Lecture 10: Attention Mechanisms and Transformers

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

Previous lecture: sequence-to-sequence models using RNNs and attention.

Today we look at self-attention and transformers:

- Convolutional sequence-to-sequence models
- Self-attention
- Transformer networks
- Pre-trained models and transfer learning (next class)

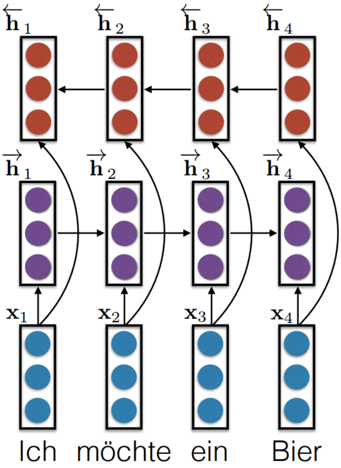
Pointers for Today's Class

- Lena Voita's seq2seq with attention: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- Marcos Treviso lecture on attention mechanisms: https://andre-martins.github.io/docs/dsl2020/attention-mechanisms.pdf
- John Hewitt's lecture on self-attention and transformers: http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture09-transformers.pdf
- Illustrated transformer: http://jalammar.github.io/illustrated-transformer/
- Annotated transformer: https://nlp.seas.harvard.edu/2018/04/03/attention.html

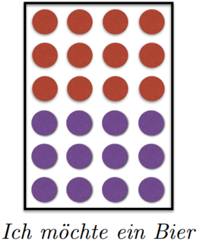
Outline

Recap: RNN with Attention (Encoder)

$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$

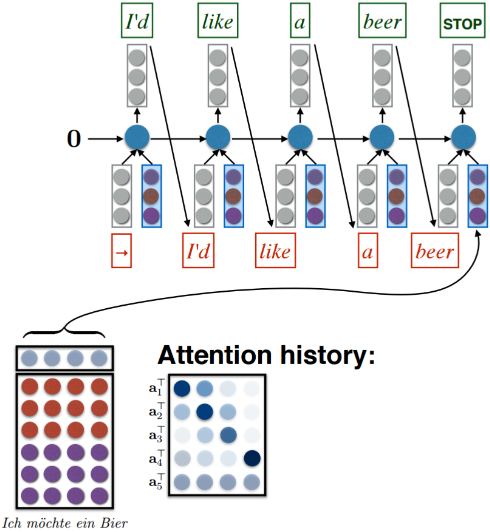


$\mathbf{F} \in \mathbb{R}^{2n \times |\boldsymbol{f}|}$



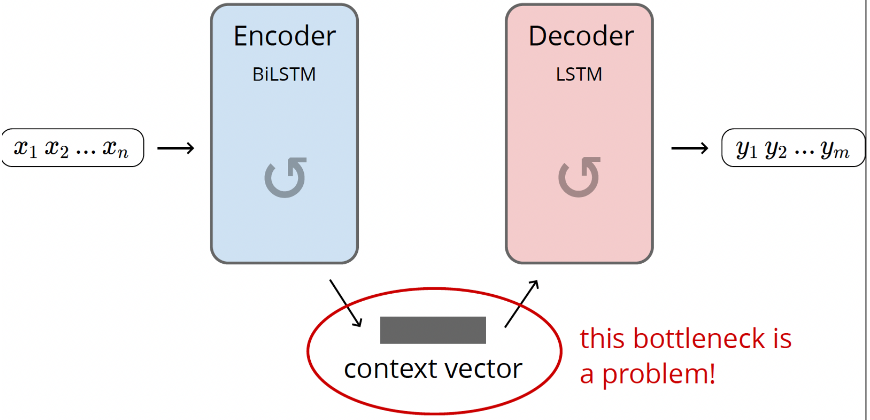
(Slide credit: Chris Dyer)

Recap: RNN with Attention (Decoder)

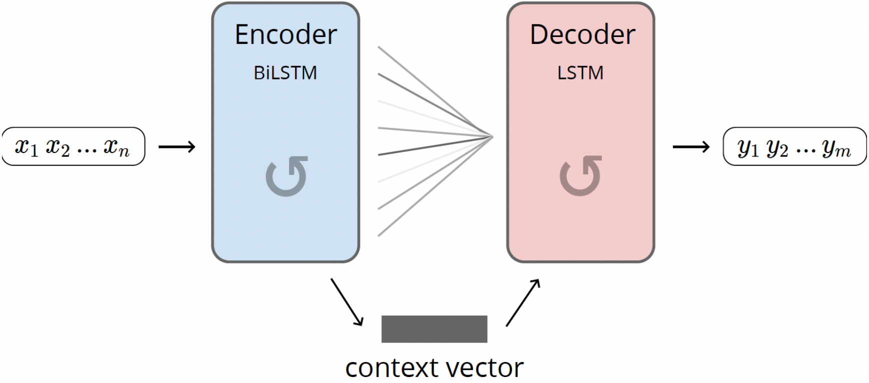


(Slide credit: Chris Dyer)

RNN-Based Encoder-Decoder



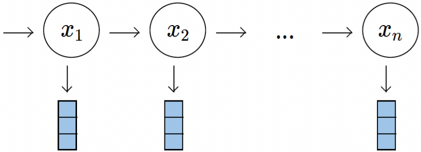
RNN-Based Encoder-Decoder



RNN-Based Encoder-Decoder

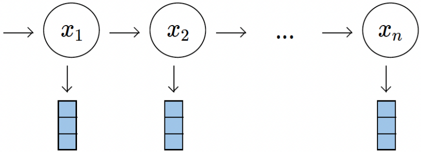
Drawbacks of RNNs

- Sequential mechanism prohibits parallelization

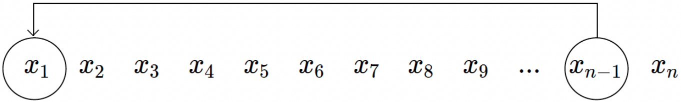


Drawbacks of RNNs

- Sequential mechanism prohibits parallelization

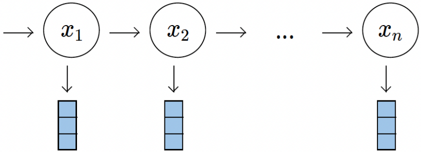


- Long-range dependencies are tricky, despite gating

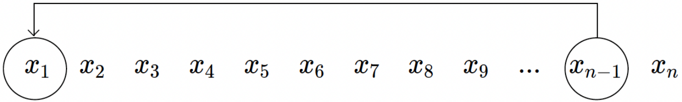


Drawbacks of RNNs

- Sequential mechanism prohibits parallelization

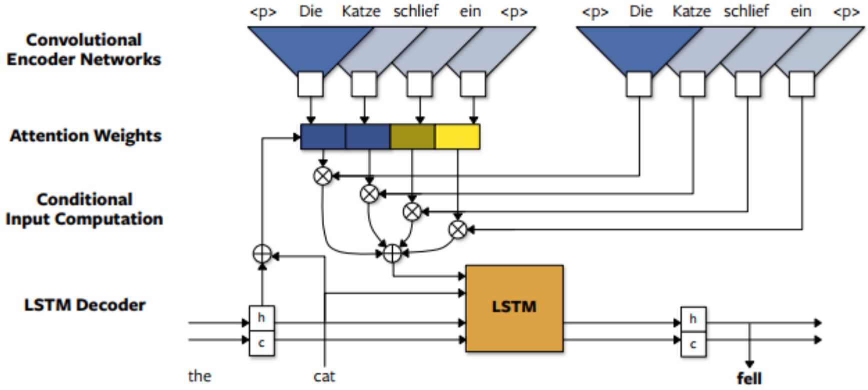


- Long-range dependencies are tricky, despite gating



- Possible solution: replace RNN encoder by hierarchical 1-D CNN

Convolutional Encoder

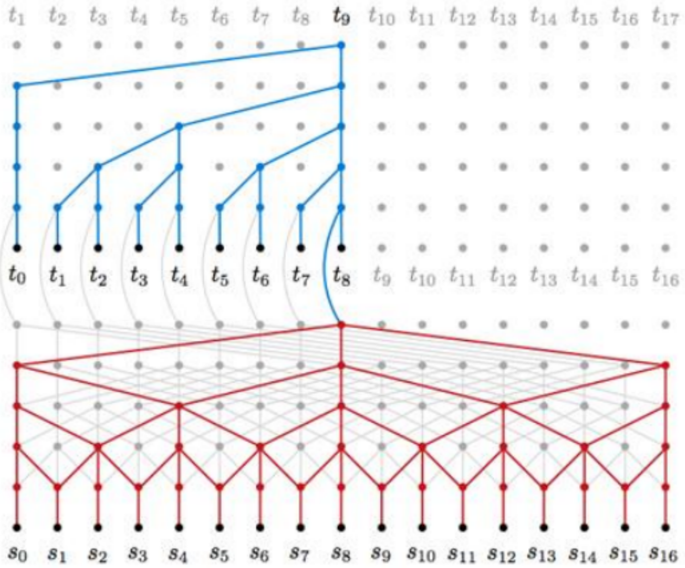


(Gehring et al., 2017)

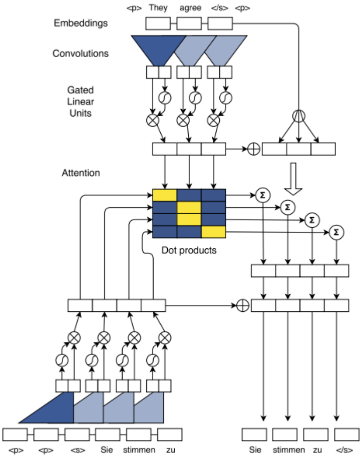
Fully Convolutional

- Can use CNN decoder too!
- Convolutions will be over output prefixes only
- Encoder is parallelizable, but decoder still requires sequential computation (the model is still auto-regressive)

Convolutional Sequence-to-Sequence



Convolutional Sequence-to-Sequence



(Gehring et al., 2017)

Next: Self-Attention

- Both RNN and CNN decoders require an attention mechanism
- Attention allows focusing on an arbitrary position in the source sentence, shortcutting the computation graph
- But if attention gives us access to any state...
  ...maybe we don't need the RNN?

Outline

1. Convolutional Encoder-Decoder
2. Self-Attention and Transformer Networks
3. Conclusions

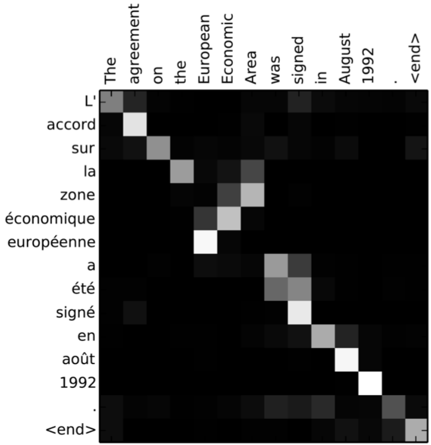
Why Attention?

We want NNs that automatically weigh input relevance

Main advantages:

- performance gain
- none or few parameters
- fast (easy to parallelize)
- tool for "interpreting" predictions

Example: Machine Translation



Dzmitry Bahdanau, KyungHuyn Cho, and Yoshua Bengio. ***Neural Machine Translation by Jointly Learning to Translate and Align****. ICLR'15.*

Example: Caption Generation

Attention over images:

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

(Slide credit to Yoshua Bengio)

Example: Document Classification

Task: Hotel location

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! **for location and price , this ca n't be beaten** , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel cleanliness

you get what you pay for . **not the cleanest rooms but bed was clean and so was bathroom** . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

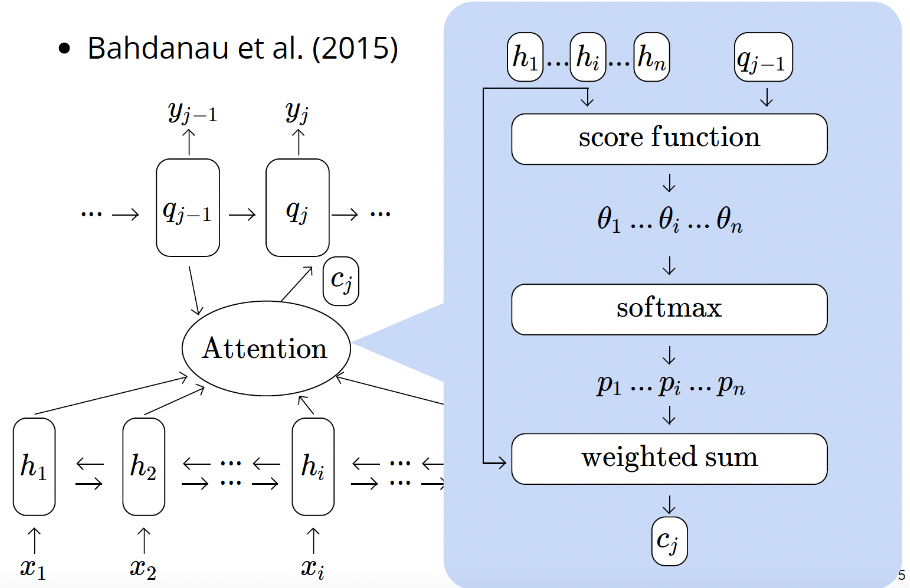
Task: Hotel service

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . **service was excellent ,** let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

(Bao et al., 2018)

Attention Mechanism

- Bahdanau et al. (2015)



Attention Mechanism: Recap

Recall how attention works:

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)
2. We have input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ (e.g. one per source word)
3. We compute affinity scores $s_1, \ldots, s_L$ by "comparing" $\boldsymbol{q}$ and $\boldsymbol{H}$
4. We convert these scores to probabilities:

$$\boldsymbol{p} = \textbf{softmax}(\boldsymbol{s})$$

5. We use this to output a representation as a weighted average:

$$\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{h}_i$$

Let's see these steps in detail!

Affinity Scores

Several ways of "comparing" a query $\boldsymbol{q}$ and an input ("key") vector $\boldsymbol{h}_i$:

- Additive attention (Bahdanau et al., 2015), what we covered in previous class:

$$s_i = \boldsymbol{u}^\top \tanh(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{B}\boldsymbol{q})$$

- Bilinear attention (Luong et al., 2015):

$$s_i = \boldsymbol{q}^\top \boldsymbol{U}\boldsymbol{h}_i$$

- Dot product attention (Luong et al., 2015) (particular case; queries and keys must have the same size):

$$s_i = \boldsymbol{q}^\top \boldsymbol{h}_i$$

The last two are easier to batch when we have multiple queries and multiple keys.

Keys and Values

The input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ appear in two places:

- They are used as keys to "compare" them with the query vector $\boldsymbol{q}$ to obtain the affinity scores
- They are used as values to form the weighted average $\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p}$

Keys and Values

The input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ appear in two places:

- They are used as keys to "compare" them with the query vector $\boldsymbol{q}$ to obtain the affinity scores
- They are used as values to form the weighted average $\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p}$

To be fully general, they don't need to be the same – we can have:

- A key matrix $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
- A value matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$

Attention Mechanism: More General Version

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)

2. We have key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
   and value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
   (e.g. one of each per source word)

3. We compute query-key affinity scores $s_1, \ldots, s_L$ "comparing" $\boldsymbol{q}$ and $\boldsymbol{K}$

4. We convert these scores to probabilities:

$$\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{s})$$

5. We output a weighted average of the values:

$$\boldsymbol{c} = \boldsymbol{V}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{v}_i \in \mathbb{R}^{d_V}$$

Self-Attention

- So far we talked about contextual attention – the decoder attends to encoder states (this is called "input context")
- The encoder and the decoder states were propagated sequentially with a RNN, or hierarchically with a CNN
- Alternative: self-attention – at each position, the encoder attends to the other positions in the encoder itself
- Same for the decoder.

Self-Attention Layer

Self-attention for a sequence of length L:

1. Query vectors $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_L]^\top \in \mathbb{R}^{L \times d_Q}$
2. Key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
3. value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
4. Compute query-key affinity scores "comparing" $\boldsymbol{Q}$ and $\boldsymbol{K}$, e.g.,

$$\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{L \times L} \qquad \text{(dot-product affinity)}$$

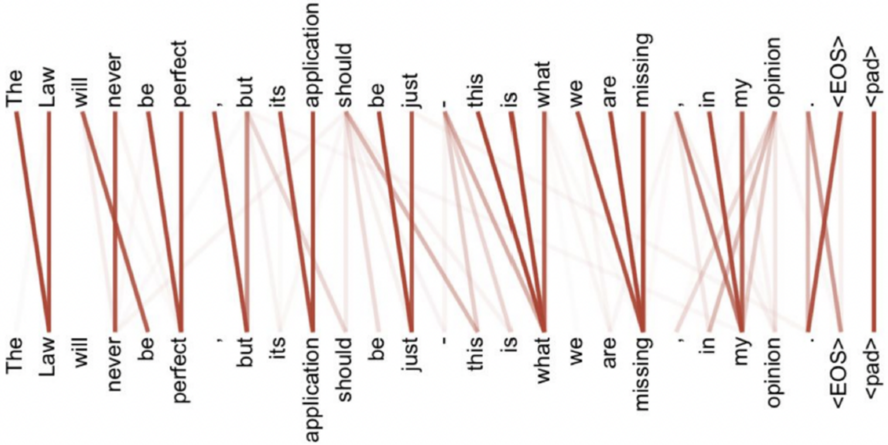
5. Convert these scores to probabilities (row-wise):

$$\boldsymbol{P} = \mathbf{softmax}(\boldsymbol{S}) \in \mathbb{R}^{L \times L}$$

6. Output the weighted average of the values:

$$\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{V} = \underbrace{\mathbf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)}_{\boldsymbol{P}} \boldsymbol{V} \in \mathbb{R}^{L \times d_V}.$$

Self-Attention



(Vaswani et al., 2017)

Transformer (Vaswani et al., 2017)

- **Key idea:** instead of RNN/CNNs, use self-attention in the encoder
- Each word state attends to all the other words
- Each self-attention is followed by a feed-forward transformation
- Do several layers of this
- Do the same for the decoder, attending only to already generated words.

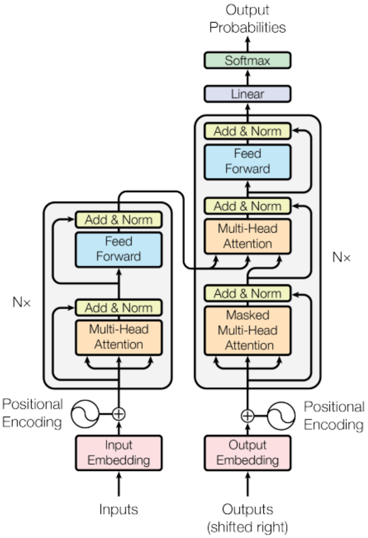
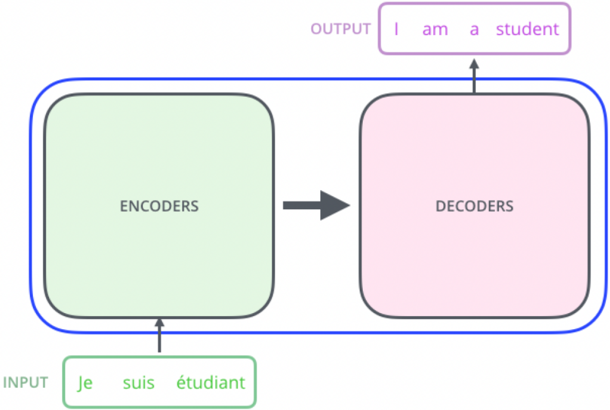


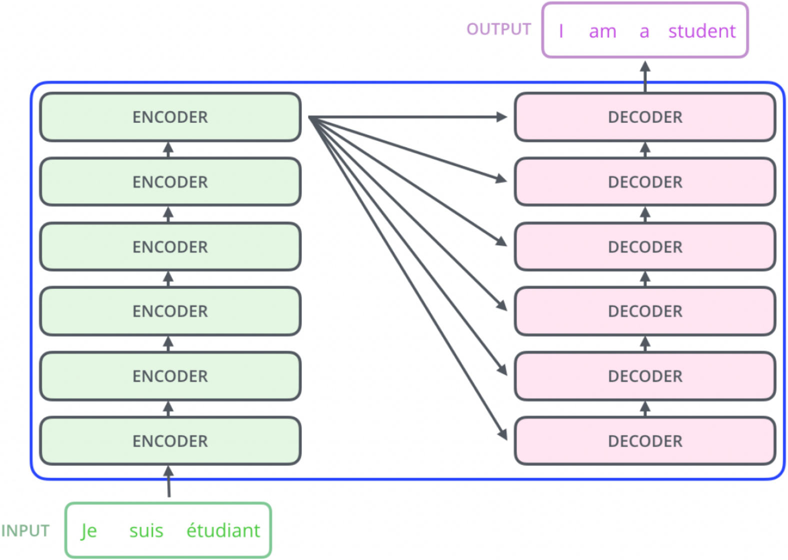
Figure 1: The Transformer - model architecture.

Transformer

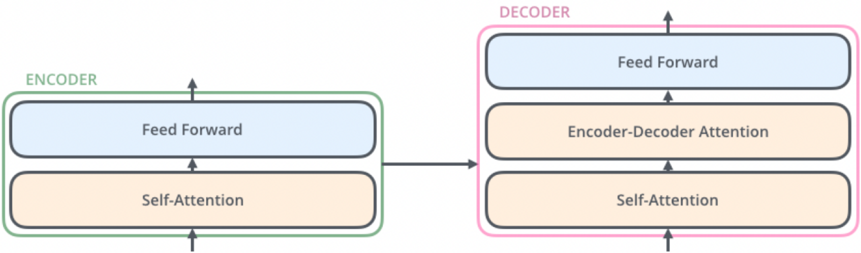


$$\boxed{x_1\, x_2 \ldots x_n} \xrightarrow{\text{encode}} \boxed{\mathbf{r}_1\, \mathbf{r}_2 \ldots \mathbf{r}_n} \xrightarrow{\text{decode}} \boxed{y_1\, y_2 \ldots y_m}$$

Transformer



Transformer Blocks



(Illustrated transformer: `http://jalammar.github.io/illustrated-transformer/`)

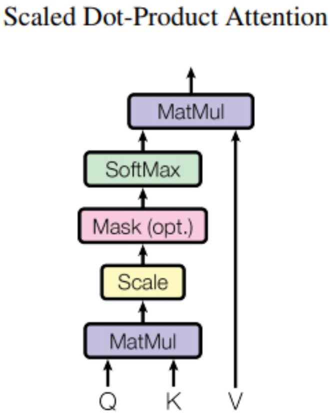
Transformer Basics

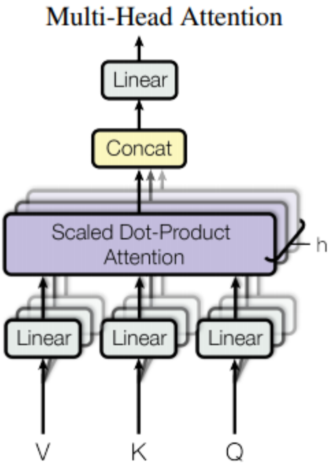
Let's define the basic building blocks of transformer networks first: new attention layers!

Two innovations:

- scaled dot-product attention
- multi-head attention

Scaled Dot-Product and Multi-Head Attention

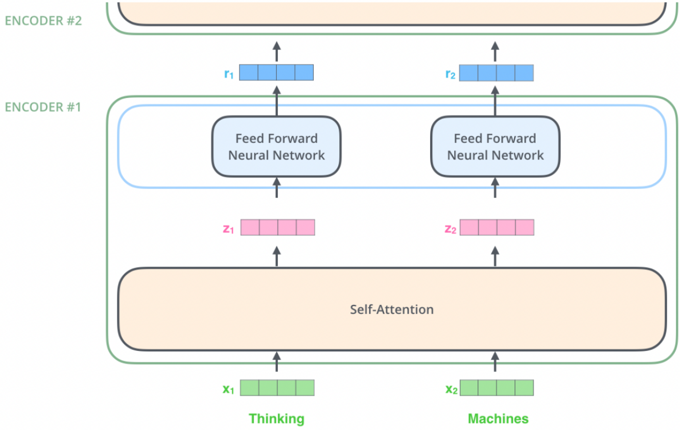




(Vaswani et al., 2017)

The Encoder

Example for a sentence with 2 words:

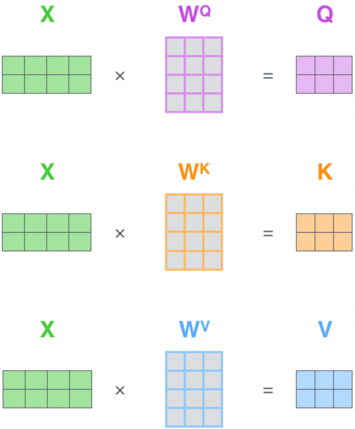


Transformer Self-Attention: Queries, Keys, Vectors

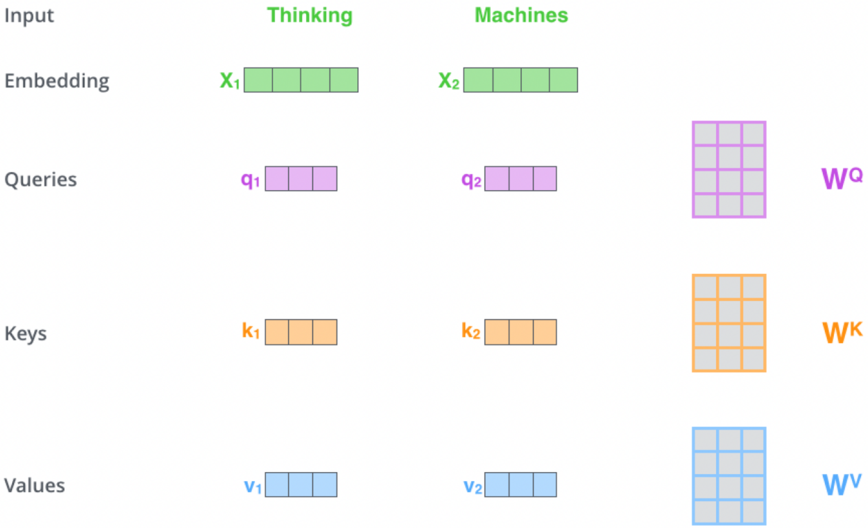
- Obtained by projecting the embedding matrix $\boldsymbol{X} \in \mathbb{R}^{L \times e}$ to a lower dimension:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{Q}$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{K}$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{V}.$$

- The projection matrices $\boldsymbol{W}^{Q}$, $\boldsymbol{W}^{K}$, $\boldsymbol{W}^{V}$ are model parameters.



Transformer Self-Attention: Queries, Keys, Vectors



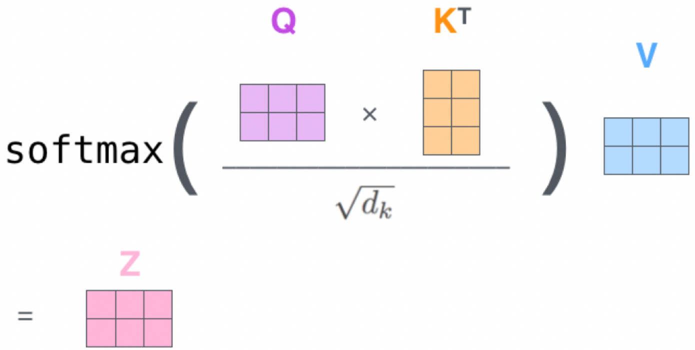
Scaled Dot-Product Attention

Problem: As d_K gets large, the variance of $\boldsymbol{q}^\top \boldsymbol{k}$ increases, the softmax gets very peaked, hence its gradient gets smaller.

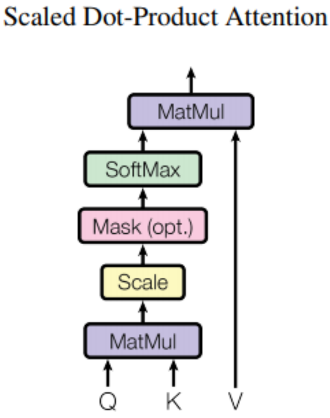
Solution: scale by length of query/key vectors:

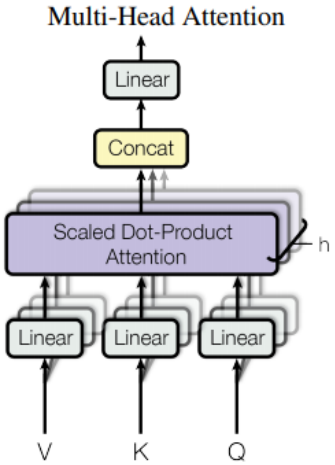
$$\boldsymbol{Z} = \mathbf{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_K}}\right)\boldsymbol{V}.$$

Scaled Dot-Product Attention



Scaled Dot-Product and Multi-Head Attention





(Vaswani et al., 2017)

Multi-Head Attention

Self-attention: each word forms a query vector and attends to the other words' key vectors

This is vaguely similar to a 1D convolution, but where the filter weights are "dynamic" is the window size spans the entire sentence!

Problem: only one channel for words to interact with one-another

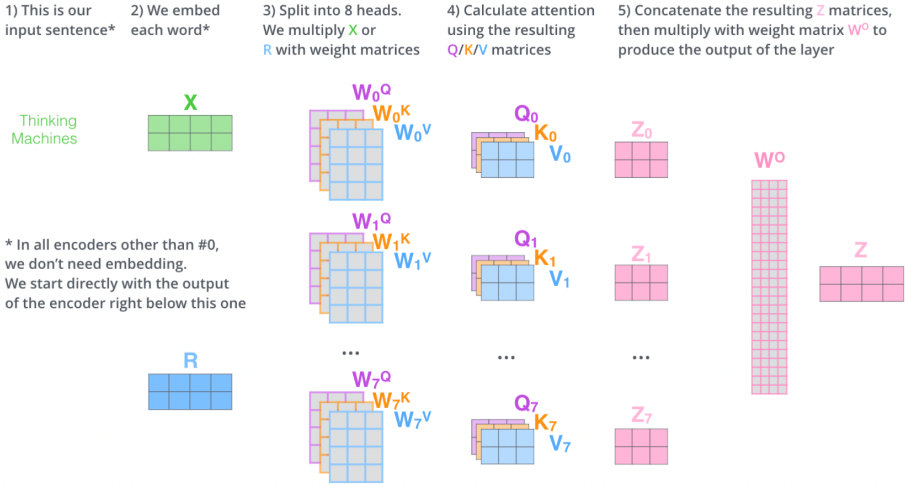
Solution: multi-head attention!

- define h attention heads, each with their own projection matrices (e.g. $h = 8$)
- apply attention in multiple channels, concatenate the outputs and pipe through linear layer:

$$\mathrm{MultiHead}(\boldsymbol{X}) = \mathrm{Concat}(\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_h)\boldsymbol{W}^O,$$

where $\boldsymbol{Z}_i = \mathrm{Attention}(\underbrace{\boldsymbol{X}\boldsymbol{W}_i^Q}_{\boldsymbol{Q}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^K}_{\boldsymbol{K}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^V}_{\boldsymbol{V}_i}).$

Multi-Head Attention



Other Tricks

- Self-attention blocks are repeated several times (e.g. 6 or 12)
- Residual connections on each attention block
- Layer normalization
- Positional encodings (to distinguish word positions)

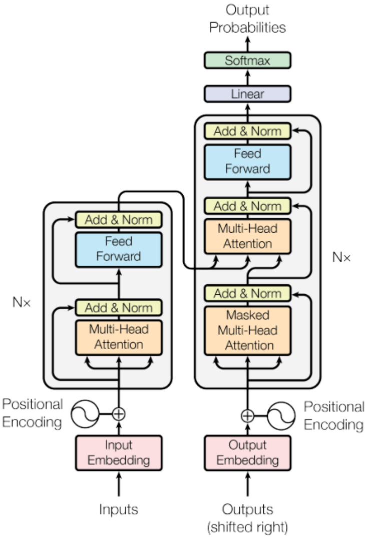
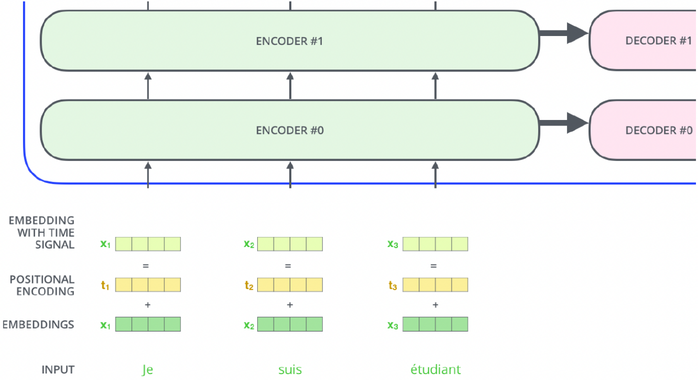


Figure 1: The Transformer - model architecture.

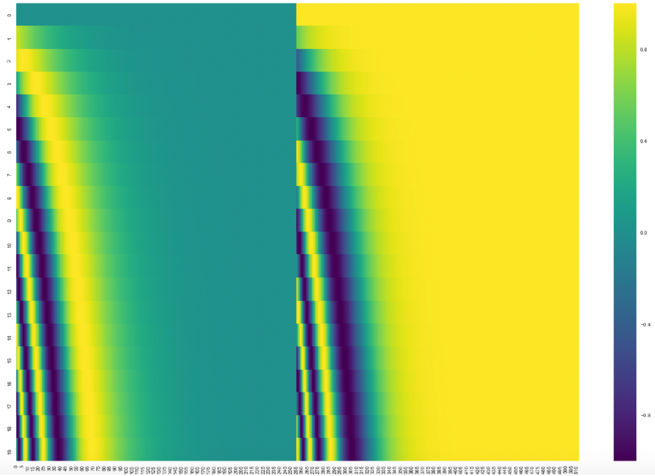
Positional Encodings

- As just described, the transformer is insensitive to word order!
  - queries attend to keys regardless of their position in the sequence
- To make it sensitive to order, we add positional encodings
- Two strategies: learn one embedding for each position (up to a maximum length) or use sinusoidal positional encodings (next)



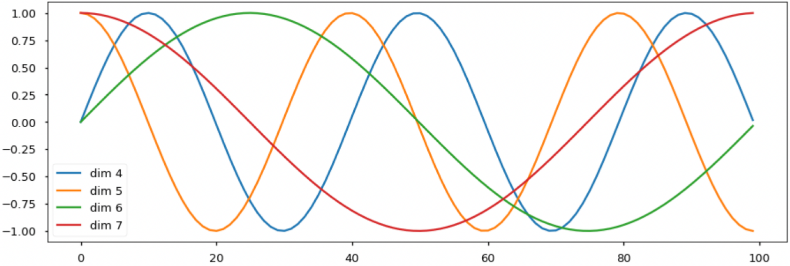
Sinusoidal Positional Encodings

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

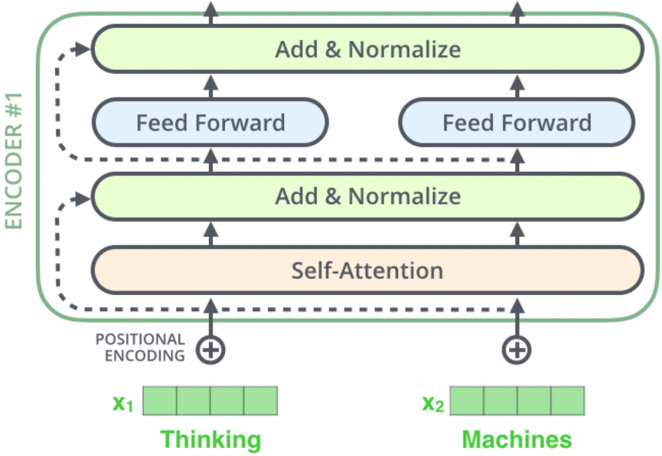


Sinusoidal Positional Encodings

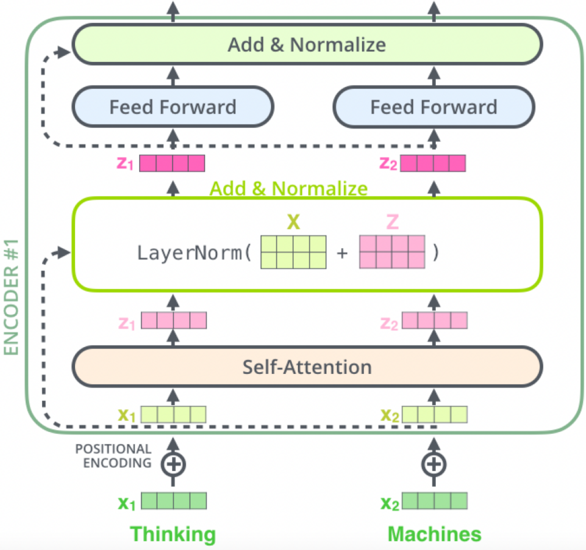
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



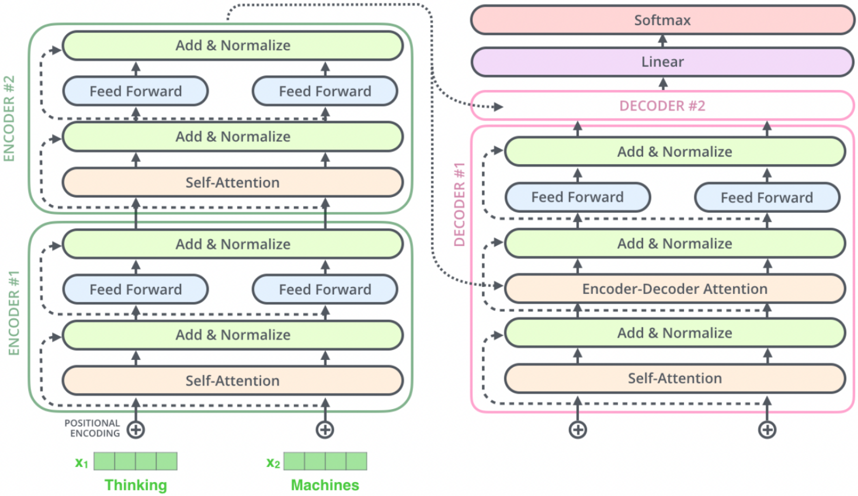
Residuals and Layer Normalization



Residuals and Layer Normalization



Residuals and Layer Normalization



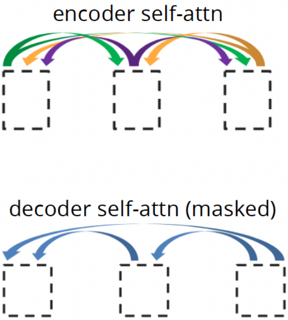
The Decoder

What about the self-attention blocks in the decoder?

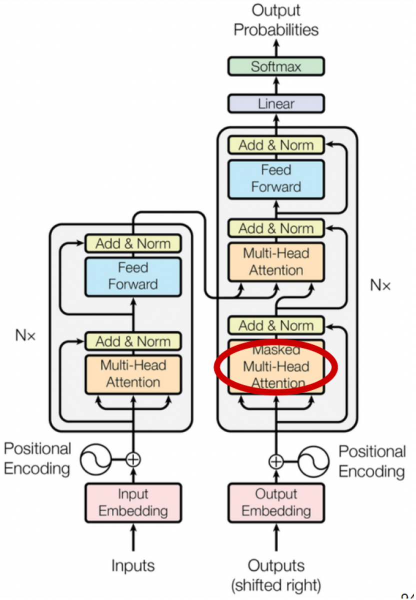
Everything is pretty much the same as in the encoder, with two twists:

- The decoder cannot see the future! Use "causal" masking
- The decoder should attend to itself (self-attention), but also to the encoder states (contextual attention).

The Decoder

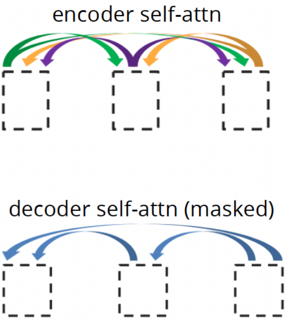


encoder self-attn
decoder self-attn (masked)

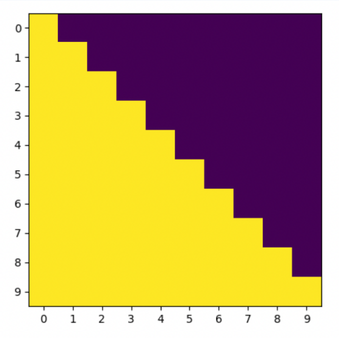


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

The Decoder



- Mask subsequent positions (before softmax)

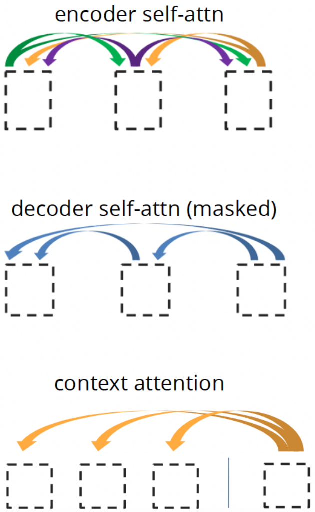


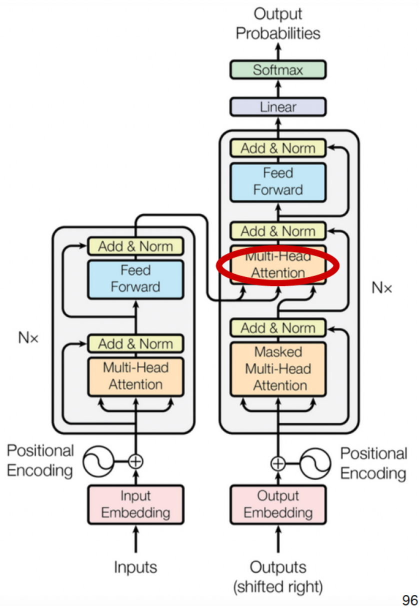
- In PyTorch

```
scores.masked_fill_(~mask, float('-inf'))
```

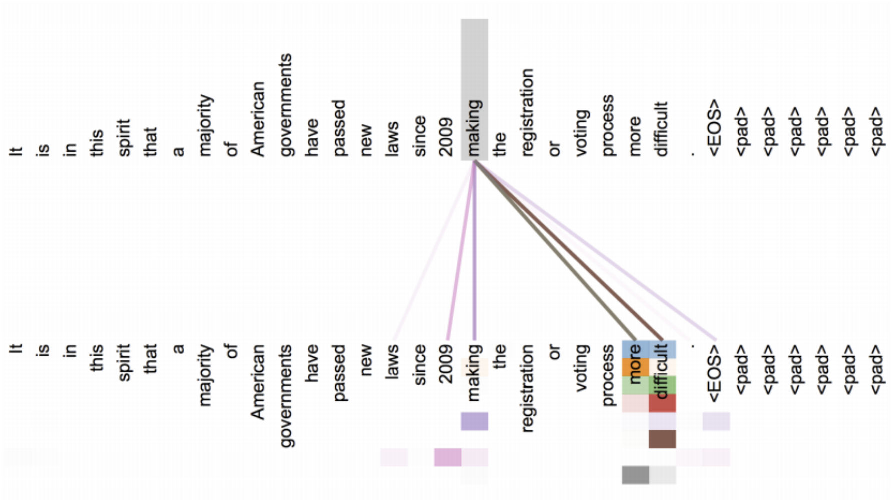
95

The Decoder

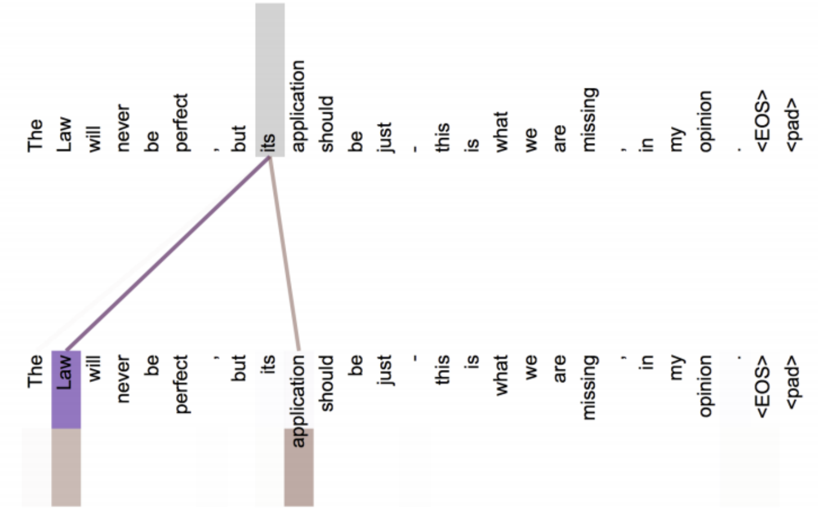

encoder self-attn
decoder self-attn (masked)
context attention


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Input
Embedding
Positional
Encoding
Output
Embedding
Inputs
Outputs
(shifted right)
96

Attention Visualization Layer 5



Implicit Anaphora Resolution



Computational Cost

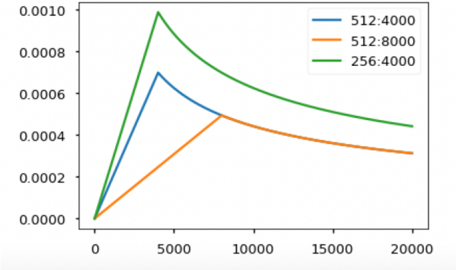
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n = seq. length d = hidden dim k = kernel size

- Faster to train (due to self-attention parallelization)
- More expensive to decode
- Scale quadratically with respect to sequence length (problematic for long sequences).

Other Tricks

- Label smoothing
- Dropout at every layer before residuals
- Beam search with length penalty
- Adam optimizer with learning-rate decay



Overall, transformers are harder to optimize than RNN seq2seq models

They don't work out of the box: hyperparameter tuning is very important.

Transformer Results

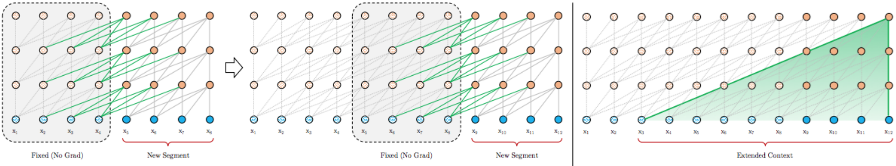
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

(Vaswani et al., 2017)'s "Attention Is All You Need"

TransformerXL

Big transformers can look at larger contexts.

TransformerXL: enables going beyond a fixed length without disrupting temporal coherence:



(a) Training phase.

(b) Evaluation phase.

(Dai et al., 2019)

Outline

Conclusions

- RNN-based seq2seq models require sequential computation and have difficulties with long range dependencies
- Attention mechanisms allow focusing on different parts of the input
- Encoders/decoders can be RNNs, CNNs, or self-attention layers
- Transformers are the current state of the art for many tasks in NLP and vision
- Other applications: speech recognition, image captioning, etc.
- Next lecture: pretrained models and transfer learning (BERT, GPT-2, GPT-3, etc.)

Thank you!

Questions?

References I

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*.

Bao, Y., Chang, S., Yu, M., and Barzilay, R. (2018). Deriving machine attention from human rationales. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1903–1913.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.