Deep Learning for Natural Language Processing

Lecture 9 – Text generation 3: Transformers

Dr. Martin Tutek

June 13, 2023

Ubiquitous Knowledge Processing
Department of Computer Science
Technical University of Darmstadt

UBIQUITOUS KNOWLEDGE PROCESSING

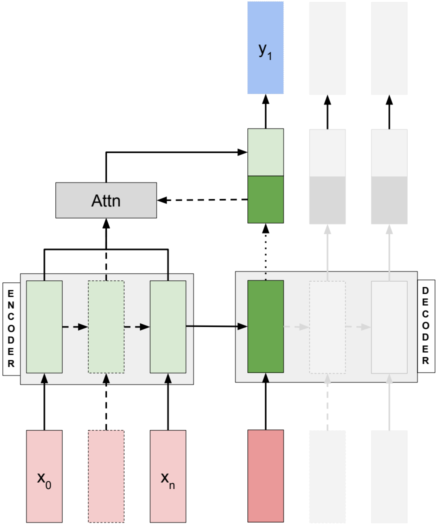
UKP Web

Recap

In the previous lecture we:

- Introduced the encoder-decoder architecture & why we need it
- Defined the three broad classes of NLP problems
- Shown that RNNs have problems when modeling long dependencies
- Introduced the attention mechanism, its abstraction and design choices

Recap: Encoder-decoder with attention



Motivation

MLP – fixed input sequence length

RNN – works well with **shorter** sequences

RNN + attention – works well with both **shorter and longer** sequences

- Why not use **only** attention?

Attention Is All You Need

Prerequisites for attention-only networks

What do we **gain** from recurrent networks?

- **Memory cells**: contain summaries of sequence read *so far*
  - **However**, they have **limited** capacity – we complement them with attention
- **Position** of a word in sequence
  - For each hidden state s_i, the current word embedding x_i is added to the previous state s_{i-1} – the network can distinguish **word order**
  - **However**, it takes n recurrence operations to process a sequence

Do recurrent networks have any other **drawbacks**?

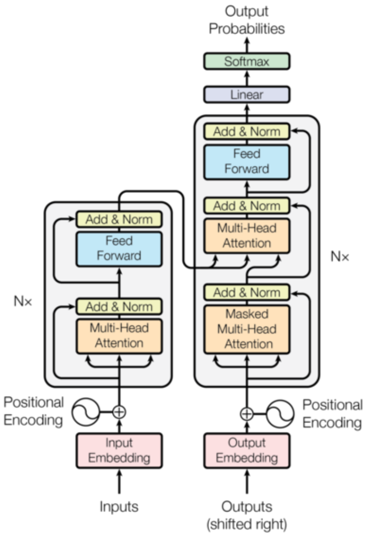
- They **scale poorly** – LSTMs are problematic to scale deeper than 4-8 layers
- **Closed vocabulary** – so far, we assumed one word = one vector (no BPE)

How to make attention-only networks work?

The Transformer

- The Transformer
  - Contextualized representations
  - The Transformer attention block
  - Byte-pair encodings
  - Positional embeddings

The Transformer (Vaswani et al., 2017)



What are the unknown elements?

- **Multi-head** attention
- Add & Norm
- **Positional** embeddings
- **Open vocabulary** through BPE

The Transformer

Contextualized representations

Contextualized representations

Recall: **limitations** of word embeddings

Polysemy, context independent representation

Some words have obvious multiple senses

A *bank* may refer to a financial institution or to the side of a river, a *star* may an abstract shape, a celebrity, an astronomical entity

How do recurrent networks handle contextualization?

$$s_i = f_{\text{rnn}}(s_{i-1}, x_i)$$

- Each state acts as a representation of the sequence **so far**

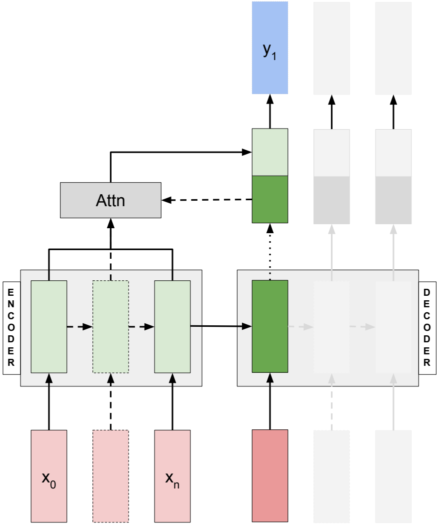
Contextualized representations

$$s_i = f_{\text{rnn}}(s_{i-1}, x_i)$$

- Each state acts as a representation of the sequence **so far**
    - Recall: **bidirectional** RNNs (left- and right-hand context)
    - A state contains **cues** about the meaning of the current word **in context**

- **However**, the state has to act as both
    1. A summary of the entire sequence
    2. The meaning of the current word in context

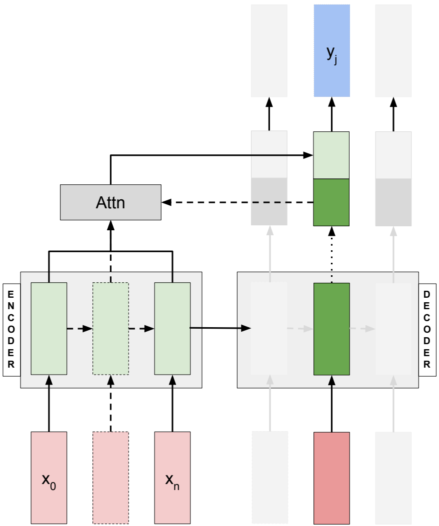
Contextualized representations

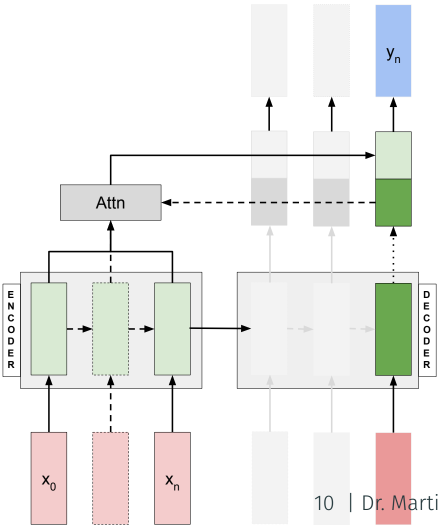


Step 1 of encoder-decoder attention:

- We obtain relevant information **for current state** from input sequence
- This result of the attention operator should also contain **contextual cues**

Contextualized representations





Contextualized representations

Why not **cut out the middleman** (RNN)?

- We use the RNN state as the **query** for attention
- We could instead use the input **word representation**

Recall: scaled dot-product attention

$$a = \sum_{i}^{n} \alpha_i v_i \qquad\qquad \hat{\alpha}_i = \frac{q^T \cdot k_i}{\sqrt{d_\text{k}}}$$

Recall: what are the query, keys & values (in encoder-decoder attention)?

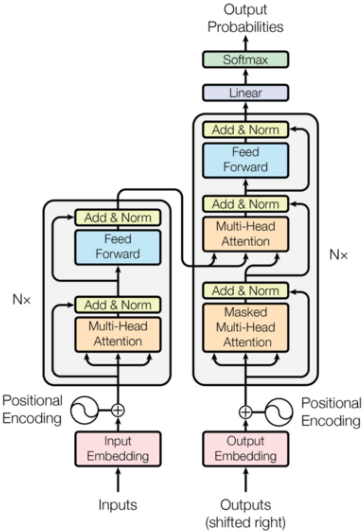
$$q = f_q(s_t^{\text{dec}}) \qquad K = f_k(\{s_i^{\text{enc}}\}_{i=1}^{n}) \qquad V = f_v(\{s_i^{\text{enc}}\}_{i=1}^{n})$$

Where f_q, f_k, f_v are arbitrary functions (neural network layers).

The Transformer

The Transformer attention block

The Transformer attention block



Encoder part of the Transformer block

- Inputs: $\{\mathbf{x}_i^l\}_{i=1}^n; \quad \mathbf{x}_i \in \mathbb{R}^{d_m}$
- $x_i^0 \rightarrow$ word embeddings

Goal: **contextualize** word embeds.

1. Transform **each** embedding to its query, key and value reprs.
2. Apply **pairwise** attention between all inputs
3. Use the outputs as word embeddings for **next layer**

The Transformer attention block

1. Each layer l has its own query, key and value linear transformation
$$W_q^l, W_k^l, W_v^l \in \mathbb{R}^{d_m \times d_m}$$
2. Transform the inputs of the current layer $\{\boldsymbol{x}_i^l\}$ into the keys, queries and values
$$\boldsymbol{Q} = W_q(\{\boldsymbol{x}_i^l\}) \quad \boldsymbol{K} = W_k(\{\boldsymbol{x}_i^l\}) \quad \boldsymbol{V} = W_v(\{\boldsymbol{x}_i^l\})$$
3. Apply scaled dot-product attention
$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_m}}\right)\boldsymbol{V}$$

The Transformer attention block: scaled dot-product

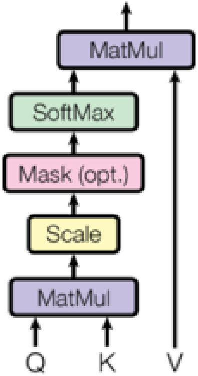
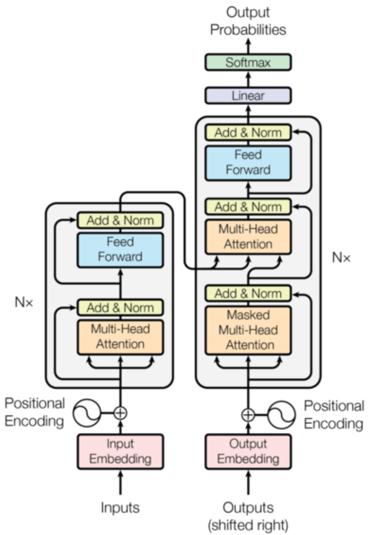


Figure from Vaswani et al., 2017

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

- Matmul between Q and K $\rightarrow$ **energy**
- Masking (why?)
  - We might not want to attend to **all** tokens
- Output $=$ weighted sum

The Transformer attention block: multi-head attention



However: we are using **multi-head** attention!

Idea: there could be **multiple aspects** in which two tokens can be similar

- Intuition: *each* hidden dimension $\approx$ one linguistic feature
- $\rightarrow$ perform **multiple** energy computations

The Transformer attention block: multi-head attention

Recall: Transform the inputs of the current layer $\{x_i^l\}$ into the keys, queries and values

$$Q = W_q(\{x_i^l\}) \quad K = W_k(\{x_i^l\}) \quad V = W_v(\{x_i^l\})$$

Each matrix $Q, K, V \in R^{n \times d_m}$, where d_m is the *model dimension*.

Split each query/key/value into h **heads** (aspects) by *reshaping*.

$$Q, K, V \in R^{n \times d_m} \rightarrow Q, K, V \in R^{n \times h \times d_m / h}$$

- **Note**: d_m **has** to be divisible by h

Remaining process continues as usual.

The Transformer attention block: multi-head attention

Recall: 3. Apply scaled dot-product attention

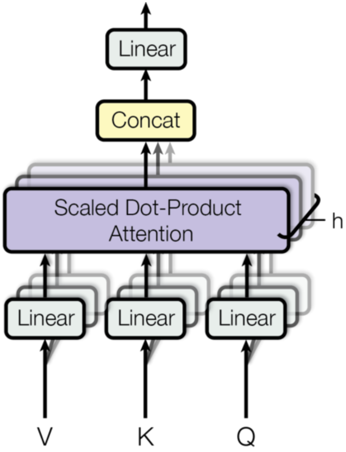
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_m}}\right)V$$

Apply attention h times **in parallel**, then **concatenate** the results.

$$\text{Attention}_j(Q_j, K_j, V_j) = \text{softmax}\left(\frac{Q_j K_j^T}{\sqrt{d_m/h}}\right)V_j$$

Where $\{Q, K, V\}_{j=1}^{h}$ are different *heads*.

The Transformer attention block: multi-head attention

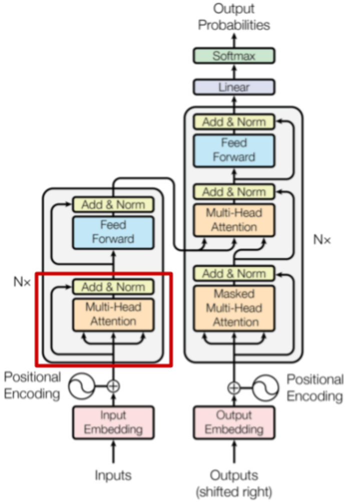


Although this entire process happens behind the scenes, we will still refer to (multi-head) attention as

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

for brevity.

The Transformer attention block: residual connection



We use *residual connections* with the input of the layer

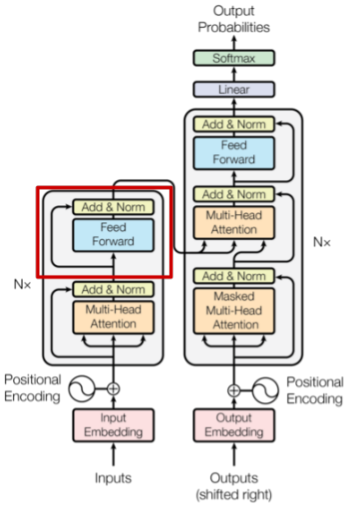
1. $\hat{x}^l$ is the output of attention

$$\hat{x}^l = \text{Attention}(Q^l, K^l, V^l)$$

2. We apply the residual connection and normalize

$$x^{l*} = \text{LayerNorm}(x^l + \hat{x}^l)$$

The Transformer attention block: position-wise linear layer



3. We apply an extra **linear transformation** to each individual representation

$$x^{l+1} = \text{LayerNorm}(x^{l*} + f_{hh}^{l}(x^{l*}))$$

Where f_{hh} is an arbitrary transformation (single hidden layer NN)

4. We use x^{l+1} as the input to the **next** layer $l+1$

The Transformer

Byte-pair encodings

Byte-pair encodings

Recall: sub-word embeddings

Sub-word embeddings

Each character $n-$gram has its own embedding.

Resolves the issues of **rare words**, **typos** and doesn't ignore the **morphology** of each word.

However – it scales poorly (there are **many** character $n-$grams)

Byte pair encodings – characters (1-grams / *bytes*) can represent **any** word.

Byte-pair encodings

Byte-pair encodings

Start at **character** level.

Merge the two **most frequently co-occuring** characters into a **new character**.

Continue until you reach desired vocabulary size. **Each word** will always be represented.

Variants: WordPiece, SentencePiece, subword-nmt (GitHub)

The differences are in the **merging criterion**:

- Sennrich et al., 2016 use **frequency** of co-occurence;
- Kudo, 2018 trains a **unigram language model**.

The Transformer

Positional embeddings

Positional embeddings

The Transformer processes all tokens **in parallel** – there is **no information** about word order which in RNNs originated from recurrence.

Idea: use functions which depend on **position of token in sequence**. The closer the tokens, the higher the similarity of the functions.

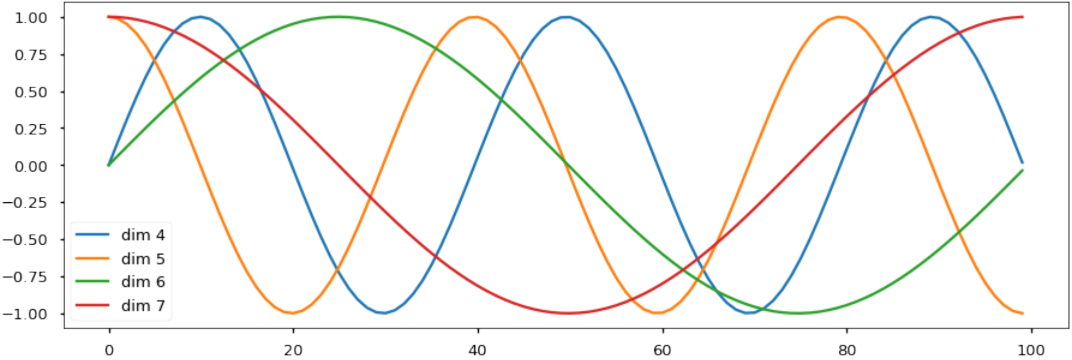
- Sine and cosine waves

$$PE_{(pos,2i)} = \underbrace{\sin(\mathrm{pos}/10000^{2i/d_m})}_{\text{Even dimensions}}$$

$$PE_{(pos,2i+1)} = \underbrace{\cos(\mathrm{pos}/10000^{2i/d_m})}_{\text{Even dimensions}}$$

- We **sum** the positional embedding vector to the token embedding

Positional embeddings



Positional embeddings

Alternative: **trained** positional embeddings

- Similar to word embeddings (byte pair embeddings)
- We randomly initialize a **position embedding matrix** and train it along with our model
    - Issues?
    - How **large** is this position embedding matrix?
    - What if test data contains sequences **longer** than training data?

Recap

The Transformer
- Contextualized representations
- The Transformer attention block
- Byte-pair encodings
- Positional embeddings

Takeaways

- Transformer networks are **fully attentional networks**
  - More efficient than RNNs (process tokens in parallel)
  - Scale better than RNNs (deeper networks)
- Multi-head attention
  - Split each token representation into h parts, perform h attention operations in parallel
  - Increased expressivity
- They require **positional embeddings**
  - Parallel processing $=$ no information about word position
- Byte pair encoding allows for **open vocabulary**

License and credits

Credits

Martin Tutek