Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

data science

CAIDAS

Chapter 7

Transformers

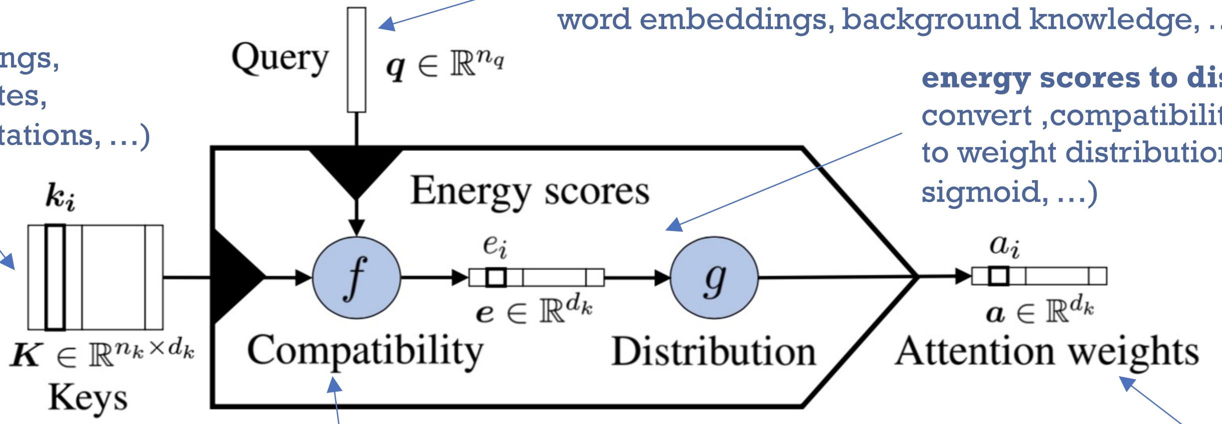
Attention, please!

- In the last few years, many attention mechanisms were introduced
- Always same idea: Compute attention weights for the input sequence to focus on more relevant input steps

query: what to ask for?
(e.g. current Decoder hidden state,
word embeddings, background knowledge, …)

keys: data features
(e.g. word embeddings,
Encoder hidden states,
document representations, …)

energy scores to distribution:
convert ‚compatibilities' of query and keys
to weight distribution (e.g. using softmax,
sigmoid, …)

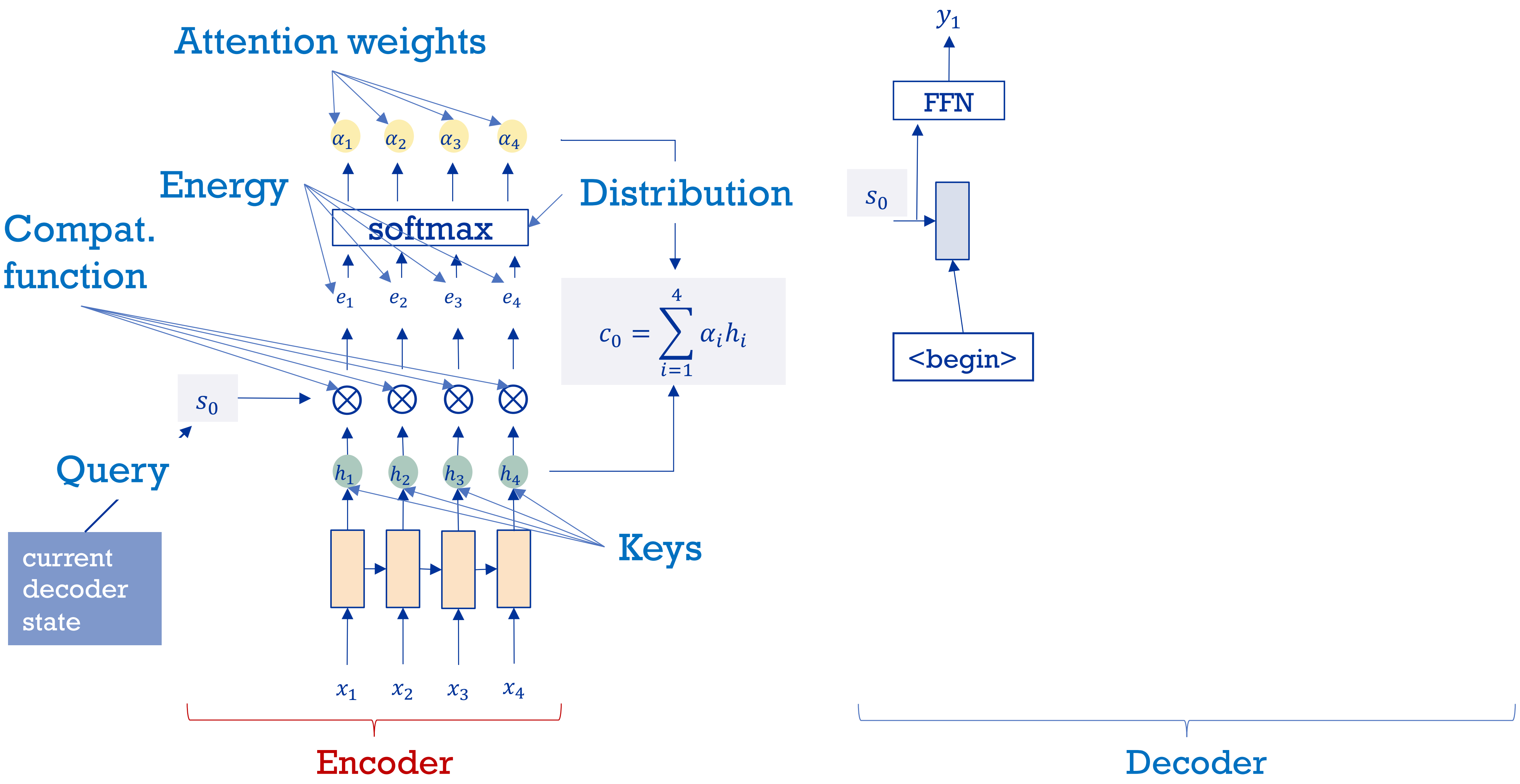


attention weights: used to weight the keys

compatibility function: which key is most compatible to query?

Galassi, Andrea, Marco Lippi, and Paolo Torroni.
"Attention, please! A Critical Review of Neural Attention Models in Natural Language Processing." *arXiv preprint arXiv:1902.02181* (2019).

Recall: Loung-Attention



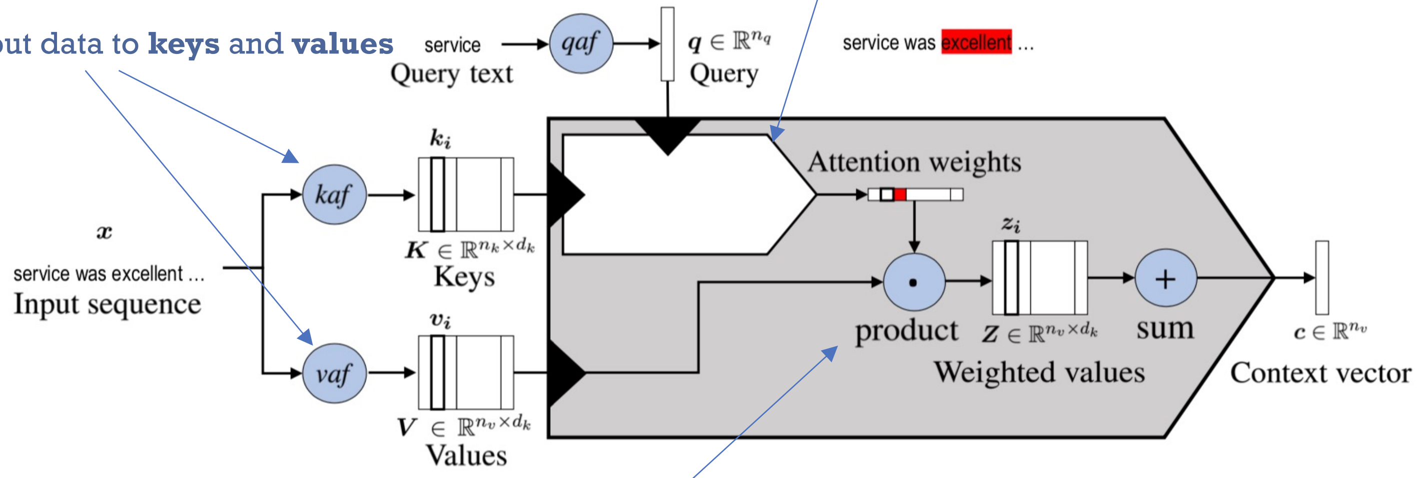
Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation.

Attention, please!

- Sometimes, new key representations are useful → introducing **values**

same mechanism as on previous slide

convert input data to **keys** and **values**



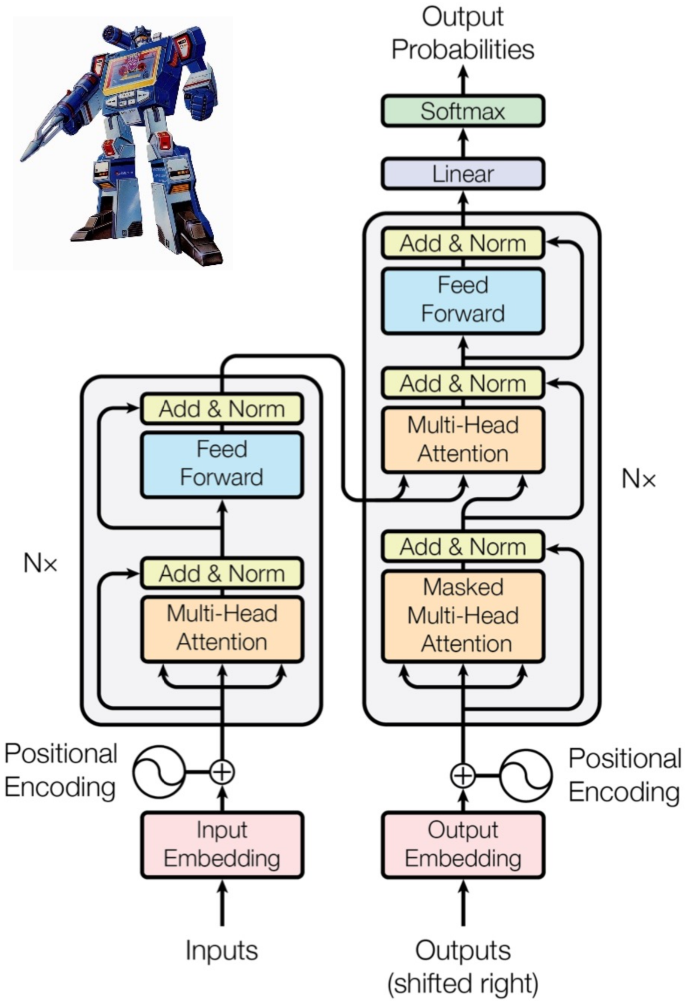
Not the keys are weighted with the resulting distribution, but the values

Galassi, Andrea, Marco Lippi, and Paolo Torroni.
"Attention, please! A Critical Review of Neural Attention Models in Natural Language Processing." *arXiv preprint arXiv:1902.02181* (2019).

Transformer — Is Attention All You Need?

- Transformer: A new neural network architecture based on attention

- Encoder-Decoder structure
- No recurrence!
  - Parallelizable → faster to train
- The encoded sentence is as long as the input sentence!
  - Capturing more information of input
  - „Transforms“ the input into an encoded form

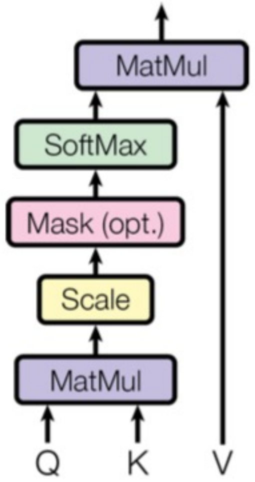


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Transformer Key Idea — (Multi-Head Self-)Attention

- **Scaled Dot-Product Attention:**
  - Introduced in Vaswani et al., 2017
  - Represents attention by matrix multiplication
  - Uses a scaling factor d → Empirically improves results

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



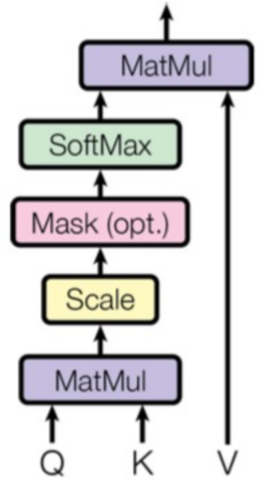
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Transformer Key Idea — (Multi-Head) Self-Attention

- **Self-Attention**
  - Transform an input sequence to a weighted sum of its **own** timesteps
  - Helps to capture long-term dependencies

  - Use Scaled Dot-Product Attention (prev. slide)

  - **Q**uery, **K**eys, and **V**alues are all computed from the input sequence
  - Difference from before:
    Query came from ‚outside' (e.g. Decoder hidden state)

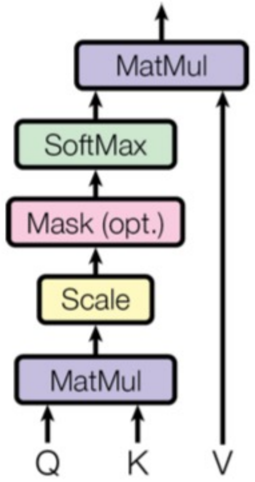


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Transformer Key Idea — (Multi-Head) Self-Attention

- **Self-Attention**
  - Input X
  - Transform X into three different „views“:
    - $K = X \cdot W_k$
    - $V = X \cdot W_v$
    - $Q = X \cdot W_q$

Trainable weights



- $Attention(Q, K, V)$ as before

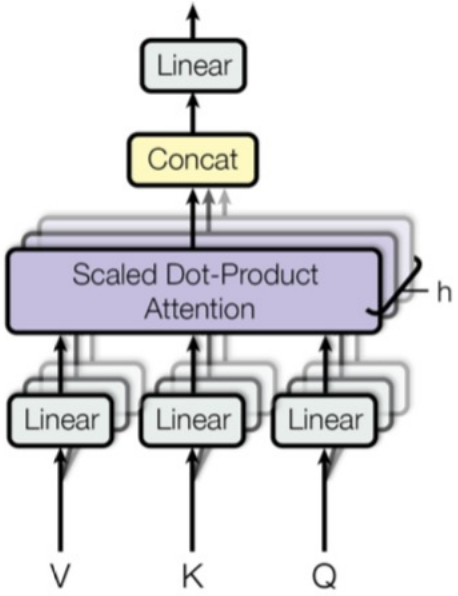
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

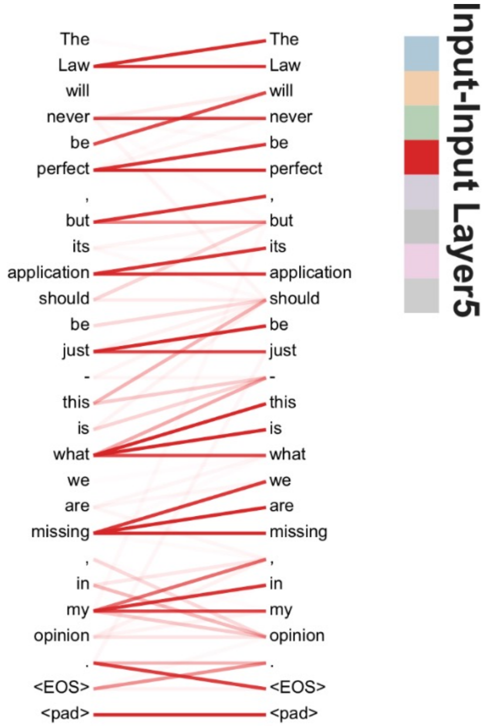
Transformer Key Idea — Multi-Head Self-Attention

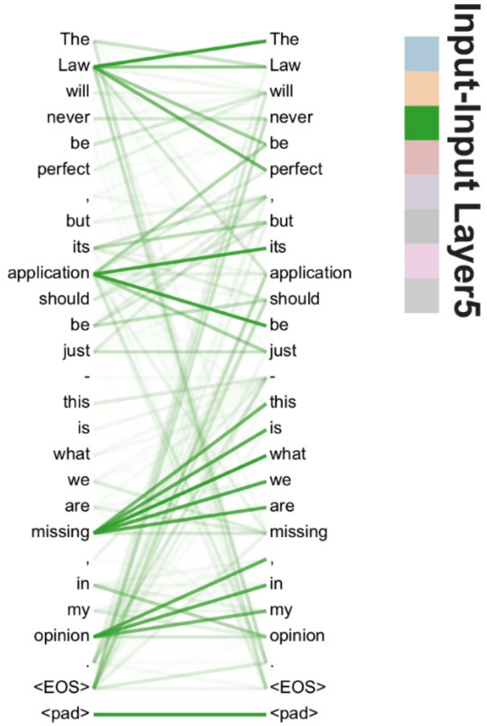
- Multi-Head Attention:
  - Apply self-attention multiple times for the same input sequence (using different weights W_q^i, W_v^i and W_k^i)

  → Attention with multiple „views“ of the original sequence

  → Enables capturing different kinds of importance





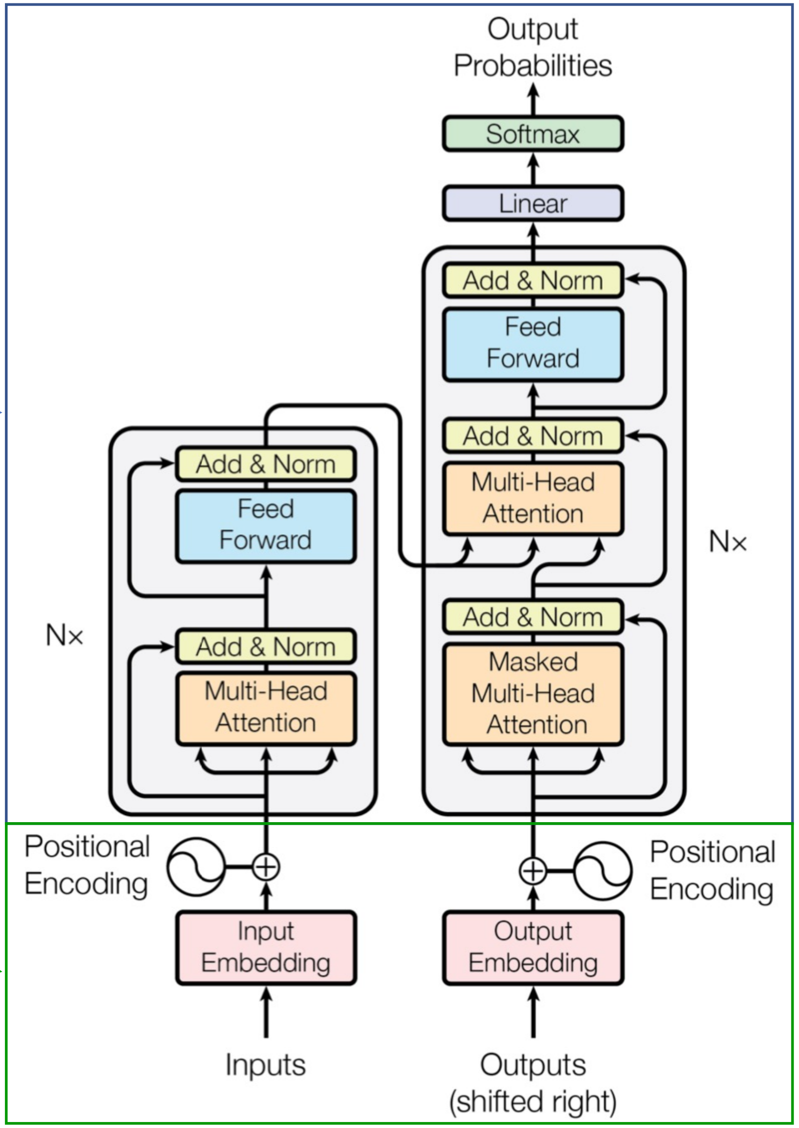


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Transformer — Is Attention All You Need?

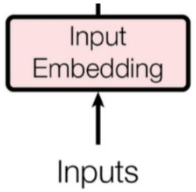
You know this part

We haven't talked about these



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Byte Pair Encoding — From Words to Subwords



- A vocabulary of 50,000 words covers ~95% of the text …
- … this gets you 95% of the way

- Imagine a translation task:
  - “The sewage treatment plant smells particularly special today”
  - “Die Abwasser Behandlungs Anlage riecht heute besonders speziell”

  - “Die **UNKNOWN** riecht heute besonders speziell”

Abwasserbehandlungsanlage?

Byte Pair Encoding — From Words to Subwords

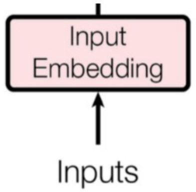
- Traditional NMT has a fixed vocabulary of 30,000 — 50,000 words
  - Rare words are problematic
  - Out-of-vocabulary words even more so
- NMT is an open-vocabulary problem
  - Especially for languages with productive word formation (compounding)
  - E. g. German

→Let's go a level deeper and use sub-word tokens

- Character-level tokens seem computationally infeasible
- Can we do better than that?

→As so often, information theory comes to rescue

Byte Pair Encoding — From Words to Subwords



- Byte Pair Encoding
  - Starting Point: Character-level representation
  - Repeatedly replace most frequent symbol pair (*a*, *b*) with (*ab*)
  - Hyperparameter **m**: When to stop → Vocabulary Size
- Bottom-up character merging
- Example with 10 merges (**m** = original vocab. + 10):

1

Word	Frequency
l o w </w>	5
l o w e r </w>	2
n e w e s t </w>	6
w i d e s t </w>	3

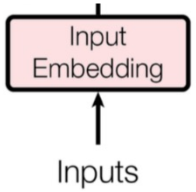
Vocabulary: l o w </w> e r n s t i d

Pairs		**Frequency**
l	o	7
o	w	7
…	…	…
e	**s**	**9**
…	…	…
t	</w>	9

Merge **e** and **s**

End-of-word symbol to restore original tokenization after translation

Byte Pair Encoding — From Words to Subwords



- Byte Pair Encoding
  - Starting Point: Character-level representation
  - Repeatedly replace most frequent symbol pair (*a*, *b*) with (*ab*)
  - Hyperparameter **m:** When to stop → Vocabulary Size
- Bottom-up character merging
- Example with 10 merges (**m** = original vocab. + 10):

2

Word	Frequency
l o w </w>	5
l o w e r </w>	2
n e w **es** t </w>	6
w i d **es** t </w>	3

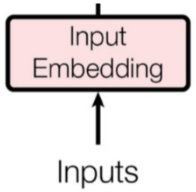
End-of-word symbol to restore original tokenization after translation

Vocabulary: l o w </w> e r n s t i d **es**

Pairs		Frequency
l	o	7
o	w	7
…	…	…
es	**t**	**9**
…	…	…
t	</w>	9

Merge **es** and **t**

Byte Pair Encoding — From Words to Subwords



- Byte Pair Encoding
  - Starting Point: Character-level representation
  - Repeatedly replace most frequent symbol pair (*a*, *b*) with (*ab*)
  - Hyperparameter **m:** When to stop → Vocabulary Size
- Bottom-up character merging
- Example with 10 merges (**m** = original vocab. + 10):

3

Word	Frequency
l o w </w>	5
l o w e r </w>	2
n e w **est** </w>	6
w i d **est** </w>	3

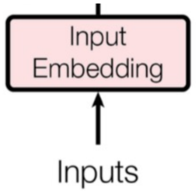
Vocabulary: l o w </w> e r n s t i d es **est**

Pairs		Frequency
l	o	7
o	w	7
…	…	…
est	**</w>**	**9**
…	…	…
d	est	3

Merge **est** and **</w>**

End-of-word symbol to restore
original tokenization after translation

Byte Pair Encoding — From Words to Subwords



- Byte Pair Encoding
  - Starting Point: Character-level representation
  - Repeatedly replace most frequent symbol pair (*a*, *b*) with (*ab*)
  - Hyperparameter **m:** When to stop → Vocabulary Size
- Bottom-up character merging
- Example with 10 merges (**m** = original vocab. + 10):

4

Word	Frequency
l o w </w>	5
l o w e r </w>	2
n e w **est</w>**	6
w i d **est</w>**	3

Vocabulary: l o w </w> e r n s t i d es est …

Pairs		**Frequency**
l	**o**	**7**
o	w	7
…	…	…
w	est</w>	6
…	…	…
d	es	3

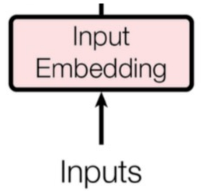
Merge **l** and **o**

End-of-word symbol to restore original tokenization after translation

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

data science
CAIDAS

Byte Pair Encoding — From Words to Subwords



- Byte Pair Encoding
  - Starting Point: Character-level representation
  - Repeatedly replace most frequent symbol pair (*a*, *b*) with (*ab*)
  - Hyperparameter **m**: When to stop → Vocabulary Size
- Bottom-up character merging
- Example with 10 merges (**m** = original vocab. + 10):

10

Word	Frequency
low</w>	5
low e r </w>	2
newest</w>	6
w i d est</w>	3

End-of-word symbol to restore original tokenization after translation

Vocabulary: l o w </w> e r n s t i d es est est</w> lo low ne new newest</w> low</w> wi

Size: Equal to initial vocabulary + amount merges

Byte Pair Encoding — From Words to Subwords

- How does Tokenization work?
  - Let‘s look at „Abwasserbehandlungsanlage“ again
  - Imagine we learned these merges, best at top to worst at bottom

A b
a s
e r
s er
w as
Ab was
Abwas ser
B e
a n
d l
h an
n g
u ng
Be han
dl ung
Behan dlung
A n
a g
l ag

Byte Pair Encoding — From Words to Subwords

- How does Tokenization work?
  - Let's look at „Abwasserbehandlungsanlage" again
  - Imagine we learned these merges, best at top to worst at bottom

A b
a s
e r
s er
w as
Ab was
Abwas ser
B e
a n
d l
h an
n g
u ng
Be han
dl ung
Behan dlung
A n
a g
l ag

1. Split word into characters

A b w a s s e r b e h a n d l u n g s a n l a g e </w>

Byte Pair Encoding — From Words to Subwords

- How does Tokenization work?
  - Let‘s look at „Abwasserbehandlungsanlage“ again
  - Imagine we learned these merges, best at top to worst at bottom

A b
a s
e r
s er
w as
Ab was
Abwas ser
B e
a n
d l
h an
n g
u ng
Be han
dl ung
Behan dlung
A n
a g
l ag

1. Split word into characters

   A b w a s s e r b e h a n d l u n g s a n l a g e </w>

2. Repeatedly pick best merge

   Ab w a s s e r b e h a n d l u n g s a n l a g e </w>

Byte Pair Encoding — From Words to Subwords

- How does Tokenization work?
  - Let's look at „Abwasserbehandlungsanlage" again
  - Imagine we learned these merges, best at top to worst at bottom

A b
a s
e r
s er
w as
Ab was
Abwas ser
B e
a n
d l
h an
n g
u ng
Be han
dl ung
Behan dlung
A n
a g
l ag

1. Split word into characters

   A b w a s s e r b e h a n d l u n g s a n l a g e </w>

2. Repeatedly pick best merge

   Ab w **as** s e r b e h a n d l u n g s a n l a g e </w>

Byte Pair Encoding — From Words to Subwords

- How does Tokenization work?
  - Let's look at „Abwasserbehandlungsanlage“ again
  - Imagine we learned these merges, best at top to worst at bottom

A b
a s
e r
s er
w as
Ab was
Abwas ser
B e
a n
d l
h an
n g
u ng
Be han
dl ung
Behan dlung
A n
a g
l ag

1. Split word into characters

   A b w a s s e r b e h a n d l u n g s a n l a g e </w>

2. Repeatedly pick best merge

   Ab w as s **er** b e h a n d l u n g s a n l a g e </w>

Byte Pair Encoding — From Words to Subwords

- How does Tokenization work?
  - Let‘s look at „Abwasserbehandlungsanlage“ again
  - Imagine we learned these merges, best at top to worst at bottom

A b
a s
e r
s er
w as
Ab was
Abwas ser
B e
a n
d l
h an
n g
u ng
Be han
dl ung
Behan dlung
A n
a g
l ag

1. Split word into characters

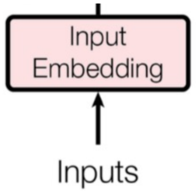
   A b w a s s e r b e h a n d l u n g s a n l a g e </w>

2. Repeatedly pick best merge

   Abwasser b e han dlung s an lag e </w>

3. We now represent our unknown word with ten subtokens

Byte Pair Encoding — From Words to Subwords



- Why Byte Pair Encoding?

- Open Vocabulary
  - Operations learned on training set can be applied to unknown words
- Compression of frequent character sequences (efficiency)

→Trade-off between text length and vocabulary size

Rico Sennrich, Barry Haddow, and Alexandra Birch. (2015). Neural Machine Translation of Rare Words with Subword Units.

Positional Encoding — A Notion of Order

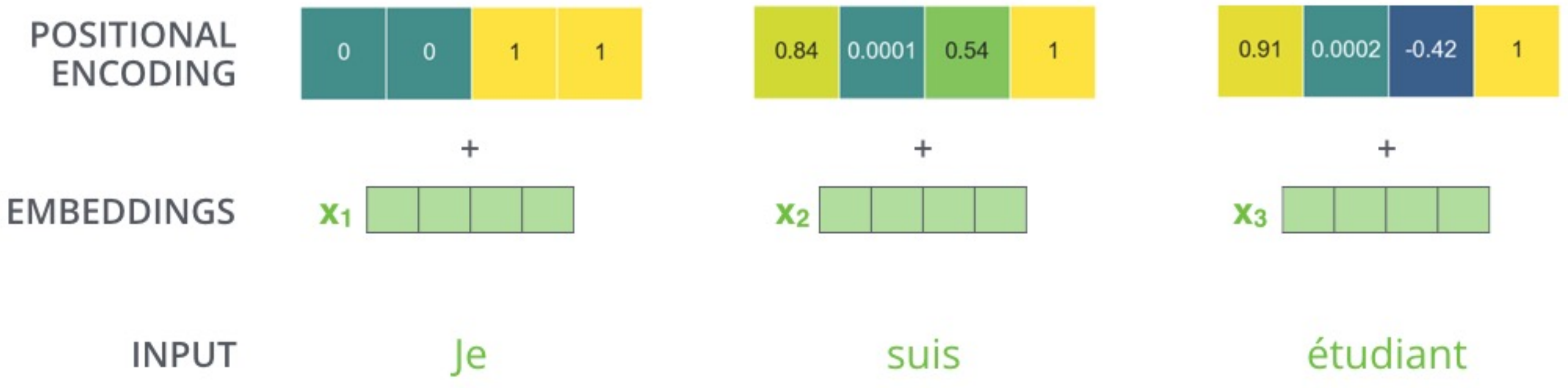
- Position and order of words are essential in any language
- RNNs model these inherently
- Transformers (intentionally) don't have recurrence
  - Massive improvements in speed
  - Potentially longer dependencies are covered
  - But: Inputs loses sequence information

- How can structure be preserved alternatively?
  - Unique encoding for each position in a sentence
  - Distances between positions must be consistent across different length sentences
  - Generalization to longer sentences

Tip: The image is quite telling

Positional Encoding — A Notion of Order

- Idea: Encode this information into our embeddings
  - Add a signal to each embedding that allows meaningful distances between vectors
  - The model learns this pattern



https://jalammar.github.io/illustrated-transformer/

Positional Encoding — A Notion of Order

- Vaswani et al. use sines and cosines of different frequencies
  - There are multiple other options, even learned ones, e. g. Shaw et al.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

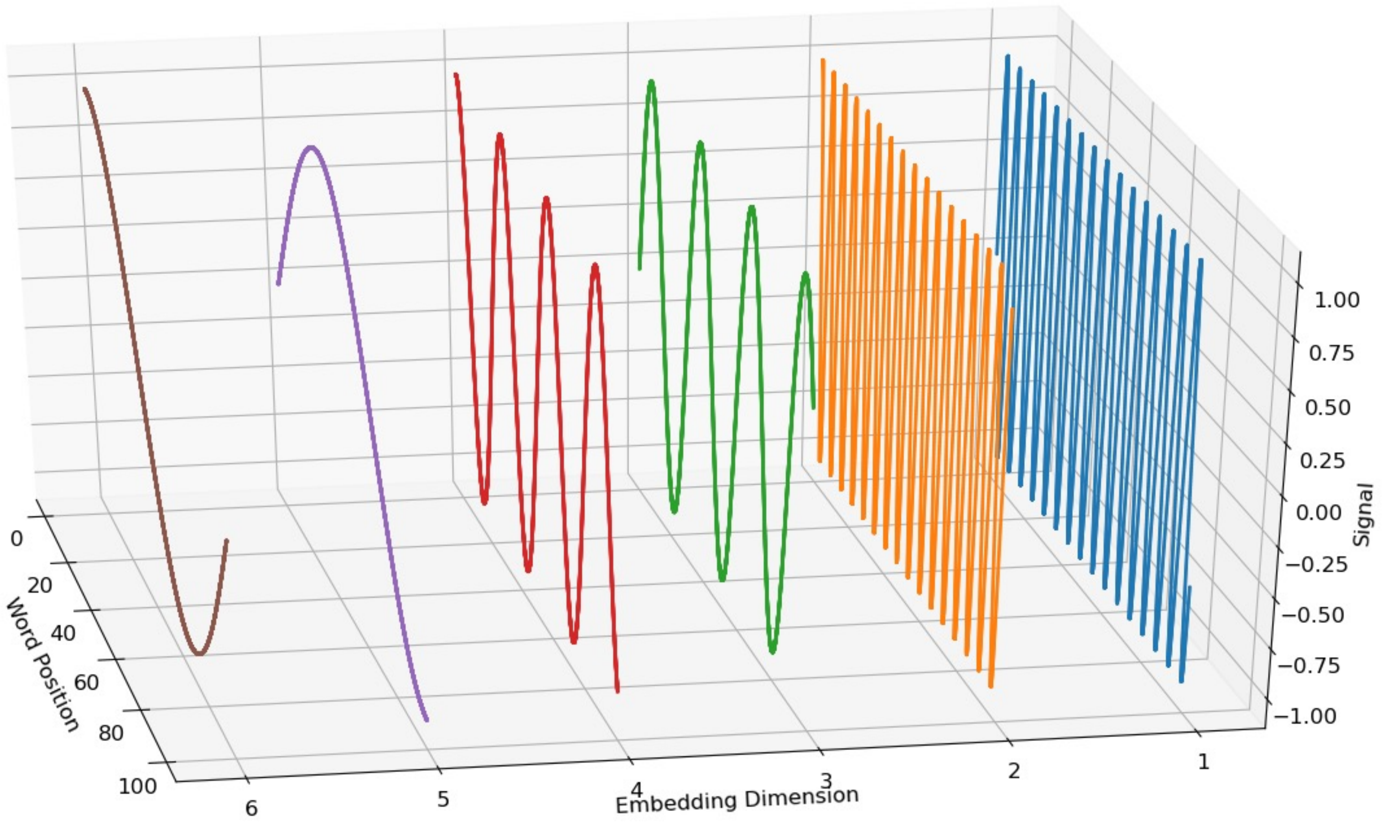
- pos = Word Position, d_{model} = Embedding Dimension, i = i-th Dimension

- Longest sequence with unique position representations: 10000 steps
- For any fixed offset k, PE_{pos+k} can be represented as linear function of PE_{pos}

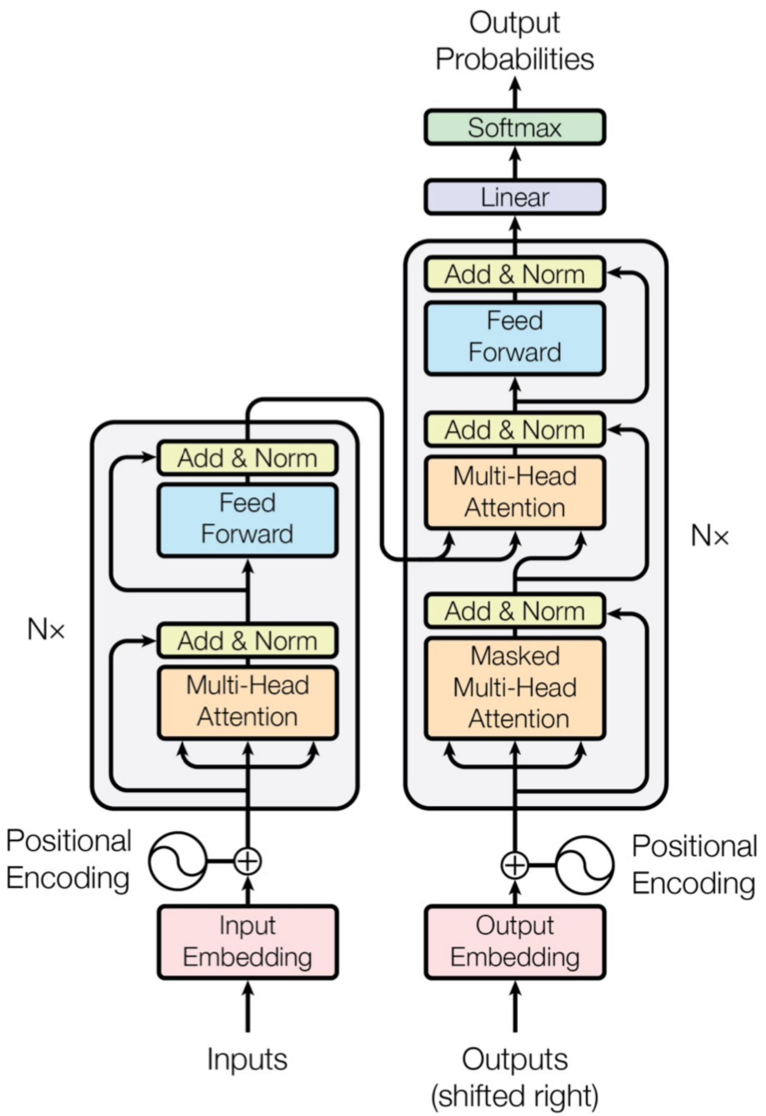
Ashish Vaswani, et al. (2017). Attention Is All You Need.
Peter Shaw, Jakob Uszkoreit, & Ashish Vaswani. (2018). Self-Attention with Relative Position Representations.

Positional Encoding — A Notion of Order

- A visualization helps to understand how this works



Transformer — Is Attention All You Need?



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Transformer — Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Next Step: The Evolved Transformer

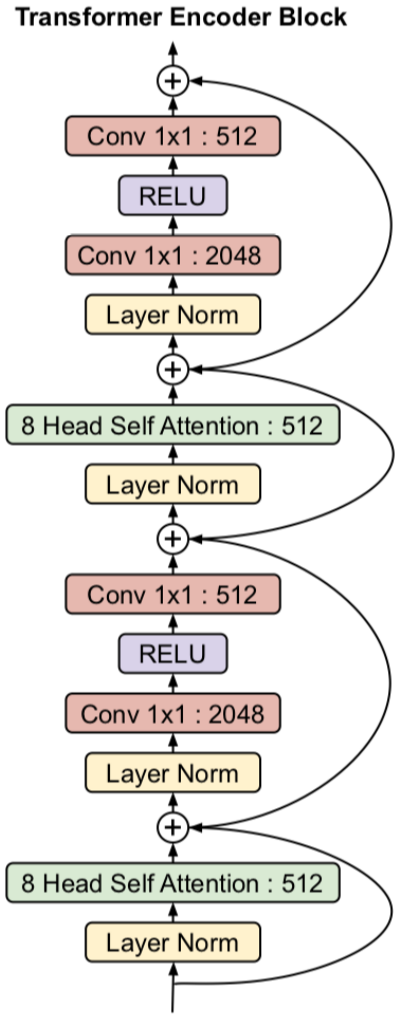
- Transformer architecture is hand-engineered
- Why not let the computer find the best architecture?
- Apply a *neural architecture search* using an Evolution Strategy
  - Randomly create different architectures and test them on the data
  - Mutate the best architectures and repeat testing

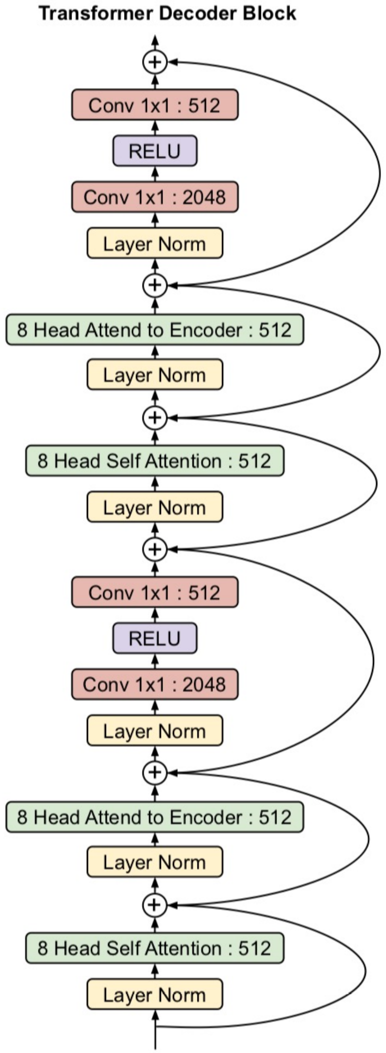
→ The Evolved Transformer

So, David R., Chen Liang, and Quoc V. Le. "The Evolved Transformer." *arXiv preprint arXiv:1901.11117* (2019).

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

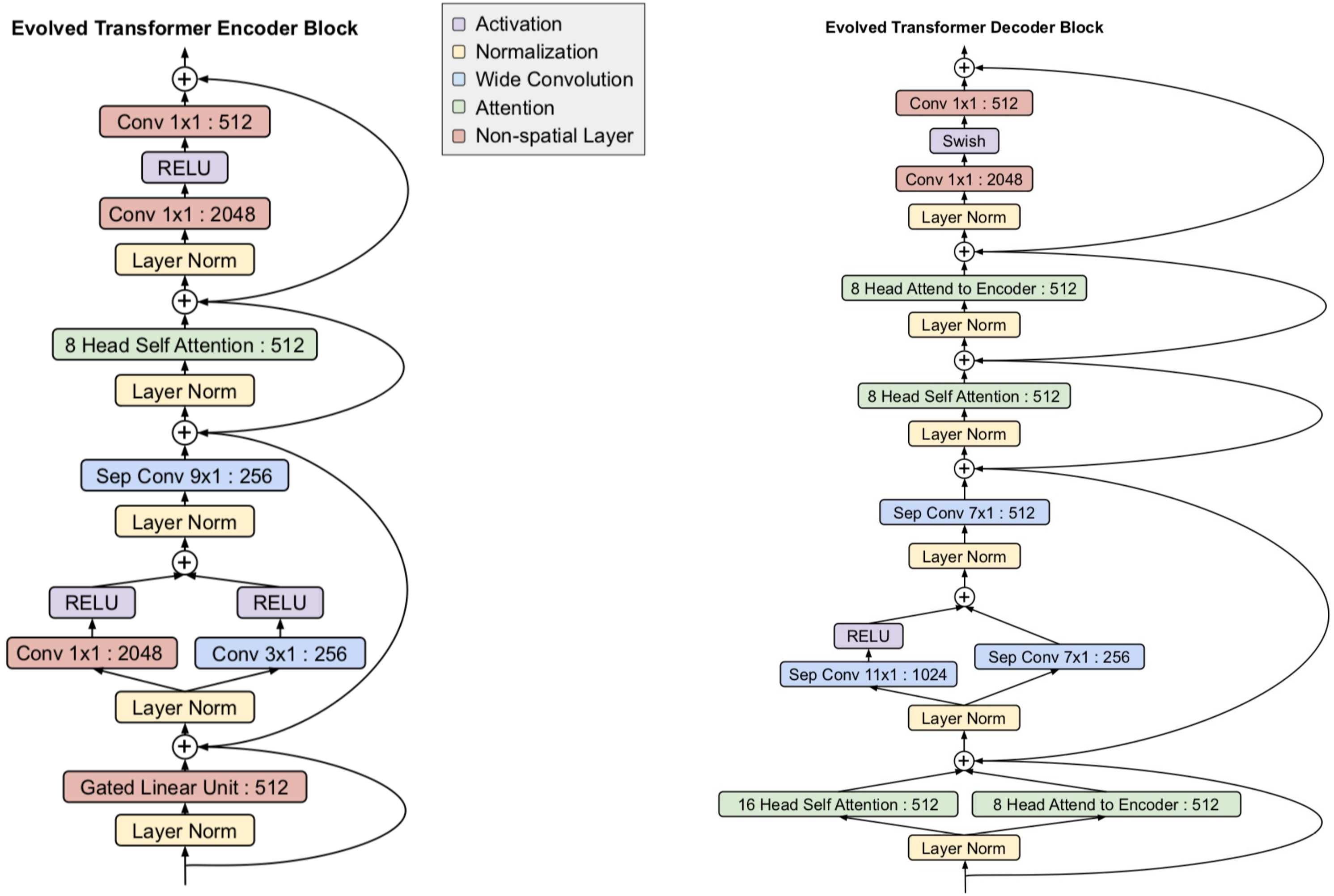
The Transformer





So, David R., Chen Liang, and Quoc V. Le. "The Evolved Transformer." *arXiv preprint arXiv:1901.11117* (2019).

The Evolved Transformer



So, David R., Chen Liang, and Quoc V. Le. "The Evolved Transformer." *arXiv preprint arXiv:1901.11117* (2019).

Evolved Transformer vs. Transformer — Results

Model	Embedding Size	Parameters	Perplexity	BLEU	Δ BLEU
Transformer	128	7.0M	8.62 ± 0.03	21.3 ± 0.1	-
ET	128	7.2M	$\mathbf{7.62} \pm 0.02$	$\mathbf{22.0} \pm 0.1$	**+ 0.7**
Transformer	432	45.8M	4.65 ± 0.01	27.3 ± 0.1	-
ET	432	47.9M	$\mathbf{4.36} \pm 0.01$	$\mathbf{27.7} \pm 0.1$	+ 0.4
Transformer	512	61.1M	4.46 ± 0.01	27.7 ± 0.1	-
ET	512	64.1M	$\mathbf{4.22} \pm 0.01$	$\mathbf{28.2} \pm 0.1$	+ 0.5
Transformer	768	124.8M	4.18 ± 0.01	28.5 ± 0.1	-
ET	768	131.2M	$\mathbf{4.00} \pm 0.01$	$\mathbf{28.9} \pm 0.1$	+ 0.4
Transformer	1024	210.4M	4.05 ± 0.01	28.8 ± 0.2	-
ET	1024	221.7M	$\mathbf{3.94} \pm 0.01$	$\mathbf{29.0} \pm 0.1$	+ 0.2

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Machine Translation — State of the Art

- Neural Machine Translation beats SMT
- Large differences between language pairs:
  Translating between English and French is much easier than between English and German!

- Current research:
  - Machine Translation without parallel data
  - Machine Translation in low resource languages