Sequence-to-sequence (encoder–decoder) networks

Statistical Methods in NLP 2
ISCL-BA-08

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

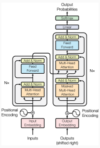
Summer Semester 2025

version: 276006a @2025-07-02

Self attention and Transformers

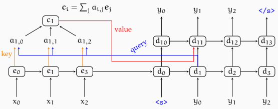
- RNNs have been the primary architecture for modeling sequences. However,
  - RNNs condition only on preceding input
  - RNNs are inherently sequential – difficult to parallelize
- *Transformers* are designed to resolve these problems
- They use rely mainly on a form of attention, called *self attention*

Transformer architecture



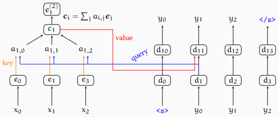
- An encoder–decoder architecture only using self attention
- The encoder is a stack of blocs consisting of self attention followed by a feed-forward layer
- The decoder is similar but,
  - Future input is 'masked'
  - Besides self attention, it also attends to the encoder states
- Both encoder and decoder are stacked
- Both of them use 'multi-head' attention

Sequence-to-sequence RNN with attention



First step towards Transformers

What if we drop the recursion?



Calculating the context vector

- The context vector is the sum of the encoder states, weighted by attention, $a_{i,j}$

$$c_i = \sum_j a_{i,j} \mathbf{e}_j$$

- Typically the weights are normalized through *softmax* the result is an attention distribution

$$a_{i,j} = \frac{e^{f(\mathbf{d}_{i-1}, \mathbf{e}_j)}}{\sum_k e^{f(\mathbf{d}_{i-1}, \mathbf{e}_k)}}$$

- The attention function, $f(\cdot)$ computes the relevance of encoder state $\mathbf{e}_j$ to the decoder state $\mathbf{d}_i$

The attention function

- The attention function returns a high value if the encoder state (the key) is relevant to the decoder state (the query)
- Most commonly, the attention function is a version of dot product between the query and the key

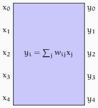
$$f(\mathbf{d}_i, \mathbf{e}_j) = \mathbf{d}_i^T \mathbf{e}_j$$

$$f(\mathbf{d}_i, \mathbf{e}_j) = \mathbf{d}_i^T W_a \mathbf{e}_j$$

$$f(\mathbf{d}_i, \mathbf{e}_j) = \frac{\mathbf{d}_i^T \mathbf{e}_j}{\sqrt{k}}$$

Self attention

a simplified first view



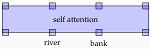
- The outputs are weighted sum of the inputs
- The weights here depend on the inputs x_i
- The general idea is to assign higher weights to inputs that act together in predicting the output
- Similar to convolution, but we are looking at the whole input range
- No recurrence
- The output is permutation invariant (order of the inputs is not important)

Simplified self attention

$$a_{ij} = \frac{e^{x_i \cdot x_j}}{\sum_k e^{x_i \cdot x_k}} \qquad y_i = \sum_j a_{ij} x_j$$

- Dot product weights the input inputs with high similarity
- We do not necessarily want similarity

An example



- We want representation of 'bank' to attend to 'river'
- But their embeddings are not likely to be similar
- Dot product by itself is not useful
- Note also the similarity with convolutions, but with unbounded distance

Self attention

typical approach

- Each input is transformed to three different vectors
  - $k_i = W_k x_i$ (key)
  - $q_i = W_q x_i$ (query)
  - $v_i = W_v x_i$ (value)
- The transformations (W_k, W_q, W_v) are learned
- The attention function becomes

$$\text{softmax}\left(\frac{q \cdot k}{\sqrt{d}}\right) v$$

- This can be computed efficiently, and in parallel

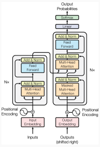
Multi-head attention

- In practical implementation multiple attention functions are learned in parallel
- conceptually, this learns different type of relations between input units

How about information from the sequence?

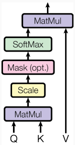
- Self attention is not sensitive to the order of input elements
- The solution is encode the position information on each embedding
- Two common approaches
  - *Position encodings*: combination of harmonic (cosine, sine) functions with different periods
  - *Position embeddings*: learned embeddings for each index of the input sequence
- Recent systems also include relative positional embeddings

Transformer architecture



- The first layer is an *embedding* layer: no information from context information
- Subsequent layers use attention followed by a non-linear transformation (feed-forward layer)
- Feed-forward layer is a projection an up-projection followed by projection back to input/output dimensions
- Input and output dimensions to each Transformer block is the same
- Layer normalization is after (sometimes before) the attention and feed-forward calculations

Transformer architecture: attention

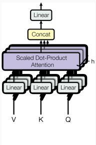


- All queries, keys and values are combined to matrices
- Instead of individual dot product, matrix multiplications is used

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

- On the decoder side, future input is masked
- The result of matrix multiplications are normalized layerwise

Transformer architecture: multihead



- All attention layers are multiplied
- Conceptually, each head learns a different property of the input to 'attend to'

Feed-forward layer

- The output of the attention module is passed to a feed-forward network (FFN)
- Typically FFN first applies a non-linear projection to a higher dimension (typically 4 times its input), and projects it back to the input/output (model) dimension
- Conceptually,
  - attention decides what lower-layer tokens in the sequence to use as input
  - the FFN transforms (non-linearly) to a new representation

Layer normalization

- Normalization helps keeping activations (and gradients) in a reasonable range
- This is important for practical reasons: most optimizers work better if activations do not have extreme values
- Original tranformers uses post-norm: normalization is done after attention/FFN calculations
- Most recent models use pre-norm
- 'Layer Norm' in original Transformers simply calculates the z-socre over the activations (features)
- A common alternative is RMSNorm
- Another common option is batch-norm: normalize across the batch rather than feature dimension

Attention in decoder

- The decoder has to decode tokens without the knowledge of the true tokens
- The attention on the decoder is 'masked', each token can only attend to the preceding tokens

$$\mathrm{mask}\left(\mathrm{softmax}\left(\frac{q \cdot k}{\sqrt{d}}\right)\right)v$$

Some closing notes

- The Transformer architecture is the main architecture used in modern (large) language models (with minor changes)
- Original Transformer is an encoder–decoder model, used for machine translation
- Modern language models can be either
  - Encoder only
  - Encoder-decoder
  - Decoder only
- Reading: Jurafsky and Martin (2025, Chapter 9)

Next:

- Transformer language models
- Reading: Jurafsky and Martin (2025, Chapters 10 & 11)

Additional reading, references, credits

- The diagrams of the Transformer architecture is from the original Transformers paper (Vaswani et al., 2017)

Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.