus). Randomly sample from the minimum set of words $V(p)$ such that $\sum_{w \in V(p)} P(w|w_1, \ldots, w_m) \geq p$, with $p \in (0, 1]$.
- Both increase diversity, but top-p aims to keep likelihood stable.

Temperature sampling

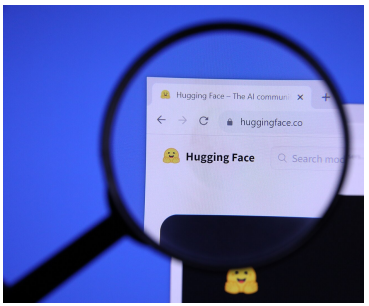
- Replace the transformer output $\mathbf{y} := \text{softmax}(\mathbf{h}^{(d)})$ by $\mathbf{y} := \text{softmax}(\mathbf{h}^{(d)}/\tau)$ with $\tau > 0$, then sample according to probability distribution.
- Low temperature ($\tau < 1$). Less diversity, as top tokens are promoted
- High temperature ($\tau > 1$). More diversity, as other tokens are promoted

Temperature sampling can be combined with top-k or top-p sampling.

Large Language Models

Overview of Available LLMs

Available LLMs



- The number of LLMs is continuously growing.
- Many but not all are provided by big tech companies.

  Often found here: `https://huggingface.co/models`

Types of LLMs

- Base. Pretrained only, mostly transformer-based
- Instruct. Instruction-following, optimized on instruction-response pairs
- Chat. Instruction-following, optimized on multi-turn dialogues

Selected LLM Families

- Paid (mostly). GPT (OpenAI), Claude (Anthropic), Pharia (Aleph Alpha)
- Open weight (partly). LLaMA (Meta), Mixtral (Mistral), Gemma (Google)
- Open source. Alpaca (Stanford), BLOOM (BigScience), LeoLM (LAION)

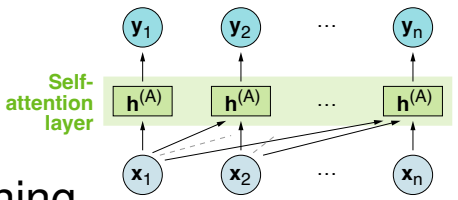
  Usually, various LLM sizes and types available; some explicitly multingual/German/...

Conclusion

Conclusion

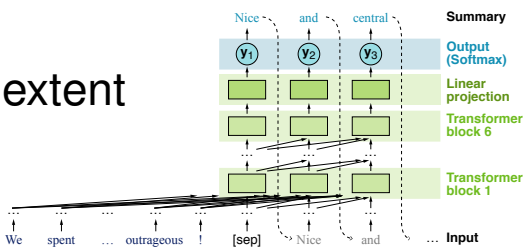
Transformer

- A neural architecture fully based on self-attention
- Types: left-to-right, bidirectional, encoder-decoder
- Transfer learning based on pretraining and fine-tuning



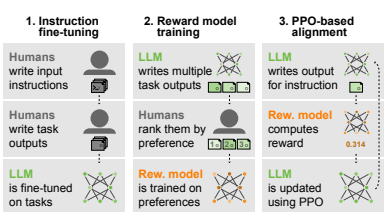
Impact of transformers

- Transformers solve context modeling to a wide extent
- Training and inference easy to parallelize
- State of the art in most NLP tasks nowadays



Large language models

- Left-to-right transformers with billions of parameters
- Instruction fine-tuned and aligned to preferences
- Prompting and sampling strategies for I/O handling



References

Much content based on

- **Jurafsky and Martin (2021).** Daniel Jurafsky and James H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. Draft or 3rd edition, December 29, 2021. https://web.stanford.edu/ jurafsky/slp3/

Other references

- **Devlin et al. (2019).** Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, 2019.
- **Houlsby et al. (2019)**. Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In Proceedings of the 36th International Conference on Machine Learning, PMLR 97:2790-2799, 2019.
- **Hu et al. (2021).** Edward J. Hu and Yelong Shen and Phillip Wallis and Zeyuan Allen-Zhu and Yuanzhi Li and Shean Wang and Lu Wang and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint arXiv:2106.09685, 2021.

References

Other references

- **Jiang et al. (2024).** Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of Experts. arXiv preprint arXiv:2401.04088, 2024.
- **Ouyang et al. (2022).** Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Advances in Neural Information Processing Systems, 2022.
- **Schulman et al. (2017).** John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
- **Vaswani et al. (2017).** Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser. Attention Is All You Need. In 31st Conference on Neural Information Processing Systems, 2017.

References

Other references

- **Wachsmuth et al. (2024).** Henning Wachsmuth, Gabriella Lapesa, Elena Cabrio, Anne Lauscher, Joonsuk Park, Eva Maria Vecchi, Serena Villata, and Timon Ziegenbein. Argument Quality Assessment in the Age of Instruction-Following Large Language Models. In Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, pages 1519–1538, 2024.
- **Wang et al. (2023).** Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 13484–13508, 2023.
- **Wei et al. (2022).** Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Advances in Neural Information Processing Systems 35, pages 24824–24837, 2022.