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

Tasks

- Language modeling
- Text summarization
- Sentiment analysis
- 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
- Conclusion

X. Practical Issues

Introduction

Transformers

Limitations of long-short term memories (LSTMs)

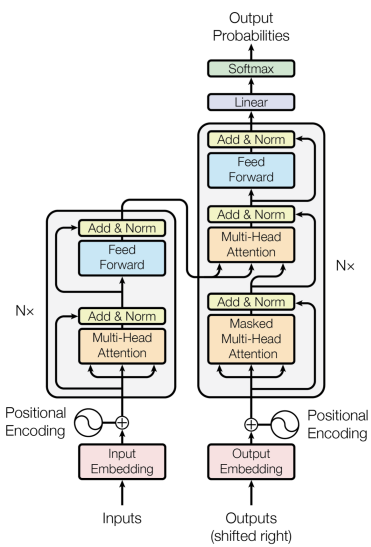
- LSTMs may still struggle with modeling long-term dependencies, since their memory is limited by the size of the hidden layer.
- Moreover, their sequential nature leaves few room for parallelization.

Idea of transformers

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

Transformer neural network

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



Vaswani et al. (2017)

Transformers

Basic Concepts

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

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

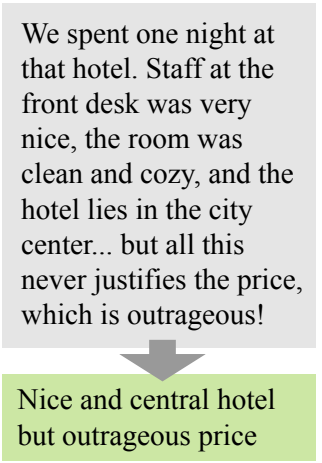
- Transformers embed a word w based on the context of the given text.
- Different contexts lead to different embeddings of w.

Transformers

Transformers in NLP

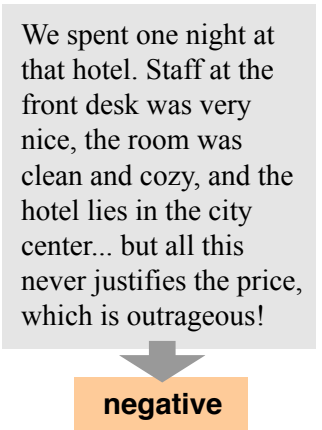
Transformers for generation tasks

- The architecture is designed for text generation.
- A left-to-right or encoder-decoder transformer is 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 is 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

Application of Transformers

Impact of transformers

- The transformer architecture is state of the art in nearly all NLP tasks.
- It does not “solve” the tasks, but often increases performance strongly.
- Any notable *large language model (LLM)* relies on transformers.

  Large not fully defined, but usually billions of parameters

Selected LLMs

- Left-to-right. GPT-1, GPT-2, GPT-3, ...
- Bidirectional. BERT, XLNet, RoBERTA, ...
- Encoder-decoder. BART, T5, ...

https://flickr.com

https://deviantart.com

Example: ChatGPT https://chat.openai.com

- A dialogue system based on GPT-3.5/-4 that answers reasonably (and often impressively) to nearly any human-written input
- Its LLM was trained to follow instructions, aligned with human values.
- Be aware that it still has clear limitations, e.g., in terms of factuality.

Left-to-Right Transformers

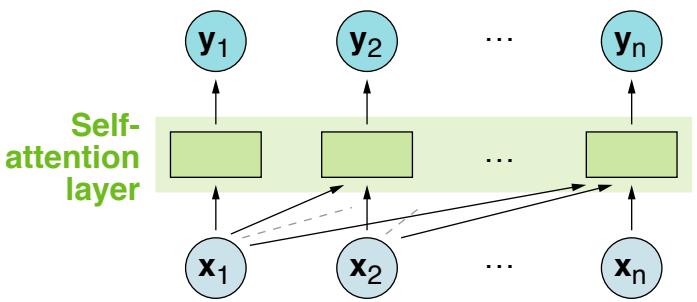
Left-to-Right Transformers

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 transformers

- 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 focus $\mathbf{x}_l$, $l \geq j$
- 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 $\mathbf{q}_j$:

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

Weight computation

- Each score may range from $-\infty$ to ∞, the larger the more similar.
- For normalization, it is often scaled based on the length m of $\mathbf{x}_k$.
- 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

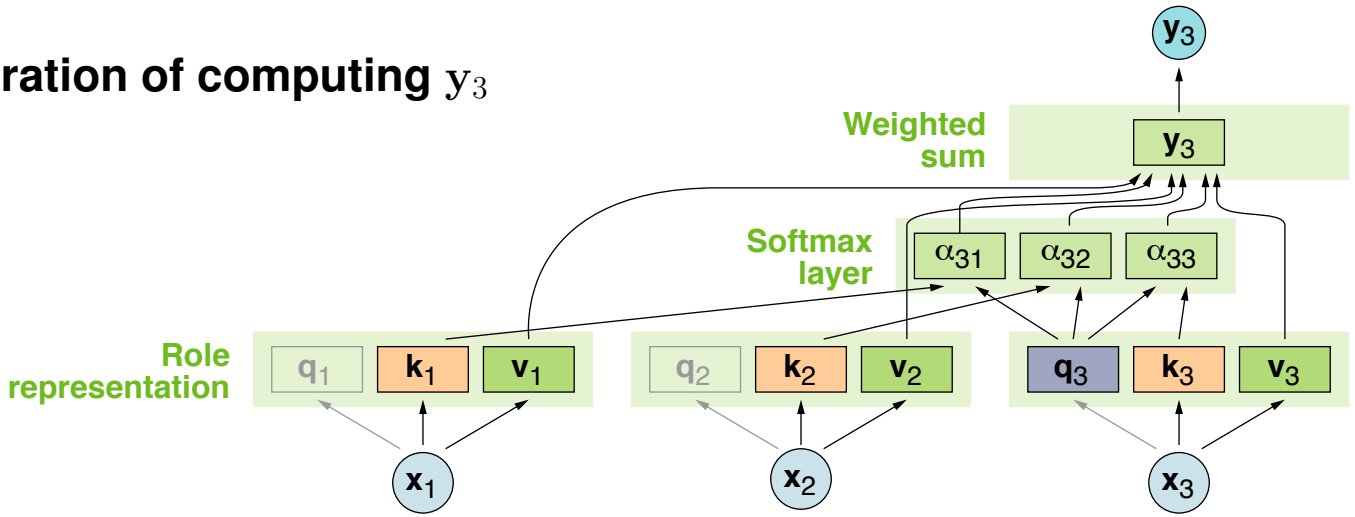
- Finally, the output $\mathbf{y}_j$ of $\mathbf{x}_j$ is the weighted sum of the values $\mathbf{v}_k$:

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

Left-to-Right Transformers

Self-Attention Layers: Efficiency

Illustration of computing $\mathbf{y}_3$



Computational efficiency

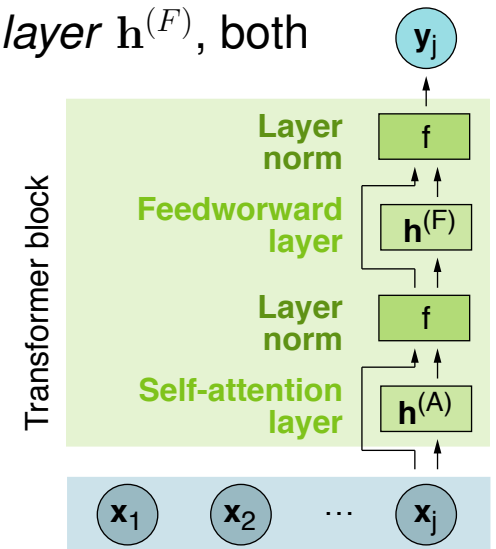
- Each output $\mathbf{y}_j \in Y$ can be computed in parallel.
- Still, each self-attention layer compares all input pairs $(\mathbf{x}_j, \mathbf{x}_k)$, which is quadratic in the length of X.
- This makes self-attention very expensive for longer inputs.
- Standard transformer libraries limit the input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

  How to deal with arbitrary lengths is an ongoing research topic.

Transformer Architecture

Transformer block

- A transformer consists of $d \geq 1$ stacked layer blocks (e.g., $d = 6$).
- Block. A self-attention layer $\mathbf{h}^{(A)}$, a *feedforward layer* $\mathbf{h}^{(F)}$, both with *layer norm* f and *residual connections*



Feedforward layer

- $\mathbf{h}^{(F)}$ maps its input to an output of the length of the block input $\mathbf{x}_j$, thus enabling stacking

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:

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

Residual connection

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

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

Transformer Architecture

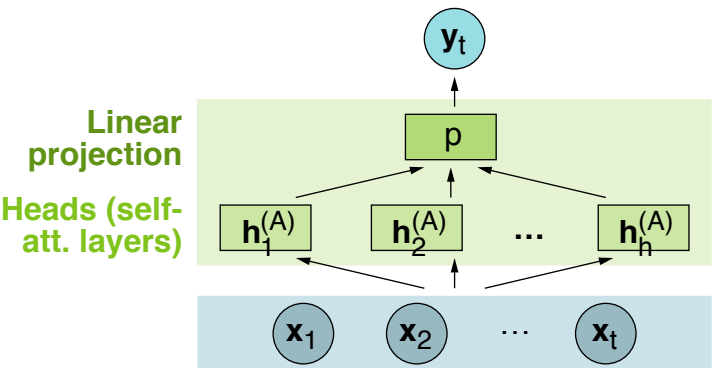
Multi-Head Self-Attention

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 p with weights $\mathbf{W}^{(P)}$:

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

Transformer Architecture

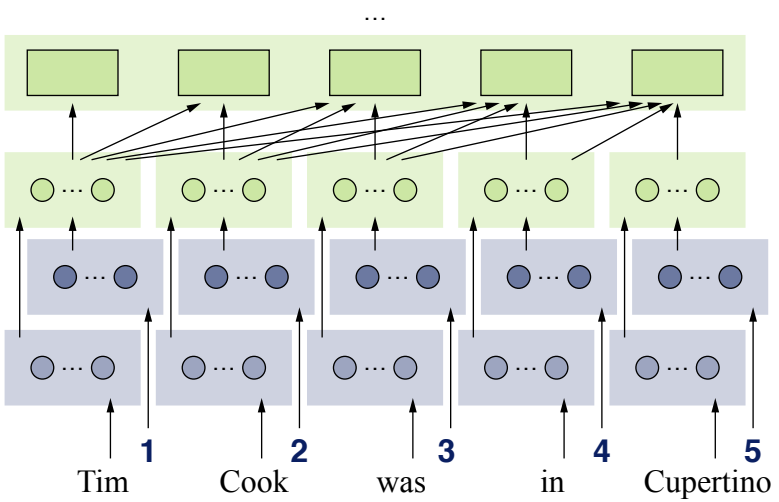
Positional Embeddings

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

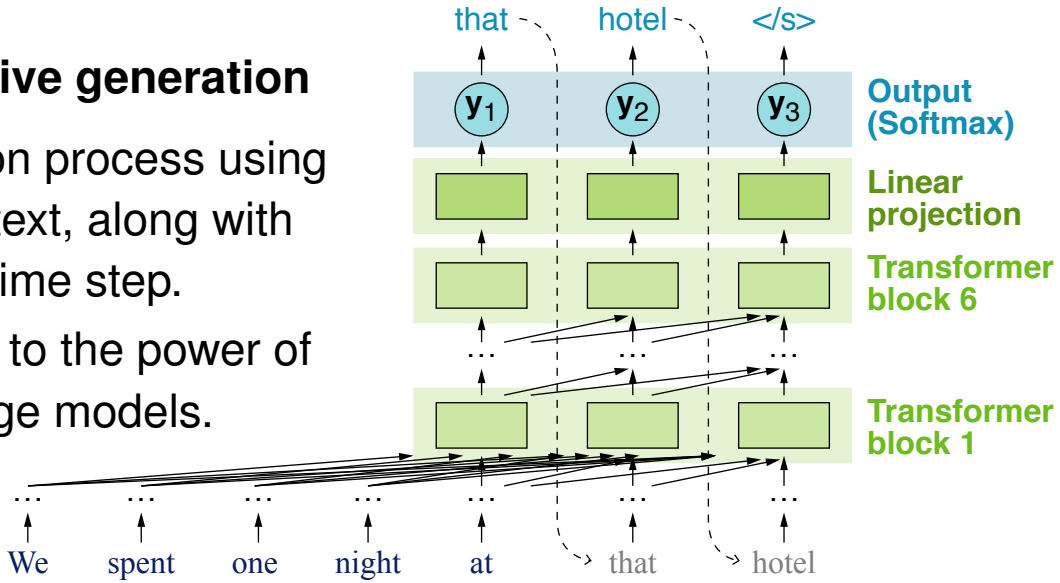
- Learning. Learn to embed j on data up to some maximum position; fewer training examples exist for later positions, though
- Static function. Map positions to embeddings in a way that captures their inherent relationship.

  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!

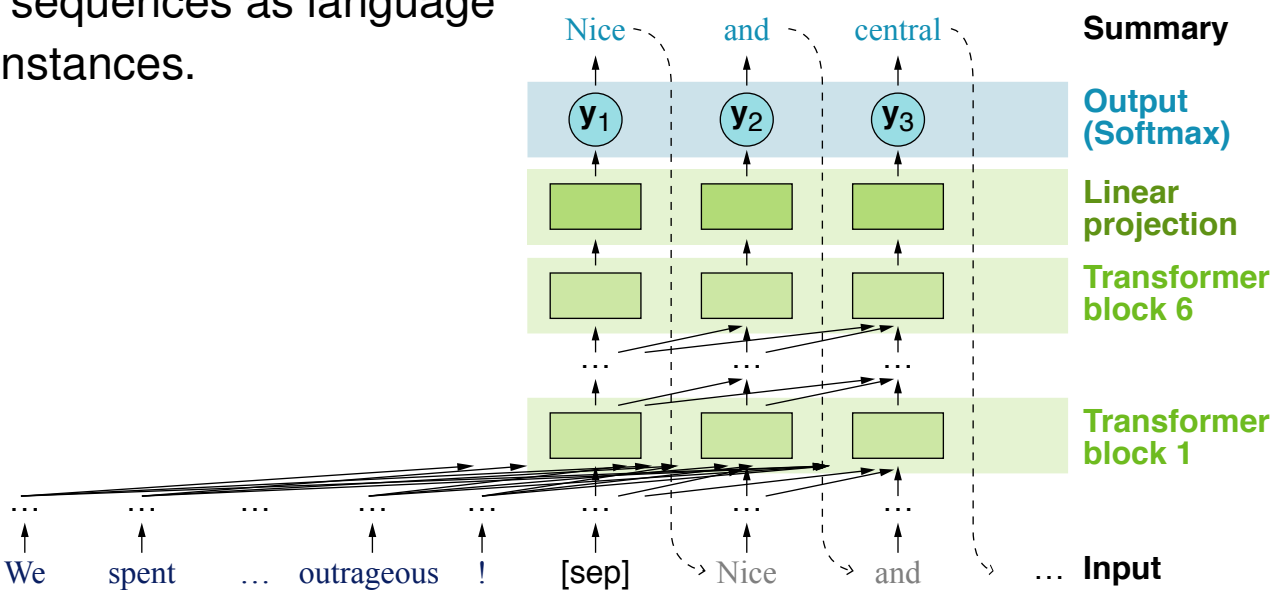
Nice and central hotel but outrageous price

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

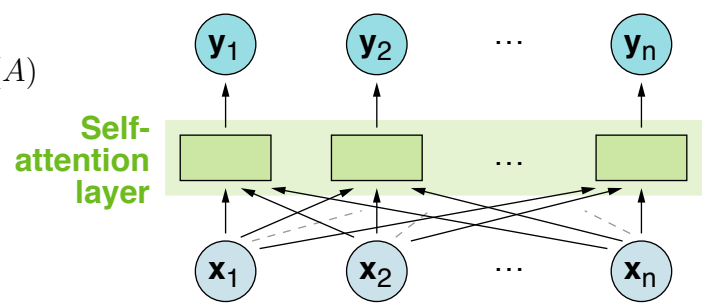
Bidirectional Transformers

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.
- Besides, most implementations rely on *subword tokenization*.

Subword tokenization

- Tokens are split into subwords that are used for all 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 sentence follow each other.

Bidirectional Transformers

Masked Language Modeling

Masked language modeling (aka the cloze task)

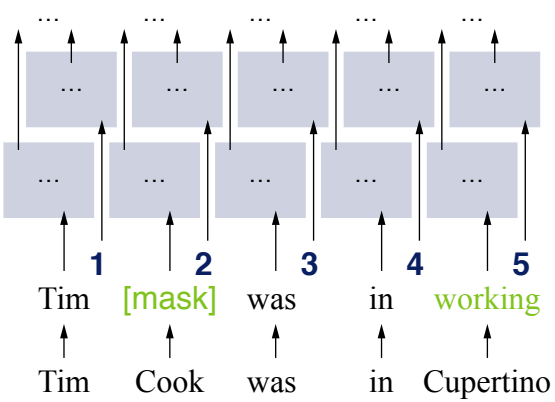
- Given a sequence of words where one or more are missing (masked), predict each missing word $\tilde{w}$.

_____ 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 the 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, each with 12 multi-head self-attention layers of size 768
- 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 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

- The 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.

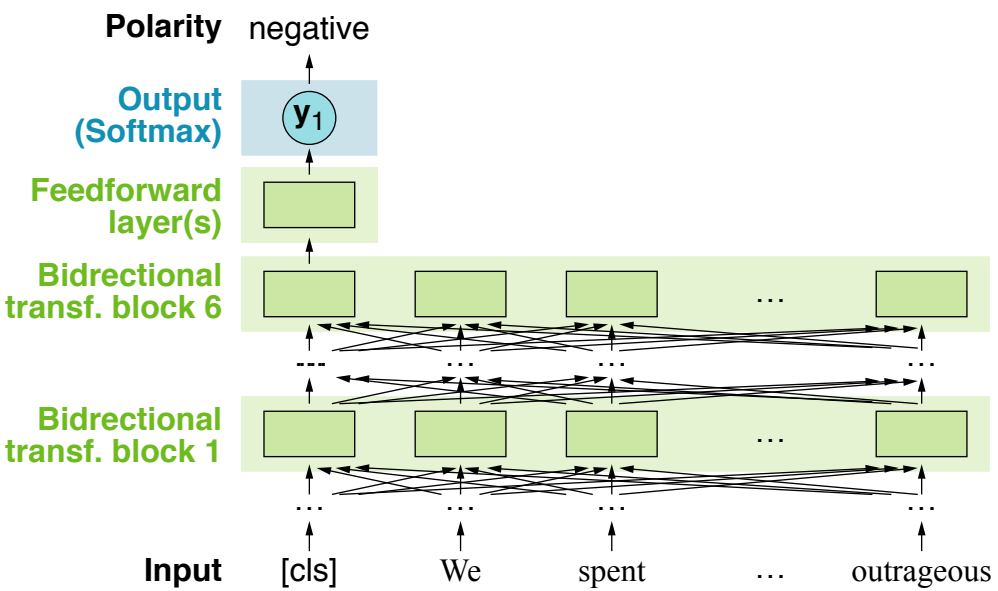
Birectional Transformers in NLP

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

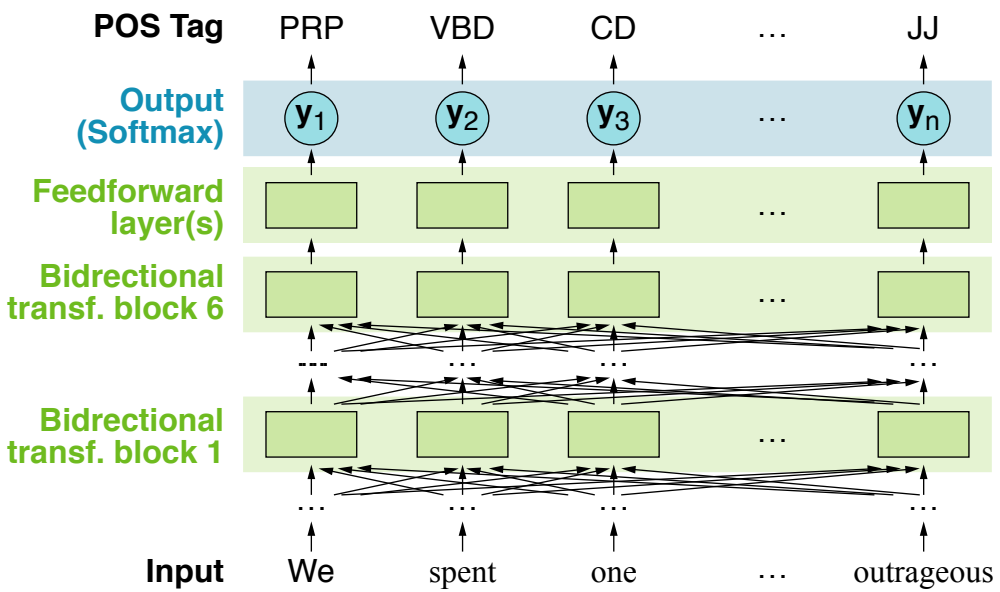


Birectional Transformers in NLP

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

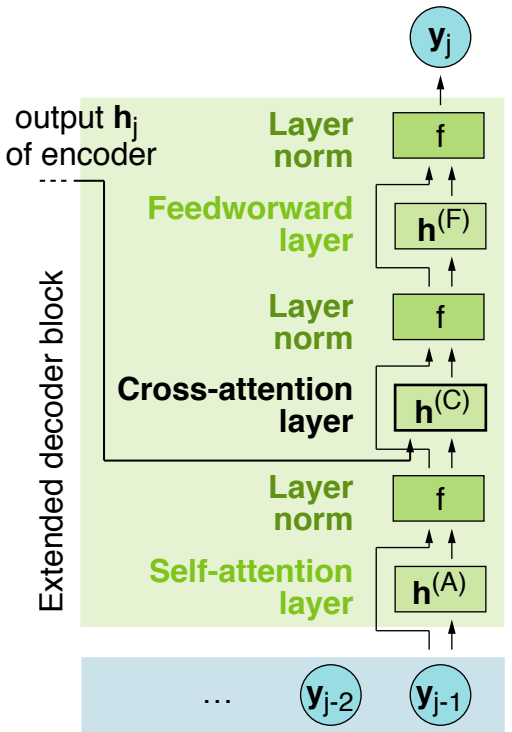


Encoder-Decoder Transformers

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 a representation $H_e = (\mathbf{h}_1, \ldots, \mathbf{h}_n)$ of X.
- Each decoder transformer block has an extra *cross-attention* layer to attend to H_e.

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 standard multi-head self-attention.

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

Encoder-Decoder Transformers

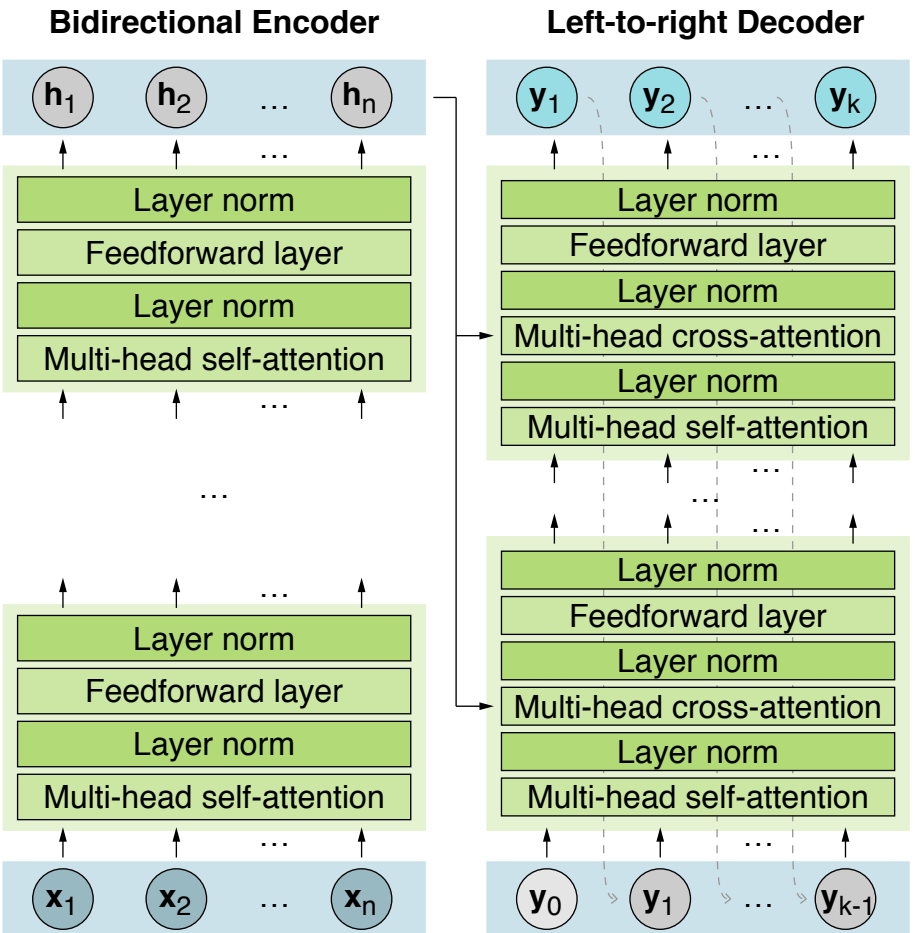
Architecture

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
- As soon as $D^{(O)}$ must fulfill defined constraints, encoder-decoders tend to be preferable

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
- Debiasing. Rewrite a text into a version free of bias
- Conclusion generation. Infer an argument's claim from its reasons

Wanna learn more?

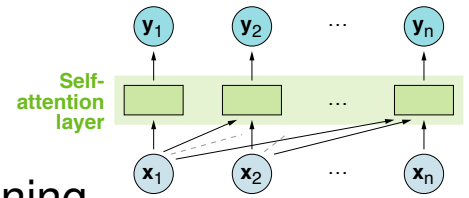
- Enroll in my summer term course "Computational Argumentation"
- Write your thesis with the NLP Group ;)

Conclusion

Conclusion

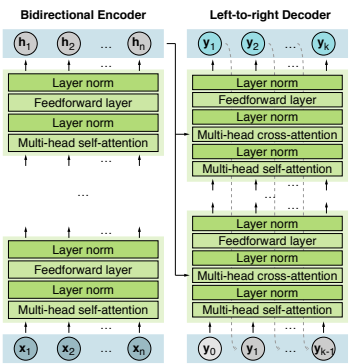
Transformer

- A neural architecture fully based on self-attention
- Training and inference easy to parallelize
- Transfer learning based on pretraining and fine-tuning



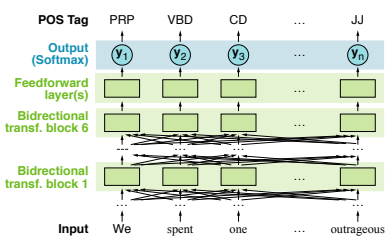
Types of transformers

- Left-to-right transformers for output decoding
- Bidirectional transformers for input encoding
- Encoder-decoder transformers for combinations



Impact of transformers

- Transformers solve context modeling to a wide extent
- State of the art in basically any NLP task nowadays
- Tools such as ChatGPT stress the real-life potential



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.
- **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.