

Multilingual NLP

4. Attention & Transformer

Prof. Dr. Goran Glavaš
Center for AI and Data Science (CAIDAS), Uni Würzburg

After this lecture, you'll...

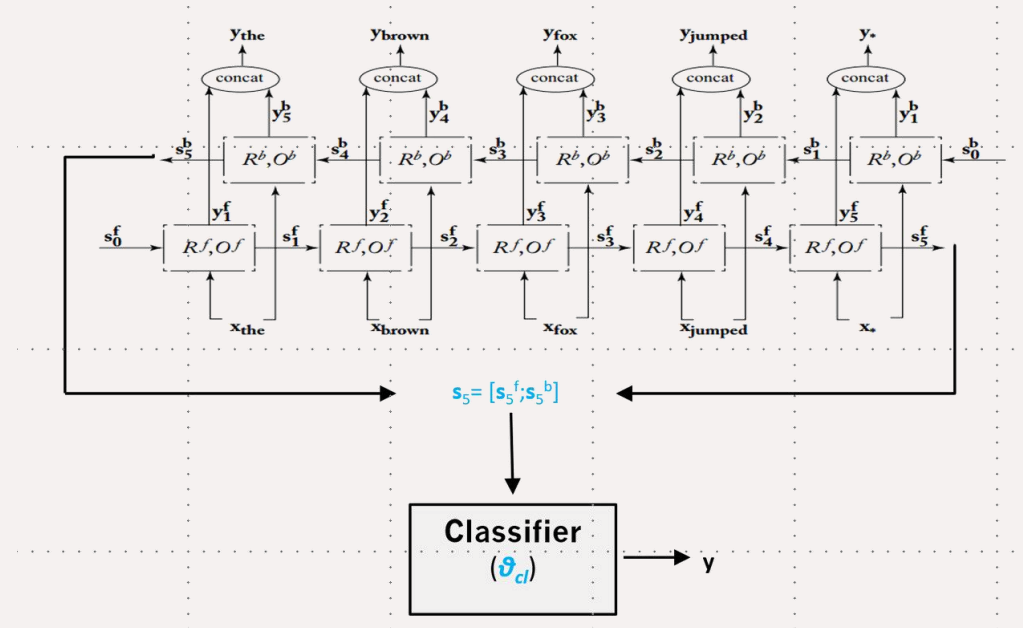
- Understand the concept of attention in NNs
- Know the exact building blocks of the Transformer architecture
- Understand the pre-training-fine-tuning paradigm

Content

- **Attention mechanism**
- Transformer – dissected
 - Positional Embeddings
 - Multi-Head Self-Attention
- Pretraining + fine-tuning

Origins of Attention

- Before the Transformer was introduced (in 2017/2018), recurrent nets were SotA for language understanding and generation
 - E.g., a bidirectional LSTM for sequence classification tasks



Origins of Attention

- Before the Transformer was introduced (in 2017, 2018), recurrent nets were SotA for language understanding and generation
 - E.g., RNN-based sequence-to-sequence models

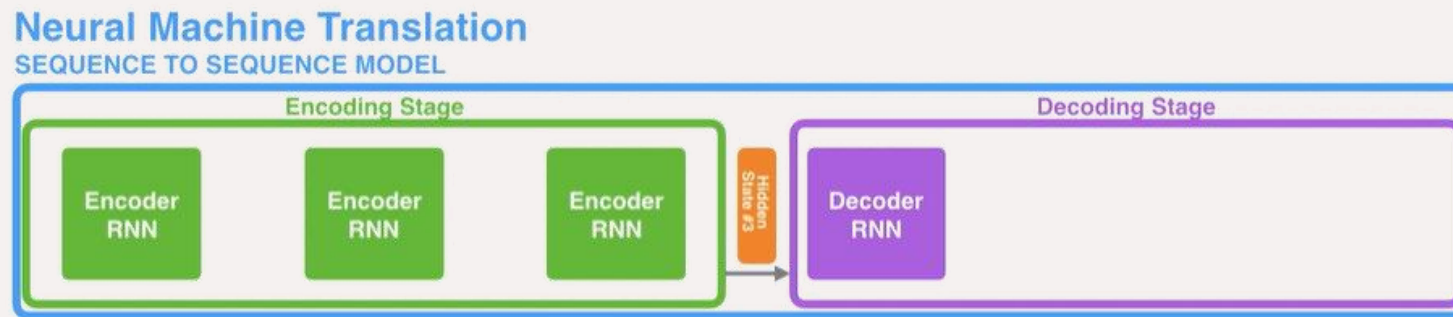
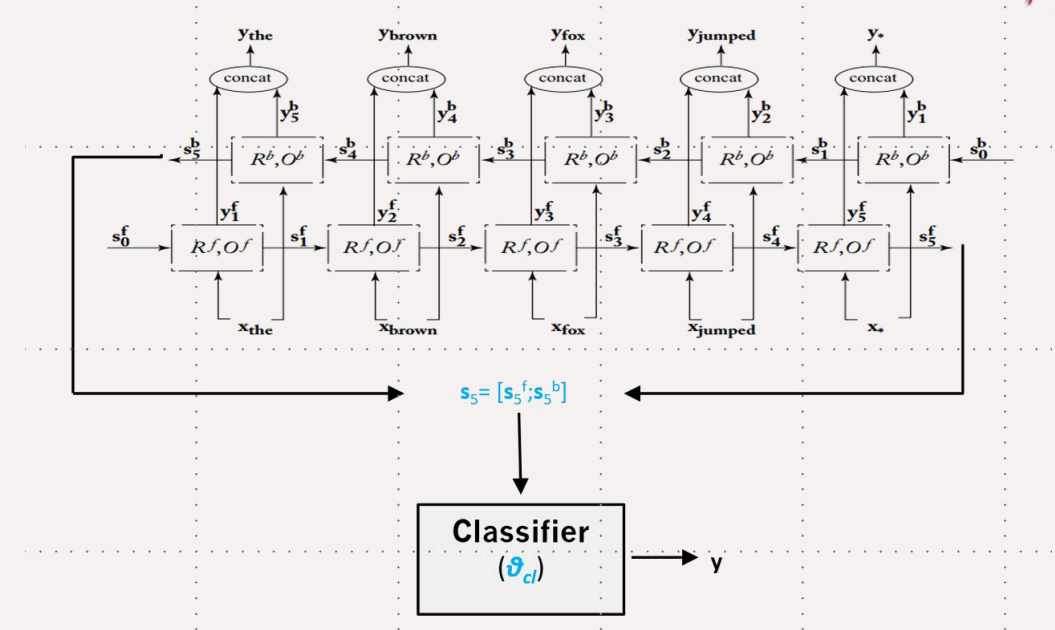


Image from: <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

Origins of Attention

- Problems with RNNs stem from their sequential nature
- Tokens last processed contribute more to the final representation
- Difficult to combine representations of mutually distant tokens (aka long dependencies)
- Tokens not given equal chance to contribute to the sequence representation



Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL

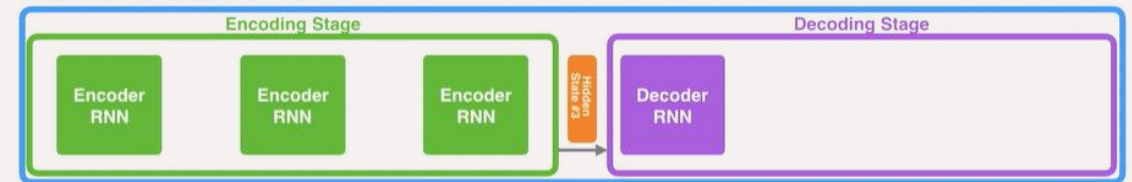
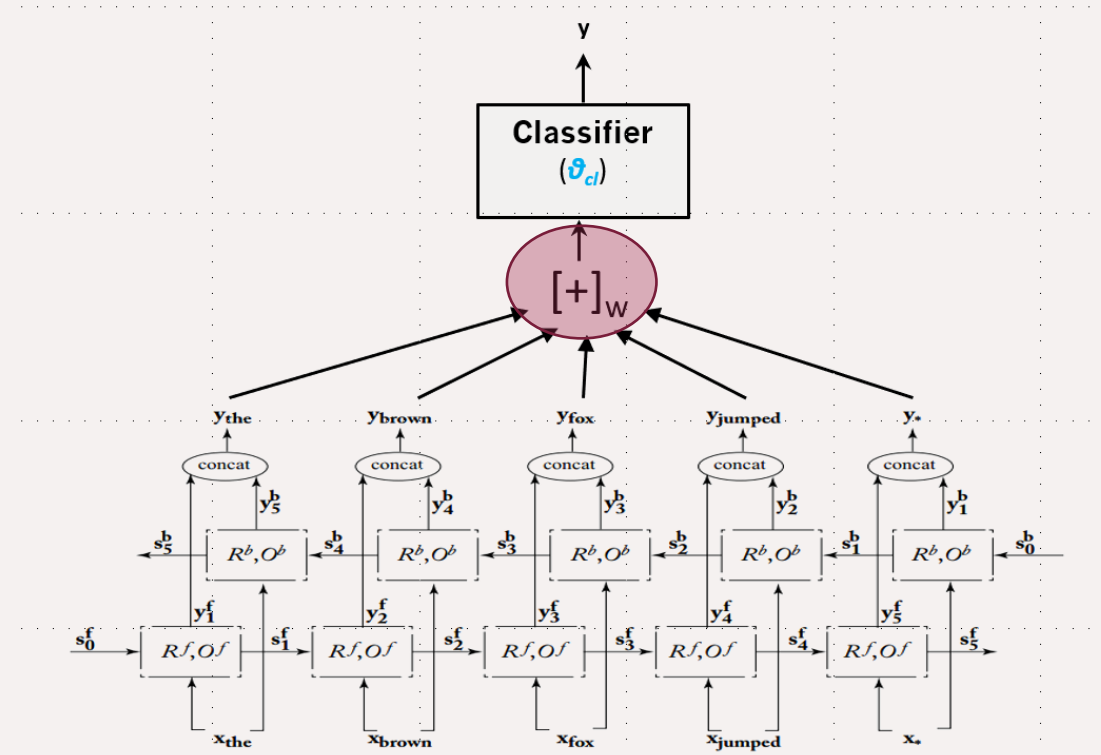


Image from: <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

Origins of Attention

- Enter **attention**: representation of the sequence as a **weighted average of token representations**
- With RNNs, token representations are the hidden state of the RNN after processing of the token
- Weights are produced by a parametrized (i.e., trainable) attention function



Origins of Attention

- Enter **attention**: representation of the sequence as a weighted average of token representations
- In **sequence-to-sequence** (encoder-decoder) for generative tasks
 - At each decoding step, we re-compute the average of the encoded tokens
 - The hidden representation of the decoder is the „**query**“ for the attention mechanism over encoded tokens

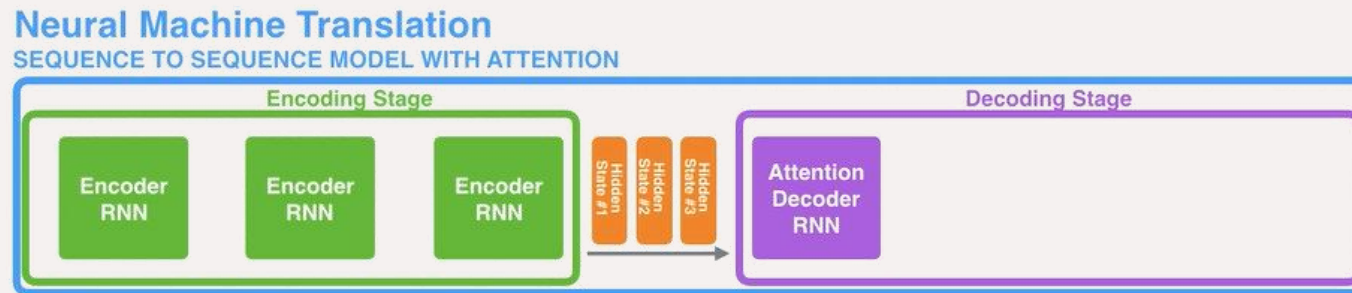


Image from: <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>



Attention Mechanism

- Given a set of objects (e.g., tokens in text), the **attention mechanism** computes a weighted average of **value vectors**
- The weights are based on the similarity between the respective **key vectors** of those same objects with the **query**
- **Query vector** represent the context with respect to which we want to aggregate object representations





Attention Mechanism

- Let t_1, t_2, \dots, t_N be a set of tokens over which we're „attending“
- Let $\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_N \in \mathbb{R}^k$ and $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N \in \mathbb{R}^v$ be the key and value vectors of those tokens, respectively
 - k is the length of key vectors
 - v is the length of value vectors
- Let $\mathbf{q} \in \mathbb{R}^q$ be the query vector
 - In most cases, the query vector must be of same length as key vectors of tokens, $q = k$



Attention Mechanism

- The **attention mechanism** is then defined with by:
 - The **scoring function** $s(\mathbf{q}, \mathbf{k})$, which produces a (single scalar) score that indicates the compatibility of a key \mathbf{k} with the query \mathbf{q}
 - The **output** of the attention mechanism is the weighted sum of value vectors, with corresponding scores as weights:

$$\sum_{i=1}^N s(\mathbf{q}, \mathbf{k}_i) * \mathbf{v}_i$$



Attention Mechanism

$$\sum_{i=1}^N s(\mathbf{q}, \mathbf{k}_i) * \mathbf{v}_i$$

- Commonly used attention types (before Transformer):

1. Additive attention (parametrized scoring function):

$$s(\mathbf{q}, \mathbf{k}) = \mathbf{v}_a \tanh(\mathbf{W}_a (\mathbf{k} \oplus \mathbf{q}))$$

- \mathbf{k} and \mathbf{q} may be of different length, \oplus denotes concatenation
- $\mathbf{W}_a \in \mathbb{R}^{h \times (k+q)}$ and $\mathbf{v}_a \in \mathbb{R}^h$: trainable params of the „attention layer“



Attention Mechanism

$$\sum_{i=1}^N s(\mathbf{q}, \mathbf{k}_i) * \mathbf{v}_i$$

- Commonly used attention types used (before Transformer):

2. Dot-product attention (non-parametrized scoring function)

$$s(\mathbf{q}, \mathbf{k}) = \mathbf{k}^T \mathbf{q}$$

- Raw (unnormalized) score for a key is a simple dot-product with the query
- Raw scores across keys are normalized with softmax:

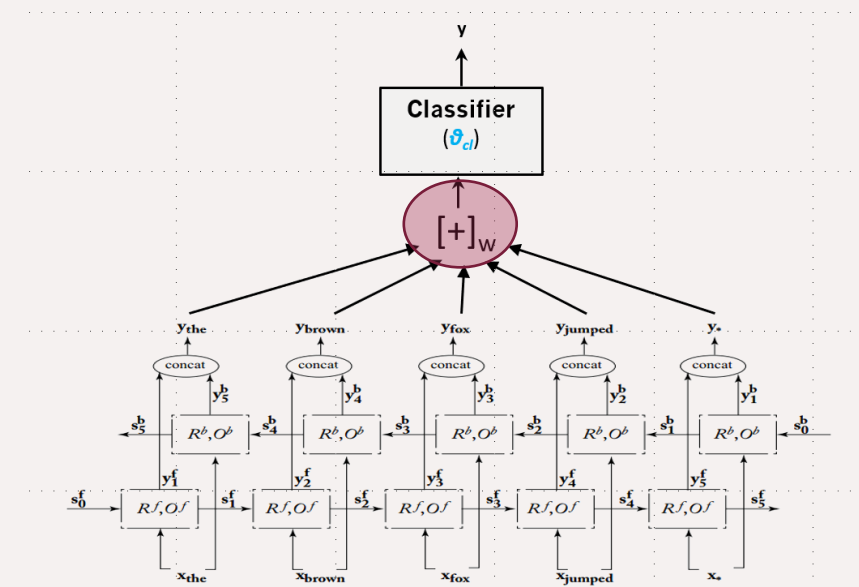
$$s(\mathbf{q}, \mathbf{k}) \rightarrow \frac{e^{s(\mathbf{q}, \mathbf{k}_i)}}{\sum_{j=1}^N e^{s(\mathbf{q}, \mathbf{k}_j)}}$$

Attention Mechanism

- Q: But where are keys, values, and queries coming from?
 - For each token we typically have only one vector
 - Token's embedding or RNN's state after processing that token
- Attention is a **general mechanism**, can be applied in various settings
 - We decide how to obtain keys, values, and queries in concrete use cases
- Use case #1: sequence classification with RNNs
 - Keys and values same vectors, one for hidden state of the RNN at each time step

$$\mathbf{k}_i = \mathbf{v}_i = \mathbf{s}_i^{\text{RNN}}$$

- No context: **q** can be any fixed vector (e.g., a vector of 1s), or a trainable vector;

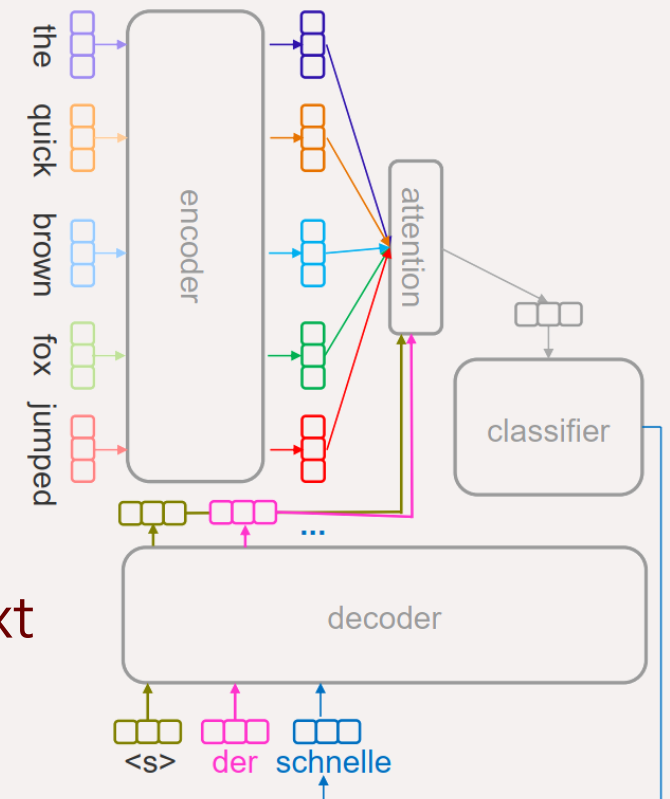


Attention Mechanism

- Attention is a **general mechanism**, can be applied in various settings
 - We decide how to obtain keys, values, and queries in concrete use cases and scenarios
- Use case #2: seq-to-seq generation (with RNNs)
 - Keys and values the same vectors, one for hidden state of encoder RNN at each time step

$$\mathbf{k}_i = \mathbf{v}_i = \mathbf{s}_i^{\text{Encoder}}$$

- Query **q**: hidden state of the decoder
 - I.e., context is the representation of the text generated so far



Content

- Attention mechanism
- **Transformer - dissected**
 - Positional Embeddings
 - Multi-Head Self-Attention
- Pretraining + fine-tuning

Is Attention All We Need?



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems (NeurIPS).

- So far, we applied attention over **keys/values** that come from a **recurrent encoder**
- RNNs are **slow** to train
 - „Backpropagation through time“
 - Computation over tokens **sequential**
- **Research question** that changed NLP and enabled LLMs:
 - Is recurrence actually **necessary**?
 - What happens if we just apply attention on top of token embeddings?

Transformer (Encoder-Decoder)



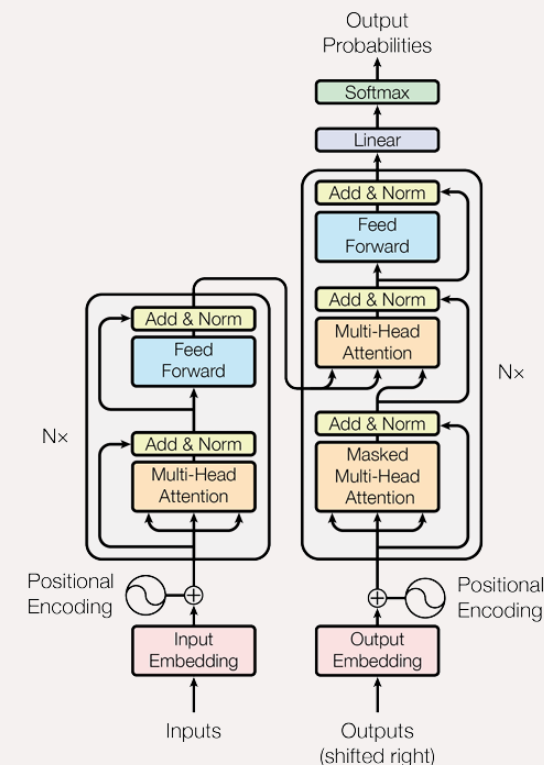
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). [Attention is all you need](#). Advances in neural information processing systems (NeurIPS).

- Enter **Transformer**: a sequence-to-sequence architecture without recurrence, based **„only“** on the attention mechanism
- Is **attention** is all we need?



Tal Linzen
@tallinzen

Attention is all you need, except residual connections, layer norm, position embeddings, extra feedforward layers, multiple heads, and masking future words

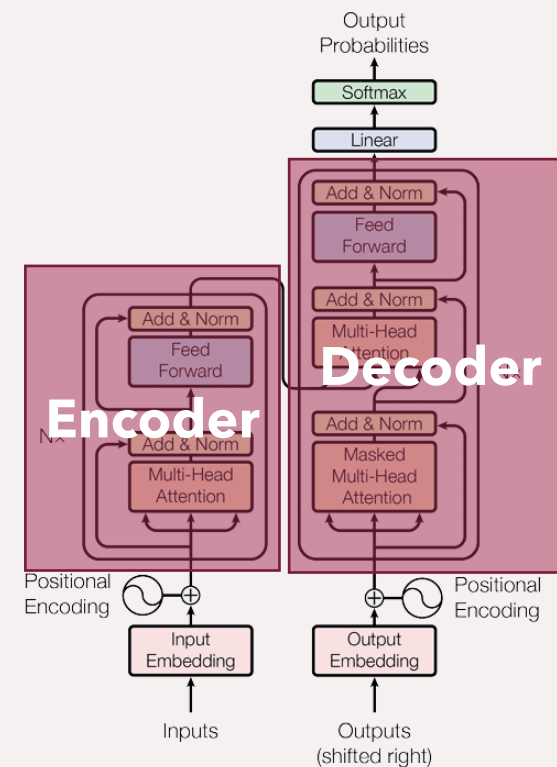


Transformer (Encoder-Decoder)



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). [Attention is all you need](#). Advances in neural information processing systems (NeurIPS).

- **Transformer** as proposed by Vaswani et al. is an encoder-decoder model
 - Introduced for machine translation
- Two types of attention
 1. **Self-attention**: only in encoder
 - Keys, Values, and Queries all derived from token representations in encoder layers
 2. **Cross-attention**: keys and values from encoder representations + previous tokens in decoder
 - **q**: from representations of decoder tokens



Transformer (Encoder-Decoder)



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). [Attention is all you need](#). Advances in neural information processing systems (NeurIPS).

- **Transformer** as proposed by Vaswani et al. is an encoder-decoder model
 - Introduced for machine translation
- Two types of attention

1. **Self-attention:** only in encoder

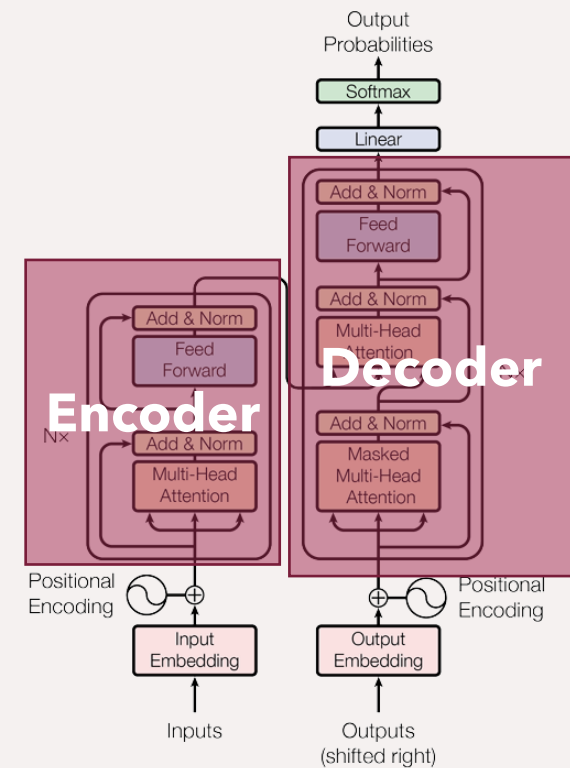
- Keys, Values, and Queries all derived from token representations in encoder layers

Today

2. **Cross-attention:** keys & values from enc. representations + previous tokens in decoder

- **q**: from representations of decoder tokens

L9: NMT



Transformer



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- **Transformer** as proposed by Vaswani et al. is an encoder-decoder model
- Most NLP tasks are not generation tasks*
 - * Most recently, with LLMs, many non-generation tasks have been successfully re-cast as generation tasks (**L11**: Prompting & LLMs)
- Most models used today are single-stack transformers
 - Encoder models (trained with masked LM-ing)
 - Decoder models (trained with autoregressive LM-ing)
- Devlin et al.'s **BERT** uses the encoder-only Transformer
 - This is the Transformer that we'll primarily dissect in this lecture

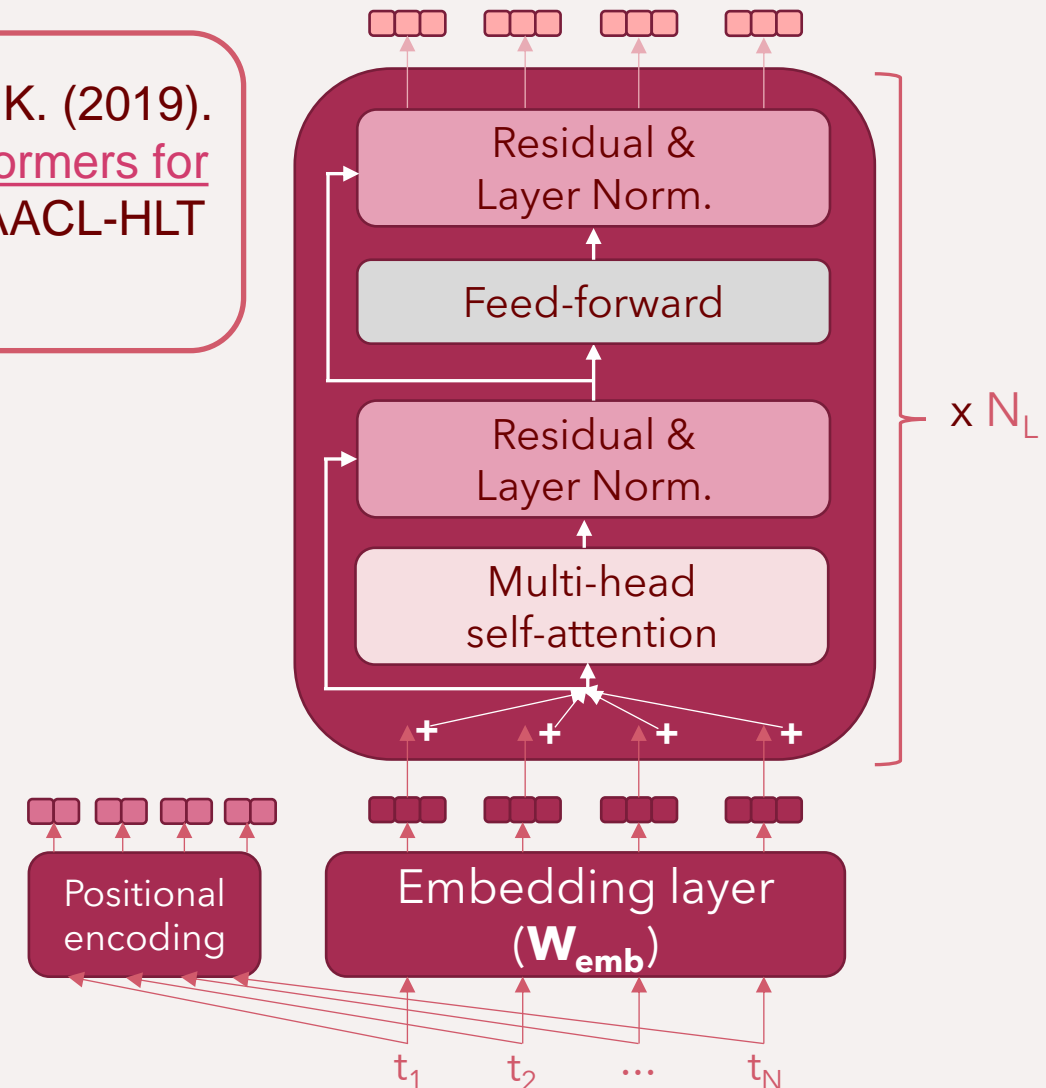


Transformer



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- **Transformer as encoder**
 - Embedding layer
 - (sub)word embeddings
 - Positional embeddings
 - N_L identical Transformer layers
 - Multi-head self-attention sublayer
 - Residual connection
 - Layer normalization
 - Feed-forward sublayer

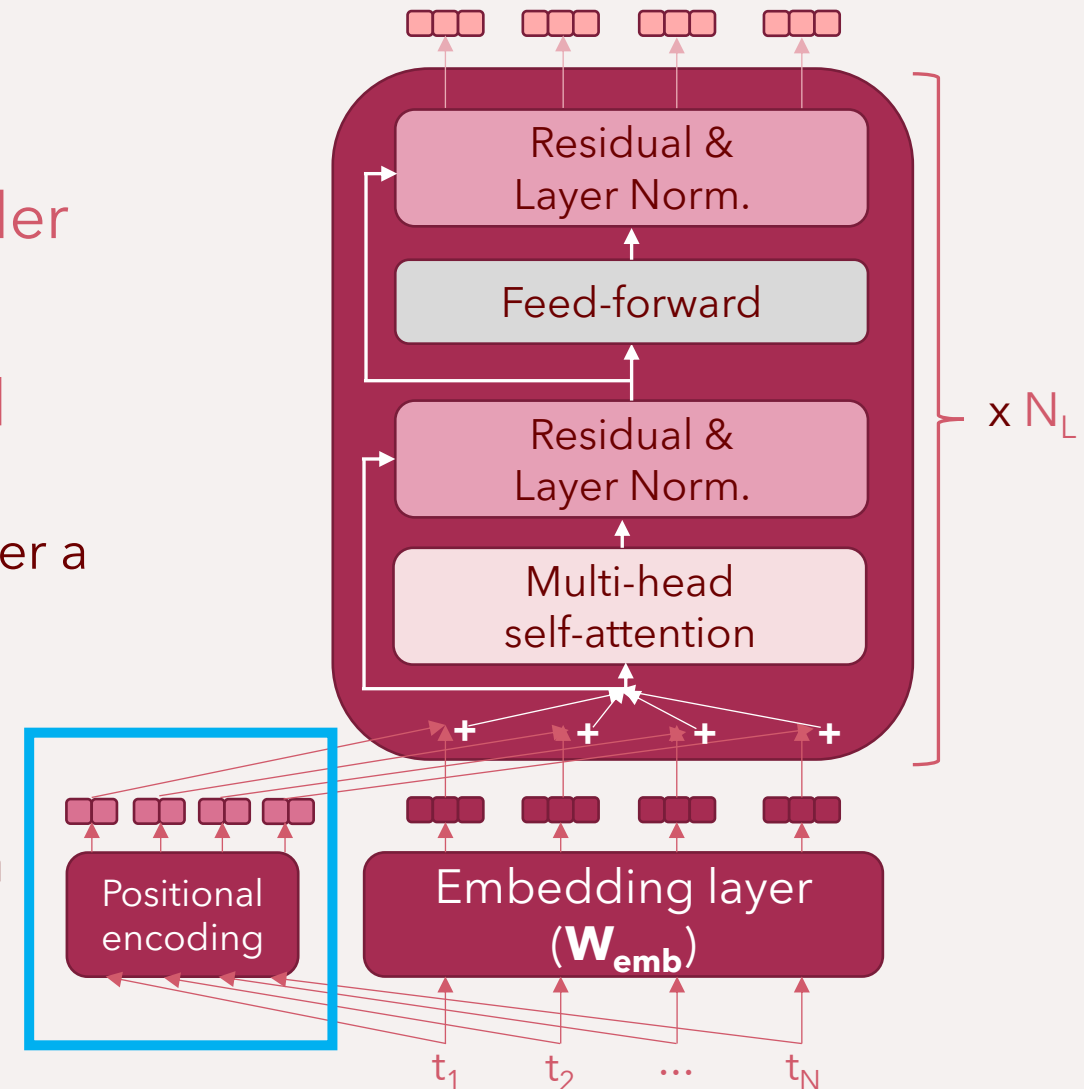


Content

- Attention mechanism
- **Transformer - dissected**
 - **Positional Embeddings**
 - Multi-Head Self-Attention
- Pretraining + fine-tuning

Positional Embeddings

- Recurrent networks implicitly retain the information about the word order
- At the core of the Transformer encoder is the so-called multi-head self-attention
 - But attention is just an aggregation over a set of vectors
 - Retains no order information
- Fix: **positional embeddings** that explicitly encode token positions in the sequence

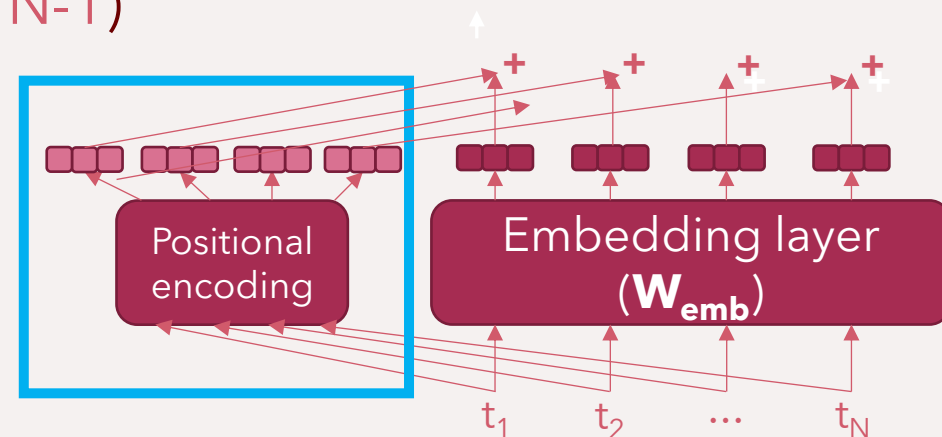


Positional Embeddings



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). [Attention is all you need](#). Advances in neural information processing systems (NeurIPS).

- Vaswani et al. propose **fixed relative positional embeddings**
- Positional embeddings added to (sub)word embeddings \rightarrow same dim. d
- Maximal sequence length: N
- Position in the sequence: **pos**, in $\{0, 1, 2, \dots, N-1\}$
- For each position/index in the pos. embedding, a different function generating the score
 - Indices: $2i$ (or $2i+1$)
for $0 \leq i < d/2$



Positional Embeddings

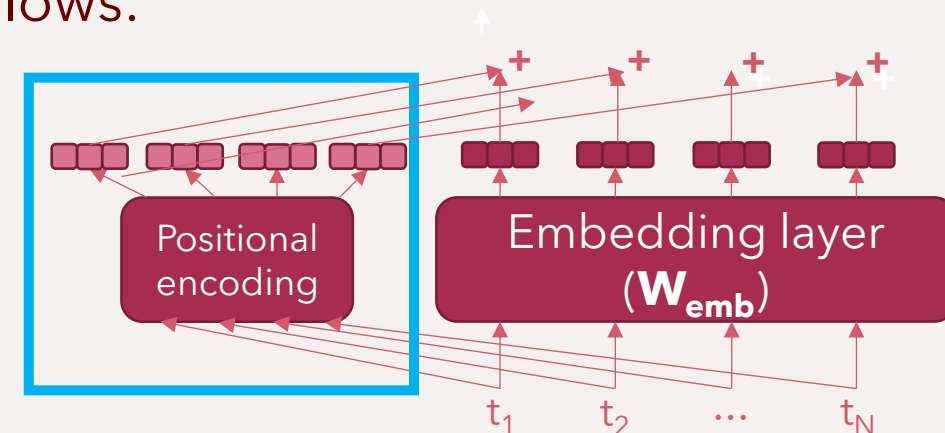


Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). [Attention is all you need](#). Advances in neural information processing systems (NeurIPS).

- Let n be an arbitrary scalar, a hyperparameter for positional embeddings
 - Vaswani et al. set $n = 10000$
- The value of the positional embedding for token position pos and embedding index $2i$ (or $2i+1$) is given as follows:

$$PE(pos, 2i) = \sin\left(\frac{pos}{n^{2i/d}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{n^{2i/d}}\right)$$



Sines and Cosines?!

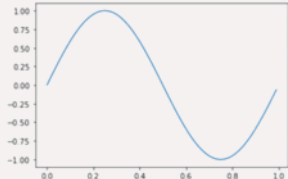
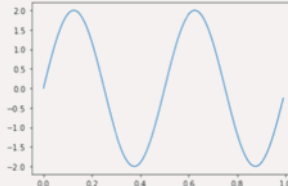
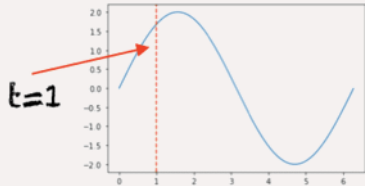
Equation	Graph	Frequency	Wavelength
$\sin(2\pi t)$		1	1
$\sin(2 * 2\pi t)$		2	1/2
$\sin(t)$		$1/2\pi$	2π
$\sin(ct)$	Depends on c	$c/2\pi$	$2\pi/c$

Image from: <https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/>

Sines and Cosines?!

- Q: Why sines and cosines?!
 - Cyclical, encode well relative values in the argument range $[0, 2\pi]$
 - We can make the range of the repetition (**wavelength**) arbitrarily long
 - Values always in the $[-1, 1]$ range

$$PE(pos, 2i) = \sin\left(\frac{pos}{n^{2i/d}}\right); \quad PE(pos, 2i+1) = \cos\left(\frac{pos}{n^{2i/d}}\right)$$

- $c = 1/n^{2i/d}$; wavelength $\rightarrow 2\pi/c = 2\pi * n^{2i/d}$
 - The bigger the index i , the bigger the wavelength (wider cycle)
- For $i = 0$ (ind. 0 and 1) $\rightarrow PE(pos, 0) = \sin(pos)$; $PE(pos, 1) = \cos(pos)$
- For $i = 1$ (ind. 2 and 3) $\rightarrow PE(pos, 2) = \sin\left(\frac{pos}{n^{2/d}}\right)$; $PE(pos, 3) = \cos\left(\frac{pos}{n^{2/d}}\right)$
- ...
- For $i = d/2$ (index d , assume d even) $\rightarrow PE(pos, d) = \sin\left(\frac{pos}{n}\right)$

Sines and Cosines?!

- For different indices of the positional embedding vectors, position-dependent values are computed with \sin/\cos of different wavelengths
 - From wavelength of 2π (for $i = 0$) to wavelength of $n * 2\pi$ (for $i = d/2$)
- Store pos. embeddings in a matrix $\mathbf{W}_{PE} \in \mathbb{R}^{N \times d}$
 - Each row corresponds to one position (from 1 to N)
 - We can visualize \mathbf{W}_{PE} - example with $d = 128$, $N = 50$, and $n = 10000$

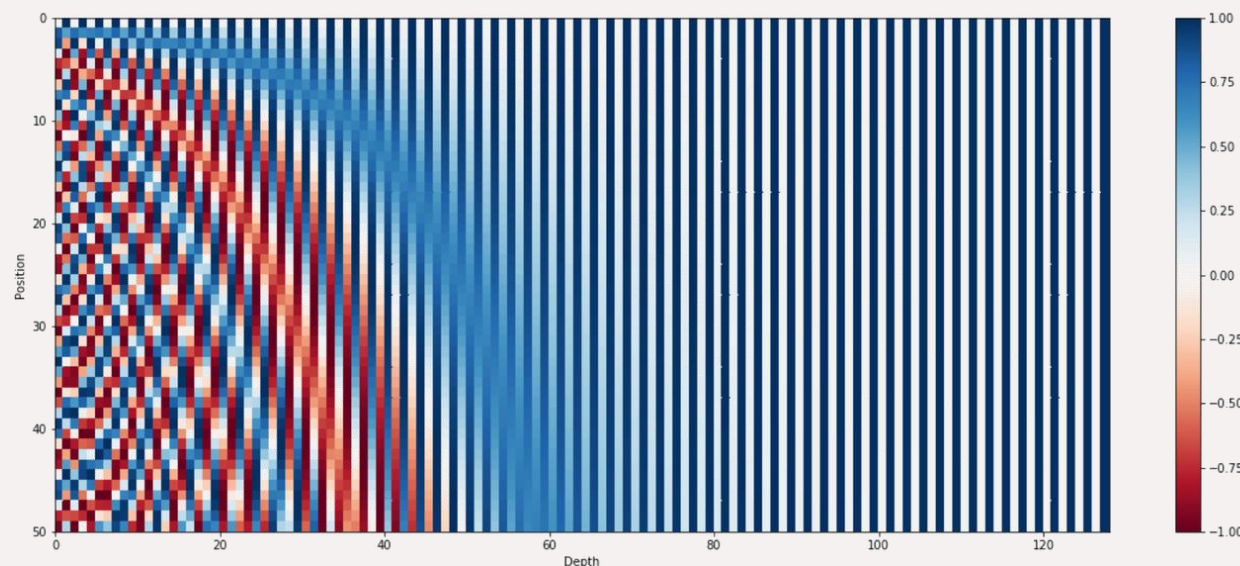


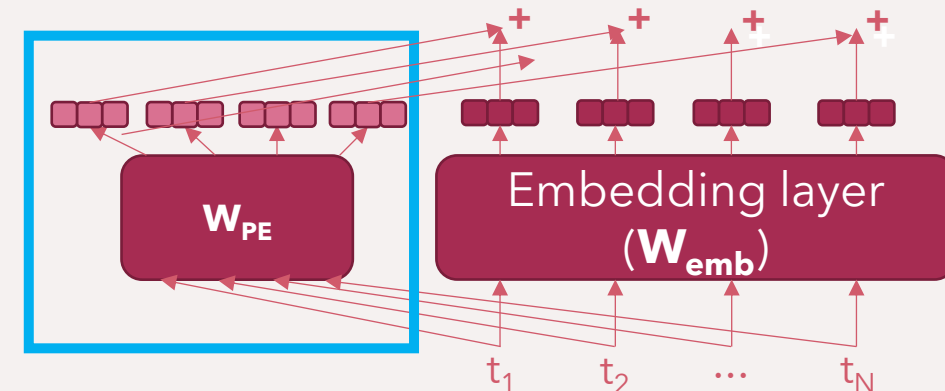
Image from: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Embeddings



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#). In Proceedings of NAACL-HLT (pp. 4171-4186).

- In BERT, Devlin et al. resort to fully trainable positional embeddings
- I.e., $\mathbf{W}_{PE} \in \mathbb{R}^{N \times d}$ another parameter matrix, along with $\mathbf{W}_{emb} \in \mathbb{R}^{|V| \times d}$
- \mathbf{W}_{PE} optimized with all other parameters of the whole neural LM

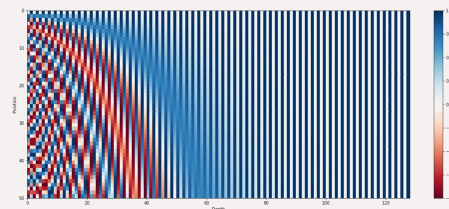


Positional Embeddings

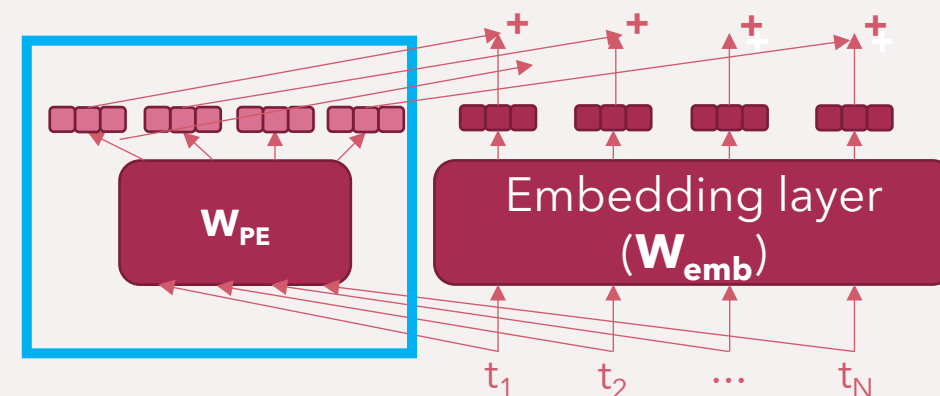


Wang, B., Shang, L., Lioma, C., Jiang, X., Yang, H., Liu, Q., & Simonsen, J. G. (2021). [On Position Embeddings in BERT](#). In International Conference on Learning Representations (ICLR).

- Which is better?
 - Fully trainable PEs or
 - Fixed relative PEs?
- The answer is not straightforward, seems to depend on the type of task
- Sequence classification: trainable PEs
- Token classification / span extraction: fixed relative PEs



vs.

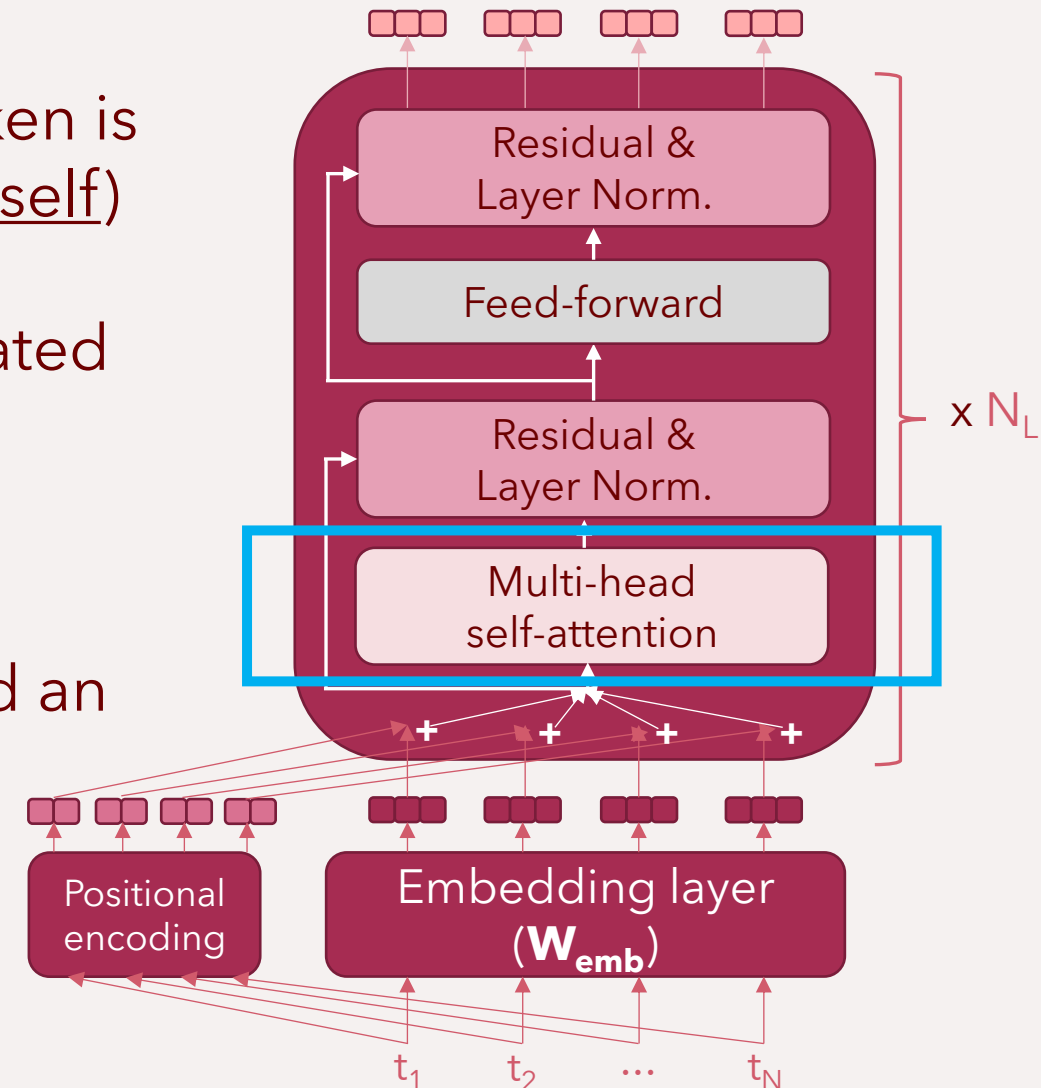


Content

- Attention mechanism
- **Transformer - dissected**
 - Positional Embeddings
 - **Multi-Head Self-Attention**
- Pretraining + fine-tuning

Multi-Head Attention

- **Self-attention** in Transformer: each token is „attending“ over all tokens (including itself)
- Each token t_i ($i = 1, \dots, N$) has an associated
 - Key vector \mathbf{k}_i
 - Value vector \mathbf{v}_i
 - Query vector \mathbf{q}_i
- One self-attention mechanism is called an attention head
 - Multi-head attention: multiple self-attention mechanisms





Self-Attention (Attention Head)

- **Self-attention:** each token is „attending“ over all tokens
- Each token t_i ($i = 1, \dots, N$) has an associated
 - Key vector \mathbf{k}_i , value vector \mathbf{v}_i , and query vector \mathbf{q}_i
- Let \mathbf{x}_i be the embedding of t_i
 - I.e., sum of subword emb. and PE
 - **Q:** how do we obtain three different vectors (\mathbf{k}_i , \mathbf{v}_i , and \mathbf{q}_i) from \mathbf{x}_i ?
 - Introduce trainable parameters that project \mathbf{x}_i into different vectors
- Stack embeddings of tokens ($\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$) into a matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$
 - In later transformer layers, \mathbf{X} is not the matrix of embeddings from the embedding layer but is the the output of the previous layer



Self-Attention (Attention Head)

- **Self-attention:** Introduce trainable parameters that project **X** into matrices **K**, **V**, and **Q**
- Query matrix: $\mathbf{Q} = \mathbf{X} \mathbf{W}^{\mathbf{Q}}$, $\mathbf{W}^{\mathbf{Q}} \in \mathbb{R}^{d \times k}$
- Key matrix: $\mathbf{K} = \mathbf{X} \mathbf{W}^{\mathbf{K}}$, $\mathbf{W}^{\mathbf{K}} \in \mathbb{R}^{d \times k}$
- Value matrix: $\mathbf{V} = \mathbf{X} \mathbf{W}^{\mathbf{V}}$, $\mathbf{W}^{\mathbf{V}} \in \mathbb{R}^{d \times v}$
- $\mathbf{Q} \in \mathbb{R}^{N \times k}$, $\mathbf{K} \in \mathbb{R}^{N \times k}$, and $\mathbf{V} \in \mathbb{R}^{N \times v}$
 - Query, key, and value vectors are not necessarily of same length d as input emb.
 - **Q** and **K** are of the same dimensionality

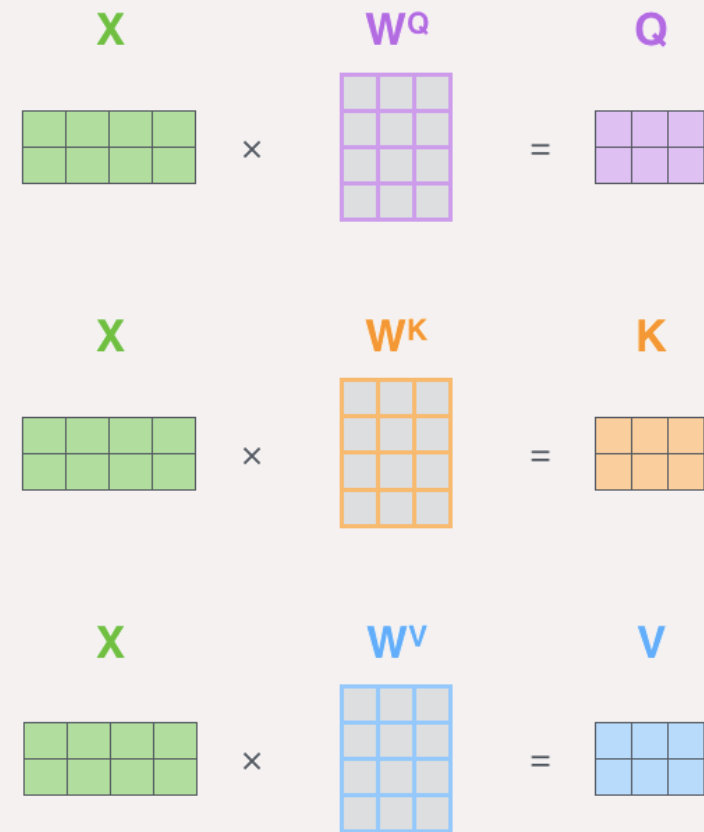


Image from: <https://jalammar.github.io/illustrated-transformer/>

Self-Attention (Attention Head)

- Query matrix: $\mathbf{Q} = \mathbf{X} \mathbf{W}^{\mathbf{Q}}, \mathbf{W}^{\mathbf{Q}} \in \mathbb{R}^{d \times k}$
- Key matrix: $\mathbf{K} = \mathbf{X} \mathbf{W}^{\mathbf{K}}, \mathbf{W}^{\mathbf{K}} \in \mathbb{R}^{d \times k}$
- Value matrix: $\mathbf{V} = \mathbf{X} \mathbf{W}^{\mathbf{V}}, \mathbf{W}^{\mathbf{V}} \in \mathbb{R}^{d \times v}$
- Output of the Transformer's self-attention is computed as:

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{k}}\right)\mathbf{V}; \quad \mathbf{Z} \in \mathbb{R}^{N \times v}$$

- The matrix $\mathbf{Q}\mathbf{K}^{\mathbf{T}} \in \mathbb{R}^{N \times N}$ is called an attention matrix
 - Often used for interpretability, how much each token attends over each other token
- softmax is applied row-wise on $\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{k}}$
- Q: Why normalization with \sqrt{k} ?

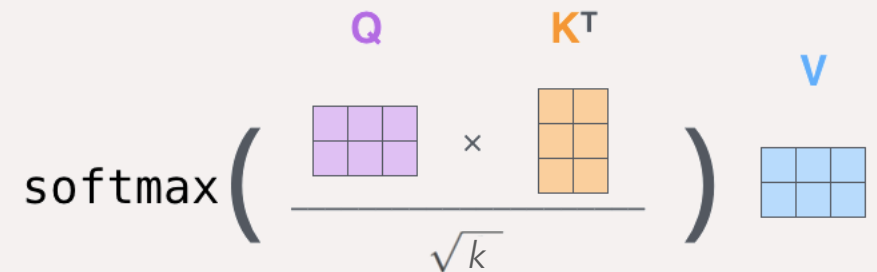


Image from: <https://jalammr.github.io/illustrated-transformer/>

Multi-Head Attention

- Simply **multiple attention heads**, independent of each other
 - Just operate on the same input **X**
 - **H** attention heads

1st attention head:

- $\mathbf{Q}_1 = \mathbf{X} \mathbf{W}_1^{\mathbf{Q}}; \mathbf{K}_1 = \mathbf{X} \mathbf{W}_1^{\mathbf{K}}; \mathbf{V}_1 = \mathbf{X} \mathbf{W}_1^{\mathbf{V}};$
- $\mathbf{Z}_1 = \text{softmax}\left(\frac{\mathbf{Q}_1 \mathbf{K}_1^T}{\sqrt{k}}\right) \mathbf{V}_1$

...

H-th attention head

- $\mathbf{Q}_H = \mathbf{X} \mathbf{W}_H^{\mathbf{Q}}; \mathbf{K}_H = \mathbf{X} \mathbf{W}_H^{\mathbf{K}}; \mathbf{V}_H = \mathbf{X} \mathbf{W}_H^{\mathbf{V}};$
- $\mathbf{Z}_H = \text{softmax}\left(\frac{\mathbf{Q}_H \mathbf{K}_H^T}{\sqrt{k}}\right) \mathbf{V}_H$

Multi-Head Attention

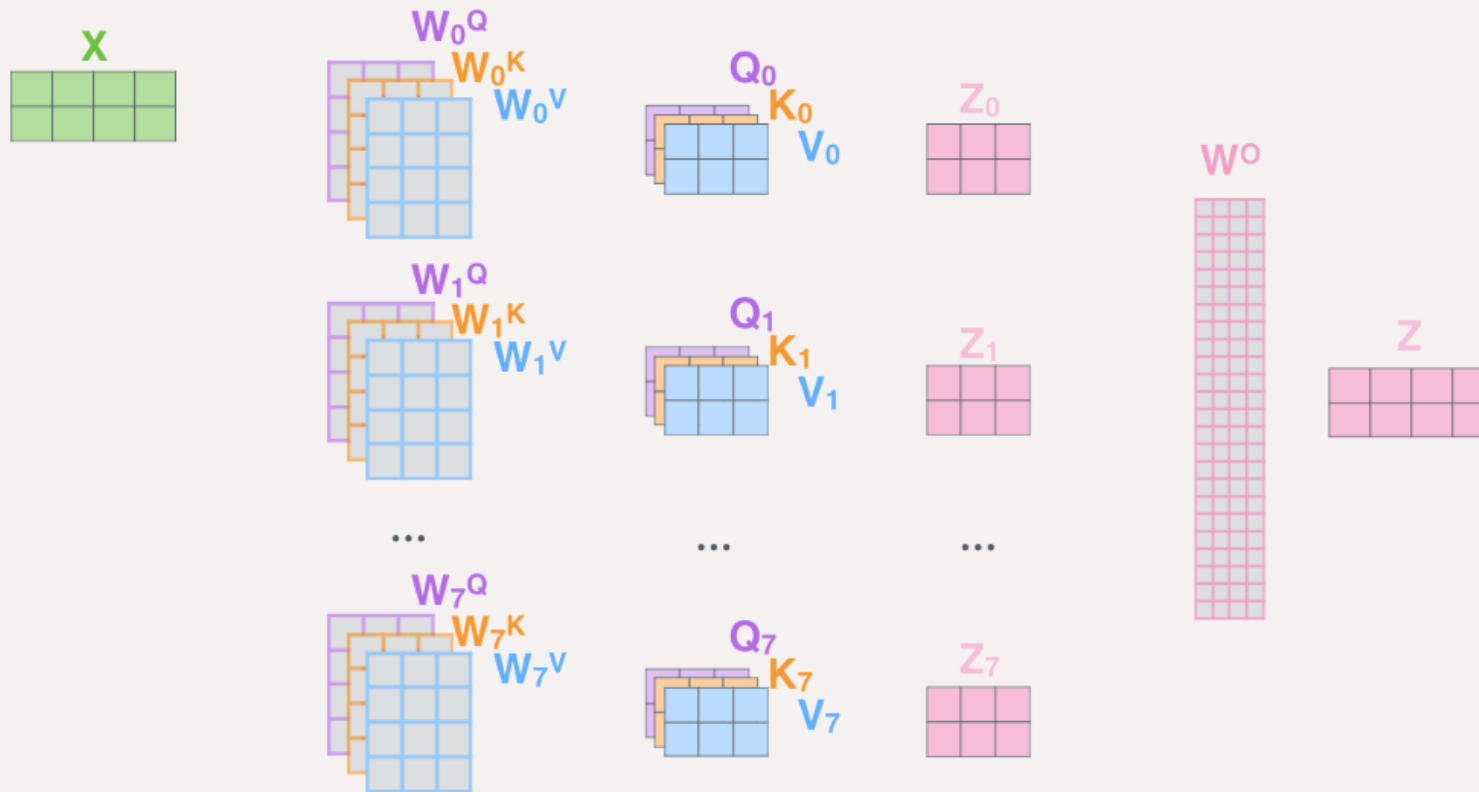
- Simply **multiple attention heads**, independent of each other
 - Just operate on the same input **X**
 - **H** attention heads
- Output of the multi-head attention layer is then a downprojection of the concatenation of the outputs of each head (i.e., each self-attention)

$$\text{mh-att}(\mathbf{X} \mid \boldsymbol{\theta}_{\text{MHA}}) = (\mathbf{Z}_1 \oplus \mathbf{Z}_2 \oplus \dots \oplus \mathbf{Z}_{H-1} \oplus \mathbf{Z}_H) \mathbf{W}^o$$

- Concatenation $(\mathbf{Z}_1 \oplus \mathbf{Z}_2 \oplus \dots \oplus \mathbf{Z}_{H-1} \oplus \mathbf{Z}_H)$ has dimensions $N \times (v \cdot H)$
 - **Desiderata** (because of multiple identical layers): output of multi-head attention has the same dimensionality as input: $\text{mh-att}(\mathbf{X}) \in \mathbb{R}^{N \times d}$
 - This mandates that the **parameter matrix** \mathbf{W}^o has dimensions $(v \cdot H) \times d$
- All parameters of one MHA (sub)layer: $\boldsymbol{\theta}_{\text{MHA}} = \{\mathbf{W}_1^q, \mathbf{W}_1^k, \mathbf{W}_1^v, \dots, \mathbf{W}_H^q, \mathbf{W}_H^k, \mathbf{W}_H^v, \mathbf{W}^o\}$

Multi-Head Attention - Visual Summary

- In the example below: **8** attention heads (indexed 0 to 7)

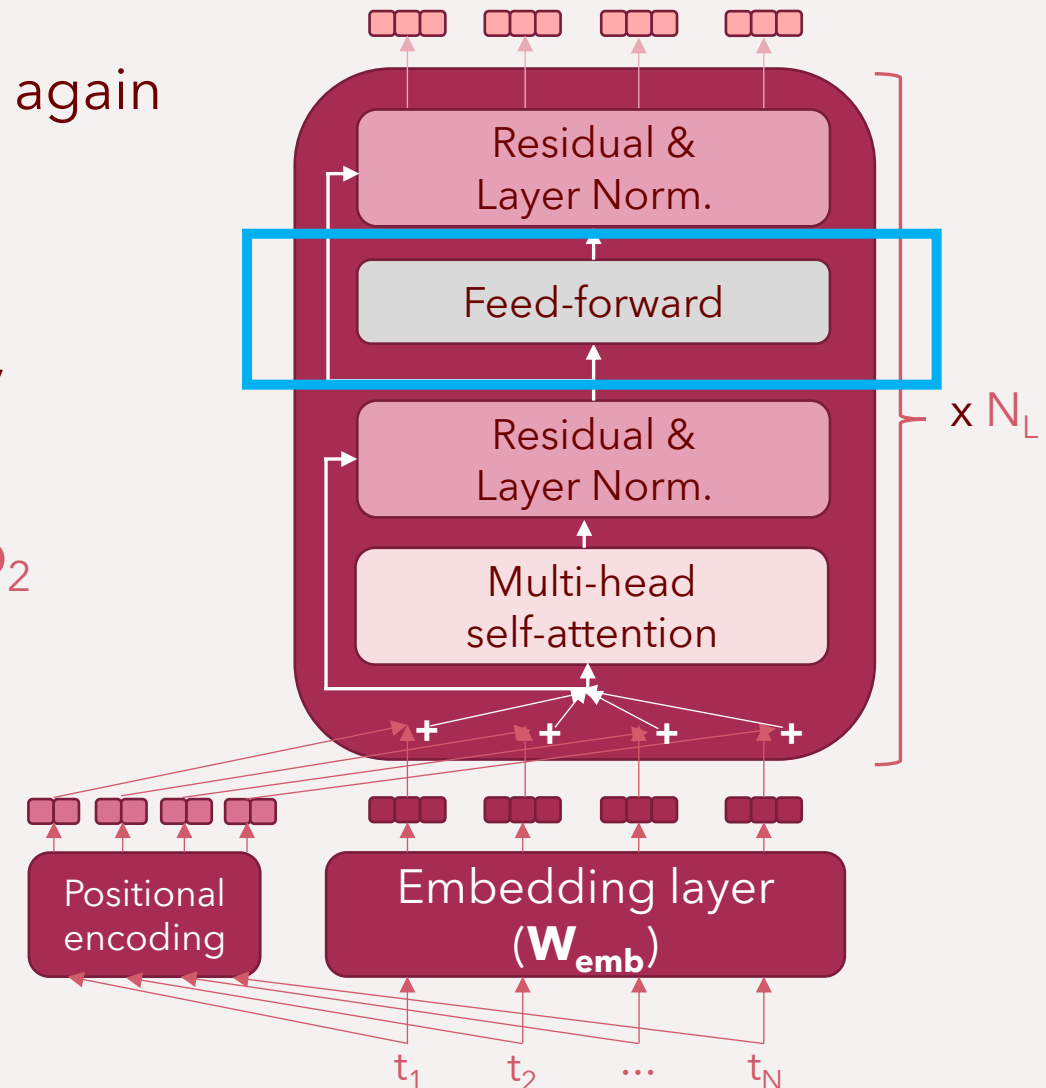


Feed-Forward Layer

- Output of the multi-head attention layer is again a matrix of dimensions $N \times d$
 - I.e., one d -dim. vector for each token
- Each token vector \mathbf{x} is then independently transformed through the following FFN:

$$\text{FFN}(\mathbf{x} \mid \boldsymbol{\theta}_{\text{ffn}}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

- $\text{ReLU}(x) = \max(0, x)$
 - Common activation function
- Trainable parameters: $\boldsymbol{\theta}_{\text{ffn}} = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2\}$
 - $\mathbf{W}_1 \in \mathbb{R}^{d \times f}, \mathbf{b}_1 \in \mathbb{R}^f$
 - $\mathbf{W}_2 \in \mathbb{R}^{f \times d}, \mathbf{b}_2 \in \mathbb{R}^d$
 - Vaswani et al. set $f = 4d$

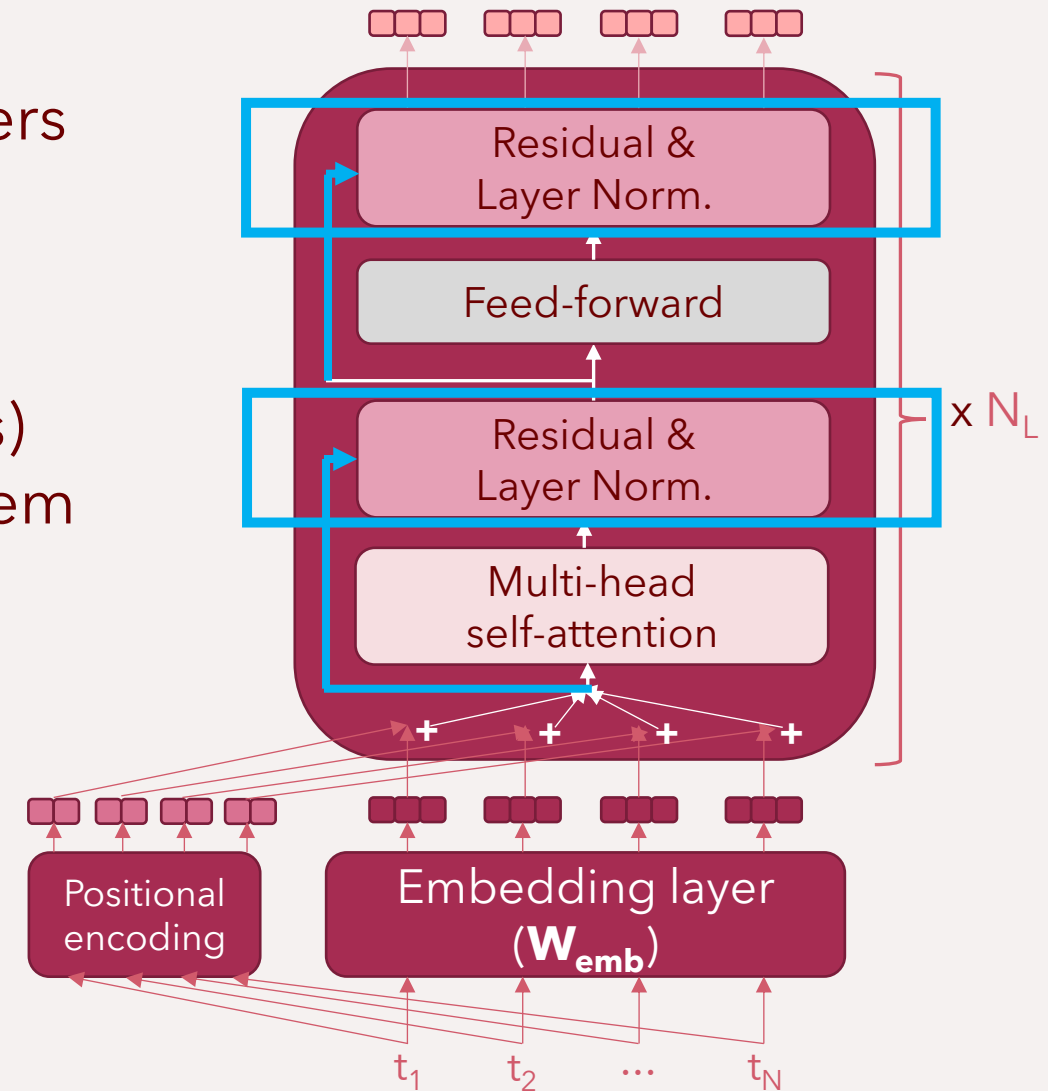


Residuals

- Transformer layer has two main sublayers
 - Multi-head attention layer
 - Feed-forward layer
- Both those layers (i.e., param. functions) have a **residual connection** around them

Residual (around a layer) – layer input added to its output

$$res(layer, \mathbf{X}) = \mathbf{X} + layer(\mathbf{X} \mid \boldsymbol{\theta}_{layer})$$

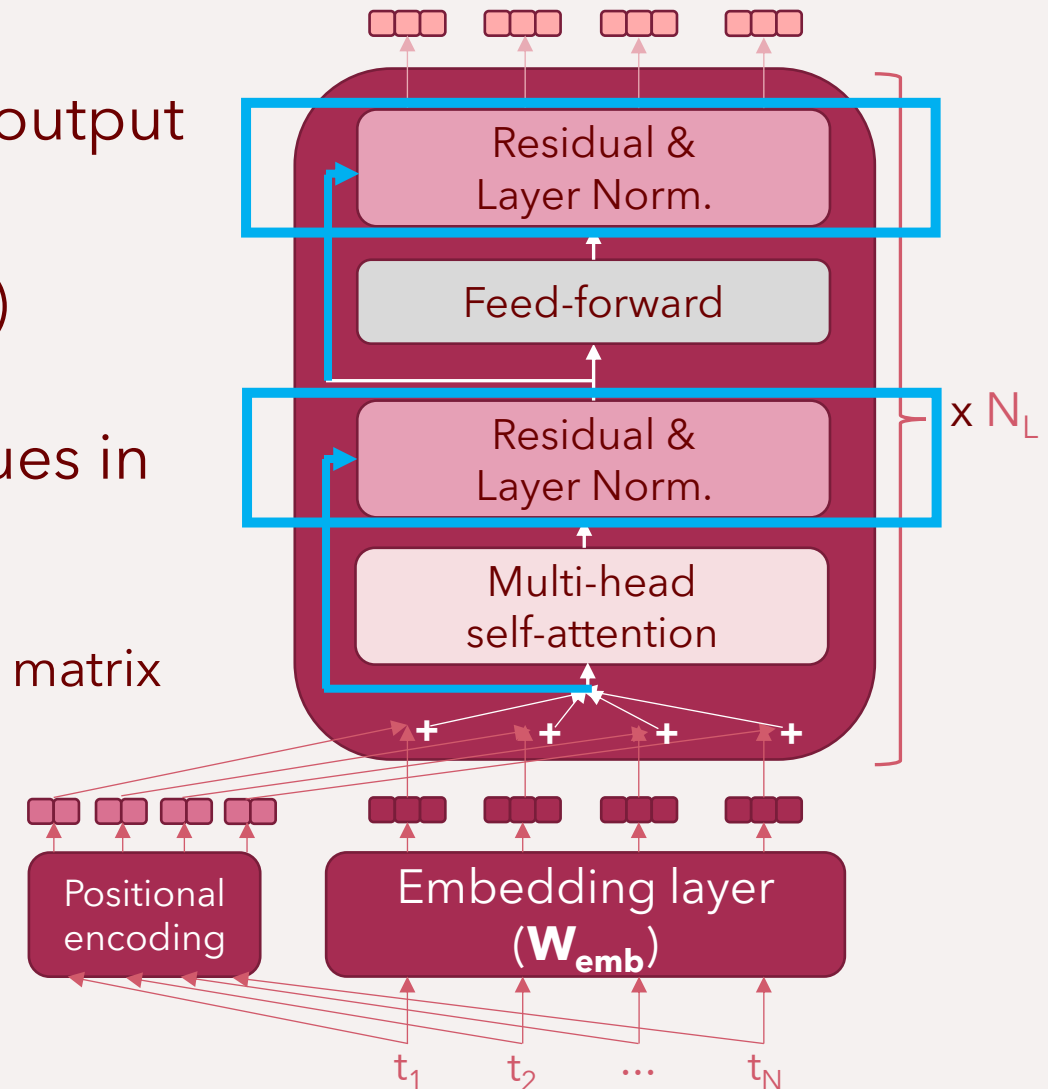


Layer Normalization

- After the residual summation, the final output is subdued to **layer normalization**

$$\text{layer_norm}(\mathbf{X}' = \mathbf{X} + \text{layer}(\mathbf{X} \mid \boldsymbol{\theta}_{\text{layer}}))$$

- Layer normalization normalizes the values in each of the row-vectors \mathbf{x} in the input
 - Input is a matrix of dimensions $N \times d$
 - Let $\mathbf{x} \in \mathbb{R}^d$ be any of the row-vectors of that matrix
- We z-normalize values in $\mathbf{x} = [x_1, x_2, \dots, x_d]$
 - $x_i \rightarrow \frac{x_i - \mu}{\sigma}$
 - μ as mean and σ as st. deviation on \mathbf{x}



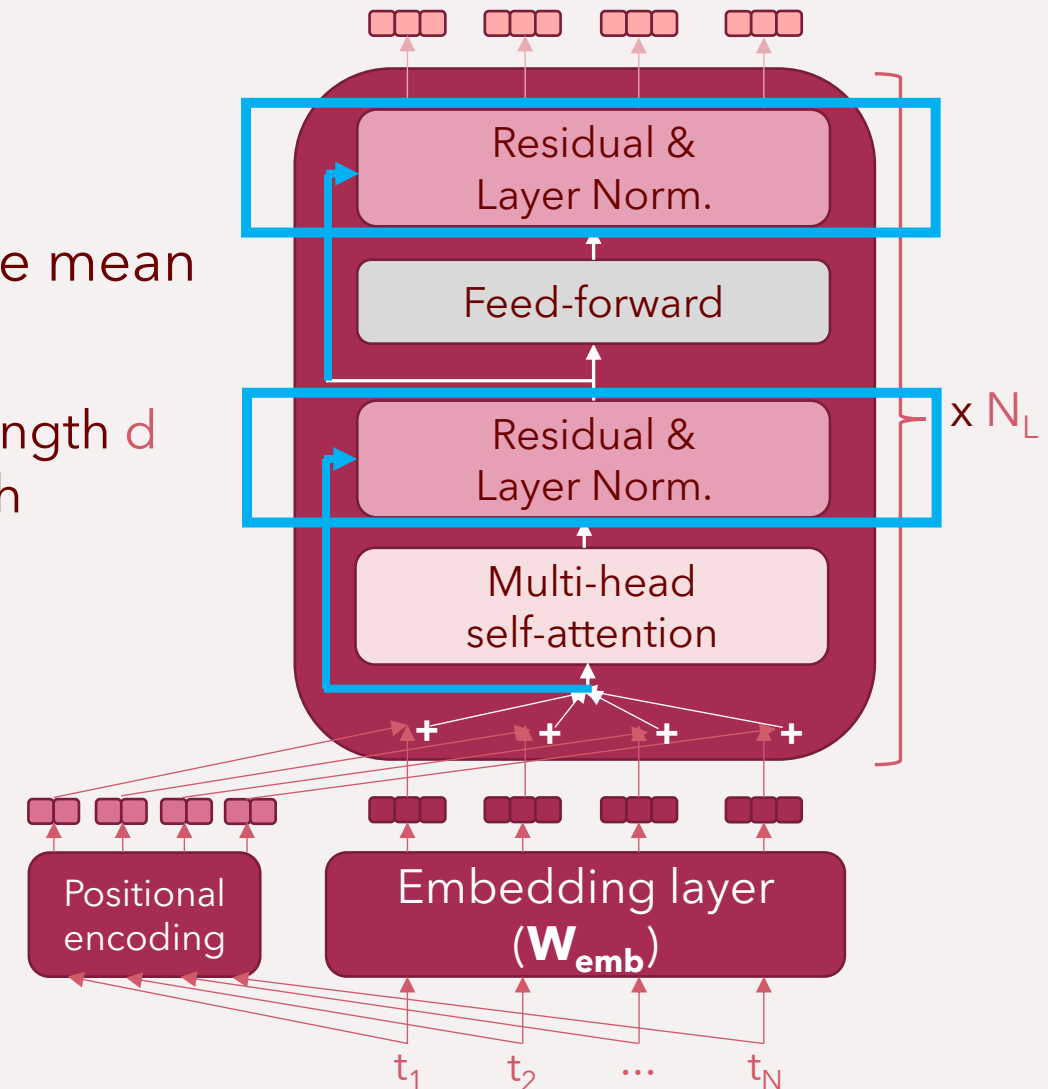
Layer Normalization

- We z-normalize (or standardize) values in \mathbf{x}
 - $x_i \rightarrow \frac{x_i - \mu}{\sigma}$
 - This centers the values in \mathbf{x} around the mean of 0, with the st. deviation of 1

- \mathbf{X}'' = matrix with N z-normalized vectors of length d
- The final layer normalized output is given with

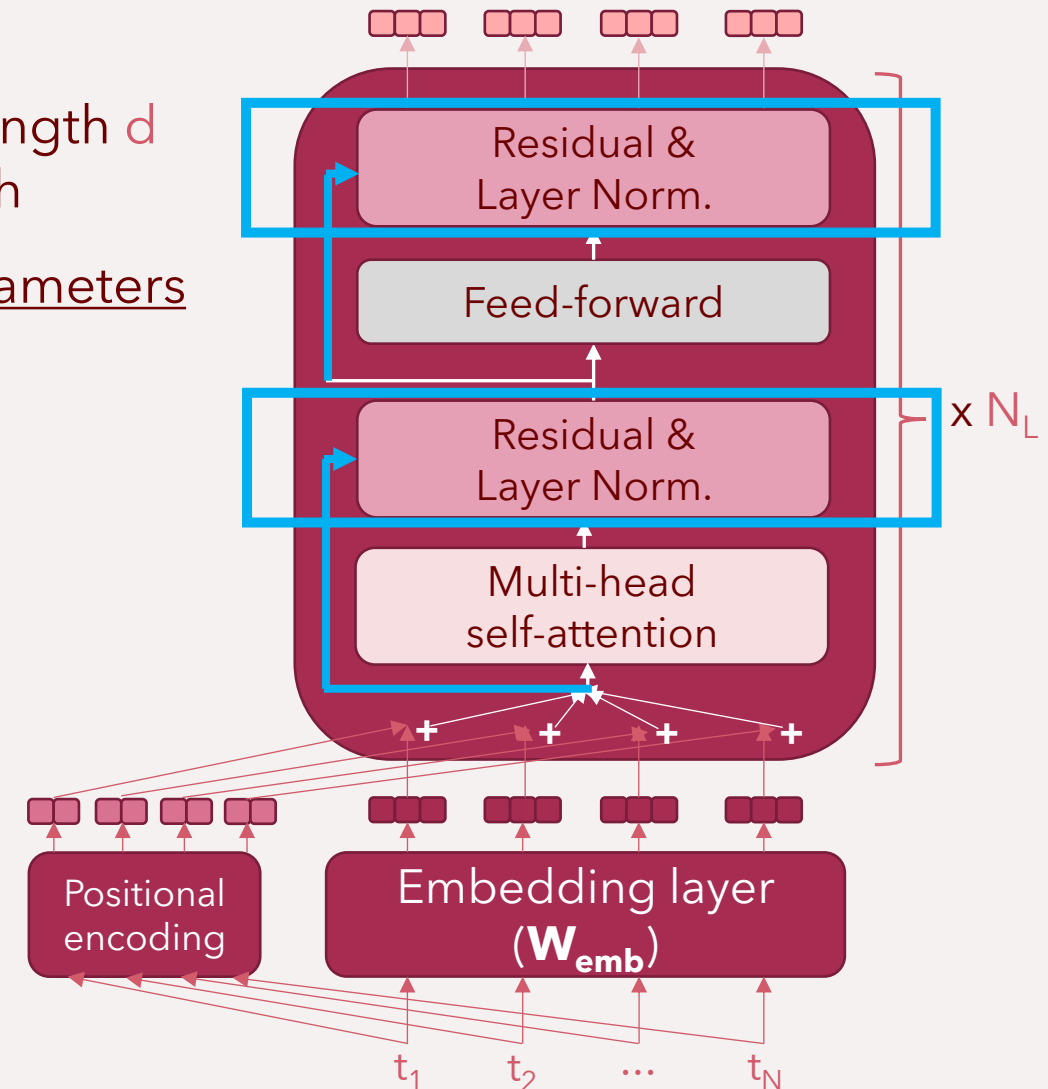
$$\boldsymbol{\gamma} * \mathbf{X}'' + \boldsymbol{\beta}$$

- Where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta} \in \mathbb{R}^d$ are trainable parameters of layer attention
 - $\boldsymbol{\gamma}$ element-wise multiplies each row of \mathbf{X}''
 - $\boldsymbol{\beta}$ is then added to each of the rows



Layer Normalization

- \mathbf{X}'' = matrix with N z-normalized vectors of length d
- The final layer normalized output is given with
- $\gamma * \mathbf{X}'' + \beta$, with γ and $\beta \in \mathbb{R}^d$ are trainable parameters of the layer normalization „layer“
 - γ element-wise multiplies each row of \mathbf{X}''
 - β is then added to each of the rows
- Layer-normalization stabilizes training
 - All instances across all mini-batches normalized the same way
 - Q: But why do we need γ and β ?
 - Normalization to $N(0, 1)$ may be too restrictive, some layers may need „more expressive“ distributions



Content

- Attention mechanism
- Transformer – dissected
 - Positional Embeddings
 - Multi-Head Self-Attention
- **Pretraining + fine-tuning**

Pretraining Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- **Pretrain - fine-tune paradigm:** the idea that we can
 - (1) pretrain the parameters of the encoder θ_{ENC} via some self-supervised training objective on large corpus and then
 - (2) further update (i.e., fine-tune) encoder's parameters θ_{ENC} while training for a concrete task - in this second step, we add task-specific classifier/regressor (head) on top of the encoder (body)
- **BERT:** pretraining–fine-tuning with a Transformer as the encoder
 - BERT not a first attempt at pretraining an encoder
 - But first where the encoder is a Transformer
 - ULMFit (Howard & Ruder, 2018), ELMo (Peters et al., 2018)



Pretraining Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).



- **Pretrain - fine-tune paradigm**
 - (1) Pretrain (self-supervised objective)
 - (2) Fine-tune (on annotated task data)
- Q: What is a suitable self-supervised objective for pretraining?

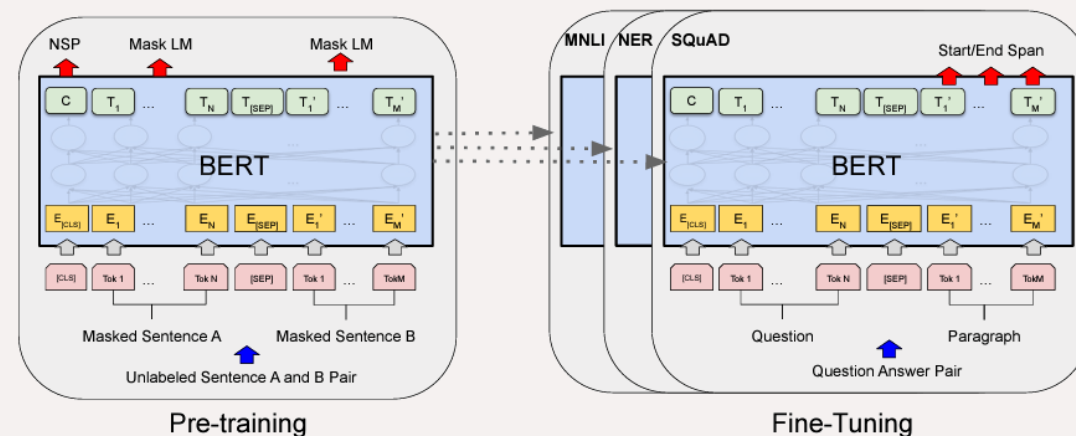


Image from Devlin et al.

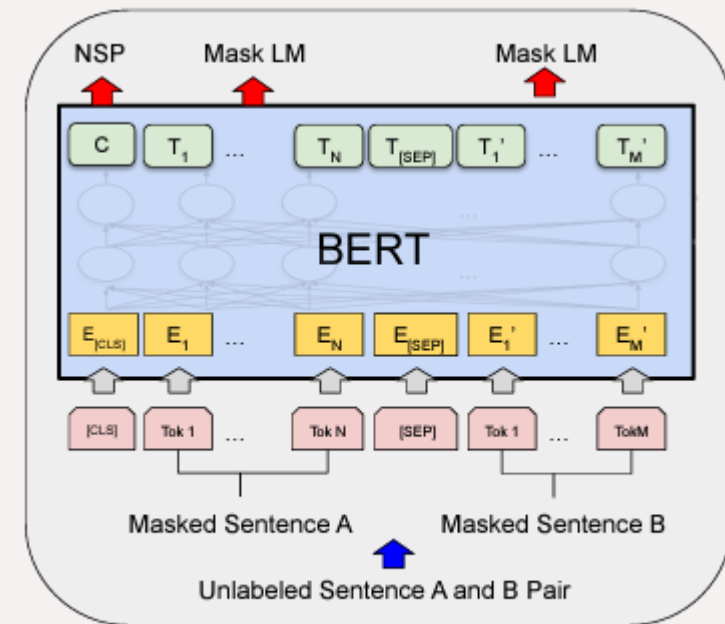
Pretraining Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- Q: What is a suitable self-supervised objective for pretraining?
- Devlin et al. use two pretraining objectives
 - Masked LM-ing (MLM)
 - Next Sentence Prediction (NSP)
- Special input: pairs of sentences with special (artificial) tokens

$[\text{CLS}] t_1^1 t_2^1 \dots t_N^1 [\text{SEP}] t_1^2 t_2^2 \dots t_M^2 [\text{SEP}]$



Pre-training

Image from Devlin et al.



Pretraining Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- Input: $[CLS] t_1^1 t_2^1 \dots t_N^1 [SEP] t_1^2 t_2^2 \dots t_M^2 [SEP]$
 - $[CLS]$ - sequence start token
 - $[SEP]$ - separator token
- The two sentences **may or may not** be adjacent in the training corpus
- Some percentage of (real) tokens **masked out** - replaced with the $[MASK]$ token

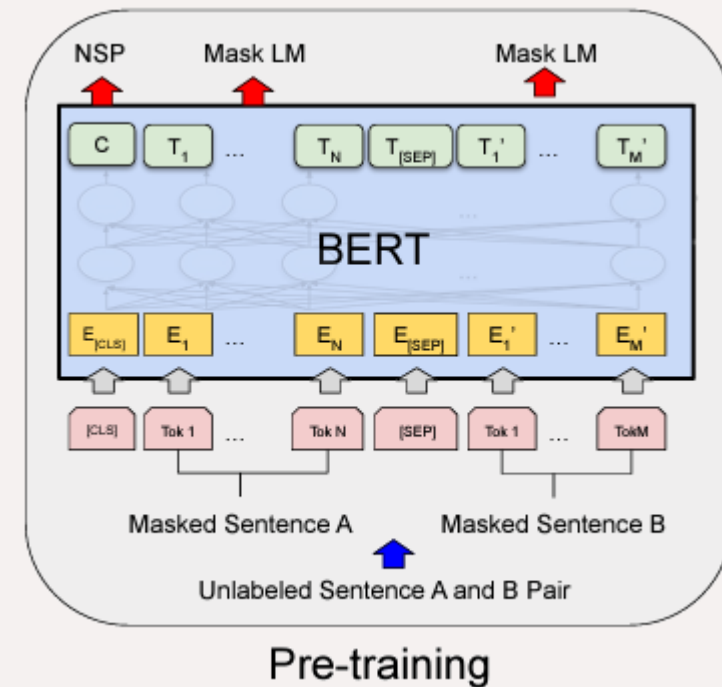


Image from Devlin et al.



Pretraining Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- BERT's pretraining objectives
 1. Masked LM-ing (MLM)
 2. Next Sentence Prediction (NSP)
- MLM: predict the original token for each masked position (in either sentence)
 - Standard LM-ing classification head + negative log-likelihood loss
- $\mathbf{x} \in \mathbb{R}^d$ = Transformer's output vector for some masked token [MASK]

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{x} \mathbf{W}_{lm}), \mathbf{W}_{lm} \in \mathbb{R}^{d \times |V|}$$
$$L(\mathbf{x}, \mathbf{y} | \boldsymbol{\theta}_{enc}, \mathbf{W}_{lm}) = -\sum_{i=1}^{|V|} y_i \ln(\hat{y}_i)$$

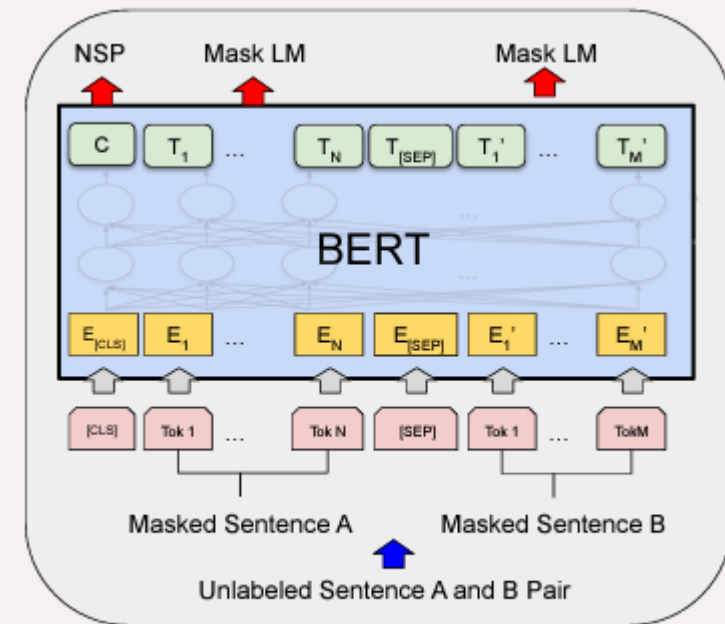


Image from Devlin et al.



Pretraining Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- BERT's pretraining objectives
 1. Masked LM-ing (MLM)
 2. Next Sentence Prediction (NSP)
- NSP: Predict if the two sentences were adjacent in the corpus or not
 - Standard binary classification head + binary cross-entropy loss
 - Q: why NSP? For text-pair tasks (QA, NLI)
- RoBERTa: same Transformer pretrained on more data and only with MLM – better performance

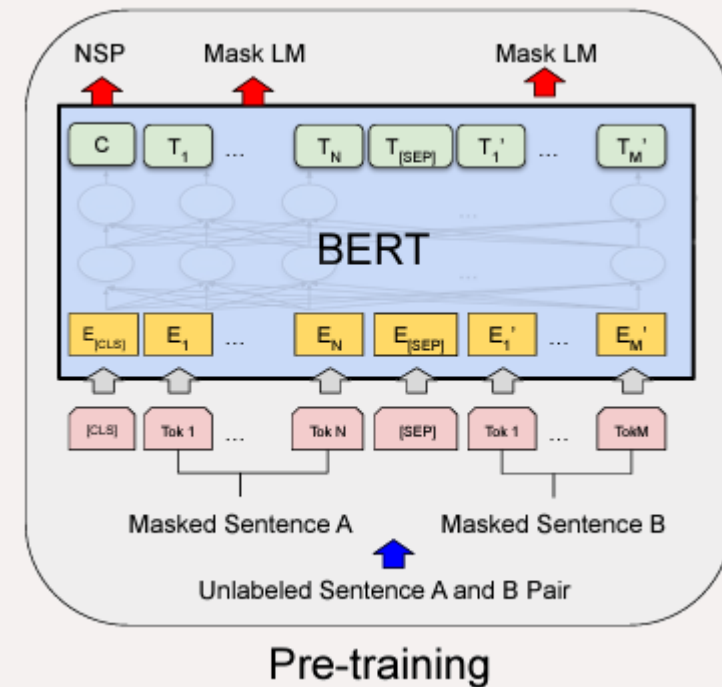


Image from Devlin et al.

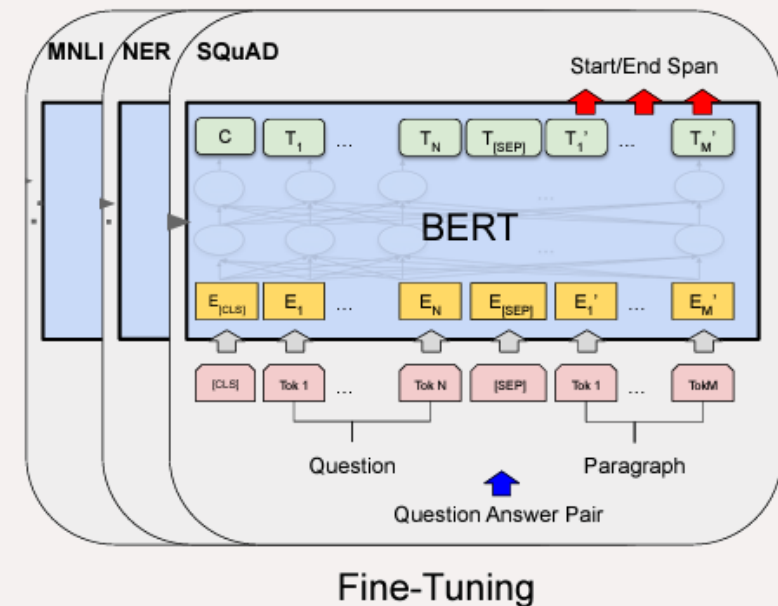


Fine-Tuning Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- **Pretrain - fine-tune paradigm**
 - (1) Pretrain (self-supervised objective)
 - (2) Fine-tune (on annotated task data)
- Q: How do we fine-tune BERT's Transformer for a concrete task?



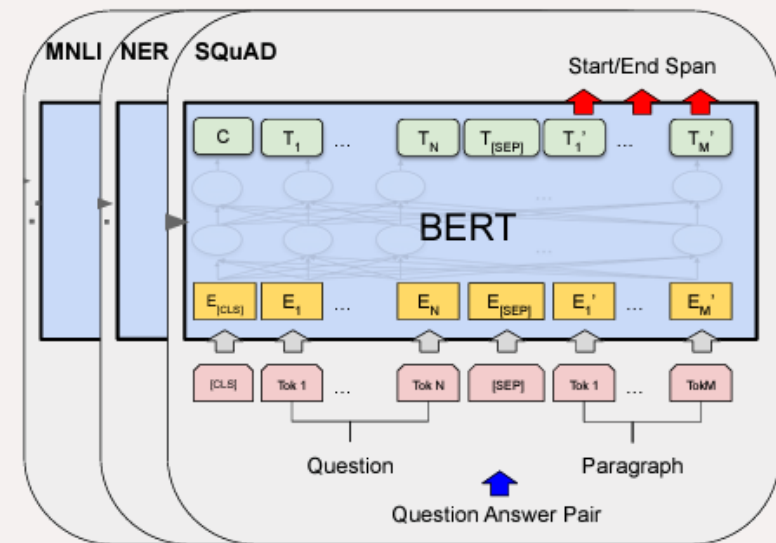
Fine-Tuning Transformers



Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#). In Proceedings of NAACL-HLT (pp. 4171-4186).

- **Sequence classification** (or regression) denotes tasks in which a label (class or score) is to be assigned to the whole input
 - $\mathbf{x}_{\text{CLS}} \in \mathbb{R}^d$ the representation of the sequence start token [CLS] output of the last Transformer layer
 - \mathbf{x}_{CLS} represents the encoding of the whole sequence, and goes into the classifier

$$\hat{\mathbf{y}} = \text{classifier}(\mathbf{x}_{\text{CLS}} | \boldsymbol{\theta}_{\text{cl}})$$



Fine-Tuning



Fine-Tuning Transformers

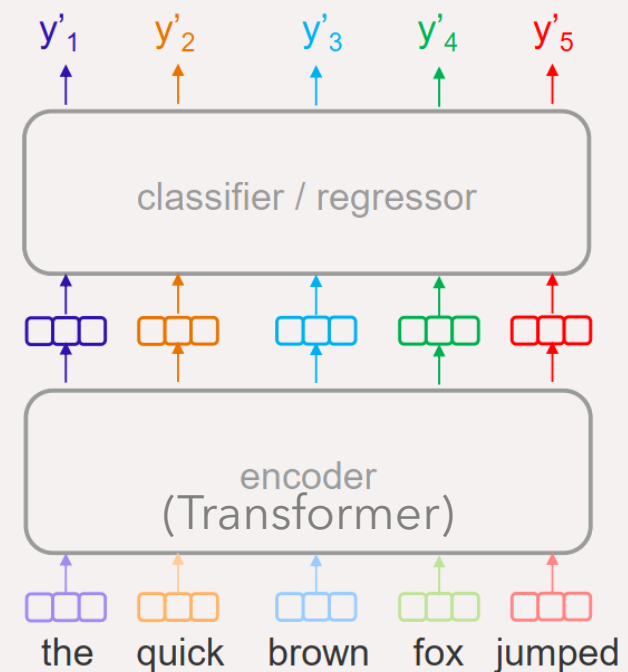


Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#). In Proceedings of NAACL-HLT (pp. 4171-4186).

- **Token classification** (or regression) denotes tasks in which a label (class or score) is to be assigned to the whole input
 - $\mathbf{x} \in \mathbb{R}^d$ the representation of a token (to be classified), output of the Transformer layer
 - \mathbf{x} is the **contextualized embedding** of the token, and goes into the classifier

$$\hat{\mathbf{y}} = \text{classifier}(\mathbf{x} | \boldsymbol{\theta}_{\text{cl}})$$

- Q: what if we're classifying word-level tokens (but have a subword tokenizer)?





The End

Image: Alexander Mikhailchik