Attention & Diffusion
Lecture 3

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University

April 25, 2023

CONTENTS

- Transformers
    - Encoder Structure (today)
    - Self Attention (today)
    - Decoder Structure (next lecture)
- Transformer Versions and Training
    - Encoder Only (next lecture)
    - Encoder-Decoder (next lecture)
    - Decoder Only (next lecture)

Transformers I

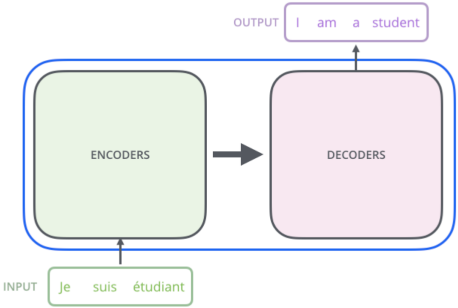
Transformers: Motivation

Transformers translate “languages”

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

- Inspiration for transformers: translating languages
- Transformers lend themselves to (maximum) parallelization
- Google: reference model for cloud TPU based computations

TRANSFORMERS: MOTIVATION II

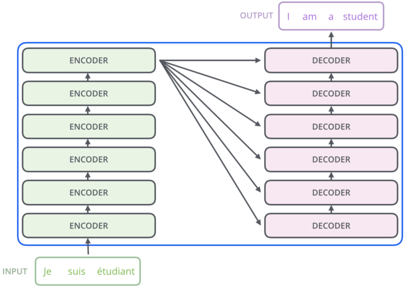


From `https://jalammar.github.io`

- Transformers employ encoder-decoder architecture
- However, neither encoder nor decoder RNN based
- *Seminal paper:* "Attention is all you need"

☞ `https://arxiv.org/abs/1706.03762`

Transformers: Structure I

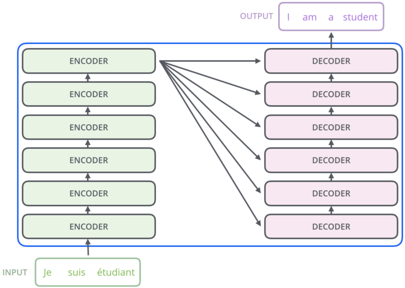


Transformers: encoders and decoders layer structured

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

- Transformers make use of stacks of encoders and decoders
  - Seminal paper: stacks are 6 layers each
  - Other numbers very well conceivable
  - Architectural design may vary by application

TRANSFORMERS: STRUCTURE II

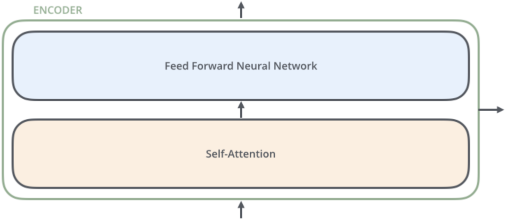


Transformers: encoders and decoders layer structured

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

- Encoders and decoders interact in different ways
  - All but last encoder provide input to next encoder
  - Last encoder provides input to all decoders
  - All but last decoder provide input to next decoder
  - Last decoder outputs translated sentence

TRANSFORMERS: ENCODER STRUCTURE I

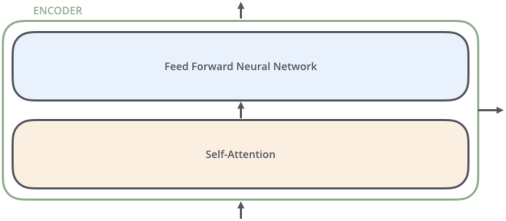


Transformers: encoders follow particular structure

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

- Encoders are identical in structure
  - But they do not share weights
- Encoders have two sublayers
  - A self-attention layer
  - A feedforward neural network layer

Transformers: Encoder Structure II

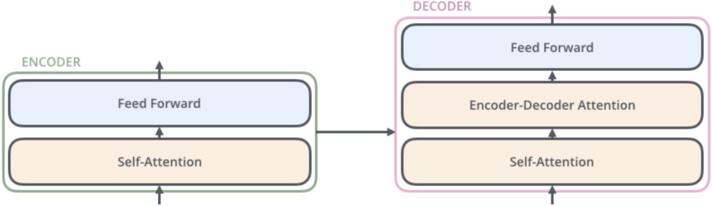


Transformers: encoders follow particular structure

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

- Self-attention layer:
  - Encoder can look at other words when encoding words
- Feedforward neural network (FFNN) layer:
  - Exact same FFNN applied for each position in sentence

Transformers: Decoder Structure I



Transformers: encoder and decoder interact in particular way

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

- Decoder shares structure with encoder, but ...
- ... has an additional encoder-decoder attention sublayer
- Helps decoder to pay attention as guided by input

Self-Attention: Reminder

Self-Attention: Definition

- Consider a sequence of tokens $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$
- Each token has its own query, key, and value
- Hence, each token can attend to each other token:
  - Pair the query vector with the key of the other token
  - This yields a weight for its own value
- Compute weighted sum of values as representation in next layer

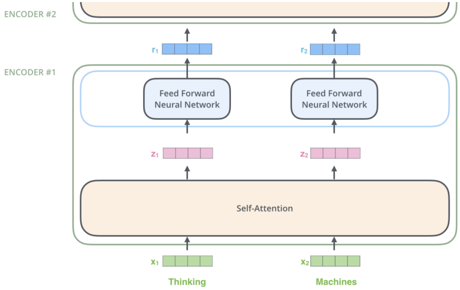
Self-Attention: Formal Summary

- Consider a sequence of tokens $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$
- Replace $\mathbf{q}$ with $\mathbf{x}$ and both $\mathbf{k}_i, \mathbf{v}_i$ with $\mathbf{x}_i$ in (1) from Lecture 2
- One obtains a new sequence $\mathbf{z}_1, ..., \mathbf{z}_n \in \mathbb{R}^d$ by

$$\mathbf{z}_i := f(\mathbf{x}_i, ((\mathbf{x}_1, \mathbf{x}_1), ..., (\mathbf{x}_n, \mathbf{x}_n)) = \sum_{j=1}^{n} \alpha(\mathbf{x}_i, \mathbf{x}_j)\mathbf{x}_j \in \mathbb{R}^d \qquad (1)$$

Transformers continued

Transformers: Encoder Structure III

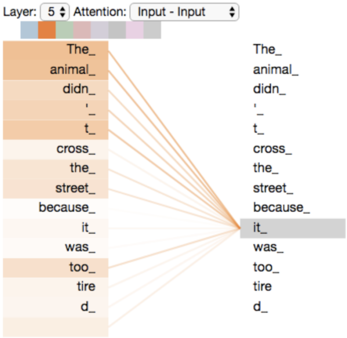


Transformers: encoders sublayer by sublayer

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

1. Words are embedded ☞ yields vectors $\mathbf{x}_i$
2. Vectors $\mathbf{x}_i$ run through self-attention sublayer ☞ yields vectors $\mathbf{z}_i$
3. Each $\mathbf{z}_i$ runs through exact same FFNN ☞ yields vectors $\mathbf{r}_i$

TRANSFORMERS: SELF-ATTENTION I

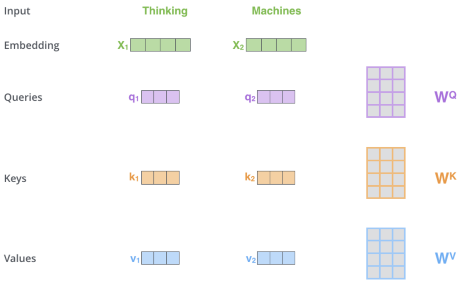


Words pay more/less attention to others

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

- 5th sublayer, 2nd out of 8 attention heads
  - Word "it" pays most attention to "the animal"
  - Word "it" pays less attention to "the street"
  - Word "it" pays no attention to "because"

Transformers: Self-Attention II

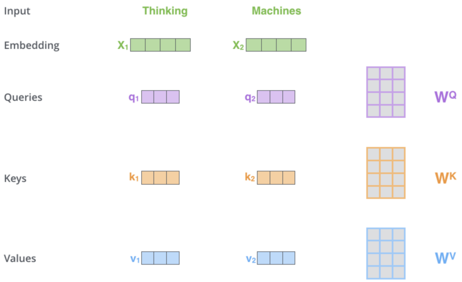


Self-attention: queries, keys and values

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 III

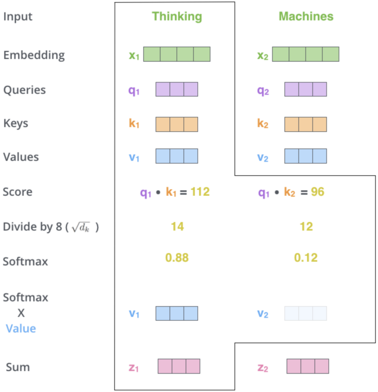


Self-attention: queries, keys and values

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

- Seminal paper: dimension of $\mathbf{x}_i$ = 512, of $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ = 64
  - So, $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{512 \times 64}$
  - *Recall:* $\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^Q$, $\mathbf{k}_i = \mathbf{x}_i\mathbf{W}^K$, $\mathbf{v}_i = \mathbf{x}_i\mathbf{W}^V$

Transformers: Self-Attention IV



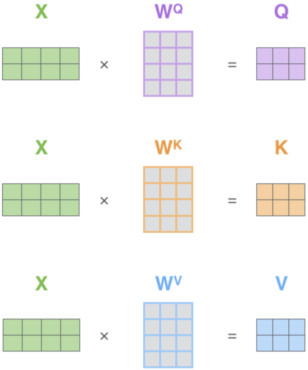
Self-attention: from input to output

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 V

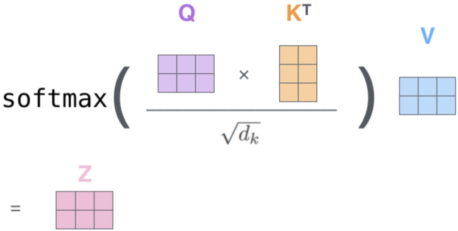


Calculating queries, keys and values

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**, **S**, **V**: 64 (here: 3)

TRANSFORMERS: SELF-ATTENTION VI

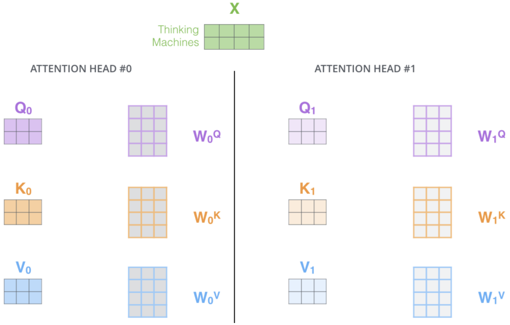


Computing values: compact matrix representation

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

1. Multiply queries with keys: $\mathbf{Q} \cdot \mathbf{S}^T$
2. Normalize relative to query/key length d_k ($= 64$ in reality)
3. Softmax across columns: $\mathbf{S} := \text{softmax}(\mathbf{Q}\mathbf{S}^T / \sqrt{d_k})$ (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 I

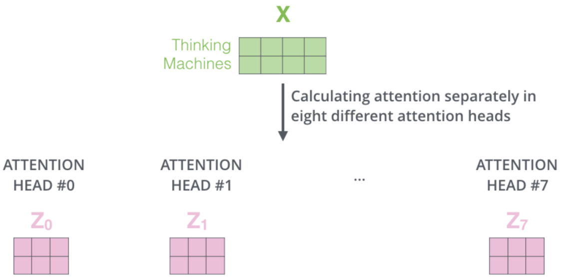


Multi-head attention with 2 heads

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 II

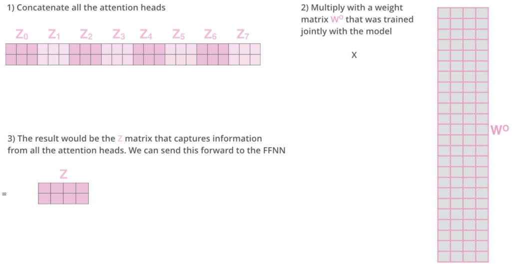


Original paper: multi-head attention with 8 heads

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

- Seminal paper uses 8 different attention heads
- How to summarize / combine the 8 resulting outputs?

TRANSFORMERS: MULTI-HEAD ATTENTION III



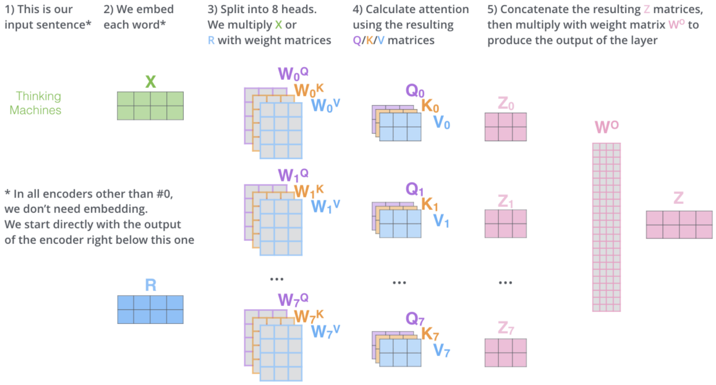
Combining outputs of different attention heads

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

- Combining attention head outputs:
  1. Concatenate all outputs
  2. Multiply resulting matrix with learned matrix $\mathbf{W}^O$
  3. Yields output being equal to input in dimension

     ☞ *Remark:* Need to learn $\mathbf{W}^O$ also for single head

TRANSFORMERS: MULTI-HEAD ATTENTION IV



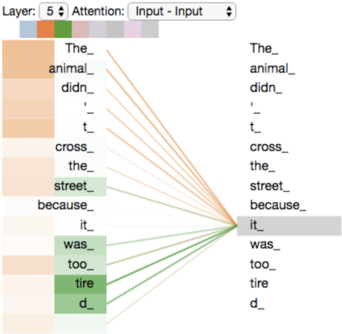
Multi-head attention: Overview / Summary

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 V

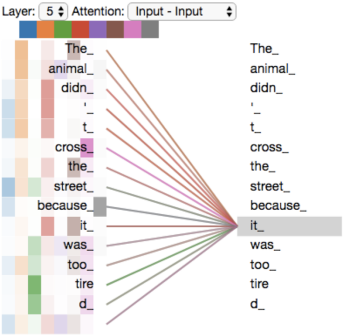


Multi-head attention:
Considering two (of eight) heads

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: Multi-Head Attention VI

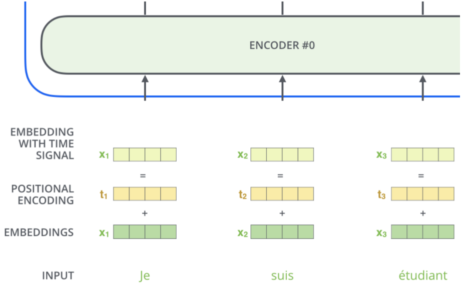


Multi-head attention:
Considering all (eight) heads

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

- Considering all eight attention heads
- Things are more difficult to interpret
- Each head reflects different relationships

Transformers: Positional Encoding

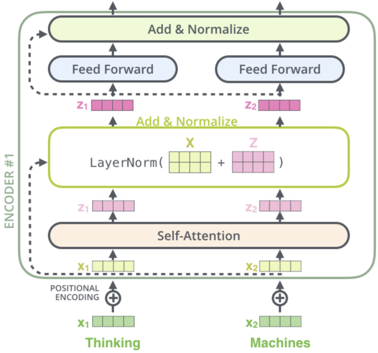


Integrating positional encodings

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

- *Problem:* Self attention unaware of order
- *Solution:* Consider vectors $\mathbf{t}_i$ that code order of word $\mathbf{x}_i$
  - Add $\mathbf{t}_i$ to $\mathbf{x}_i$ ☞ position i of $\mathbf{x}_i$ can be determined
  - Details of generation of $\mathbf{t}_i$ not discussed here

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

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

UNIVERSITÄT BIELEFELD

Thanks for your attention!!