Attention & Diffusion
Lecture 4

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

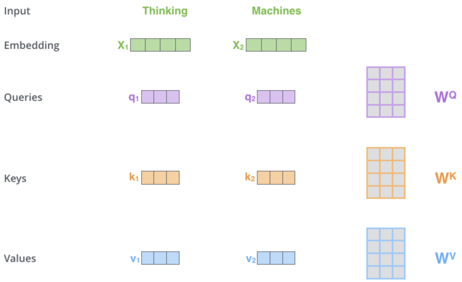
Bielefeld University

May 9, 2023

CONTENTS

Self Attention: Illustrated Reminder

Transformers: Self-Attention Reminder I

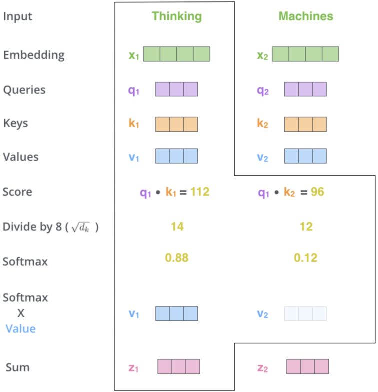


Self-attention: queries, keys and values

From `https://jalammar.github.io`

- Input vectors $\mathbf{x}_i$ are transformed to
  - queries $\mathbf{q}_i$, keys $\mathbf{k}_i$, values $\mathbf{v}_i$ by
  - applying matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ to $\mathbf{x}_i$ from the right

Transformers: Self-Attention Reminder II



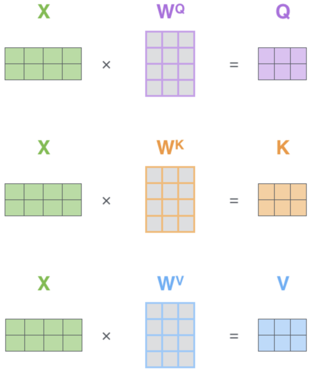
Self-attention: from input to output

From `https://jalammar.github.io`

- Scores for $\mathbf{x}_1$ w.r.t. $\mathbf{v}_1, \mathbf{v}_2$
  - $\mathbf{v}_1$: Compute $\mathbf{q}_1 \cdot \mathbf{k}_1$, divide by 8, yields 112
  - $\mathbf{v}_2$: Compute $\mathbf{q}_1 \cdot \mathbf{k}_2$, divide by 8, yields 96
- Softmax'ing: Probabilities $0.88, 0.12$ for $\mathbf{v}_1, \mathbf{v}_2$
- Final output for $\mathbf{x}_1$:

$$0.88 \cdot \mathbf{v}_1 + 0.12 \cdot \mathbf{v}_2$$

Transformers: Self-Attention Reminder III

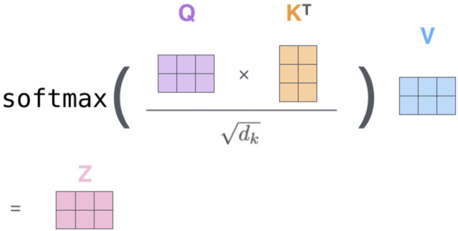


Calculating queries, keys and values

From `https://jalammar.github.io`

- Pack embedded words into matrix **X**
  - Each row corresponds to one word
- Multiply X with trained matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$
- Recall real dimensions:
  - Words: 512 (here: 4); **Q**, **K**, **V**: 64 (here: 3)

TRANSFORMERS: SELF-ATTENTION REMINDER IV

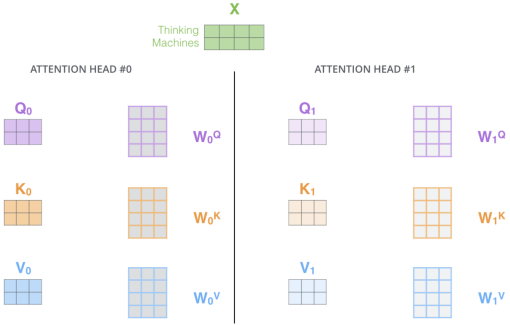


Computing values: compact matrix representation

From `https://jalammar.github.io`

1. Multiply queries with keys: $\mathbf{Q} \cdot \mathbf{K}^T$
2. Normalize relative to query/key length d_k ($= 64$ in reality)
3. Softmax across columns: $\mathbf{S} := \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d_k})$ (here: $\in \mathbb{R}^{2\times 2}$)
4. Compute weighted sum for each word: $\mathbf{Z} = \mathbf{S} \cdot \mathbf{V}$

Transformers: Multi-Head Attention Reminder I

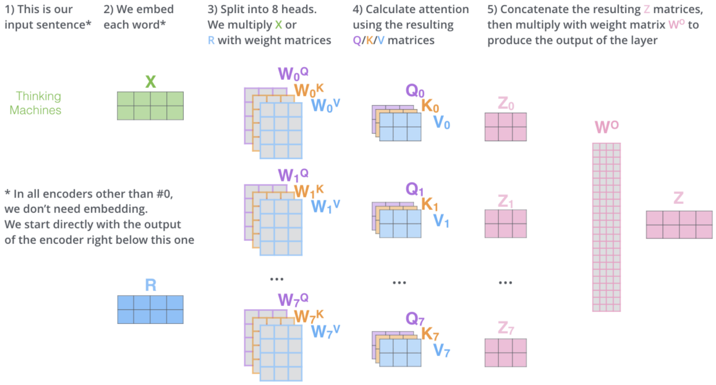


Multi-head attention with 2 heads

From `https://jalammar.github.io`

- Learn several "heads", attending to different interactions
  - Learn several different $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^K$ (here: $i = 2$)
  - Establish differences by randomized initialization

Transformers: Multi-Head Attention Reminder III



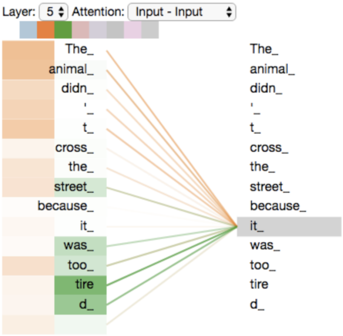
Multi-head attention: Overview / Summary

X: embedded words, input for first attention layer

R: output of earlier layer input for all but first layer

From `https://jalammar.github.io`

TRANSFORMERS: MULTI-HEAD ATTENTION REMINDER IV

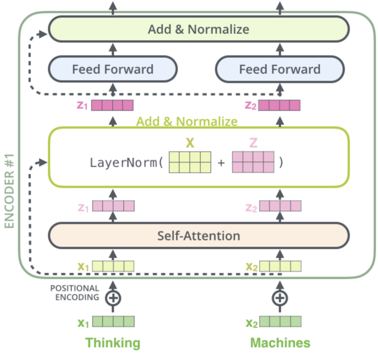


Multi-head attention:
Considering two (of eight) heads

From `https://jalammar.github.io`

- Considering two attention heads, orange and green
- *Orange:* "it" mostly attends to "the animal"
- *Green:* "it" mostly attends to "tired"

Transformers: Encoder Details

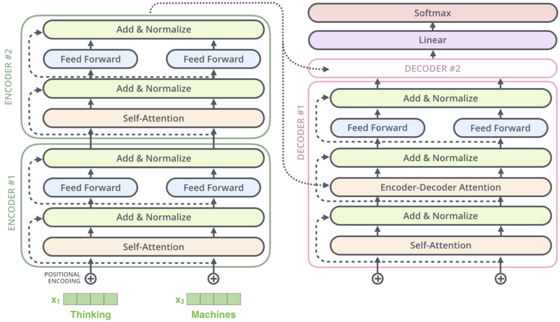


Transformer encoder block: details

From `https://jalammar.github.io`

1. Embedded words are equipped with positional encodings
2. Self attention is applied
   - 2.1 Original $\mathbf{x}_i$ is added to $\mathbf{z}_i$
     ☞ Residual skip connection
   - 2.2 Layer norm is applied
     ☞ Normalizes values across layer
3. Each resulting $\mathbf{z}_i$ passed through identical feedforward NN (FFNN)
   - 3.1 Original $\mathbf{z}_i$ added to FFNN output
     ☞ Residual skip connection
   - 3.2 Layer norm is applied
     ☞ Normalizes values across layer

TRANSFORMERS: ENCODER-DECODER INTERACTION



Transformer with two encoder and two decoder blocks

From `https://jalammar.github.io`

- Decoder blocks integrate encoder-decoder attention layers
  - Between decoder self attention and FFNN layer
  - Encoder output transformed into keys and values
  - Decoder output transformed into queries

TRANSFORMER: DECODER I

From `https://jalammar.github.io`

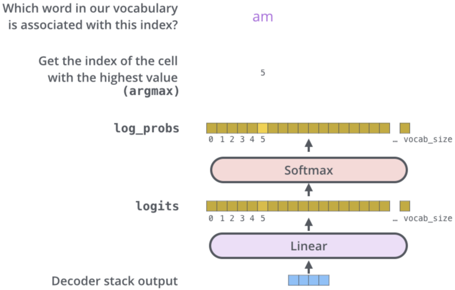
1. Encoder processes input sequence (here: with positional encoding)
2. Output of top encoder transformed into keys $\mathbf{K}_{\text{encdec}}$ and values $\mathbf{V}_{\text{encdec}}$
3. Decoder uses $\mathbf{K}_{\text{encdec}}$ and $\mathbf{V}_{\text{encdec}}$ in encoder-decoder attention layer

Transformer: Decoder II

From `https://jalammar.github.io`

1. Decoder takes in already generated tokens (words)
2. Self-attention: decoder only attends to already generated tokens
   - Achieved by masking future positions
3. Encoder-decoder attention layer generates its own queries
   - but uses keys and values from topmost encoder output

Transformers: Decoder Final Layer



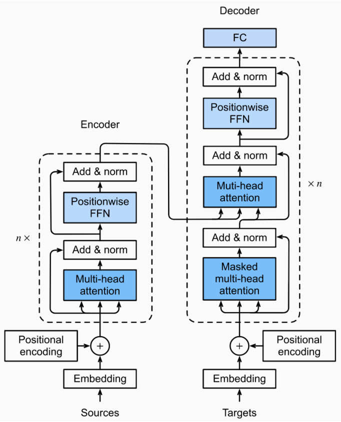
Transformer decoder: final layer consists of linear and softmax sublayer

From `https://jalammar.github.io`

- Linear layer takes decoder output, computes a value for each word
  - See *logits* layer in figure; number of words equal to size of vocabulary
- Softmax layer turns values into probabilities
  - Yields *log_probs* layer; word with greatest probability is output

Transformer Variants

Transformers: Architecture Summary I



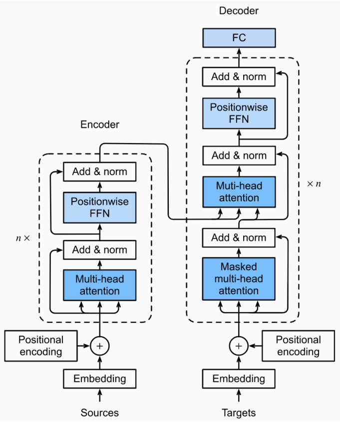
Transformer: Summary.
n encoder and n decoder layers

From `https://jalammar.github.io`

Encoder

- Both encoder and decoder consist of n layers
  ☞ original paper: $n = 6$
- Stacks identical layers
- Each layer has two sublayers
  - Multi-head attention layer
  - Positionwise feedforward neural network
- Contains skip connections
  ☞ inspired by ResNet

Transformers: Architecture Summary II



Transformer: Summary.
n encoder and n decoder layers

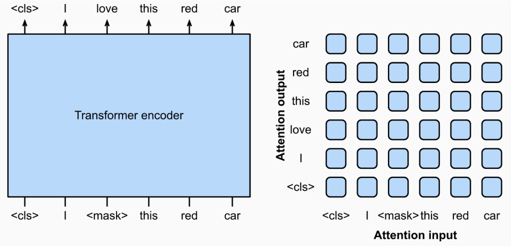
From `https://d2l.ai`

Decoder

- Also stacks identical layers
- Each layer: three sublayers
  - Multi-head self attention
  - Encoder-decoder attention
  - Positionwise feedforward neural network
- Encoder-decoder attention does not exist in encoder
- Contains skip connections
  ☞ inspired by ResNet
- Each position only attends to earlier positions
  - *Masked* attention preserves autoregressive property

Transformer Variants: Encoder Only

Transformer Variants: Encoder Only I

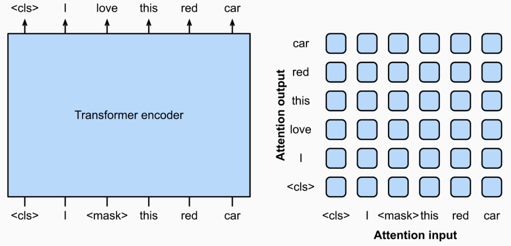


Transformer encoder only: pretraining

From `https://d2l.ai`

- *Attention input* $\leftrightarrow$ embedded words $\mathbf{x}_i$
- *Attention output* $\leftrightarrow$ new "words" $\mathbf{z}_i$
- Meaning right panel: each $\mathbf{x}_i$ contributes to each $\mathbf{z}_i$

Transformer Variants: Encoder Only II

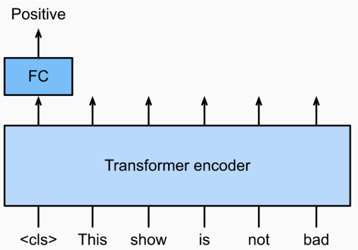


Transformer encoder only: pretraining

From `https://d2l.ai`

- Prominent example: *Bidirectional Encoder Representations from Transformers (BERT)*, see `https://arxiv.org/abs/1810.04805`
- *Pretraining* supposed to pick up basic language structure
- *Principle:* Learn masked words in sentences

Transformer Variants: Encoder Only III



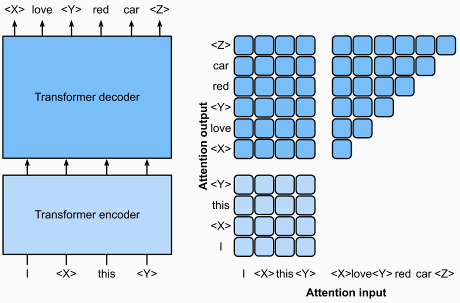
Transformer encoder only: finetuning for sentiment analysis

From `https://d2l.ai`

- After pretraining, encoder-only transformer is *finetuned*
  - Involves different kind of training
- *Example:* Sentiment analysis
  - Predicting sentiments inherent to sentences
- *Principle:* Use final representation of special token $< \text{cls} >$

Transformer Variants: Encoder-Decoder

Transformer Variants: Encoder-Decoder I

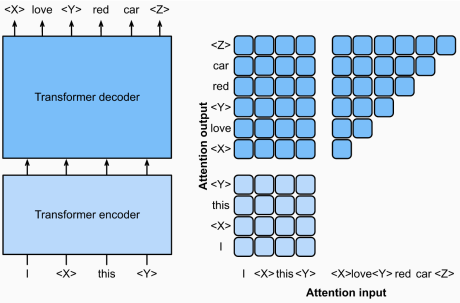


Transformer encoder-decoder: pretraining

From `https://d2l.ai`

- *Advantage:* Output can vary in length
- Prominent example: *T5*, see `https://arxiv.org/abs/1910.10683`

Transformer Variants: Encoder-Decoder II

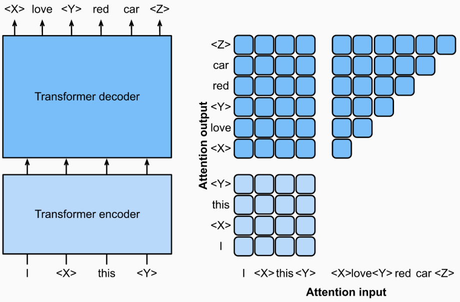


Transformer encoder-decoder: pretraining

From `https://d2l.ai`

- *Pretraining Example:* Predict consecutive spans
- *Here:* Replace "$\langle X\rangle$" with "$\langle X\rangle$ love" and "$\langle Y\rangle$" with "$\langle Y\rangle$ red car"

Transformer Variants: Encoder-Decoder III

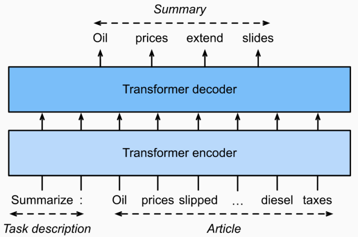


Transformer encoder-decoder: pretraining

From `https://d2l.ai`

- *Encoder:* Each input token attends to each other
- *Decoder:* Target tokens attend to
  - all input tokens (*encoder-decoder attention*)
  - only past and present target tokens (*causal attention*)

Transformer Variants: Encoder-Decoder IV



Transformer encoder-decoder: Finetuning for generating text summaries

From `https://d2l.ai`

- After pretraining, encoder-decoder transformer is *finetuned*
  - Involves different training principle
- *Example:* Summarization of large texts
  - *Input:* Task description and large text
  - *Output:* Brief summary of large text

Transformer Variants: Encoder-Decoder V

Teddy bears swimming at the Olympics 400m Butterfly event.

A cute corgi lives in a house made out of sushi.

A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.

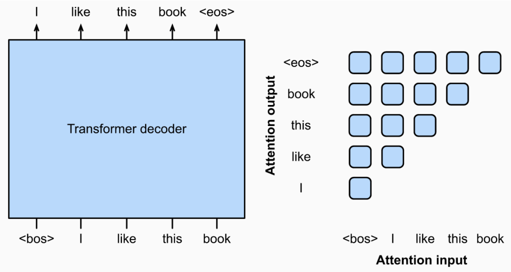
Imagen, based on T5 encoder: Turning texts into images

From `https://d2l.ai`

- Generate image that reflects text contents
- Text-to-image model "Imagen", see `https://arxiv.org/abs/2205.11487`
- Imagen based on "frozen" T5 encoder

Transformer Variants: Decoder Only

Transformer Variants: Decoder Only I

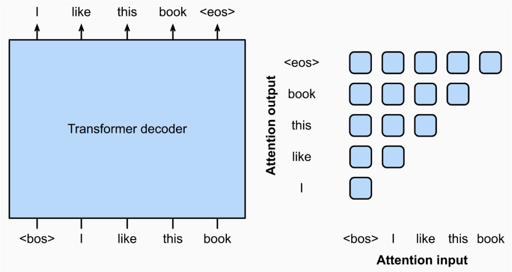


Transformer decoder only: pretraining

From `https://d2l.ai`

- De facto architecture in large-scale language modeling
- Encoder-decoder attention sublayers removed
- *Pretraining:* Teacher forcing
  - Target sequence is input sequence shifted by one token

TRANSFORMER VARIANTS: DECODER ONLY II

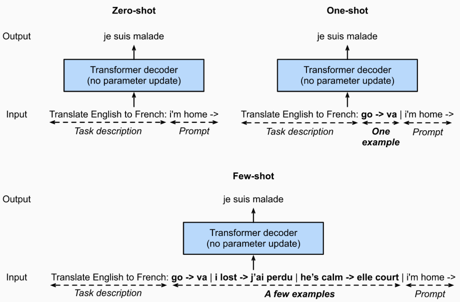


Transformer decoder only: pretraining

From `https://d2l.ai`

- *Self-supervised learning:* Learns structures in unlabeled data
  - Leverages abundantly existing, unlabeled text corpora
- Prominent example: *GPT-3*, see `https://arxiv.org/abs/2005.14165`
  - Basis of *ChatGPT*, for example

TRANSFORMER VARIANTS: DECODER ONLY III

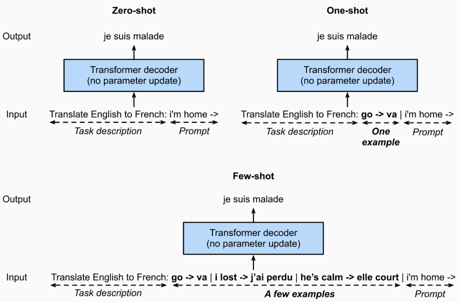


Transformer decoder only: finetuning

From `https://d2l.ai`

- GPT-2 demonstrated that model can be re-used for other tasks
  - without parameter re-training / updating (!), so no finetuning
- GPT-3 exploits the *in-context learning* principle further

Transformer Variants: Decoder Only IV



Transformer decoder only: finetuning

From `https://d2l.ai`

- In-context learning requires task description and prompt, as task input
- In addition, in-context learning may involve no examples (*zero-shot*), one example (*one-shot*) or few examples: *few-shot*

References

- Jay Alammar's ML blog:
  - "Visualizing A Neural Machine Translation Model", see `https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attentio`
  - "The Illustrated Transformer", see `https://jalammar.github.io/illustrated-transformer/`
- `http://d2l.ai`, 11.4–11.7, 11.9

Thanks for your attention!!