Attention Mechanisms in Neural Networks

UvA, Advanced Topics in Computational Semantics

Phillip Lippe

April 5, 2024

Today's learning goals

- What is "attention"?
- What different kinds of attention layers exist in NLP?
- Why and when to use attention
- Special focus: Self-attention and the Transformer architecture
  - Building blocks, design choices, training tips, and more!

What is attention?

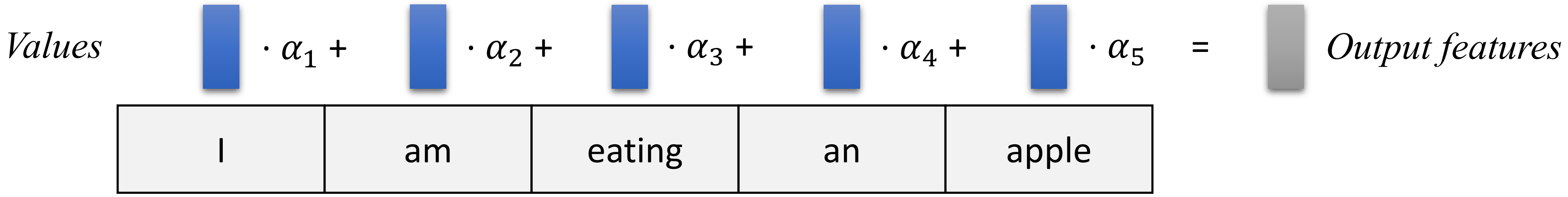
A weighted average of (sequence) elements with the weights depending on an input query.

Query: Feature vector, describing what we are looking for, what might be important

Key: One feature vector per element/word. What is this word “offering”? When might it be important?

Value: One feature vector per element/word. The actual features we want to average

Score function f_{attn} : maps query-key pair to importance weight. Commonly MLP or dot product



What is attention?

A weighted average of (sequence) elements with the weights depending on an input query.

Query: Feature vector, describing what we are looking for, what might be important

Key: One feature vector per element/word. What is this word “offering”? When might it be important?

Value: One feature vector per element/word. The actual features we want to average

Score function f_{attn} : maps query-key pair to importance weight. Commonly MLP or dot product

$$\alpha_i = \frac{\exp(f_{attn}(\mathrm{key_i}, \mathrm{query}))}{\sum_j \exp\left(f_{attn}(\mathrm{key_j}, \mathrm{query})\right)}$$

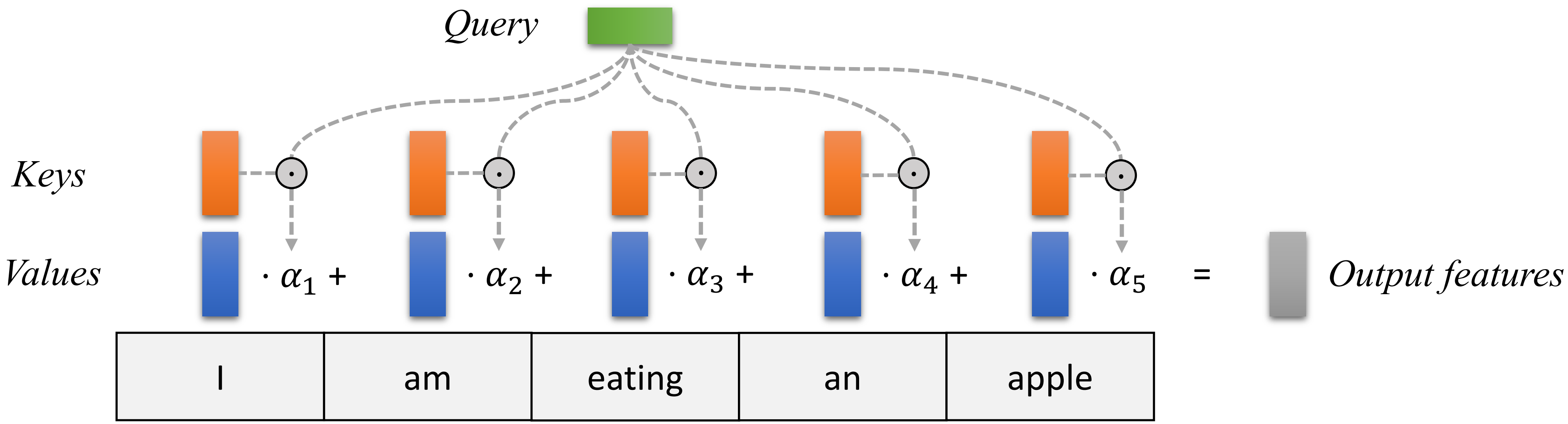
$$\mathrm{out} = \sum_i \alpha_i \cdot \mathrm{value}_i$$

What is attention?

A weighted average of (sequence) elements with the weights depending on an input query.

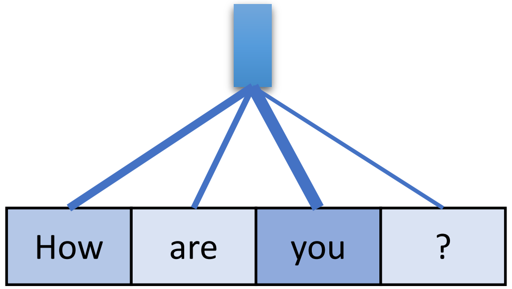
$$\alpha_i = \frac{\exp(f_{attn}(\mathrm{key_i}, \mathrm{query}))}{\sum_j \exp\left(f_{attn}(\mathrm{key_j}, \mathrm{query})\right)} \qquad \mathrm{out} = \sum_i \alpha_i \cdot \mathrm{value}_i$$

Example

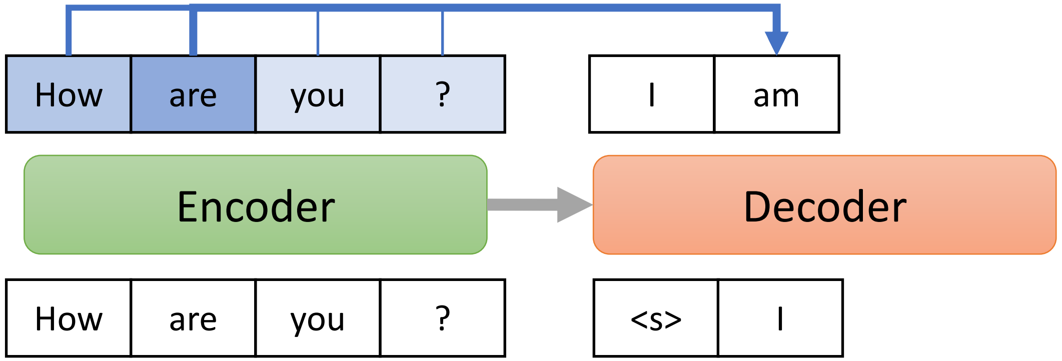


Attention mechanisms

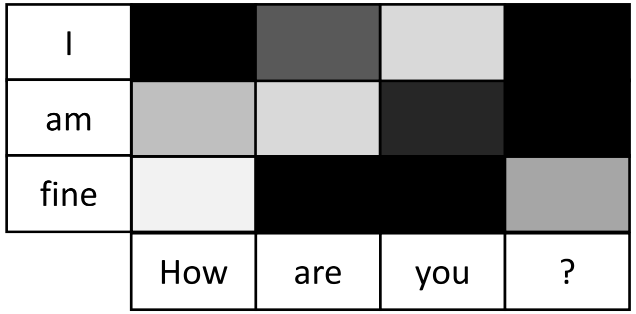
Aggregation



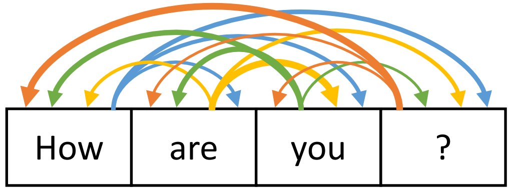
Encoder-Decoder



Cross-Attention



Self-Attention

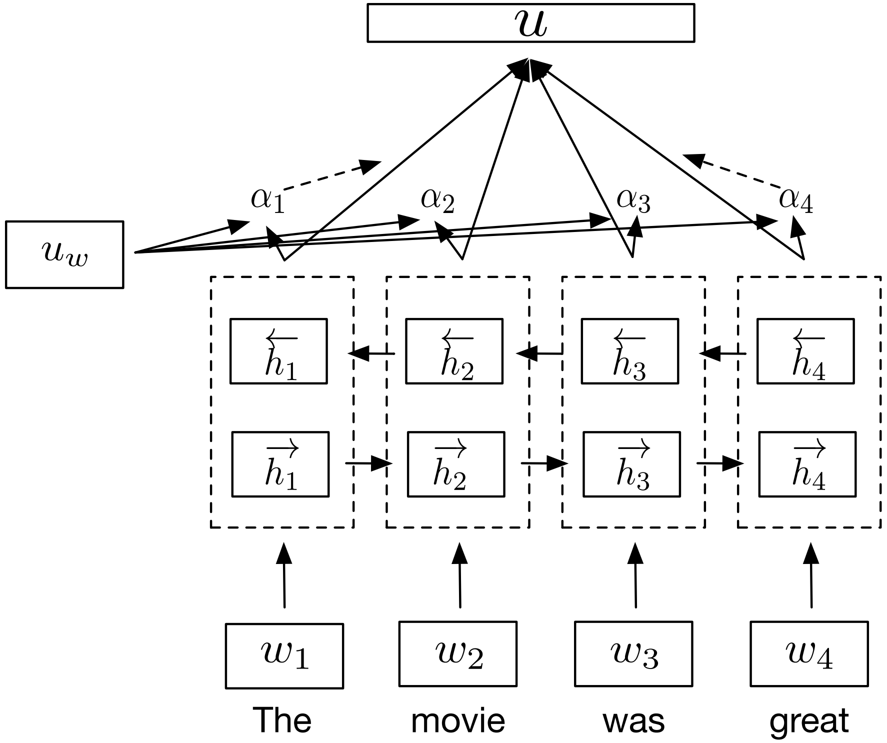


Aggregation

Natural Language Inference

- Sentence representation as weighted average of (contextualized) word representations
- Attention calculation:

$$\bar{h}_i = \tanh(W h_i + b_w)$$
$$\alpha_i = \frac{e^{\bar{h}_i^T u_w}}{\sum_i e^{\bar{h}_i^T u_w}}$$
$$u = \sum_t \alpha_i h_i$$



<u>Formula legend</u>
h_i - hidden state of i-th word
u_w - learned query vector

Credit: Conneau et al. Supervised learning of universal sentence representations from NLI data. (2017)

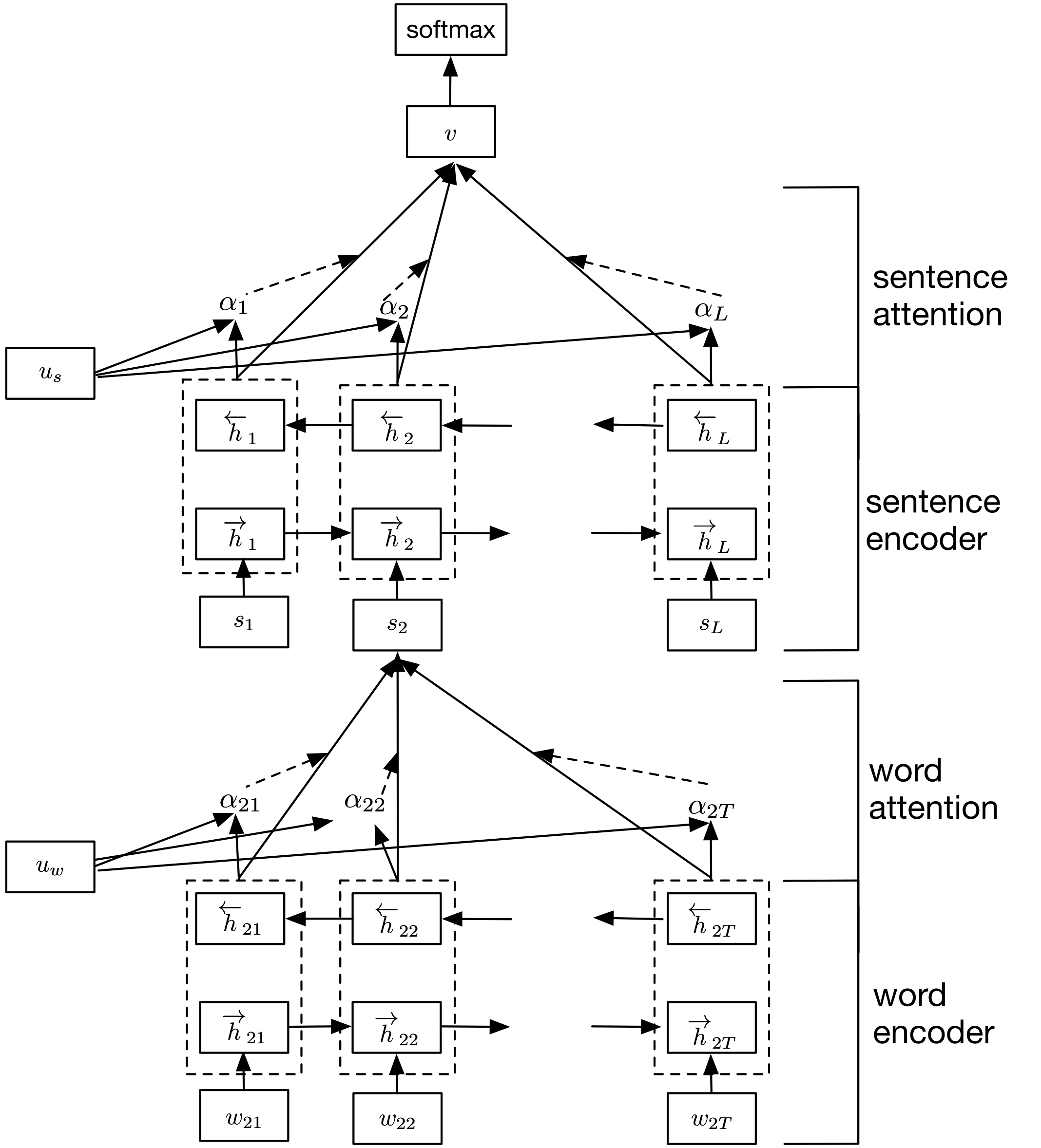
Aggregation

Hierarchical Attention Network

- Extends idea to document representations
- Summarizing hidden states per word into sentence representation

$$u_{it} = \tanh(W_w h_{it} + b_w)$$
$$\alpha_{it} = \frac{\exp(u_{it}^\top u_w)}{\sum_t \exp(u_{it}^\top u_w)}$$
$$s_i = \sum_t \alpha_{it} h_{it}.$$

- Sentences can again be weighted and summed to obtain a document representation

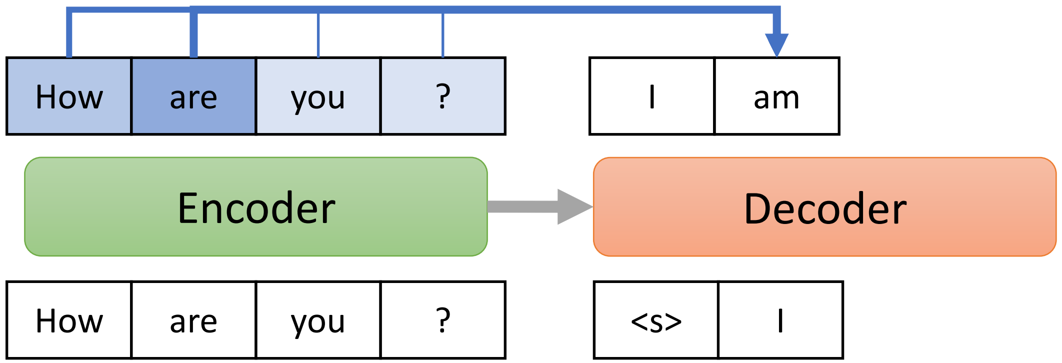


Credit: Yang et al., "Hierarchical Attention Networks for Document Classification" (2016)

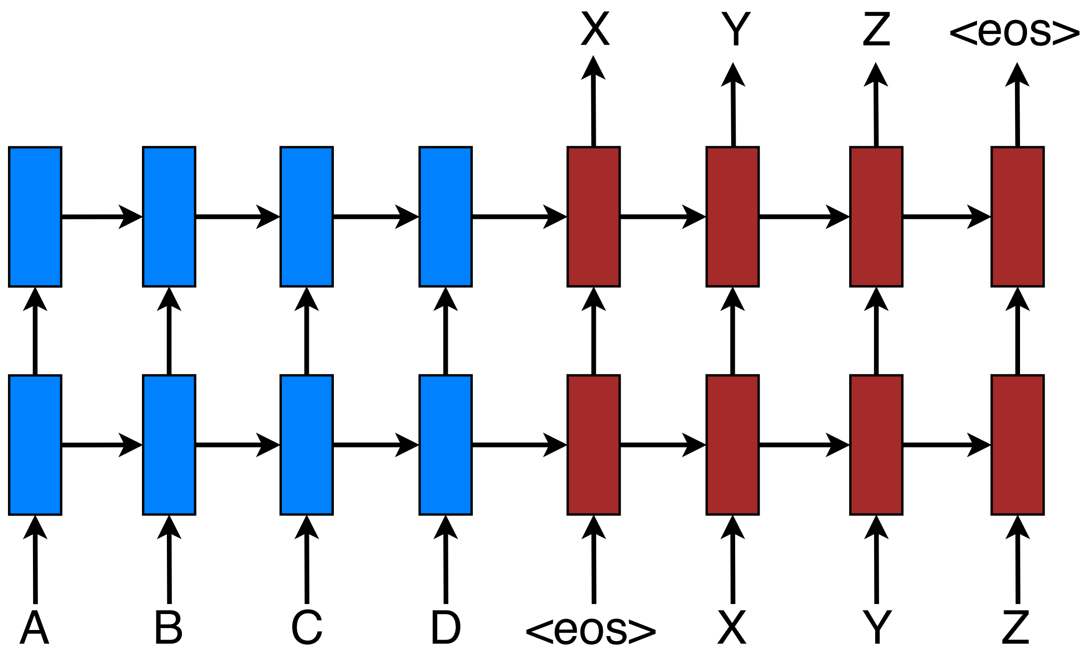
Formula legend
h_{it} - hidden state of t-th word in the i-th sentence
u_w - learned query vector

Encoder-Decoder Attention

- General setup
- Global vs Local Attention
- Applications



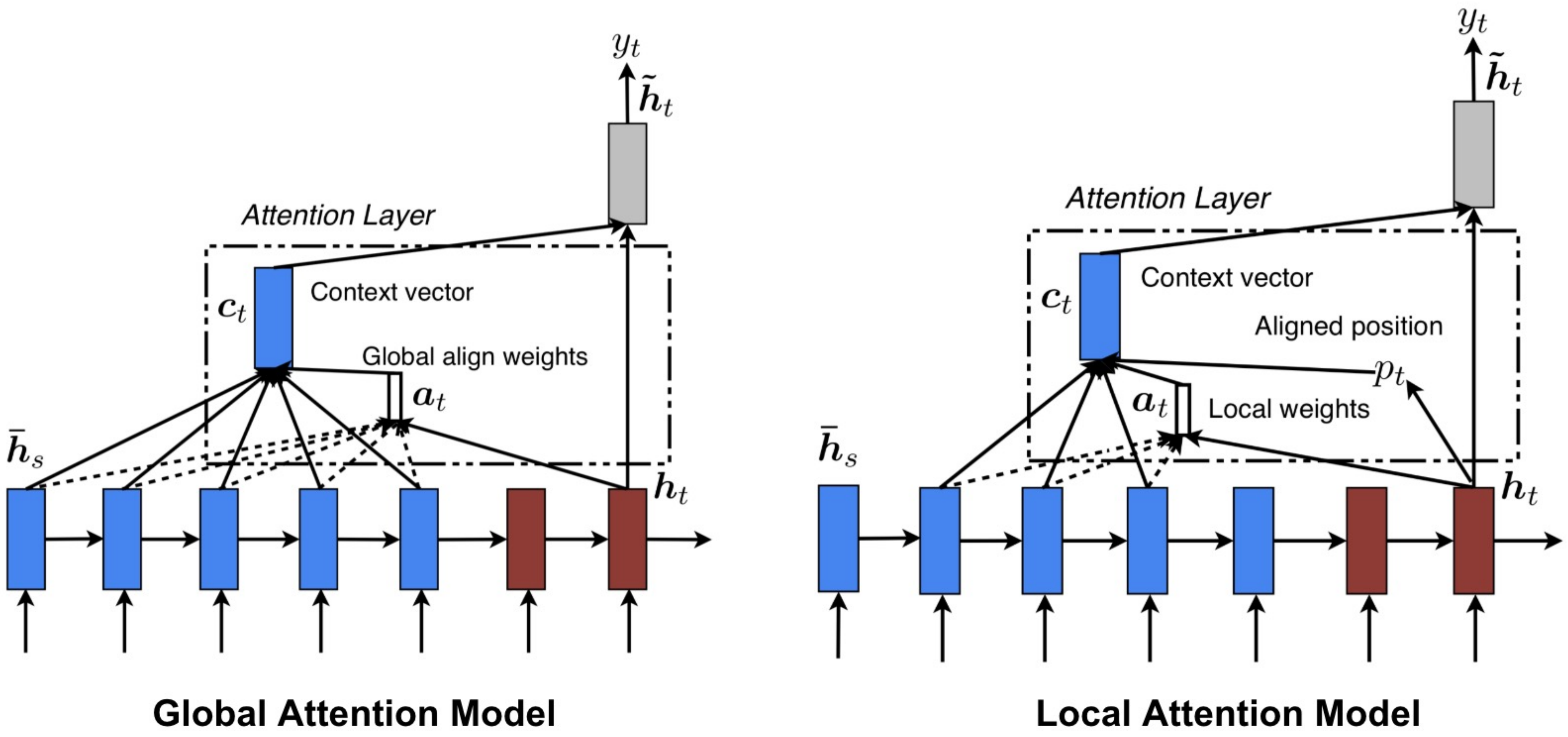
Encoder-Decoder



Credit: Luong et al., "Effective Approaches to Attention-based NMT" (2015)

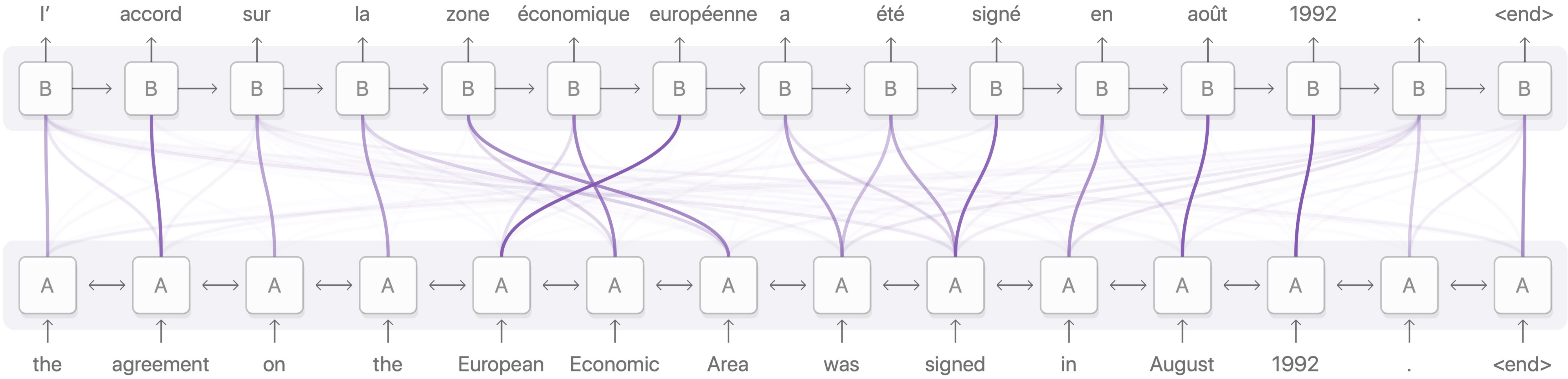
- Suffering from long-term dependencies
- Encoder output must summarize the whole sentences with all its details
- Especially difficult if there are many different possible outputs

Global vs Local Attention



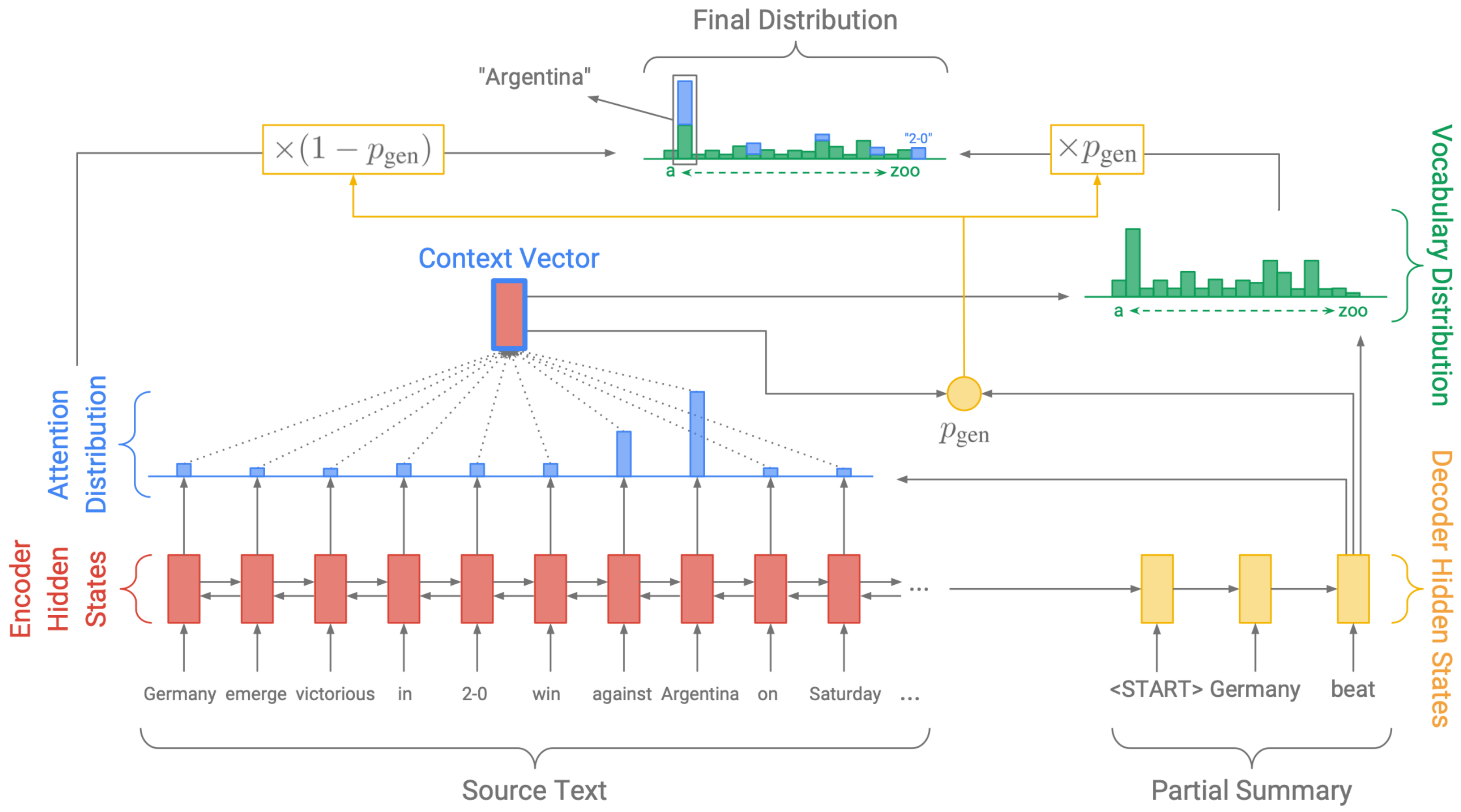
Credit: Luong et al., "Effective Approaches to Attention-based NMT" (2015)

Applications – Machine Translation



Credit: Olah, Chris and Carter, Shan, “[Attention and Augmented Recurrent Neural Networks]”

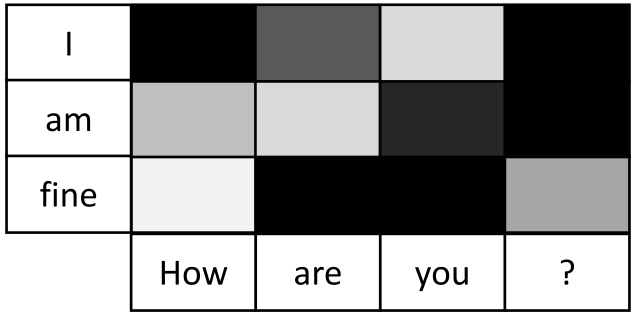
Applications – Summarization



Credit: See et al., "Get To The Point: Summarization with Pointer-Generator Networks"

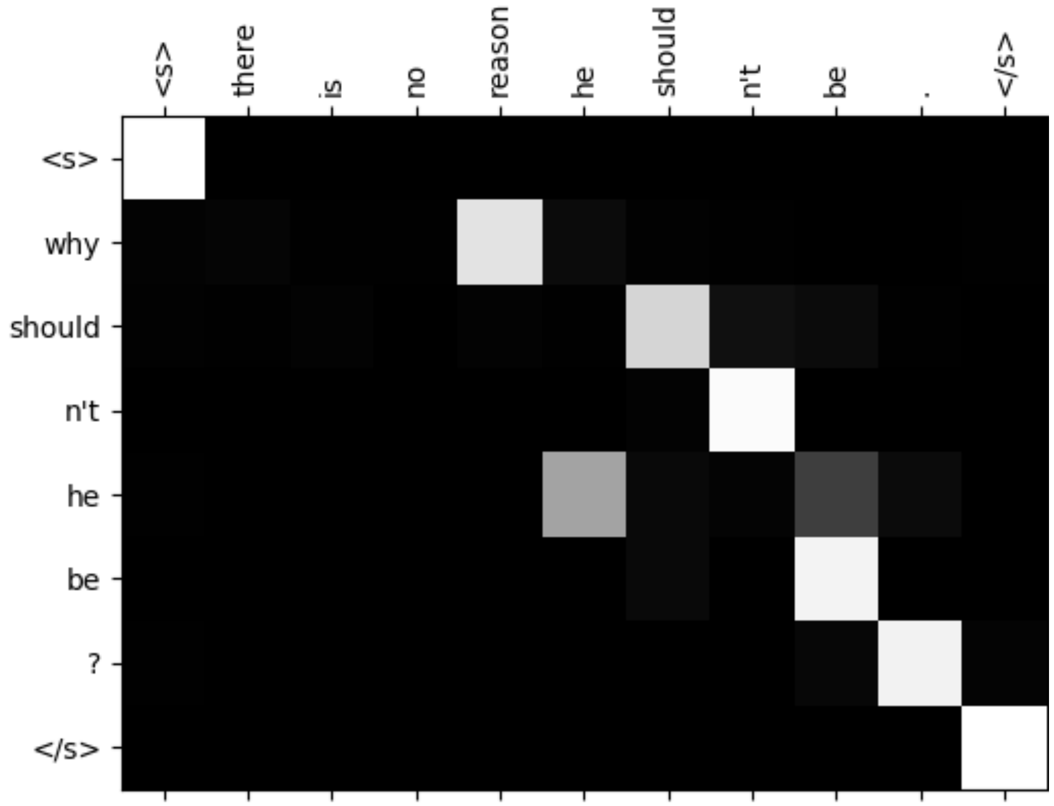
Cross-Attention

- General setup
- Applications



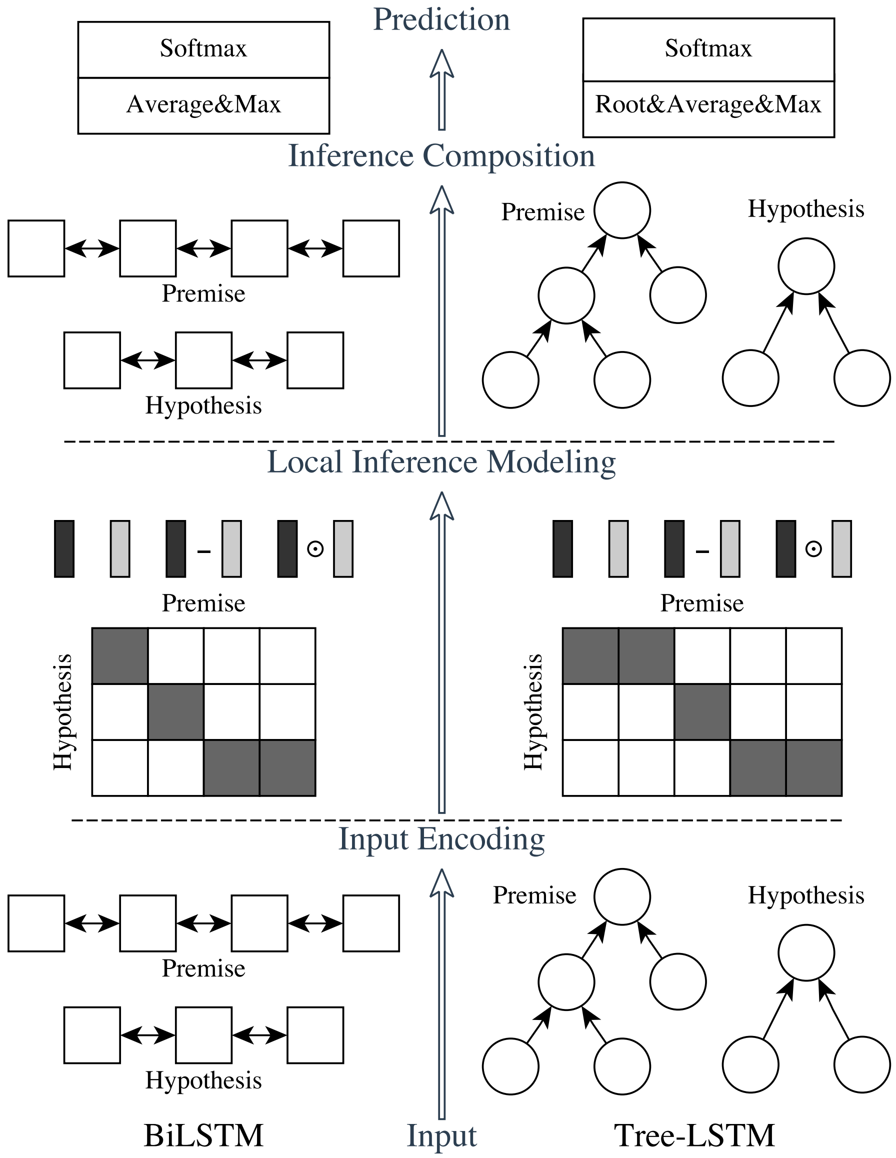
Cross-Attention

- Input: two sentences or sequences
- Task: reason/compare those sentences
- Attention: queries for each word from one sentences, key and value for each word from second sentence



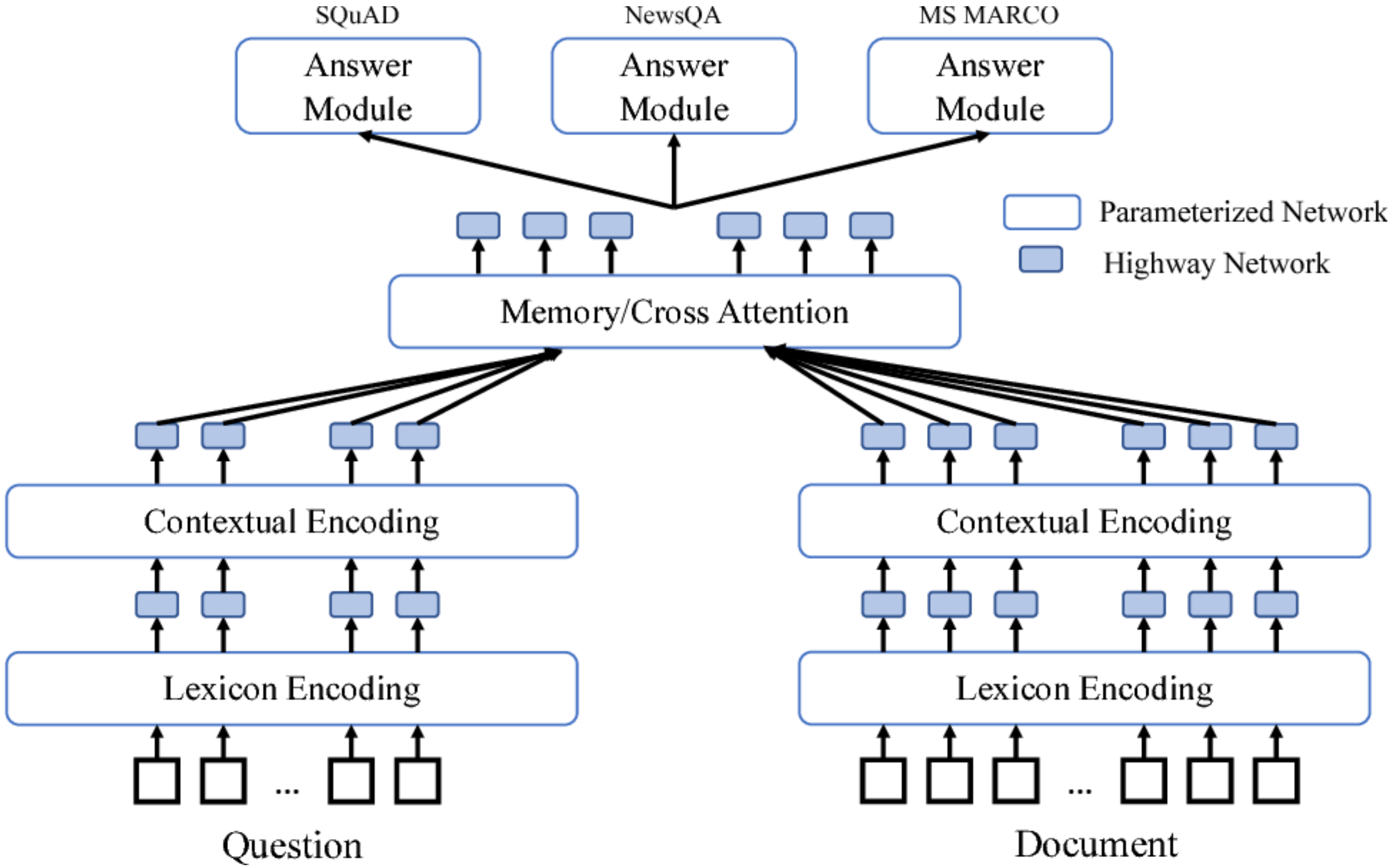
Applications – NLI

- Combining sentence-level with word-level inference
- Premise and hypothesis words can align to find small differences much easier (e.g. “blue” vs “red” bag)



Credit: Chen et al., “Enhanced LSTM for NLI” (2016)

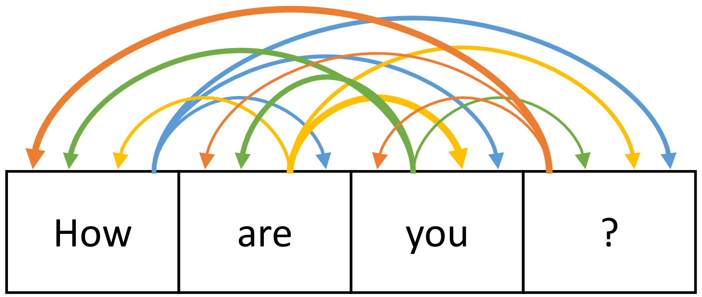
Applications – Question-Answering



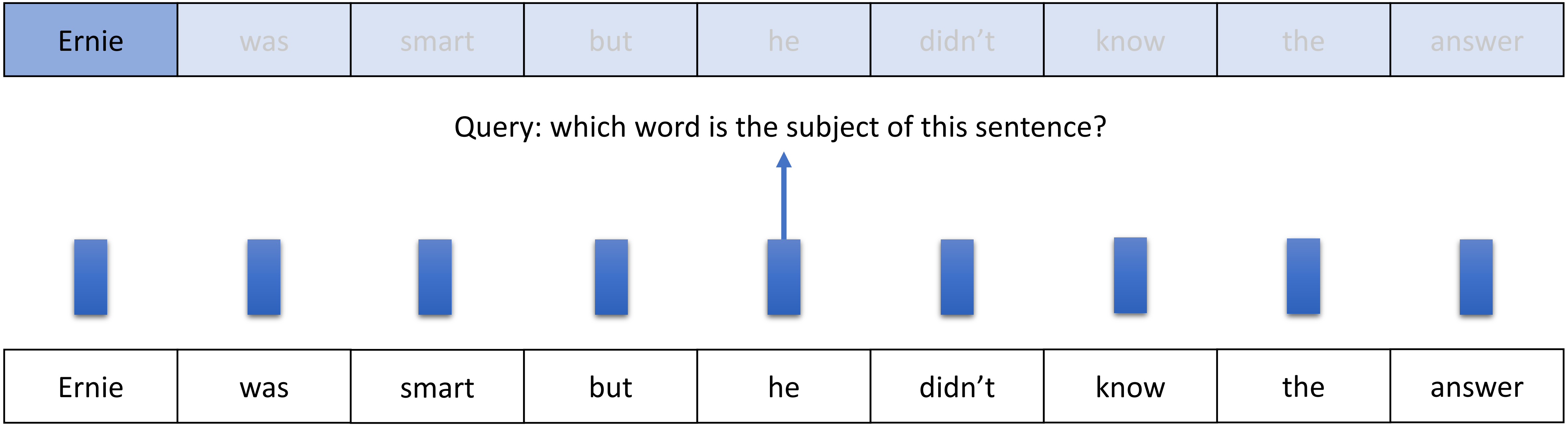
Credit: Xu et al. „Multi-Task Learning for Machine Reading Comprehension.“ (2018)

Self-attention

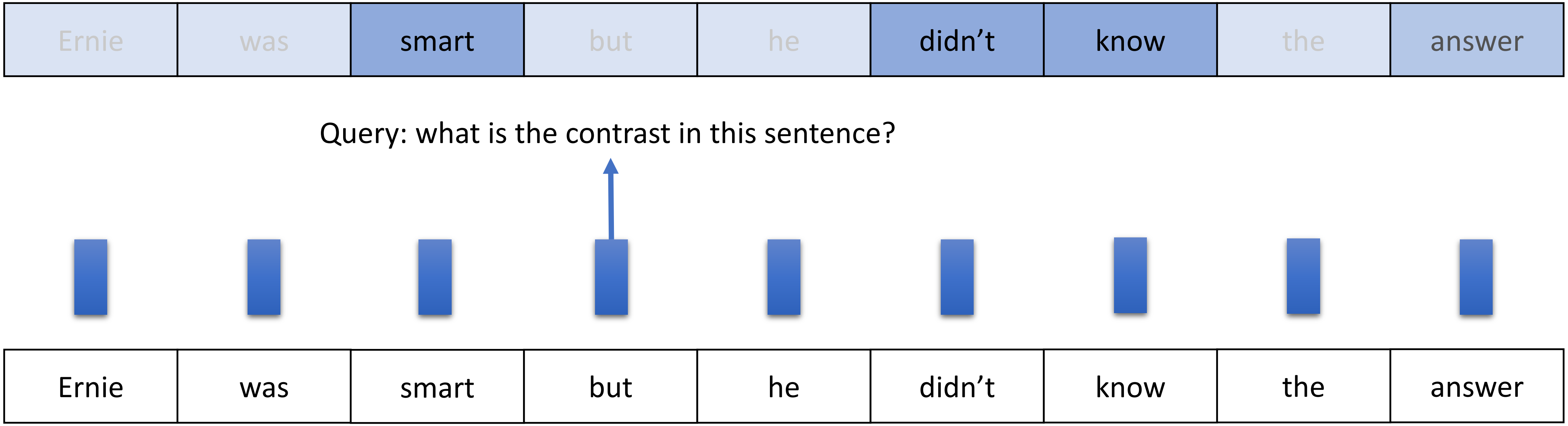
- Intuition and Motivation
- Self-attention layer
- Transformer architecture
- Optimization issues and training tips
- Advanced topics



Intuition

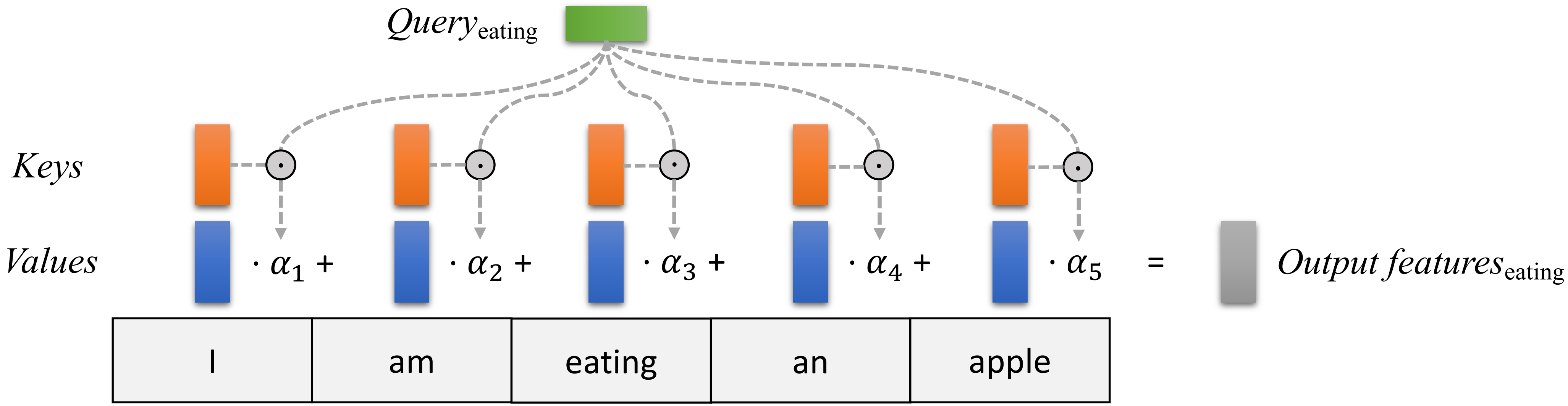

Ernie
was
smart
but
he
didn’t
know
the
answer
Query: which word is the subject of this sentence?
Ernie
was
smart
but
he
didn’t
know
the
answer

Intuition


Ernie
was
smart
but
he
didn’t
know
the
answer
Query: what is the contrast in this sentence?
Ernie
was
smart
but
he
didn’t
know
the
answer

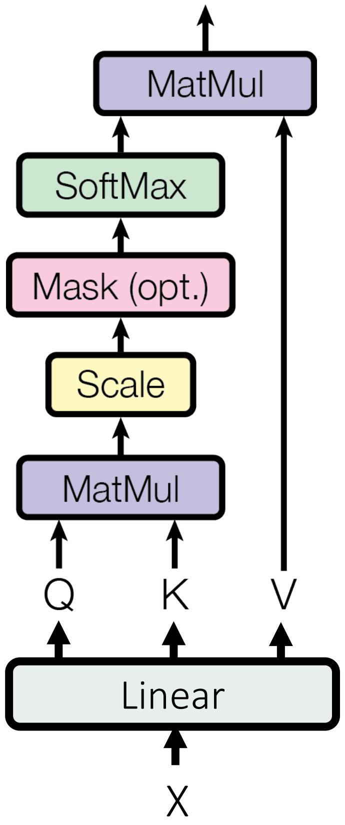
Self-attention

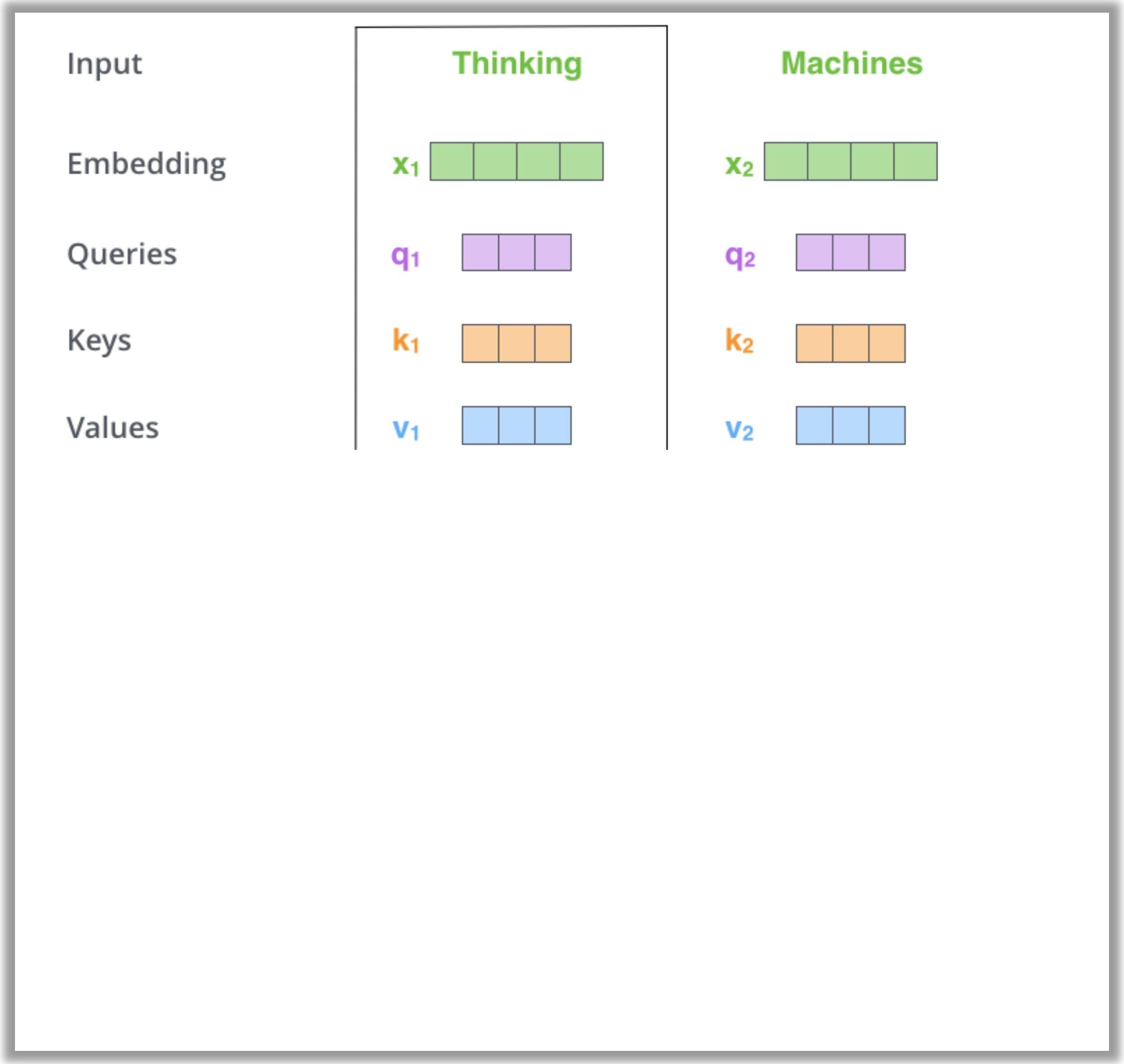
- Self-attention is cross-attention of a sentence to itself
- For every word, we calculate an attention vector as before:



Self-attention layer

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



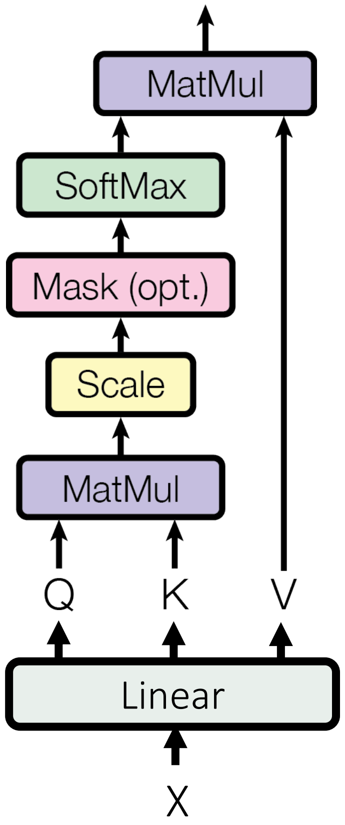


Formula legend
d_k - hidden size of key/query

Credit: Alammar, Jay: The Illustrated Transformer, http://jalammar.github.io/illustrated-transformer/

Self-attention layer

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



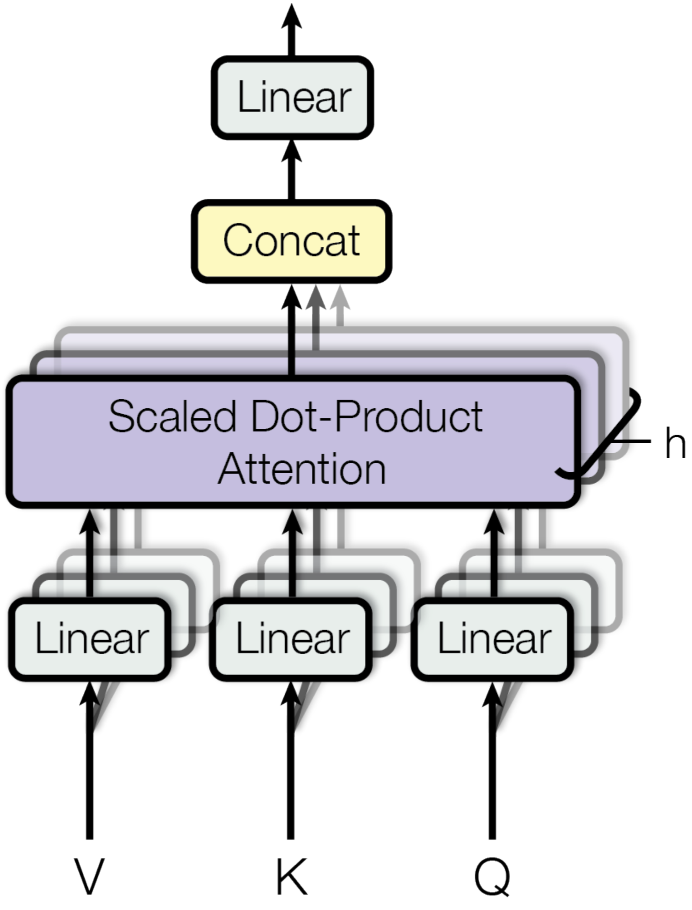
Why scaling by $1/\sqrt{d_k}$?

- The variance of the dot product scales linearly with d_k
  $\Rightarrow$ Scaling brings it back to 1

$$q_i \sim \mathcal{N}(0,\sigma^2), k_i \sim \mathcal{N}(0,\sigma^2) \rightarrow \text{Var}\left(\sum_{i=1}^{d_k} q_i \cdot k_i\right) = \sigma^4 \cdot d_k$$

- High initial values significantly harm gradient flow

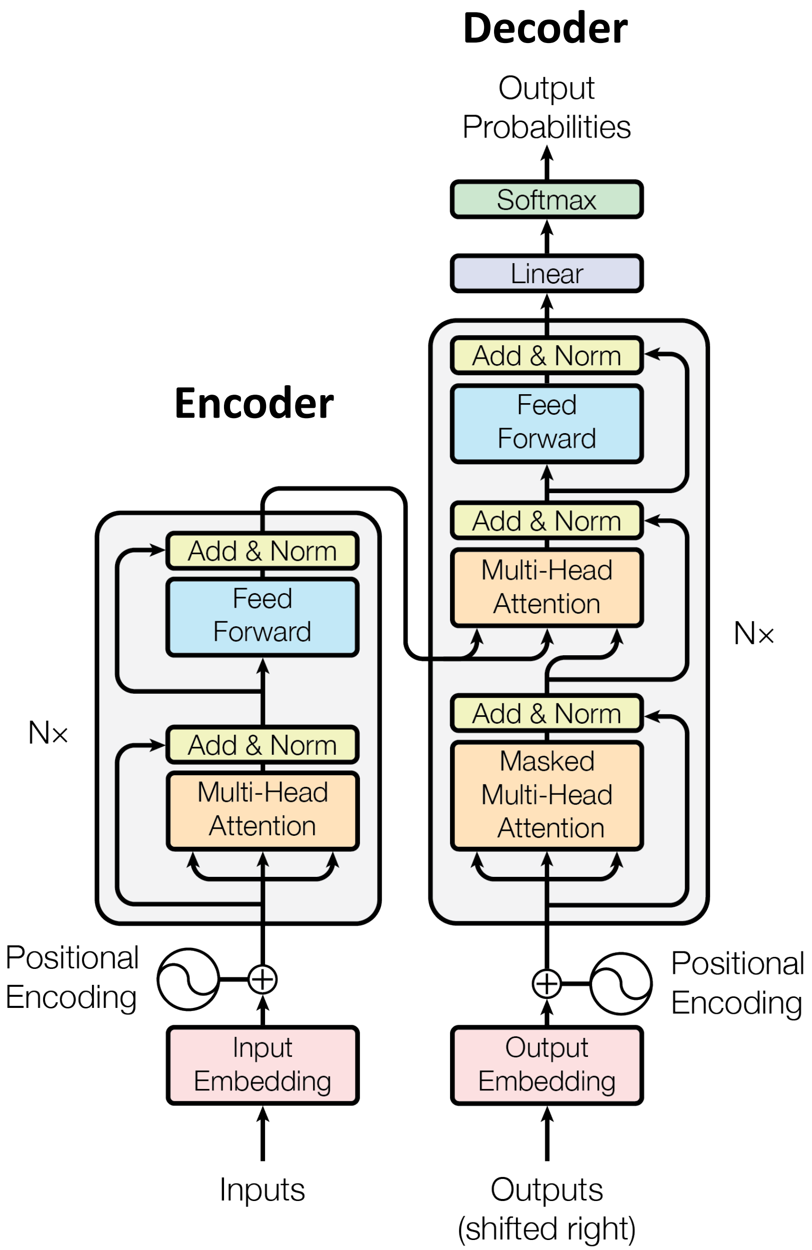
Multi-Head self-attention



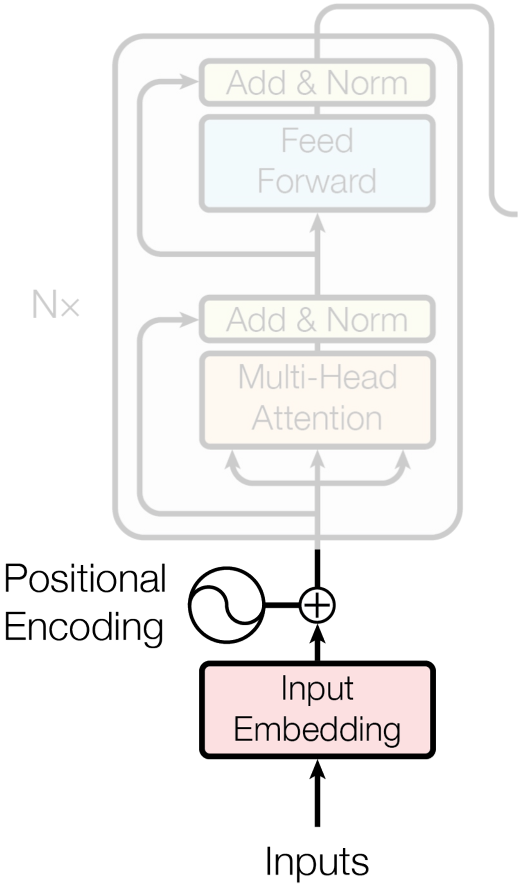
- Single head offers only one perspective on the data
  ⇒ Often not enough, can harm gradients again
- Performing several self-attentions in parallel increases flexibility and non-linearity/complexity
- Output projection to scale down the concatenation if necessary

Transformer architecture

- The (original) Transformer has an encoder-decoder structure
- Both parts consist of N blocks with self-attention layers
- Initially designed for machine translation
  - Encoder analyses input sentence
  - Decoder predicts output sentence autoregressively



Transformer - Encoder



Byte-pair encoding

- Encode common subtokens instead of only words

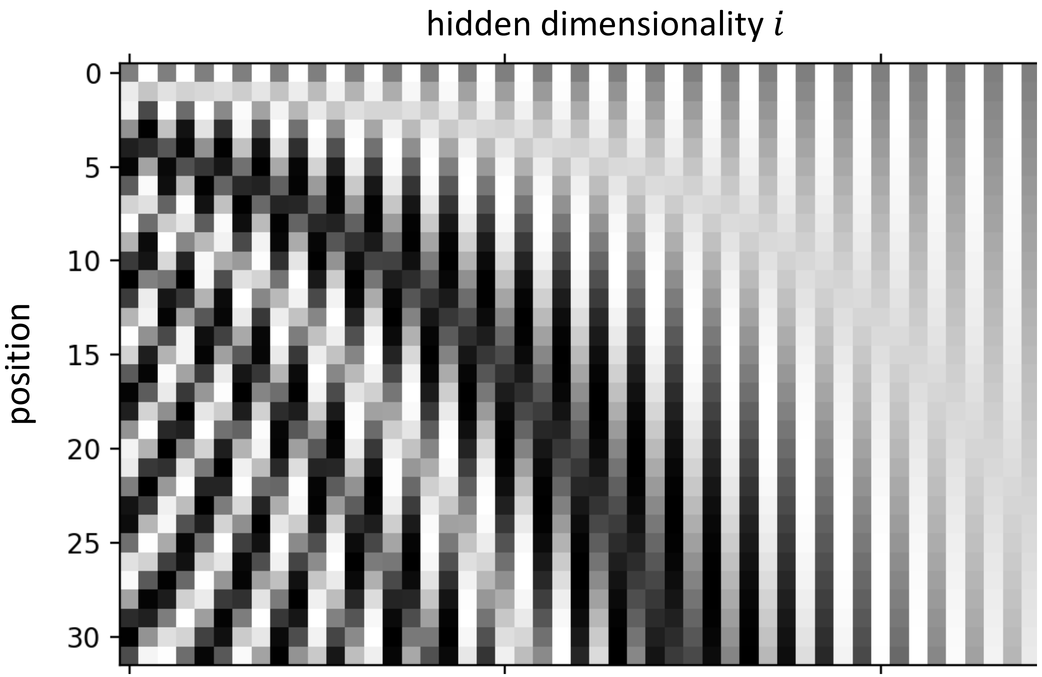
  smarter ⇒ smart-er, tokenized ⇒ token-ized

- Easier adaptation to unseen words in the training corpus
- Sharing of common word parts ("-ing", "re-", etc.)

Positional embeddings

- Self-attention layers do not encode positions but view the input as a **set** (permutation equivariant)
- Sinusoidal positional encoding added to embeddings

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

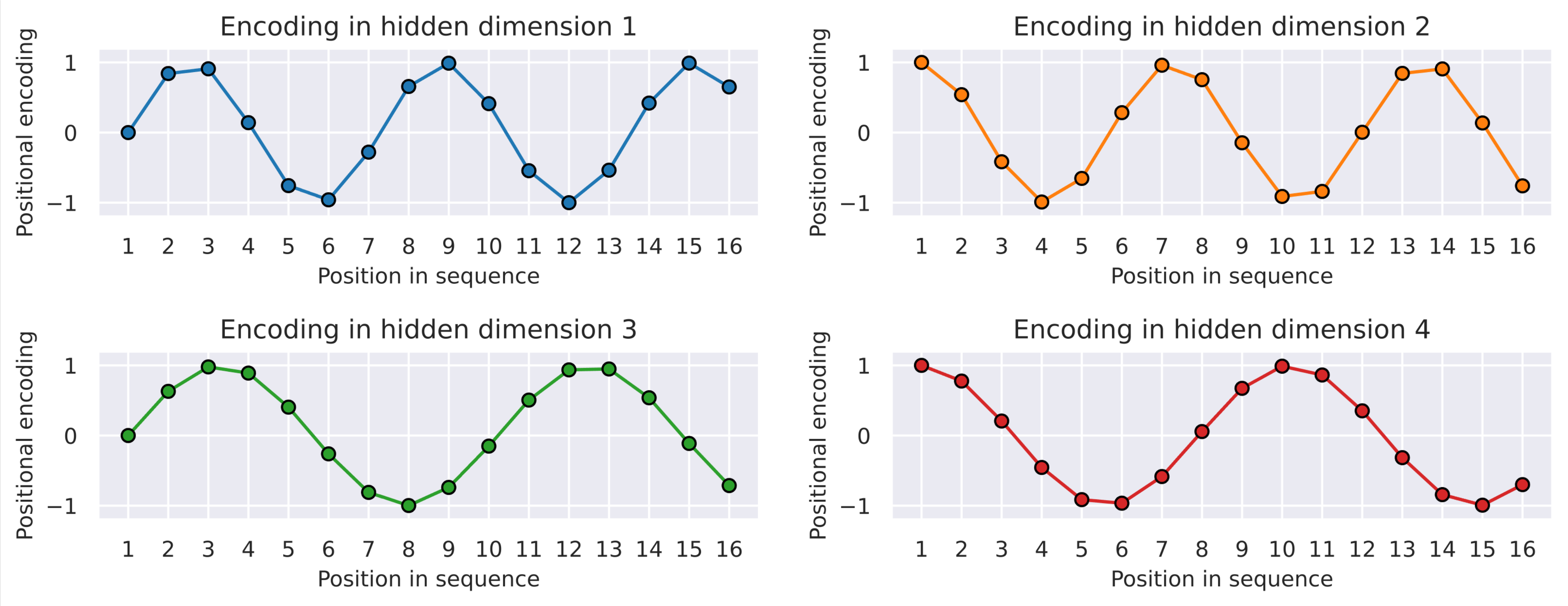


Credit: Weng, Lilian: [The Transformer family]

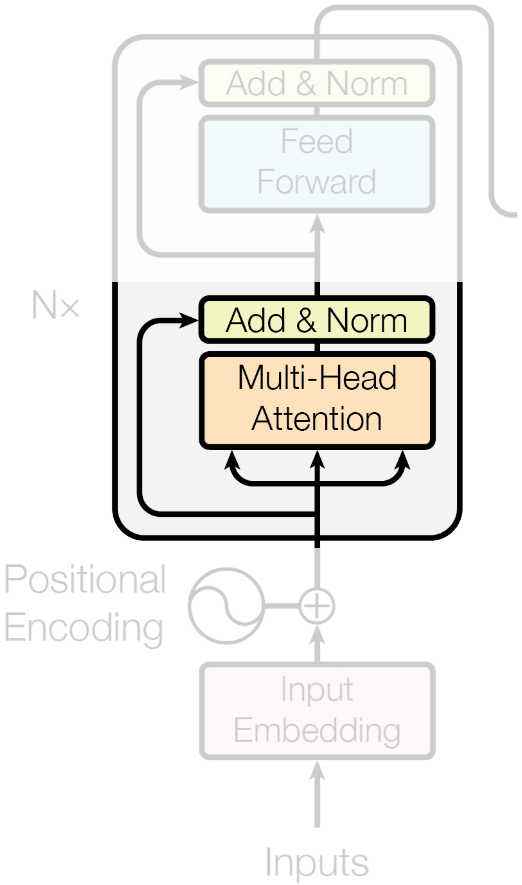
- Scales to unseen lengths
- Encodes relative positions as linear functions

Formula legend
d_{model} - hidden size of embedding
i – index over the hidden dimension
pos – position of word in sentence

Positional embeddings

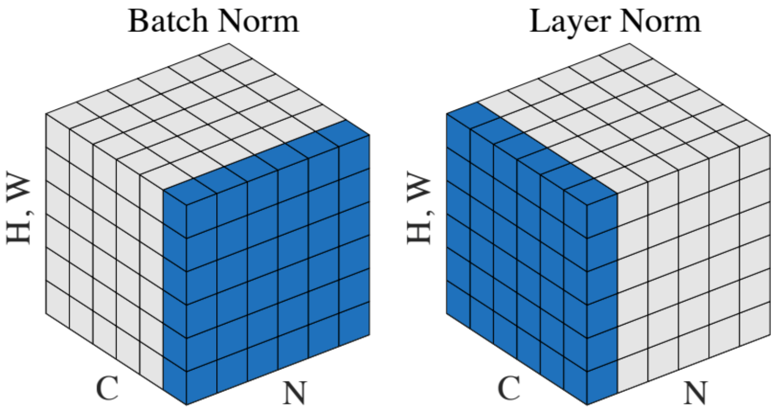


Transformer - Encoder



- Residual connection combined with Layer normalization

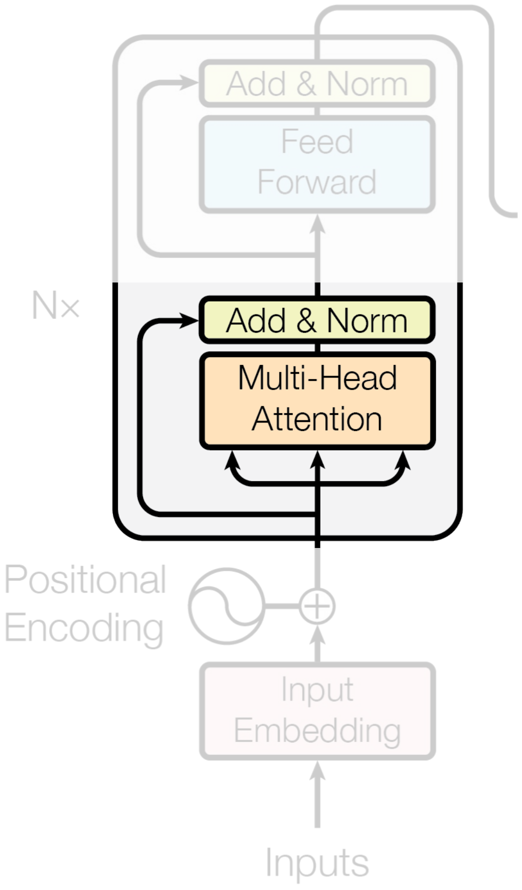
$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$



Credit: Wu, Yuxin and He, Kaiming. Group Normalization. (2018),

*Statistics only over channels here, not sequence length

Transformer - Encoder



- Residual connection combined with Layer normalization

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

Why do we need residual connections?

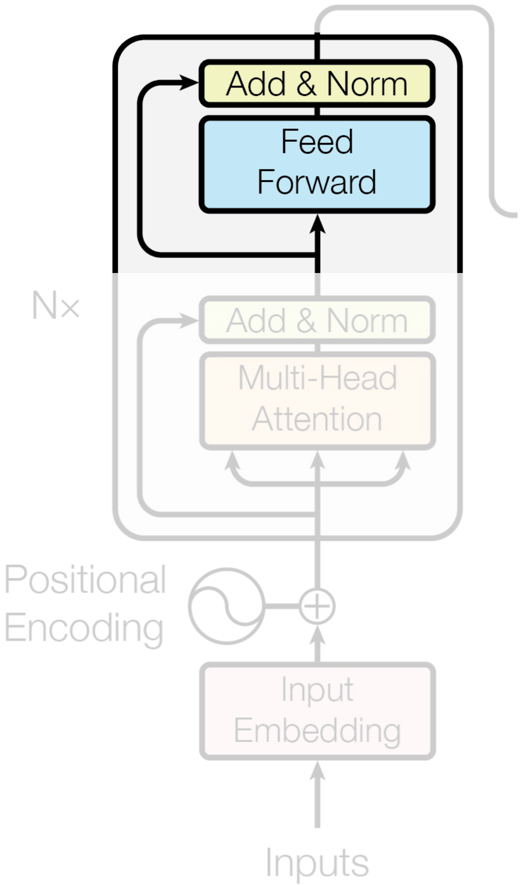
- Better gradient flow
- Word/position information would get lost, especially after init

  ⇒ Over-smoothing

Why do we need Layer normalization?

- Faster training and prevent variance explosion
- Not batch normalization due to high variance in language features

Transformer - Encoder



- Point-wise feed-forward network with ReLU activation

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

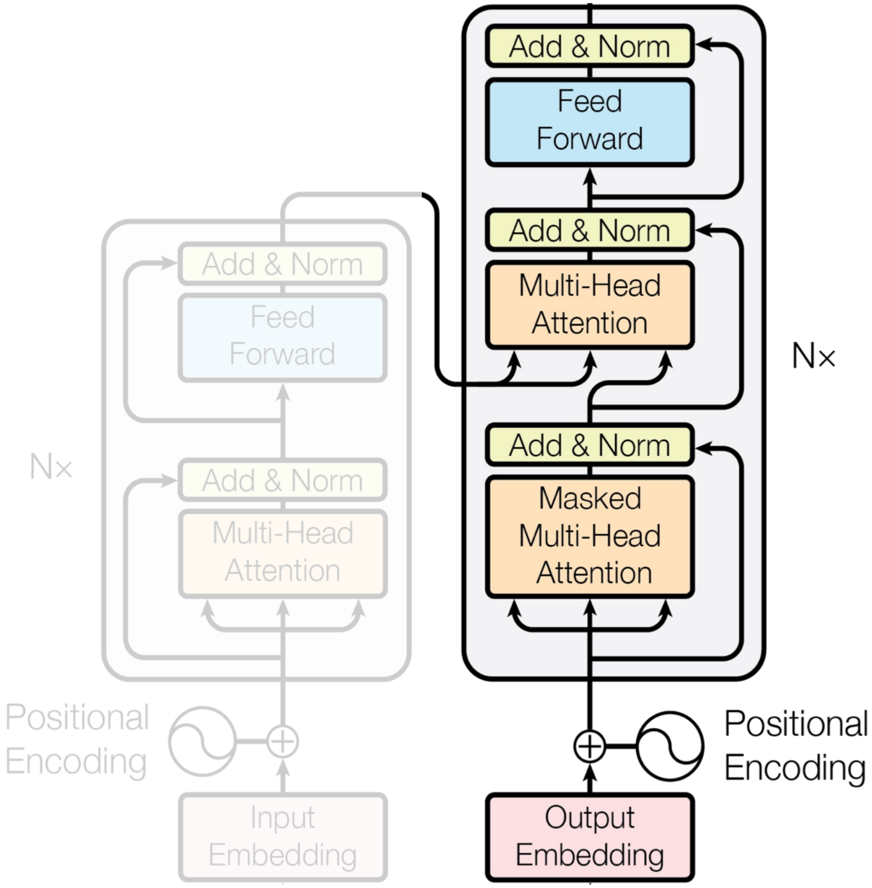
- Adds complexity with classical non-linearity to network
- Prepares features for the next attention layer
- Inner hidden dimensionality commonly 4-8x larger

Why larger hidden dimensionality instead of deeper MLP?

- Faster computation (can be run in parallel)
- Single layer complexity sufficient

<u>Formula legend</u>
W – weight matrix
b – bias vector

Transformer - Decoder



- Multi-head self-attention masked for autoregressive prediction
- Additional attention sublayer over encoder output layer
  - Key and value features from encoder
  - Query features from decoder
- Linear output layer and softmax over vocabulary

Transformer - Performance

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**	$2.3 \cdot 10^{19}$	
SOTA Transformer (2022)	35.1	46.4		

Is attention all we need?

Transformers

+ State-of-the-art on most benchmarks
+ Scalable to trillions of parameters (Switch transformer – 1.6 trillion params)
+ Computation in parallel (feedforward network)

– Recurrence needs to be learned
  ⇒ lots of data required or pretrained model
– Many parameters for suitable model necessary
  ⇒ can easily overfit
– Memory scales quadratically with seq length

RNNs

+ Language is naturally recurrent
+ Higher non-linearity and more complex composition ⇒ Single-layer RNN outperforms single-layer transformer
+ Efficient inference due to compact state

– Naïvely, does not scale well beyond 5 layers
– Slower to train for long sequences
– Long-term dependencies problematic

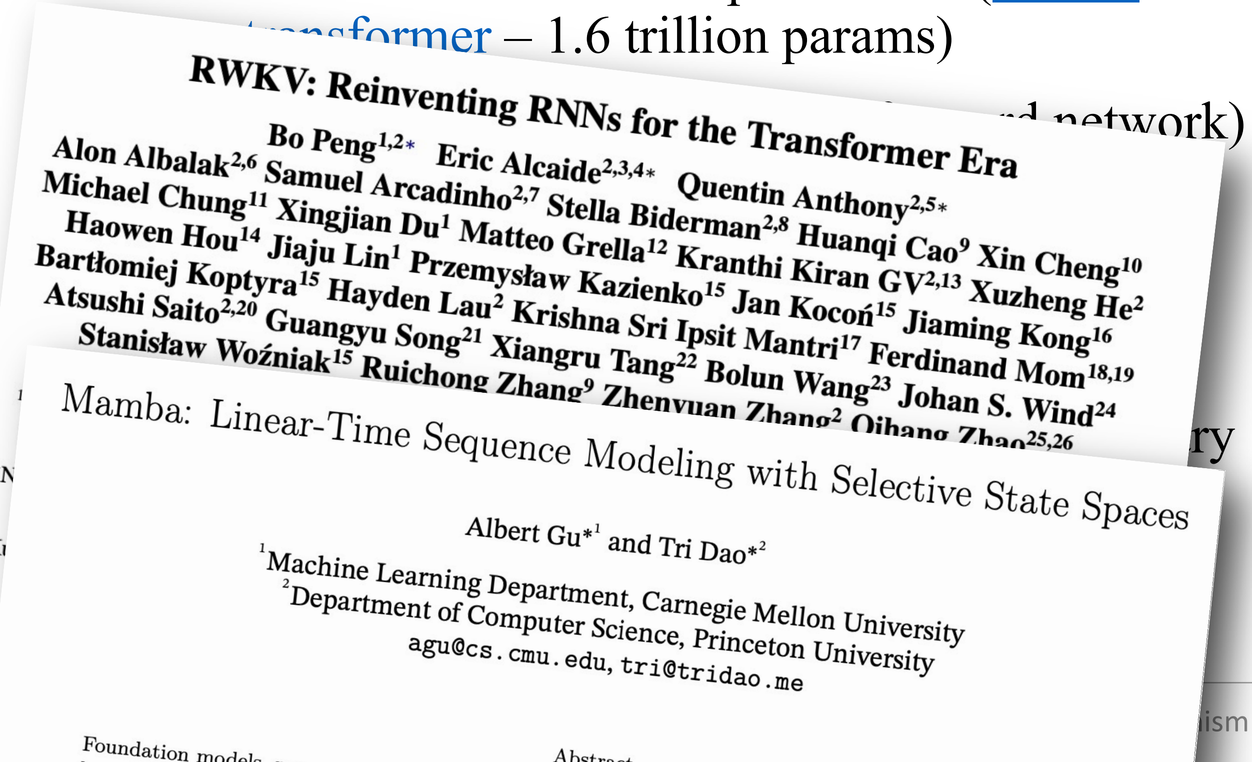
Is attention all we need?

Transformers

+ State-of-the-art on most benchmarks
+ Scalable to trillions of parameters (Switch transformer – 1.6 trillion params)

...rd network)

...ry


RWKV: Reinventing RNNs for the Transformer Era

Bo Peng[1,2*] Eric Alcaide[2,3,4*] Quentin Anthony[2,5*]
Alon Albalak[2,6] Samuel Arcadinho[2,7] Stella Biderman[2,8] Huanqi Cao[9] Xin Cheng[10]
Michael Chung[11] Xingjian Du[1] Matteo Grella[12] Kranthi Kiran GV[2,13] Xuzheng He[2]
Haowen Hou[14] Jiaju Lin[1] Przemysław Kazienko[15] Jan Kocoń[15] Jiaming Kong[16]
Bartłomiej Koptyra[15] Hayden Lau[2] Krishna Sri Ipsit Mantri[17] Ferdinand Mom[18,19]
Atsushi Saito[2,20] Guangyu Song[21] Xiangru Tang[22] Bolun Wang[23] Johan S. Wind[24]
Stanisław Woźniak[15] Ruichong Zhang[9] Zhenyuan Zhang[2] Qihang Zhao[25,26]

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu*[1] and Tri Dao*[2]
[1]Machine Learning Department, Carnegie Mellon University
[2]Department of Computer Science, Princeton University
agu@cs.cmu.edu, tri@tridao.me

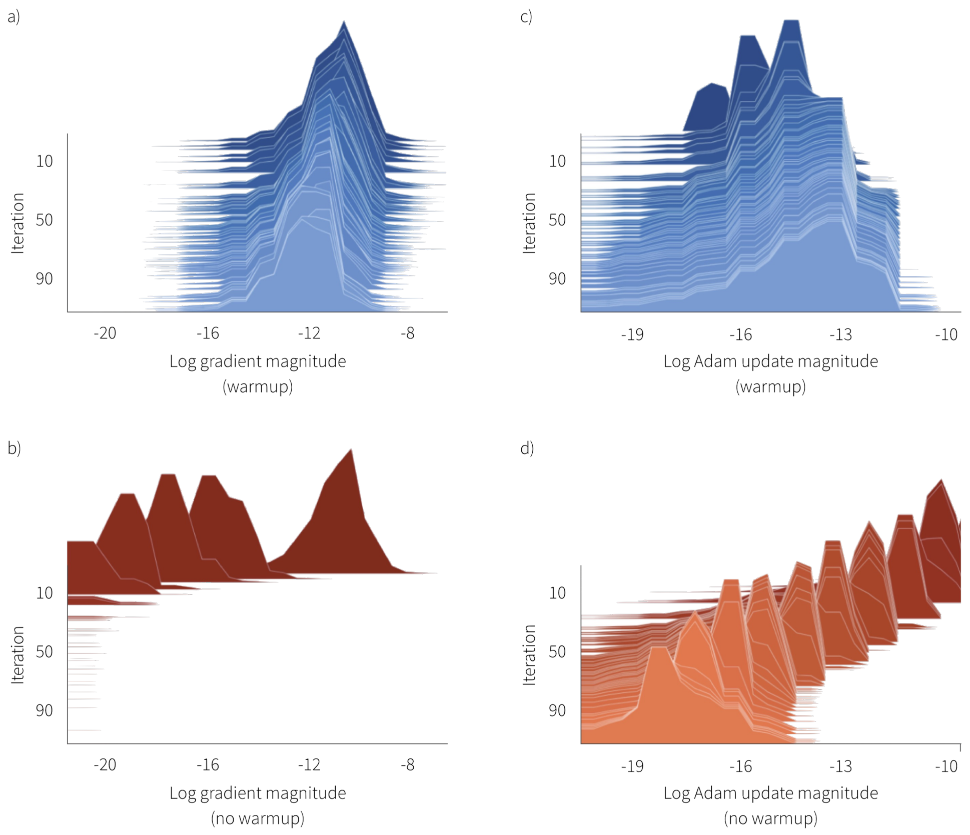
RNNs

+ Language is naturally recurrent
+ Higher non-linearity and more complex composition ⇒ Single-layer RNN outperforms single-layer transformer
+ Efficient inference due to compact state

– Naïvely, does not scale well beyond 5 layers
– Slower to train for long sequences
– Long-term dependencies problematic

Transformers – Advanced Topics

- Training and finetuning tips
- Why is warmup so critical?
- Improvements since Vanilla Transformers
- Transformers as Graph NNs
- Efficient Transformers



Credit: Xu, P. and Prince, S. [Transformers III, Training](), (2021)

Transformers – Training tips

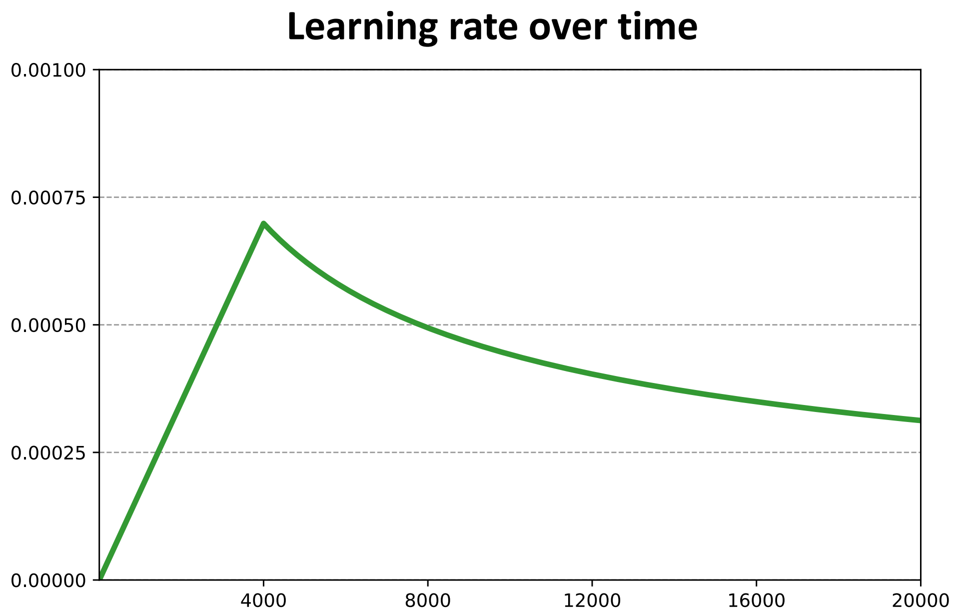
- Training Transformers can be painful on a single small GPU…
- Use optimizer with adaptive learning rate (e.g. Adam) with learning rate warmup
- Higher batch sizes are often beneficial. To reduce memory, consider removing the (significantly) largest sentences from training. **But**…
  - Transformers have been shown to generalize poorly to sentence lengths differing from training set
  - Only remove if there are very few very long sentences
- Training with huge batch size across many GPUs comes with new challenges
  But don't worry if you're not Google, OpenAI or NVIDIA (Lamb, ZeRO, Scaling Tutorials)
- Reduce memory footprint with mixed precision training and gradient checkpointing.
- BPE vocabulary must be trained on sufficient data. Otherwise, it easily overfits

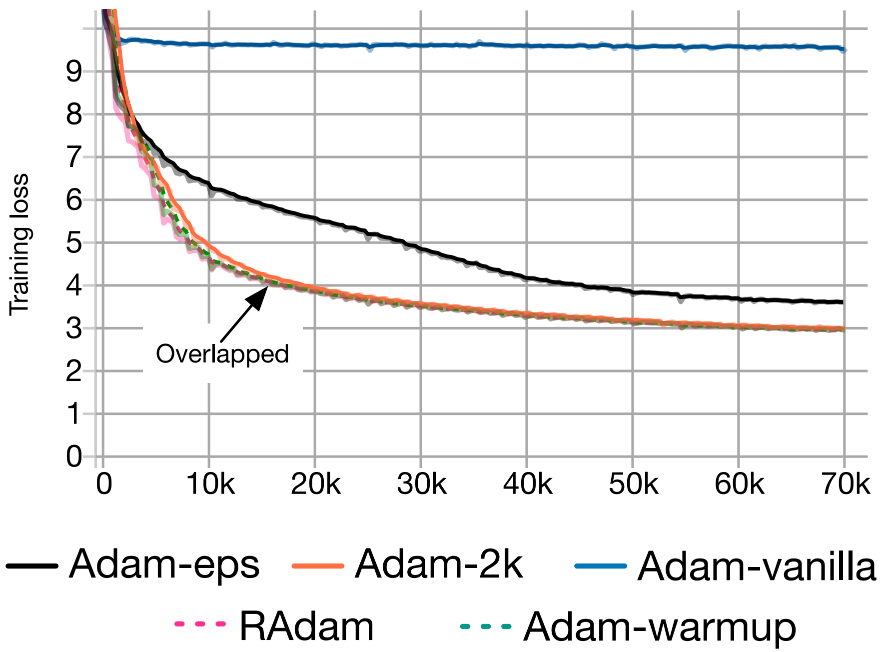
Transformers – Finetuning

- Many state-of-the-art performances can be achieved by finetuning large pre-trained language models or prompting tuning
- If you want to finetune yourself, use libraries such as Hugging Face
- Use optimizer with adaptive learning rate (e.g. Adam) with learning rate warmup
  - Good learning rates usually between 1e-5 and 2e-4
- If you want to find good initial hyperparameters, consider:
  - The following paper on hyperparameter search: Dodge et al., 2020
  - The examples in the Hugging Face library for different tasks (link)
- Don't finetune whole model, but only the last few layers or with Low-Rank Adapters (LoRA)
- Regularization like weight decay or dropout often helps

Transformers – Warmup

- Learning rate warmup is one of the most important hyperparameters

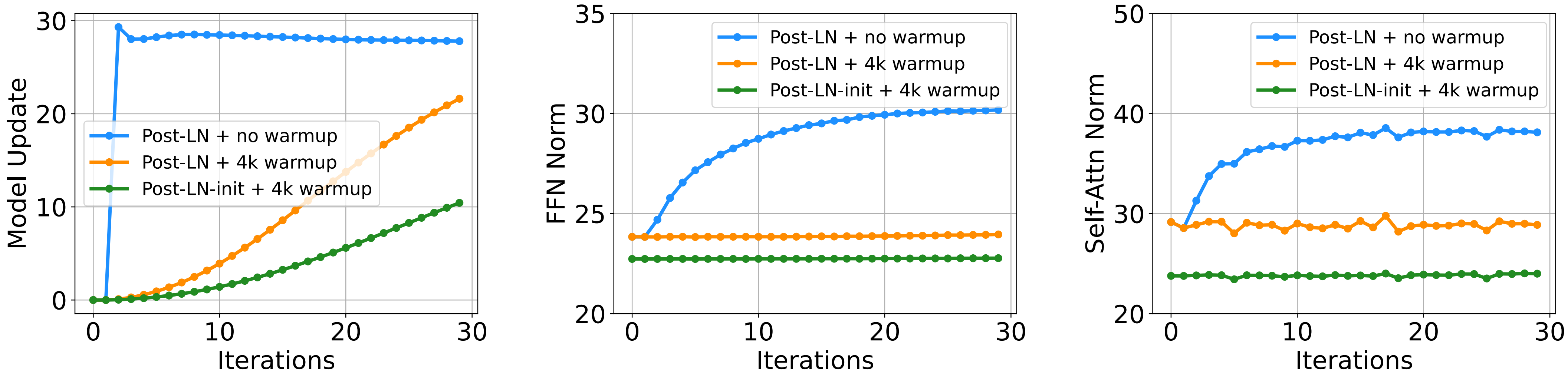




Credit: Liu et al., “On the variance of the adaptive learning rate and beyond” (2020)

Transformers – Warmup

- Why is warmup so critical?
- Without warmup, the model heavily changes in the first step, and then almost stops



(a) Accumulated model update (b) Input from FFN to LN (c) Input from attention to LN

Wang et al., 2022: DeepNet: Scaling Transformers to 1000 Layers

Transformers – Warmup

- Why is warmup so critical?

(1) **Variance in adaptive learning rate**

Adam:
$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1)\cdot g^{(t)}$$
$$v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2)\cdot \left(g^{(t)}\right)^2$$
$$\boxed{\hat{m}^{(t)} = \frac{m^{(t)}}{1-\beta_1^t}, \hat{v}^{(t)} = \frac{v^{(t)}}{1-\beta_2^t}}$$
$$w^{(t)} = w^{(t-1)} - \frac{\eta}{\sqrt{v^{(t)}}+\epsilon}\circ \hat{m}^{(t)}$$

High variance in first iterations.

Better: RAdam ([Liu et al., 2020](#))

[Hugging Face](#): skip bias correction

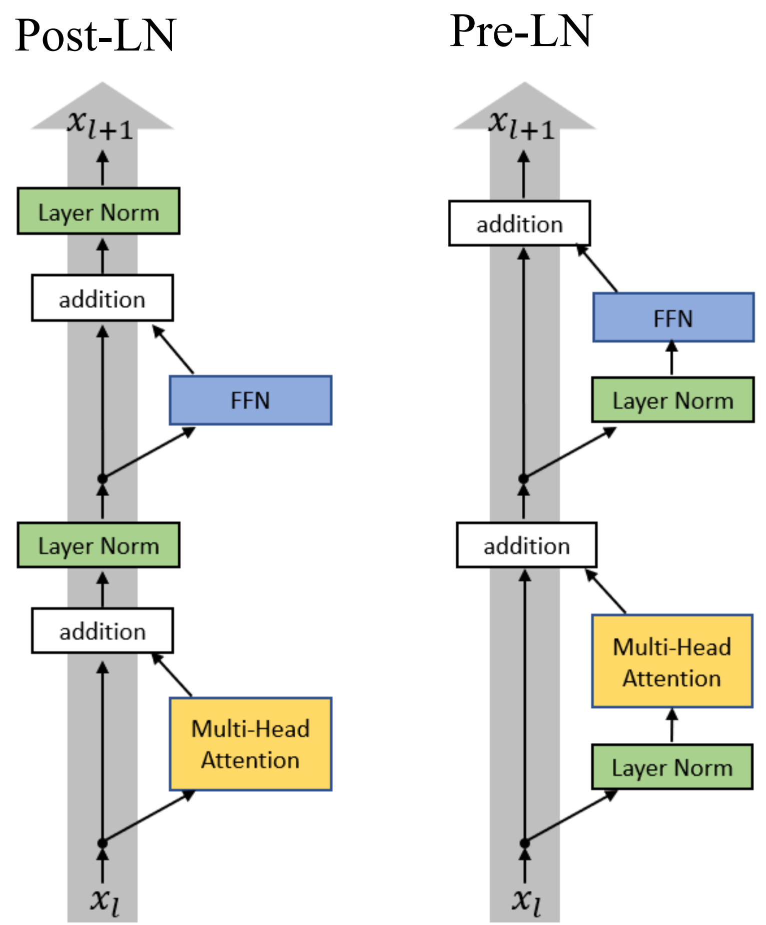
<u>Formula legend</u>
g^t - gradient at iteration t
m – momentum
v – second-order momentum (adaptive lr)
w – weight parameters
β_1, β_2 - Adam hyperparameters

Transformers – Warmup

- Why is warmup so critical?

(2) **Layer Normalization**

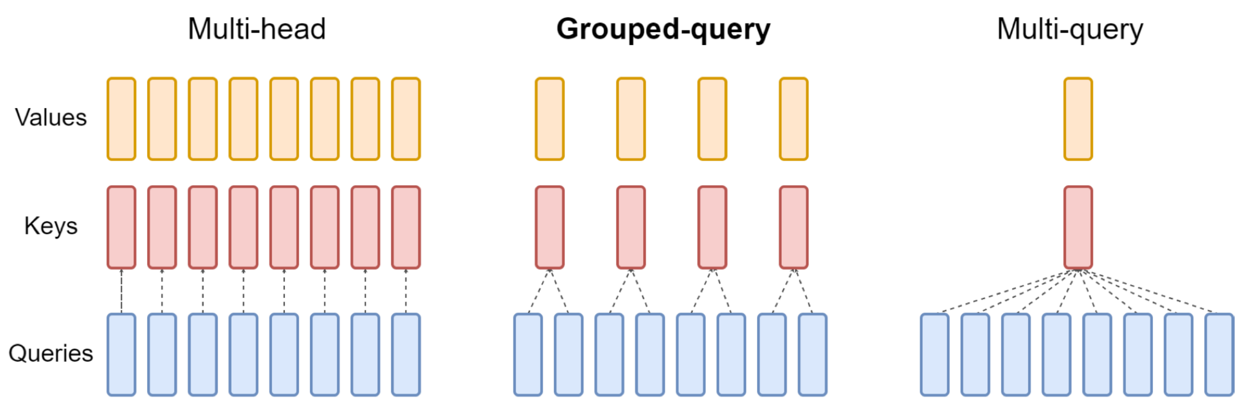
- After initialization, the expected gradients of the parameters near the output layer are very large
- LayerNorm downscales variance after each layer
  ⇒ last FFN and Multi-head attention layer
  have gradients independent of number of layers
- More stable: use Pre-Layer Normalization or alternatives (Adaptive Normalization, Power Normalization)
- However, Post-LN used to get best performance



Credit: Xiong et al., "On Layer Normalization in the Transformer Architecture" (2020)

Transformers – Improvements

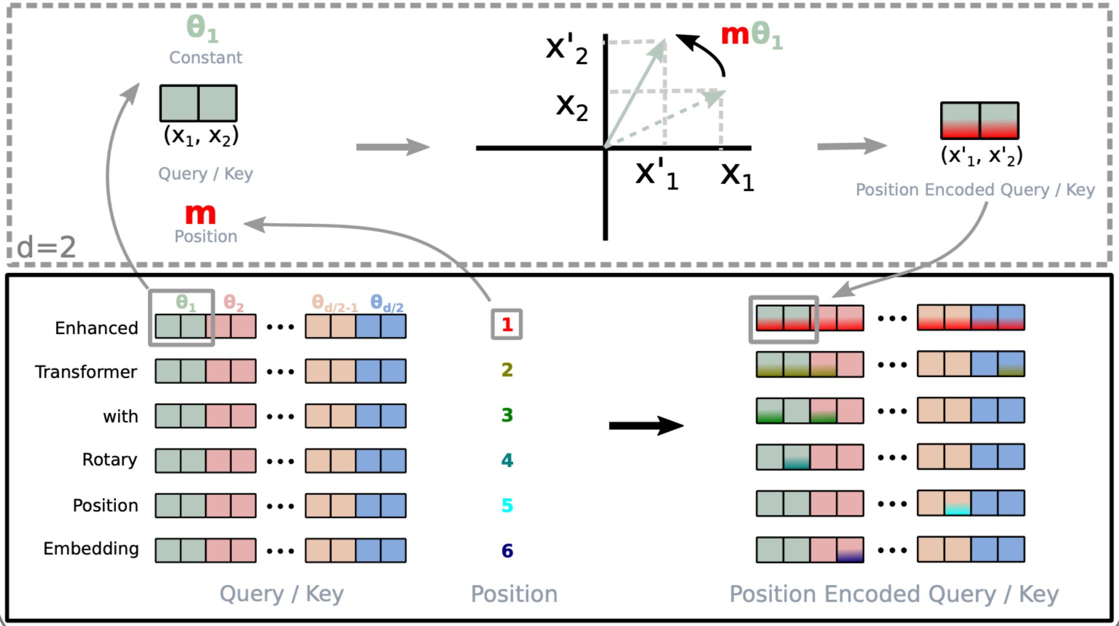
Multi-Query Attention



RMSNorm

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

RoPE: Rotary Position Embeddings

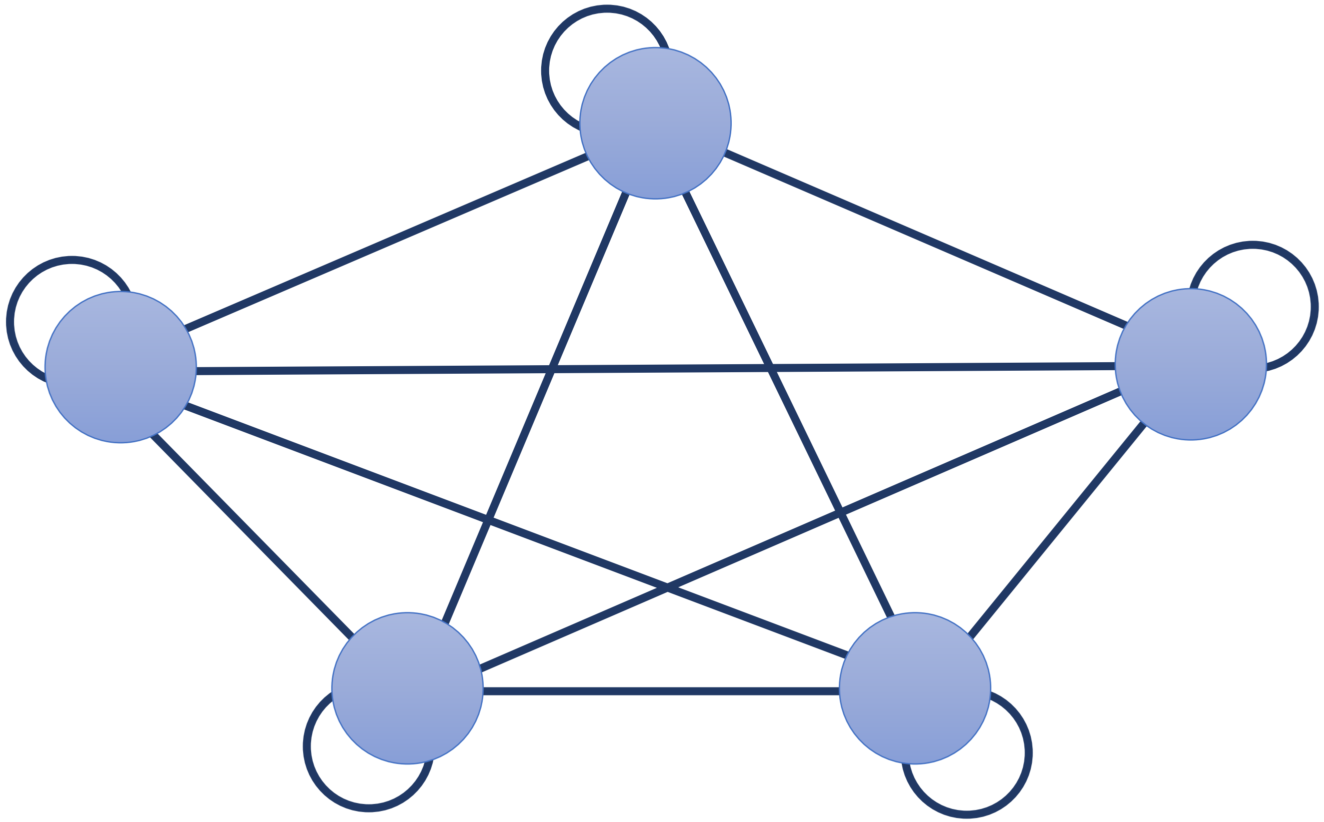


SwiGLU: Gated MLPs

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

Transformers as Graph NNs

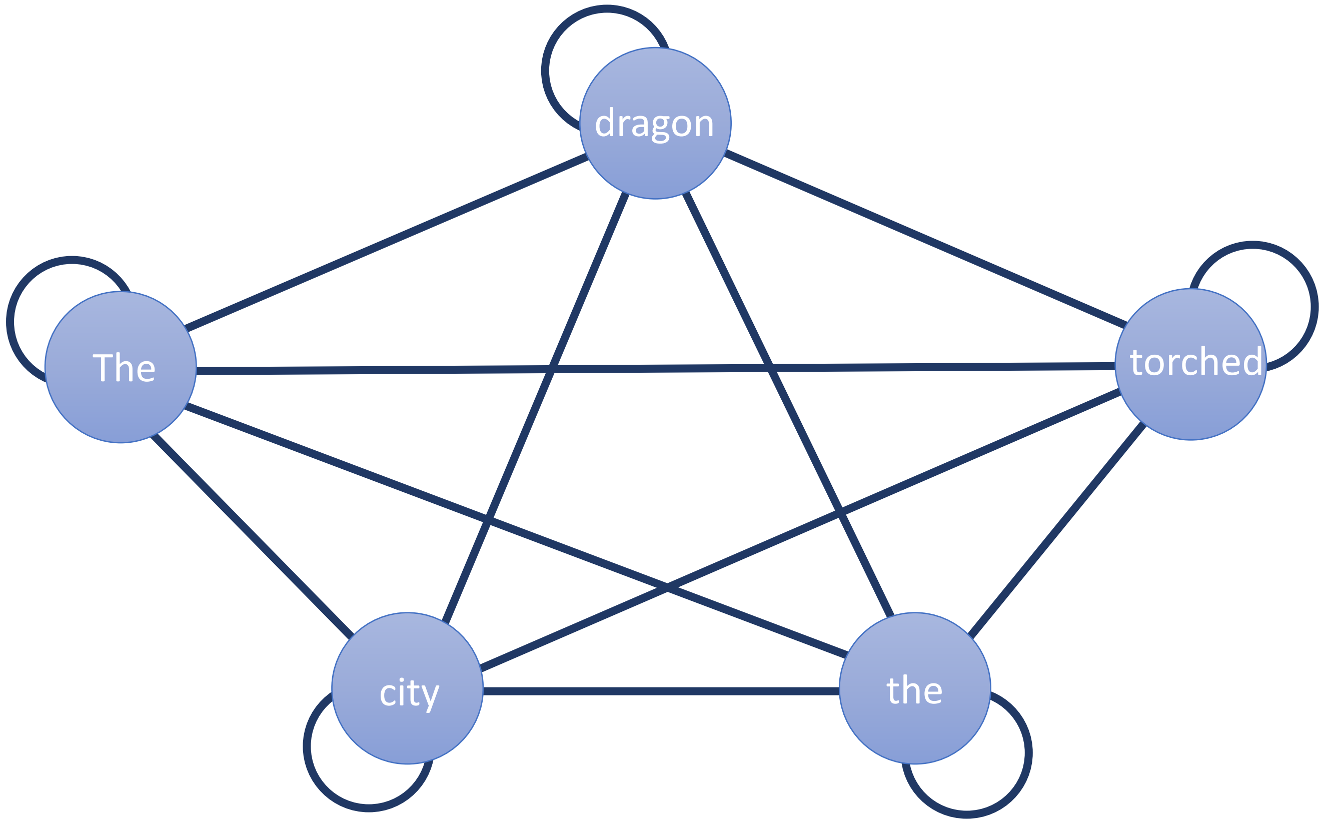
Claim: Transformers are just graph convolutions over dense graphs



- Each node sends a "message" to all its neighbors
- Nodes can weight their input messages based on features from the sender and receiver

Transformers as Graph NNs

Claim: Transformers are just graph convolutions over dense graphs



- Each node sends a value vector to all its neighbors
- Nodes can weight their input messages based on the dot product between the query from the sender and key from the receiver

Transformers as Graph NNs

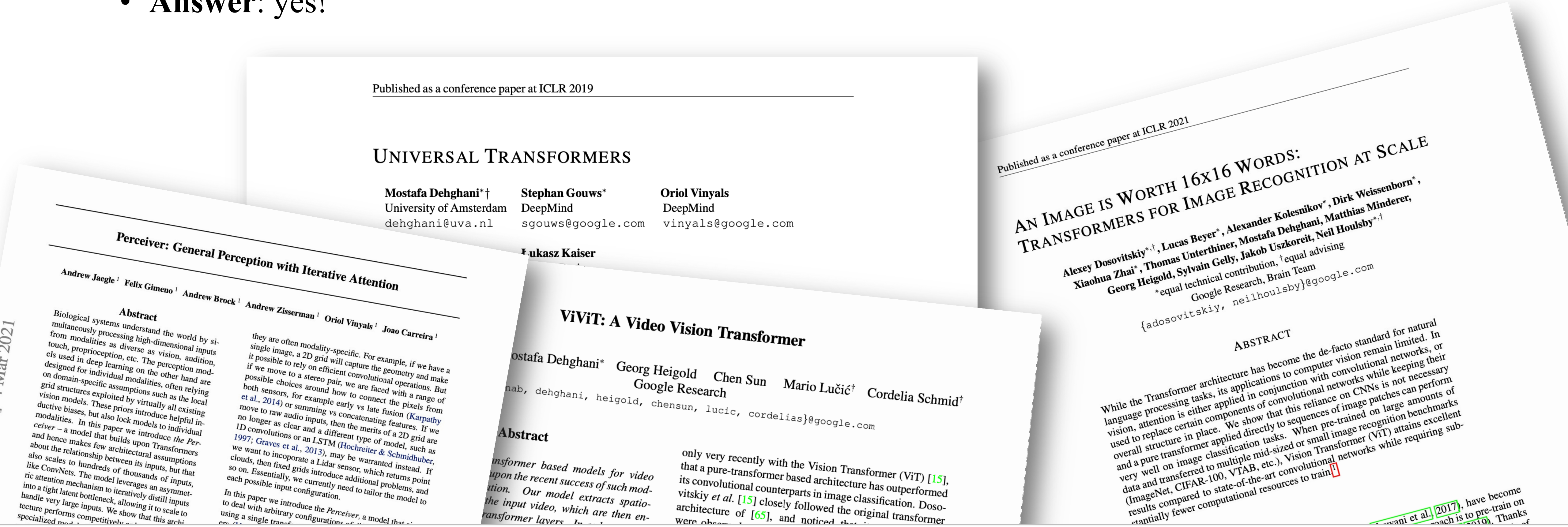
Claim: Transformers are just graph convolutions over dense graphs

Implications:

- Positional encoding necessary as self-attention considers input as graph and not as sequence
- Long-term dependencies not an issue as distance is equal among all words
- Dense graph has N^2 edges
  ⇒ Graph sparsification based on syntax trees etc. corresponds to masking
- Self-attention can be used for permutation-equivariant/-invariant tasks
  - Data like sets, graphs, etc.

Transformers as Graph NNs

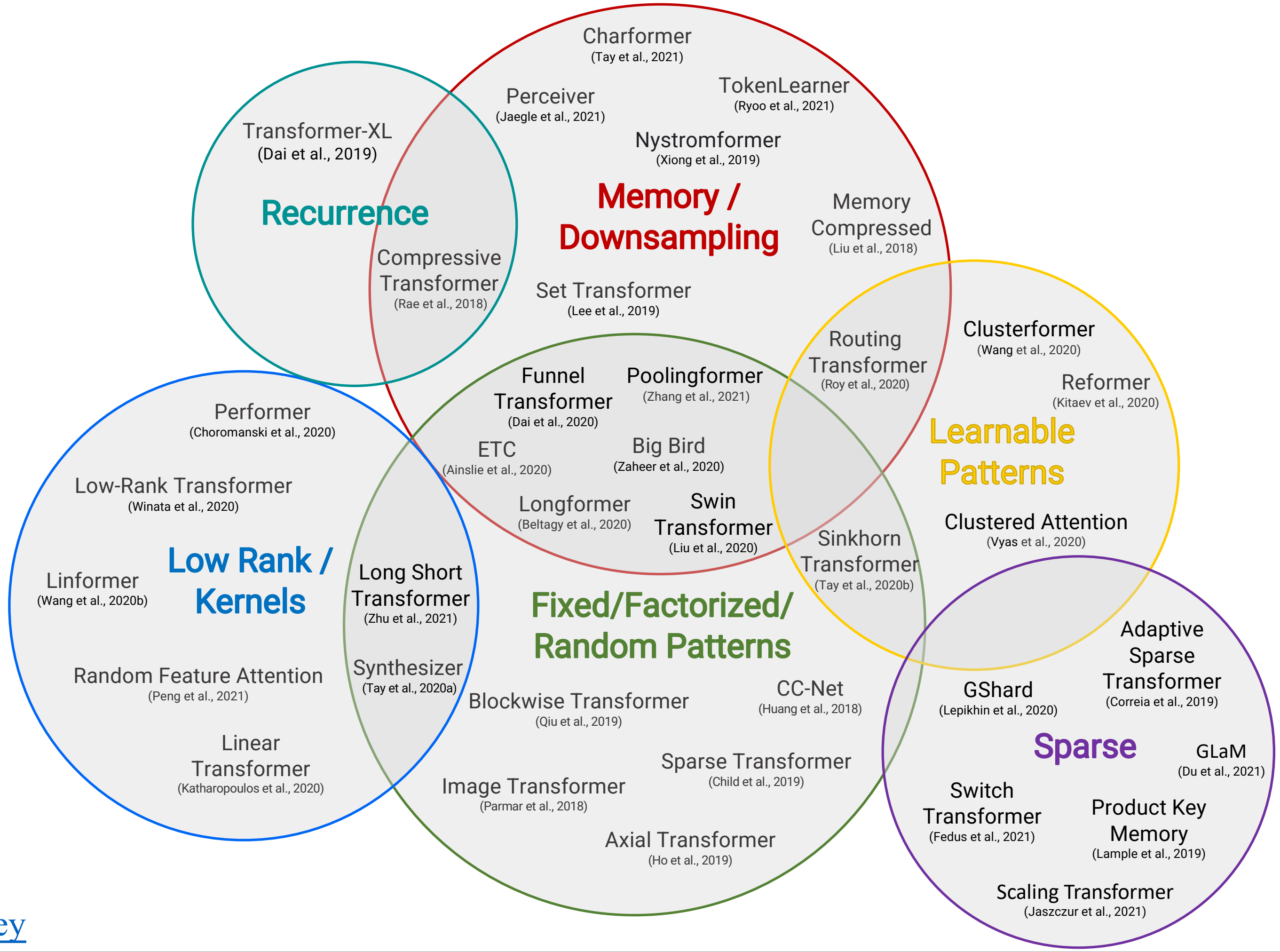
- If we can view sentences as graphs, can we also view other data structures as graphs?
- **Answer**: yes!



Efficient Transformers

- Memory cost of Transformers scales quadratically with sequence length. Can we be more efficient?

- Limiting field of view for attention
  - Learning hard attention mask
- Using memory tokens that can access the full sequence, others only the memory
- Mathematical rewriting to kernels or using low-rank approximations
- Segment-based recurrence
- Sparsely using sets of parameters



Credit: Tay et al., 2022: Efficient Transformers: A Survey

Conclusion

Four main attention mechanisms:

1. **Aggregation**: compressing sequence to single feature vector, pooling
   Applications: creating sentence representations
2. **Encoder-Decoder attention**: allowing the decoder to take a second look at the input based on the current word.
   Applications: any Seq2Seq task like Machine Translation, Summarization, Dialogue Modeling
3. **Cross-Attention**: comparing two sequences on word-level.
   Applications: Natural Language Inference, Question-Answering
4. **Self-Attention**: message passing among words within a sentence or document.
   Applications: stand-alone architecture for almost any task
   - Transformers constitute current state-of-the-art, but don't forget about RNNs!
   - Self-attention views sentence as graph, not as sequence

Useful blogposts

- Transformers and Multi-Head Attention, a Jupyter notebook tutorial about the details on the Transformer architecture and its implementation in PyTorch. Contains a written-down version of many parts of this presentation
- Google AI Blog explaining the transformer paper.
- The Illustrated Transformer, nice illustrations and detailed explanation of self-attention and the transformer model.
- The transformer family, review of many different transformer variants
- A Survey of Long-Term Context in Transformers, reviews transformer variants with the goal of more efficient models for long sequences
- Attention? Attention!, explaining different forms of attention. Takes a different perspective and does not only focus NLP
- Attention and Augmented Recurrent Neural Networks, although from 2016, gives a nice review of attention before transformers, especially with insights to Machine Translation. Written by Chris Olah who also wrote the most cited LSTM blog.
- Transformers III: Training, explains all the training difficulties (warmup, gradient vanishing, etc.) in detail

Useful papers

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

Papers extending the original Transformer architecture

- Dehghani, Mostafa, et al. "Universal transformers." arXiv preprint arXiv:1807.03819 (2018). *Combining Transformers with recurrence over layer depth, making it Turing complete. Especially useful for complex reasoning tasks like question-answering.*
- Kitaev, Nikita, et al. "Reformer: The Efficient Transformer" arXiv preprint arXiv:2001.04451 (2020). *Making transformers more memory efficient by local-sensitive hasing and using reversible layers to re-calculate activations during backpropagation.*
- Sukhbaatar, Sainbayar, et al. "Adaptive Attention Span in Transformers" arXiv preprint arXiv:1905.07799 (2019). *Allowing the attention layers to learn the optimal receptive field/span to reduce memory footprint and computational time.*

Useful papers

Papers about training details – general tips

- Popel, Martin, Bojar, Ondrej, "Training Tips for the Transformer Model" (2018). *Review of a large hyperparameter grid search and sharing insights.*
- Dodge, Jesse et a., "Fine-Tuning Pretrained Language Models" (2020). *Review of hyperparameters for finetuning large transformer-based language models.*

Useful papers

Papers about training details – Layer Normalization

- Shen, Sheng, et al. "Rethinking Batch Normalization in Transformers." arXiv preprint arXiv:2003.07845 (2020). *Analyzing Batch normalization for language and proposing alternative to Layer normalization*
- Xu, Jingjing, et al. "Understanding and Improving Layer Normalization." Advances in Neural Information Processing Systems. 2019. *Analyzing gain and bias in Layer normalization and proposing alternative*
- Xiong, Ruibin, et al. "On Layer Normalization in the Transformer Architecture." arXiv preprint arXiv:2002.04745(2020). *Analyzing and comparing PreNorm vs PostNorm*

Useful papers

Papers about training details – investigating issues of Transformer architectures

- Wang, Hongyu, et al. "DeepNet: Scaling Transformers to 1,000 Layers." arXiv preprint arXiv:2203.00555 (2022). *Discusses initializations for training very deep Transformers with up to 1000 layers.*
- Liu, Liyuan, et al. "Understanding the difficulty of training transformers." EMNLP (2020). *Discusses variance amplifications of small parameter changes in early layers.*

Q&A