Biological Applications of Deep Learning
Lecture 13

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University

January 25, 2023

Contents Today

- Transformers
  - Encoder Structure
  - Self Attention
  - Decoder Structure
- Transformer Versions and Training
  - Encoder Only
  - Encoder-Decoder
  - Decoder Only
- AlphaFold
  - Predicting Protein Structure from Primary Sequence
  - The Evoformer Module
  - Attention and Multiplicative Updates

Transformers

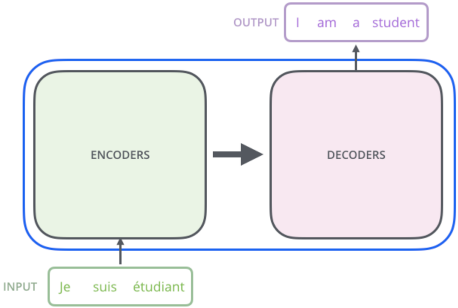
TRANSFORMERS: MOTIVATION

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

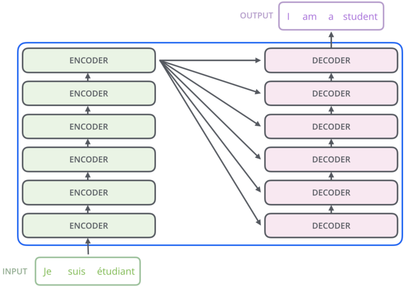


From `https://jalammar.github.io`

- 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

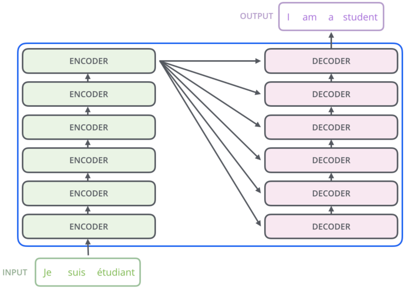


Transformers: encoders and decoders layer structured

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

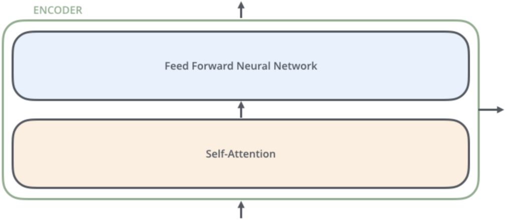


Transformers: encoders and decoders layer structured

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

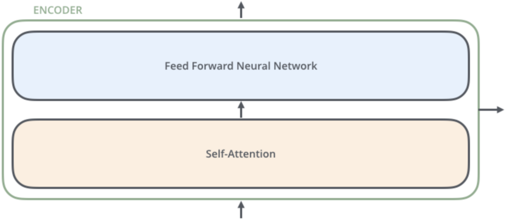


Transformers: encoders follow particular structure

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

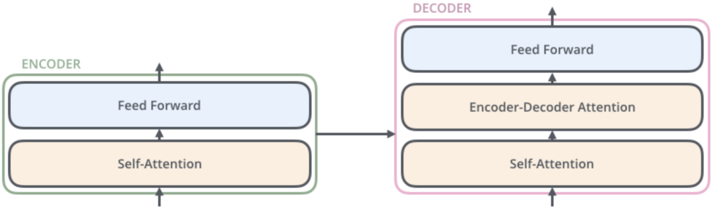


Transformers: encoders follow particular structure

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

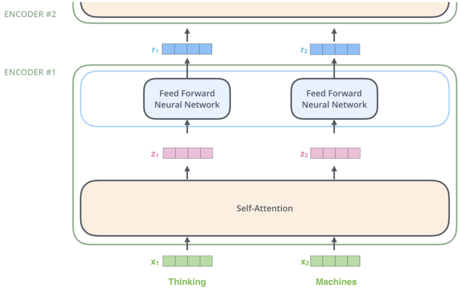


Transformers: encoder and decoder interact in particular way

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

TRANSFORMERS: ENCODER STRUCTURE III

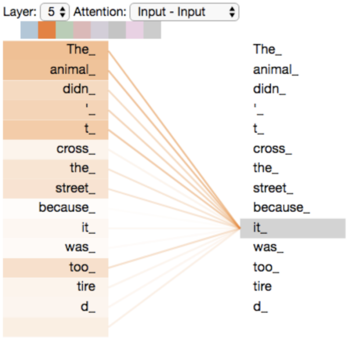


Transformers: encoders sublayer by sublayer

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

1. Words are embedded ☞ yields vectors x_i
2. Vectors x_i run through self-attention sublayer ☞ yields vectors y_i
3. Each y_i runs through exact same FFNN ☞ yields vectors z_i

TRANSFORMERS: SELF-ATTENTION I

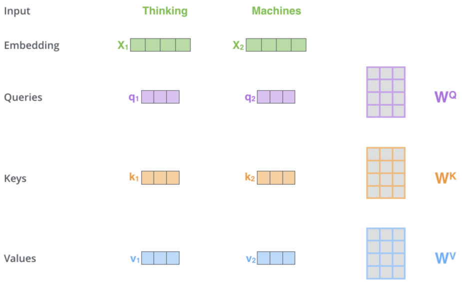


Words pay more/less attention to others

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

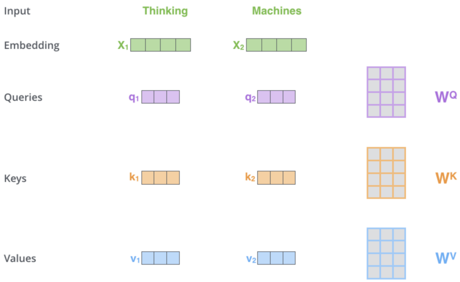


Self-attention: queries, keys and values

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

- Input vectors x_i are transformed to
  - queries q_i, keys k_i, values v_i by
  - applying matrices W_Q, W_K, W_V to x_i from the right

TRANSFORMERS: SELF-ATTENTION III

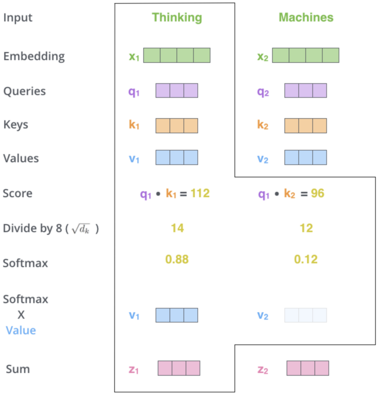


Self-attention: queries, keys and values

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

- Seminal paper: dimension of x_i = 512, of q_i, k_i, v_i = 64
  - So, $W_Q, W_K, W_V \in \mathbb{R}^{512\times 64}$
  - *Recall:* $q_i = x_i W_Q$, $k_i = x_i W_K$, $v_i = x_i W_V$

Transformers: Self-Attention IV



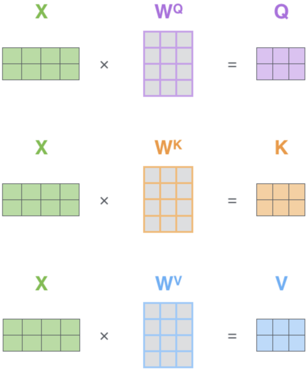
Self-attention: from input to output

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

- Scores for x_1 w.r.t. v_1, v_2
  - v_1: Compute $q_1 \cdot k_1$, divide by 8, yields 112
  - v_2: Compute $q_1 \cdot k_2$, divide by 8, yields 96
- Softmax'ing: Probabilities $0.88, 0.12$ for v_1, v_2
- Final output for x_1:

$$0.88 \cdot v_1 + 0.12 \cdot v_2$$

Transformers: Self-Attention V

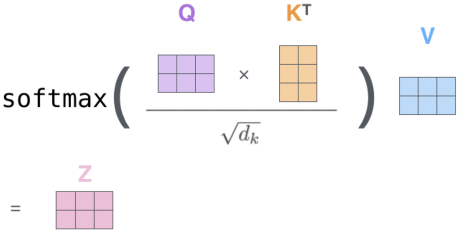


Calculating queries, keys and values

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

- Pack embedded words into matrix X
  - Each row corresponds to one word
- Multiply X with trained matrices W_Q, W_K, W_V
- Recall real dimensions:
  - Words: 512 (here: 4); Q, K, V: 64 (here: 3)

Transformers: Self-Attention VI

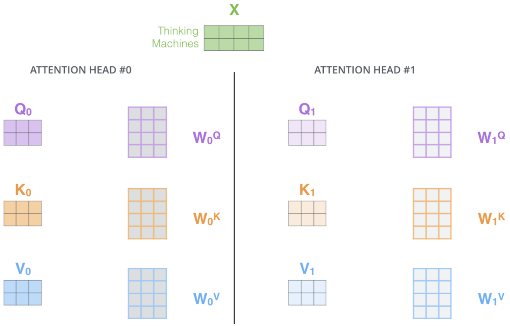


Computing values: compact matrix representation

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

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

Transformers: Multi-Head Attention I

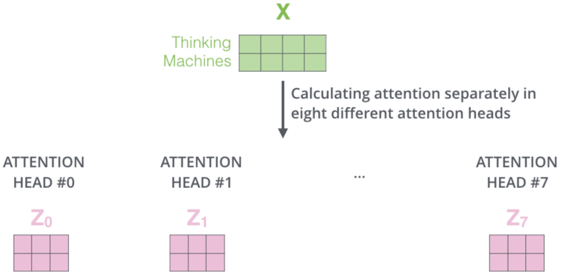


Multi-head attention with 2 heads

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

- Learn several "heads", attending to different interactions
  - Learn several (here: 2) different W_Q, W_K, W_K
  - Establish differences by randomized initialization

Transformers: Multi-Head Attention II

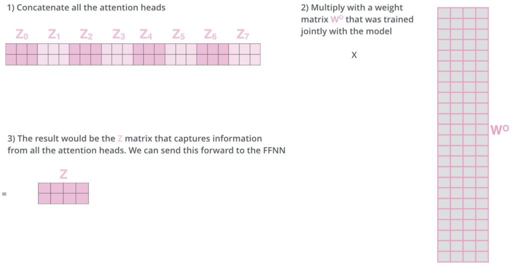


Original paper: multi-head attention with 8 heads

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

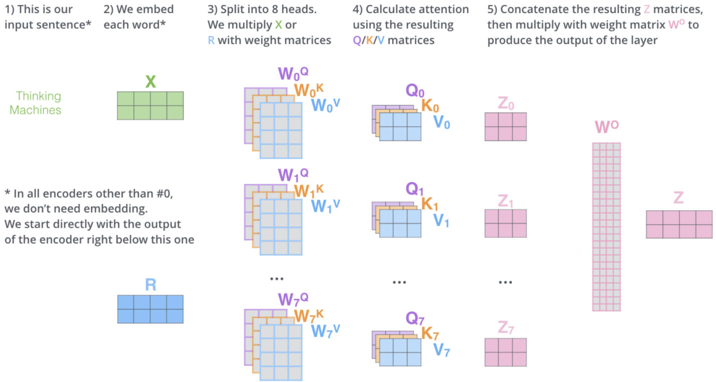


Combining outputs of different attention heads

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

- Combining attention head outputs:
  1. Concatenate all outputs
  2. Multiply resulting matrix with learned matrix W_O
  3. Yields output being equal to input in dimension
     ☞ *Remark:* Need to learn W_O also for single head

Transformers: Multi-Head Attention IV



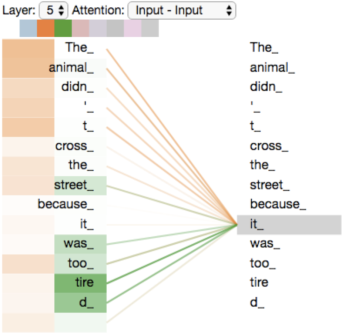
Multi-head attention: Overview / Summary

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

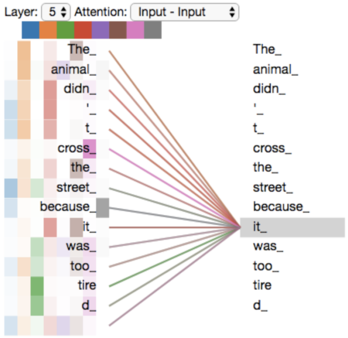


Multi-head attention:
Considering two (of eight) heads

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

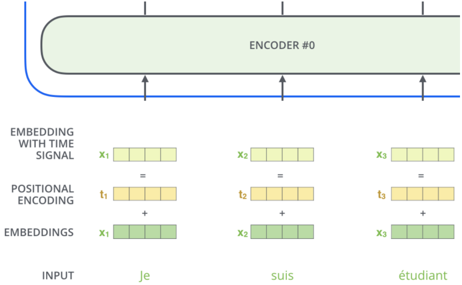


Multi-head attention:
Considering all (eight) heads

From https://jalammar.github.io

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

Transformers: Positional Encoding

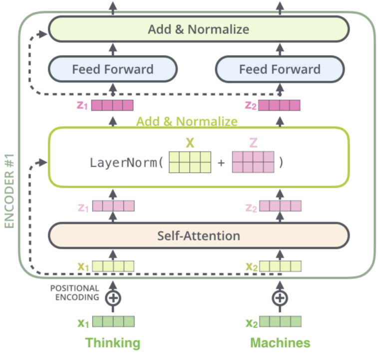


Integrating positional encodings

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

- *Problem:* Self attention unaware of order
- *Solution:* Consider vectors t_i that code order of word x_i
  - Add t_i to x_i ☞ order i can be determined
  - Details of generation of t_i not discussed here

Transformers: Encoder Details

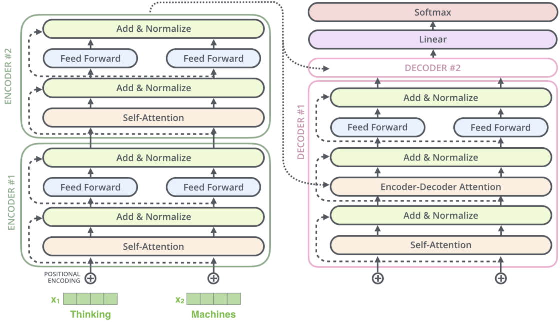


Transformer encoder block: details

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

1. Embedded words are equipped with positional encodings
2. Self attention is applied
   - 2.1 Original x_i is added to z_i
     ☞ Residual skip connection
   - 2.2 Layer norm is applied
     ☞ Normalizes values across layer
3. Each resulting z_i passed through identical feedforward NN (FFNN)
   - 3.1 Original 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`

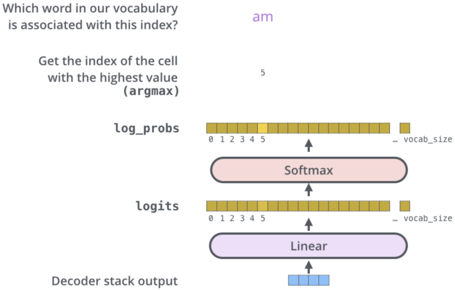
1. Encoder processes input sequence (here: with positional encoding)
2. Output of top encoder transformed into keys K_{encdec} and values V_{encdec}
3. Decoder uses K_{encdec} and V_{encdec} in encoder-decoder attention layer

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

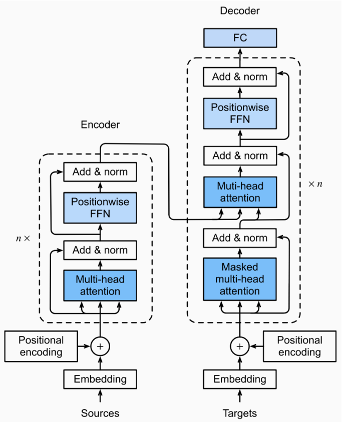


Transformer decoder: final layer consists of linear and softmax sublayer

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

Transformers: Architecture Summary I



Transformer: Summary. n encoder and n decoder layers

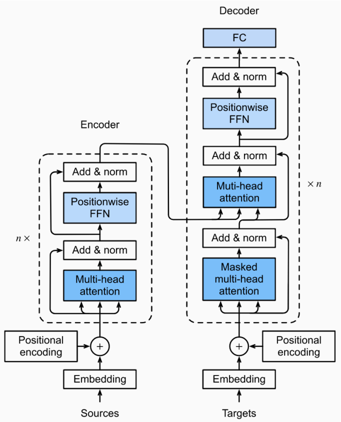
From `https://jalammar.github.io`

Encoder

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

Transformer Variants

Transformers: Architecture Summary II



Transformer: Summary. n encoder and n decoder layers

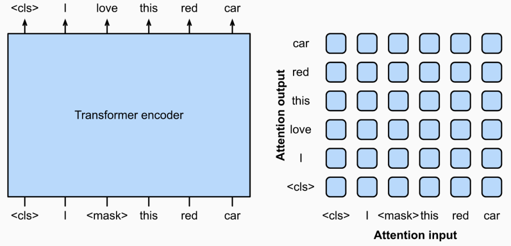
From https://d2l.ai

Decoder

- Decoder 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

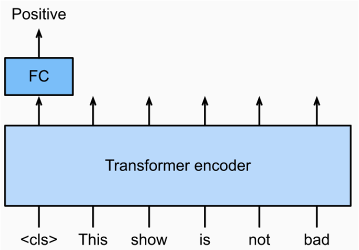


Transformer encoder only: pretraining

From `https://d2l.ai`

- 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 II



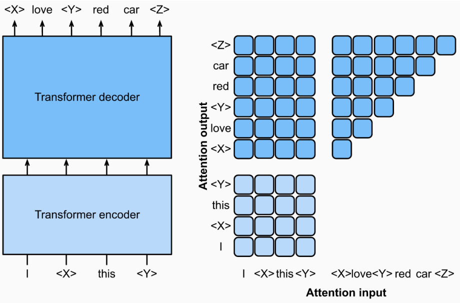
Transformer encoder only: finetuning for sentiment analysis

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

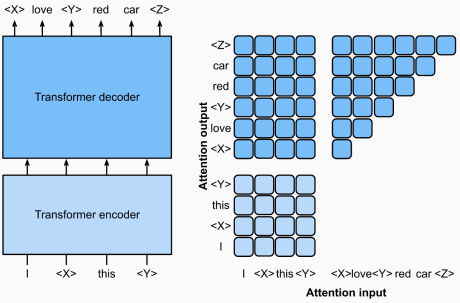


Transformer encoder-decoder: pretraining

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

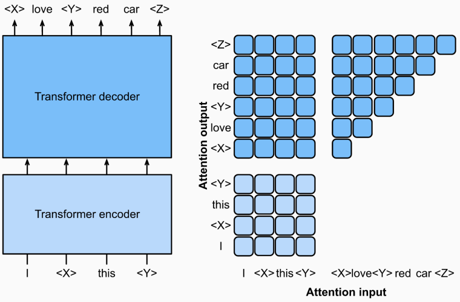


Transformer encoder-decoder: pretraining

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

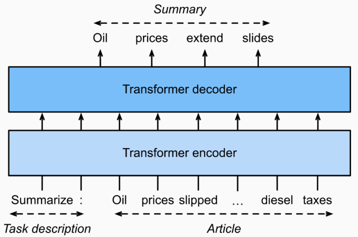


Transformer encoder-decoder: pretraining

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.

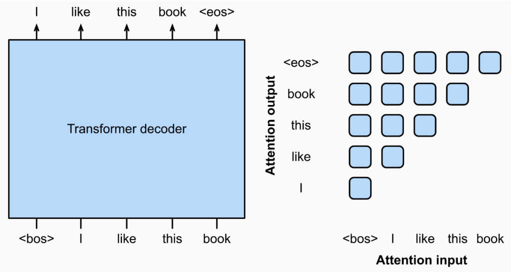
Imagen, based on T5 encoder: Turning texts into images

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

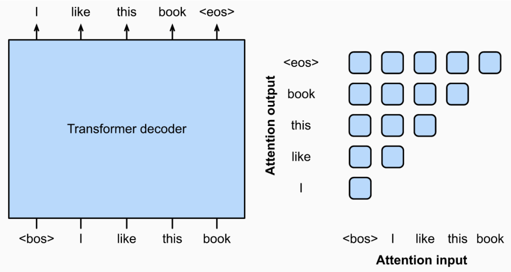


Transformer decoder only: pretraining

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

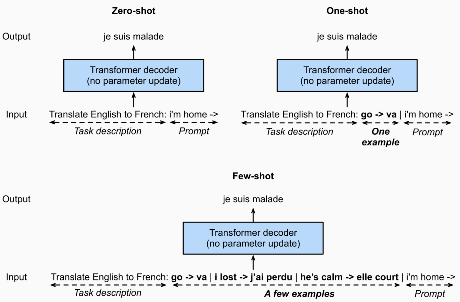


Transformer decoder only: pretraining

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

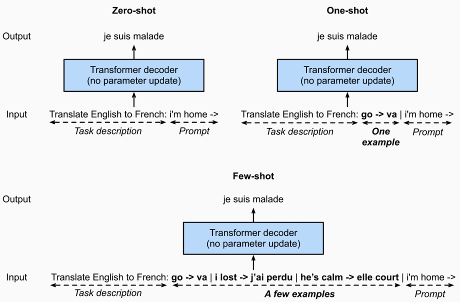


Transformer encoder-decoder: pretraining

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 encoder-decoder: pretraining

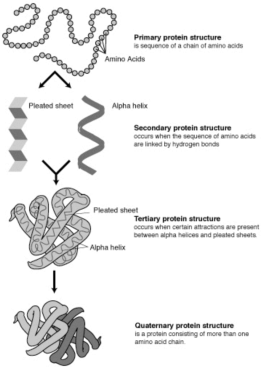
From `https://d2l.ai`

- 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*

UNIVERSITÄT BIELEFELD

AlphaFold
Predicting Protein Structure from Primary Sequence

AlphaFold: Motivation

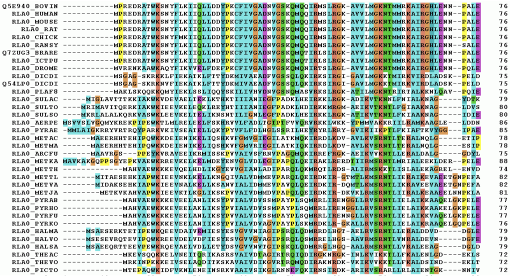


From protein sequence to structure

From https://en.wikipedia.org

- Protein structure traditionally determined by cristallography
  ☞ Time consuming and expensive
- Databases have been filling up with high quality protein structures for decades
- *Idea:* Exploit the existing knowledge
  ☞ Predict structure directly from sequence
- AlphaFold predicts *tertiary* structure
- Reference: [Jumper et al., Nature, 2021], see https://www.nature.com/articles/s41586-021-03819-2

AlphaFold: Multiple Sequence Alignments

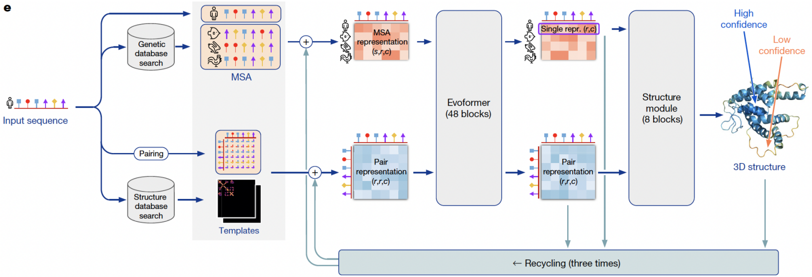


Multiple sequence alignment of same protein from different species

From `https://en.wikipedia.org`

- AlphaFold makes use of multiple sequence alignments (MSA)
- MSA's align evolutionarily related protein sequences
- Each row reflects organism / species; each column amino acid residue

ALPHAFOLD: WORKFLOW

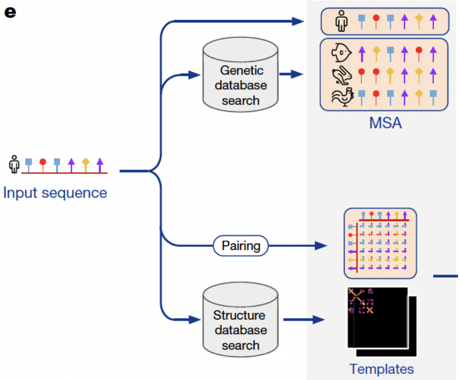


AlphaFold workflow

From [Jumper et al., 2021]

- Very deep: 3 × 48 Evoformer blocks plus 8 Structure Module blocks

AlphaFold: Input I

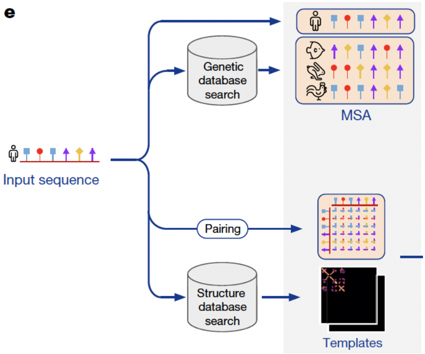


AlphaFold input

From [Jumper et al., 2021]

- Primary amino acid sequence of interest
- Evolutionarily related (homologous) sequences
  ☞ From genetic databases
- Related structures
  ☞ From structure databases

ALPHAFOLD: INPUT II



AlphaFold input

From [Jumper et al., 2021]

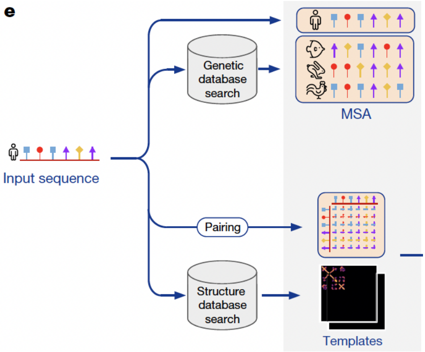
Multiple sequence alignment representation
(MSA-R in the following)

- Relies on relationships with homologous sequences (homologs)
- Captures evolutionary constraints among residues, across homologs

Pair representation
(Pair-R in the following)

- Builds on input from structure databases
- Captures structural / 3D distance constraints among residues

AlphaFold: Input III

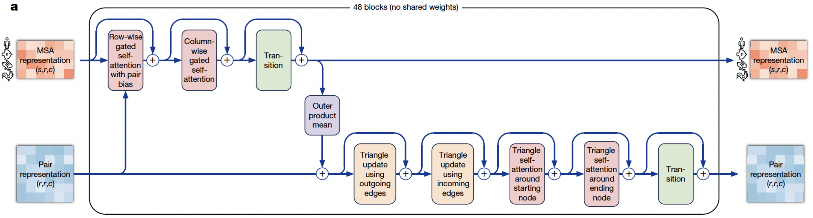


AlphaFold input

From [Jumper et al., 2021]

- Let r be length of primary sequence
- Let s be number of homologs
- MSA-R $\leftrightarrow$ $r \times s$ matrix $\mathbf{M}$
  - $\mathbf{M}_{ij} \in \mathbb{R}^{c_m}$ for each $1 \leq i \leq r, 1 \leq j \leq s$
  - $c_m = 256$
- Pair-R $\leftrightarrow$ $s \times s$ matrix $\mathbf{Z}$
  - $\mathbf{Z}_{ij} \in \mathbb{R}^{c_z}$ for each $1 \leq i \leq r, 1 \leq j \leq r$
  - $c_z = 128$

ALPHAFOLD: EVOFORMER BLOCK

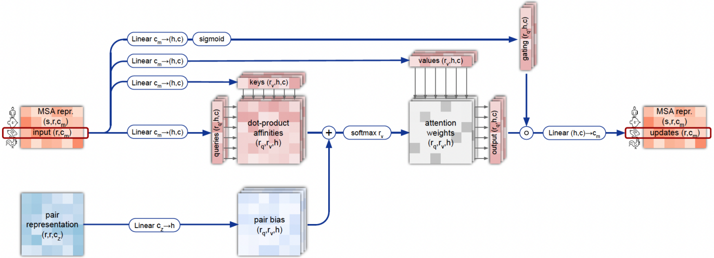


AlphaFold EvoFormer block

From [Jumper et al., 2021]

- MSA-R draws from self attention as major mechanism
- Pair-R draws from “triangular” updates
- MSA-R and Pair-R interact to generate updates
- After 48 updates, output is passed on to structure module

AlphaFold: Row-Wise Self Attention I

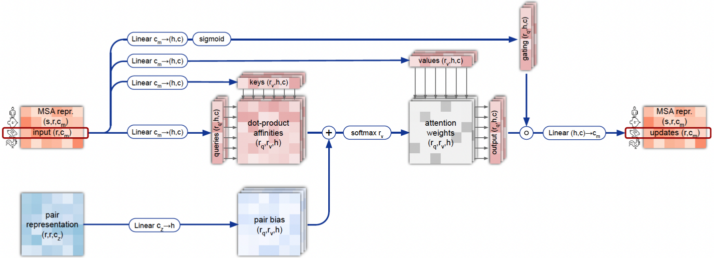


AlphaFold: Row wise self attention

From [Jumper et al., 2021]

- Applied for each row (homolog) $1 \leq i \leq s$ separately
  - Attention between $\mathbf{M}_{ij} \in \mathbb{R}^{c_m}$, $1 \leq j \leq r$ for particular homolog i
- Lower three "Linear $c_m \to (h, c)$":
  - Turns $\mathbf{M}_{ij} \in \mathbb{R}^{c_m}$ to values, keys, queries $\in \mathbb{R}^c$
  - h is number of heads; here $h = 8$

AlphaFold: Row-Wise Self Attention II

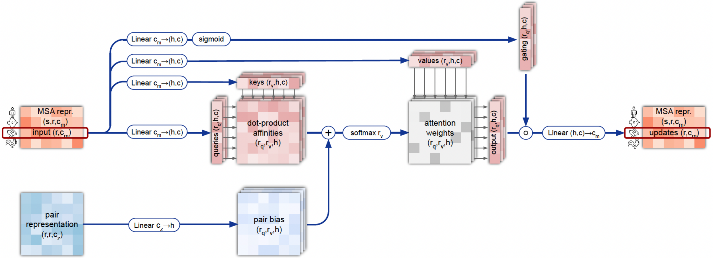


AlphaFold: Row wise self attention

From [Jumper et al., 2021]

- Lower three "Linear $c_m \to (h,c)$":
  - Turns $\mathbf{M}_{ij} \in \mathbb{R}^{c_m}$ to values, keys, queries $\in \mathbb{R}^c$
  - h is number of heads; here $h = 8$
  - Reflect applying (h different) matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{c \times c_m}$ to $\mathbf{M}_{ij}$
  - Usually, $r_q = r_v \leftrightarrow$ each residue transformed in both query and key

AlphaFold: Row-Wise Self Attention III

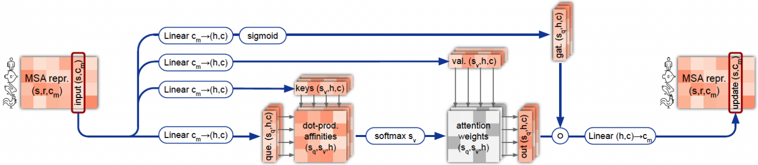


AlphaFold: Row wise self attention

From [Jumper et al., 2021]

- Uppermost "Linear $c_m \rightarrow (h, c)$" reflects generation of gating values
  - Remember principles of LSTM RNN's
- "Linear $c_z \rightarrow h$" turns vectors $\mathbb{R}^{c_z}$ into h biases $\in \mathbb{R}$
  - One bias for each attention head, to be added to dot-product affinities
  - Further softmax'ed to obtain attention weights

AlphaFold: Column-Wise Self Attention

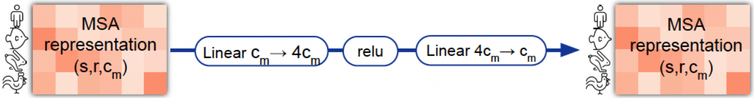


AlphaFold: Column wise self attention

From [Jumper et al., 2021]

- Applied for each column (residue) $1 \leq j \leq s$ separately
  - Attention between $\mathbf{M}_{ij} \in \mathbb{R}^{c_m}, 1 \leq i \leq r$ for particular residue j
- Further, analogous to row-wise self attention
- Only exception: Pair-R has no influence on column-wise self attention
  - Pair-R reflects relationship between residues, not homologs (!)

AlphaFold: Transition



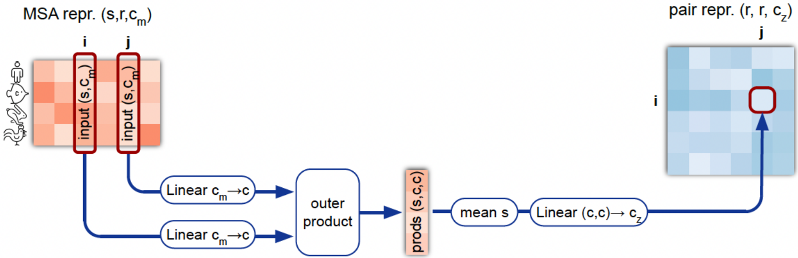
AlphaFold: Transition

From [Jumper et al., 2021]

- Transition reflects application of 2-layer MLP
- Hidden layer has $4c_m$ channels

Summary: No particularly advanced techniques

AlphaFold: Outer Product Mean I



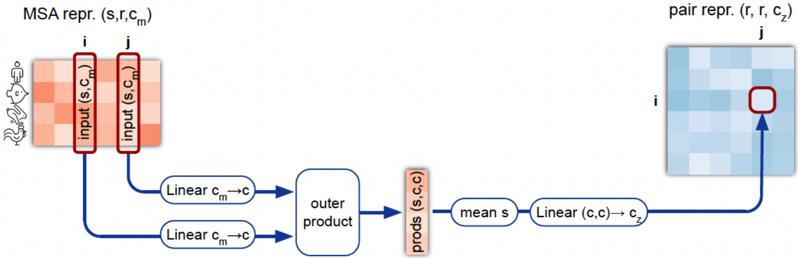
AlphaFold: Outer product mean

From [Jumper et al., 2021]

Intention: Relate residues in MSA-R with each other

- Transforms MSA-R into Pair-R compatible format
- Subsequently added to Pair-R
- Computes outer products for each pair of MSA-R columns

AlphaFold: Outer Product Mean II

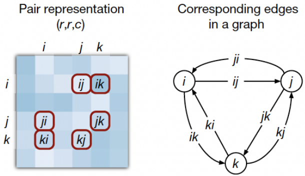


AlphaFold: Outer product mean

From [Jumper et al., 2021]

- Outer product of two c-dimensional vectors yields matrix $\in \mathbb{R}^{c \times c}$
- Matrix entries averaged across homologs
- Yields one $c \times c$ matrix for each pair of residues
- Further transformed into one c_z-dimensional vector

AlphaFold: Triangular Updates I

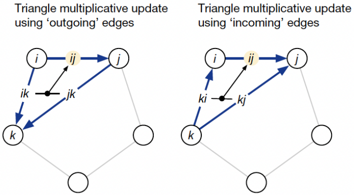


AlphaFold: Triangular updates

From [Jumper et al., 2021]

- *Intention:* Update entries $\mathbf{Z}_{ij}$ in Pair-R based on related entries
  - Related entries share row or column with $\mathbf{Z}_{ij}$ within Pair-R
- *Remember:* Relationships reflect structural constellations (distances etc.)
  - $\mathbf{Z}_{ij}$ influenced by combination of $\mathbf{Z}_{ik}$ and $\mathbf{Z}_{jk}$, for example
- *Procedure:* View residues as nodes in graph; systematically evaluate influence of relationships on other

AlphaFold: Multiplicative Updates II

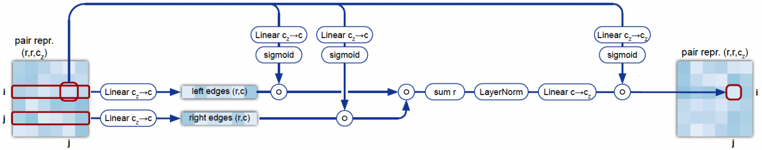


AlphaFold: Multiplicative updates

From [Jumper et al., 2021]

- Combines information in each triangle of edges $(i, j), (i, k), (j, k)$
- Each triangle receives update from other two edges where it is involved
- Two versions:
  - Outgoing edges: (i, j) (and (j, i)) updated based on $(i, k), (j, k)$
  - Incoming edges: (i, j) (and (j, i)) updated based on $(k, i), (k, j)$

AlphaFold: Multiplicative Updates

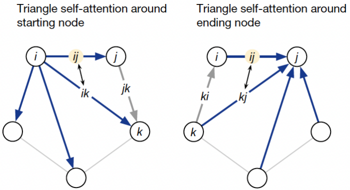


AlphaFold: Multiplicative updates, outgoing edges

From [Jumper et al., 2021]

- *Outgoing edges* from i, j relate to *rows* i and j in $\mathbf{Z}$
- *Insight:* All triangles in one go by summing entries in these rows
- Operations refer to standard operations:
  - Computing weighted sums (e.g. "Linear $c_z \rightarrow c$") and sigmoid activation
  - Hadamard products for gate-type computations; LayerNorm
- *Incoming edges:* Analogous by doing *columns*, not rows

AlphaFold: Triangular Self Attention I

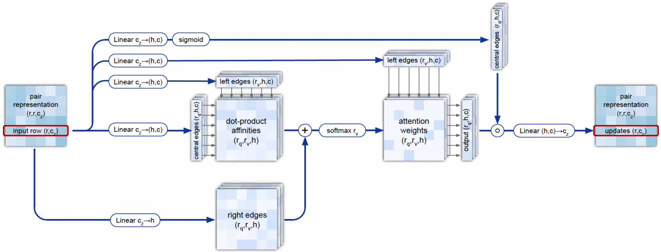


AlphaFold: Triangular Self Attention

From [Jumper et al., 2021]

- Uses self attention as basic mechanism
- *Starting node:* Updates each $\mathbf{Z}_{ij}$ with values from $\mathbf{Z}_{ik}, 1 \leq k \leq r$
  - Corresponds to edges leaving from $i \leftrightarrow$ left panel in figure
  - Pair of edges $(i, j), (i, k)$ "controlled" by (j, k)
- *Ending node:* Updates each $\mathbf{Z}_{ij}$ with values from $\mathbf{Z}_{kj}, 1 \leq k \leq r$
  - Corresponds to edges going into $j \leftrightarrow$ right panel in figure
  - Pair of edges $(i, j), (k, j)$ "controlled" by (k, i)

AlphaFold: Triangular Self Attention II

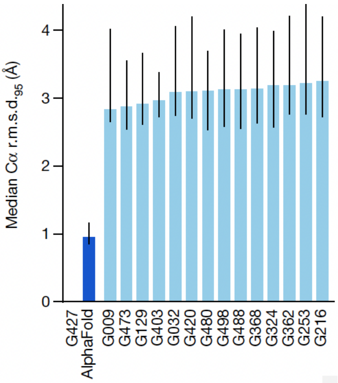


AlphaFold: Triangular self attention from starting node

From [Jumper et al., 2021]

- Queries, keys, values determined for $\mathbf{Z}_{ij}, 1 \leq j \leq r$
- "Controlled" by biases computed from Pair-R
- Entirely analogous to row-wise self attention for MSA-R
  - *Difference:* Replace rows in MSA-R with rows in Pair-R
  - *Ending node:* Use columns, not rows in Pair-R

ALPHAFOLD: RESULTS



AlphaFold: Results

From [Jumper et al., 2021]

- Statistics from Critical Assessment of Structure Prediction 14 (CASP-14)
  - Regularly recurring, renowned structure prediction competition
- X-axis: ID's of ifferent competitors
  - AlphaFold's ID: G427
- Y-axis: Root-mean-square deviation, measured in Ångström ($= 10^{-10}\,m$)
  - Blue bar: median across 10 000 bootstrap samples
  - Black line: 95% confidence interval

REFERENCES

- Jay Alammar's ML blog: "The Illustrated Transformer", see https://jalammar.github.io/illustrated-transformer/
- http://d2l.ai, 11.7, 11.9
- [Jumper et al., Nature, 2021], see https://www.nature.com/articles/s41586-021-03819-2

Thanks for your attention!!