Attention Mechanism in Neural Networks

UvA, Advanced Topics in Computational Semantics

Phillip Lippe

16/04/2020

Practical things

- If you have a question, simply raise your hand
- If my connection breaks, let me know in the chat
- If I ask a question, feel free to turn on your audio and answer
- If I ask simple yes/no questions, you can also answer by reactions

Today’s learning goals

- What is “attention”?
- What different kind 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

What is attention?

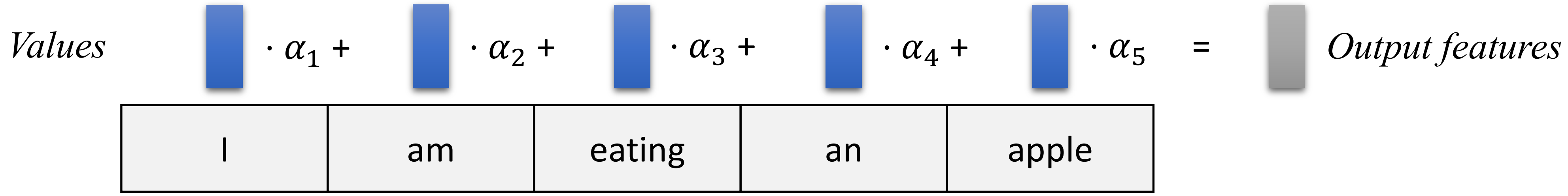
A weighted average of (sequence) elements with the weights depending on an input query.

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

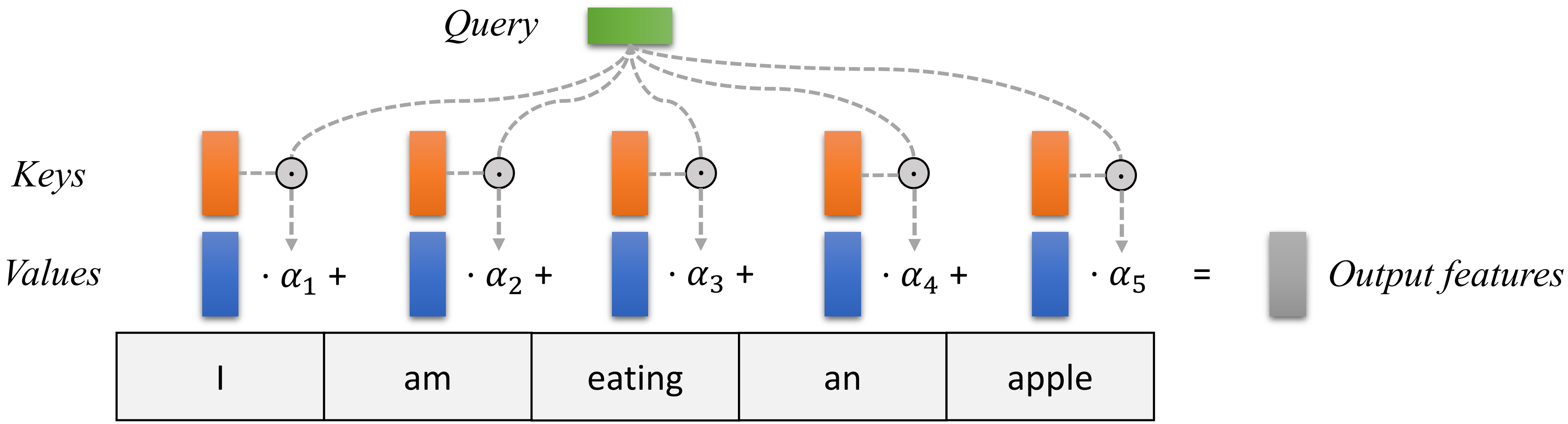
$$\alpha_i = \frac{\exp(f_{attn}(\mathrm{key_i}, \mathrm{query}))}{\sum_j \exp\left(f_{attn}(\mathrm{key_j}, \mathrm{query})\right)} \qquad \mathrm{out} = \sum_i \alpha_i \cdot \mathrm{value}_i$$

What is attention?

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

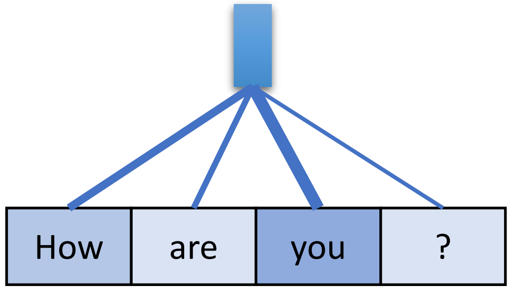
$$\alpha_i = \frac{\exp(f_{attn}(\mathrm{key_i}, \mathrm{query}))}{\sum_j \exp\left(f_{attn}(\mathrm{key_j}, \mathrm{query})\right)} \qquad \mathrm{out} = \sum_i \alpha_i \cdot \mathrm{value}_i$$

Example

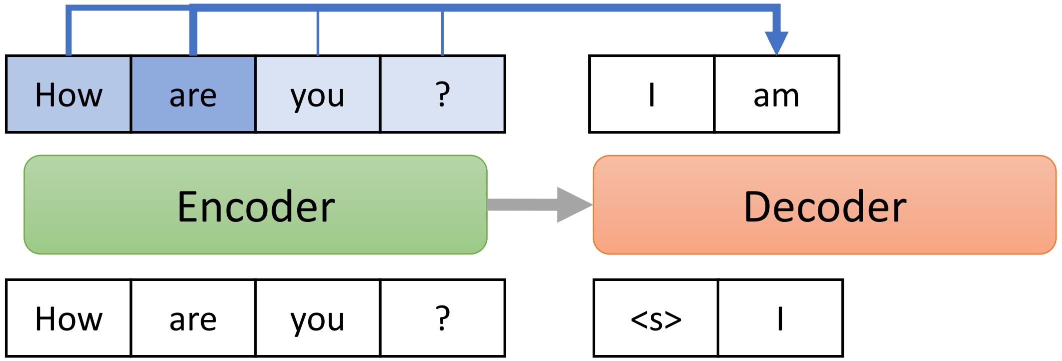


Attention mechanisms

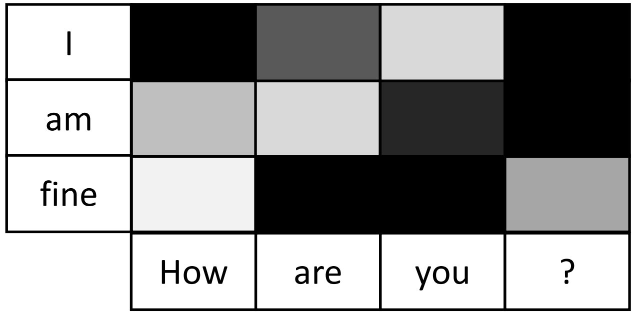
Aggregation



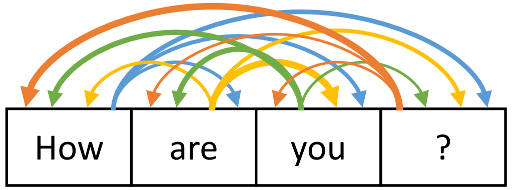
Encoder-Decoder



Cross-Attention



Self-Attention



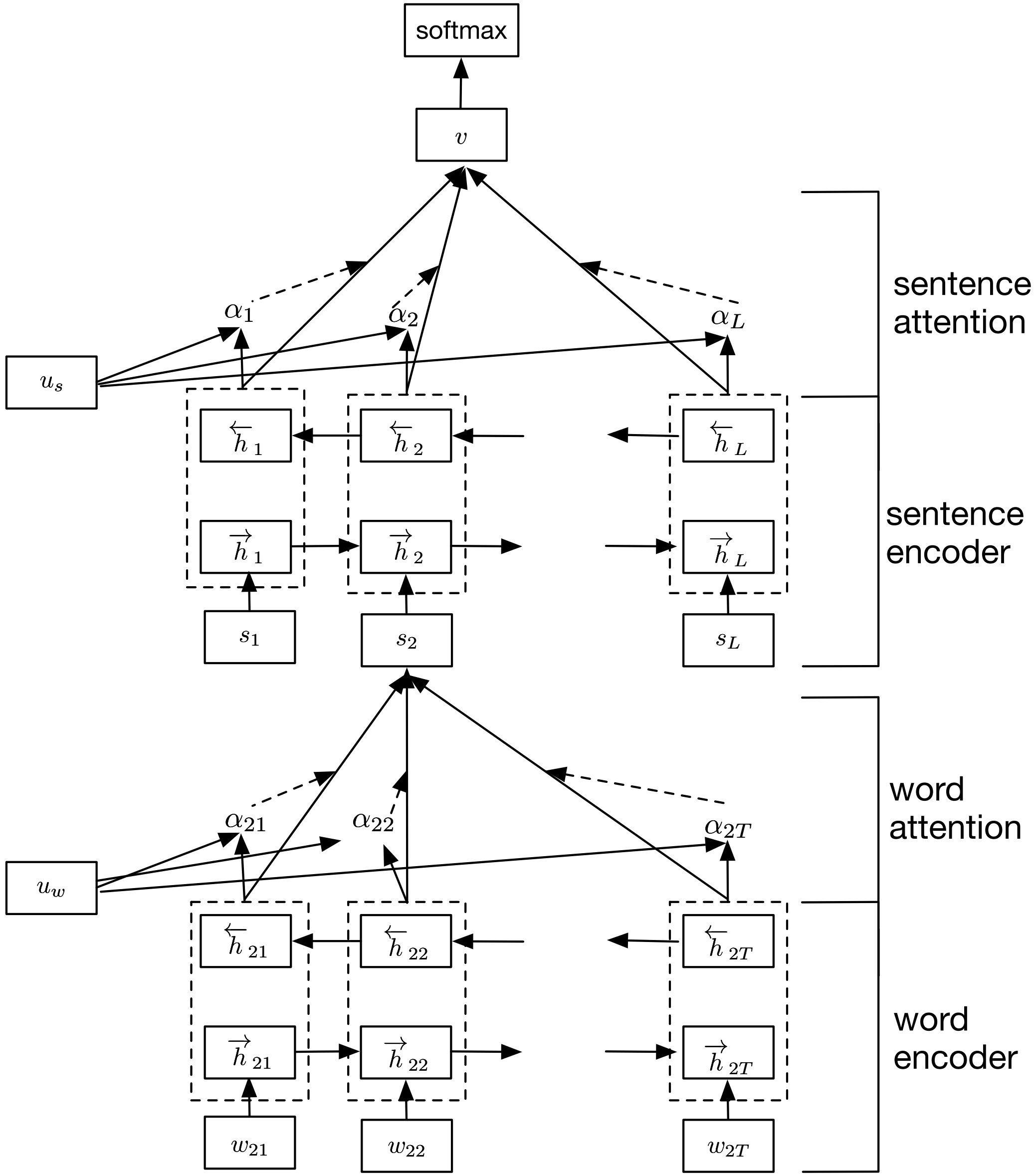
Aggregation

Recap NLP1: **Hierarchical Attention Network**

- 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

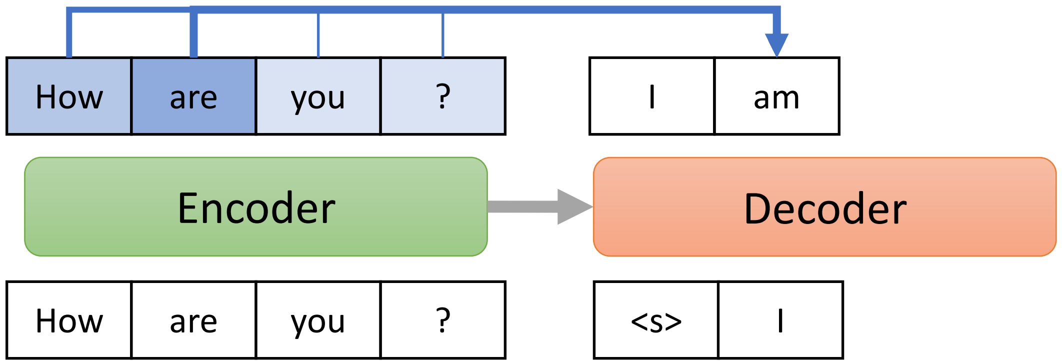


Credit: Yang et al., "Hierarchical Attention Networks for Document Classification" (2016)

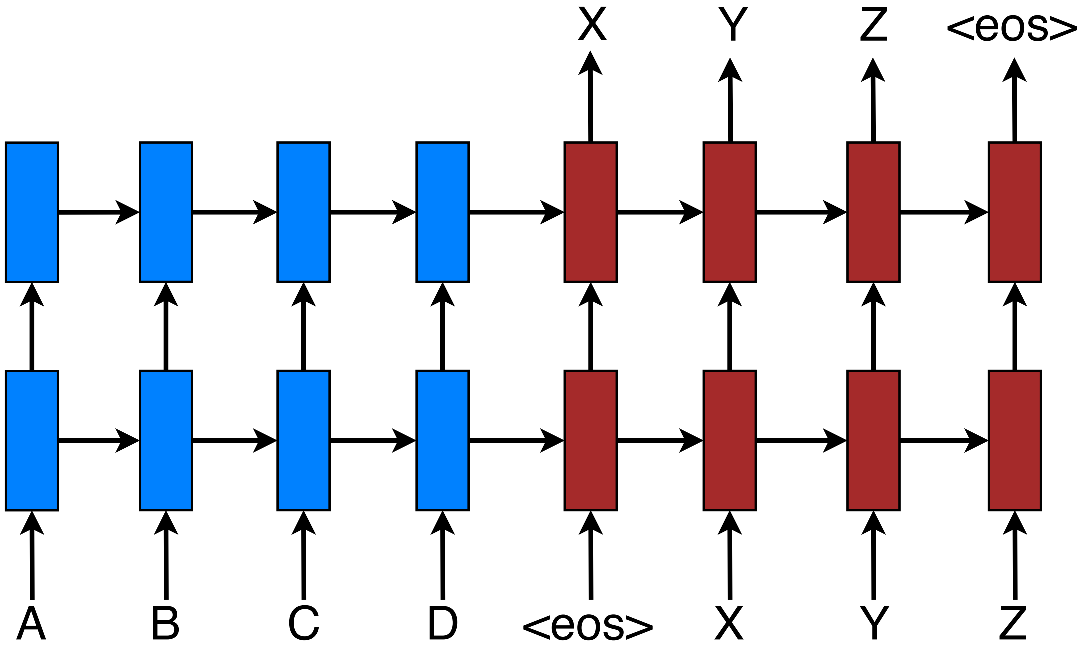
<u>Formula legend</u>
h_{it} - hidden state of t-th word in the i-th sentence
u_w - learned query vector

Encoder-Decoder Attention

- General setup
- Global vs Local Attention
- Applications



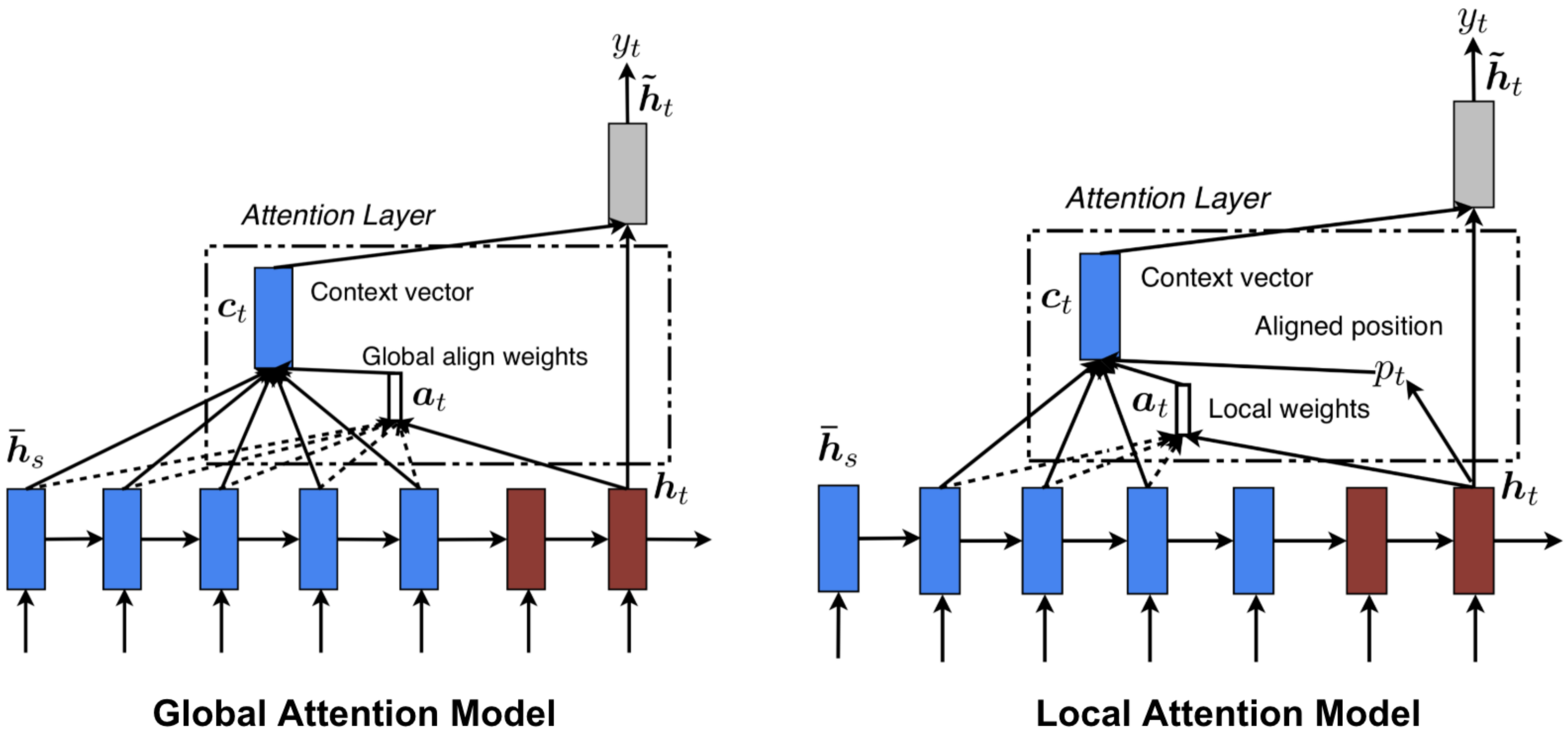
Encoder-Decoder



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

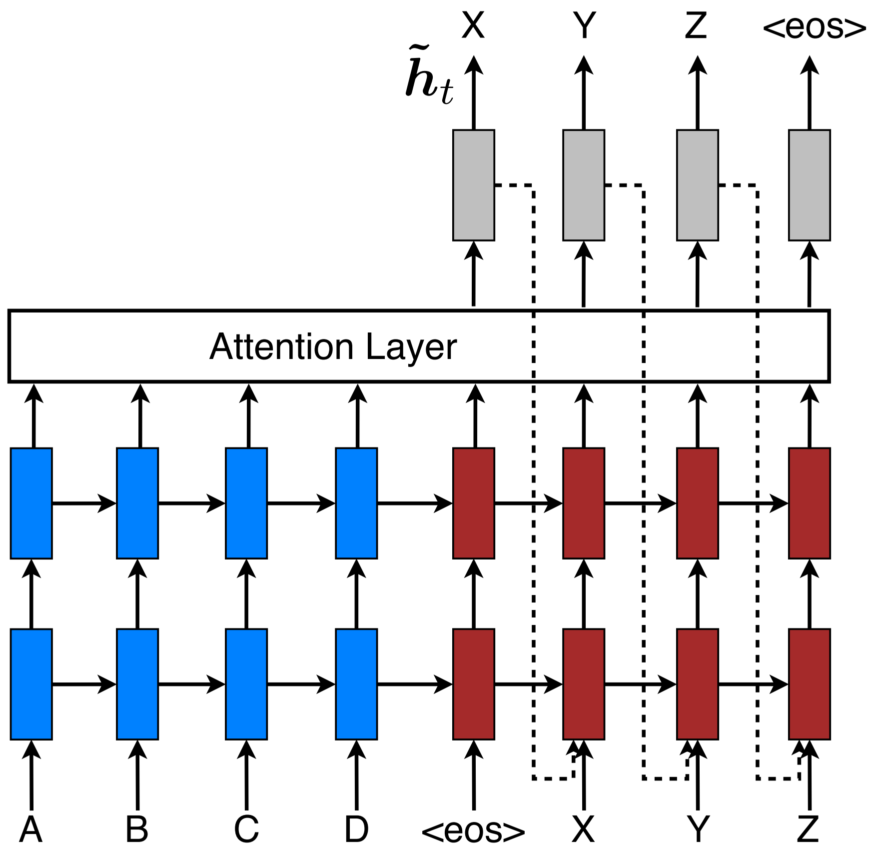
- Suffering from long-term dependencies
- Encoder output must summarize the whole sentences with all its details
- Especially difficult if there are many different possible outputs

Global vs Local Attention



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

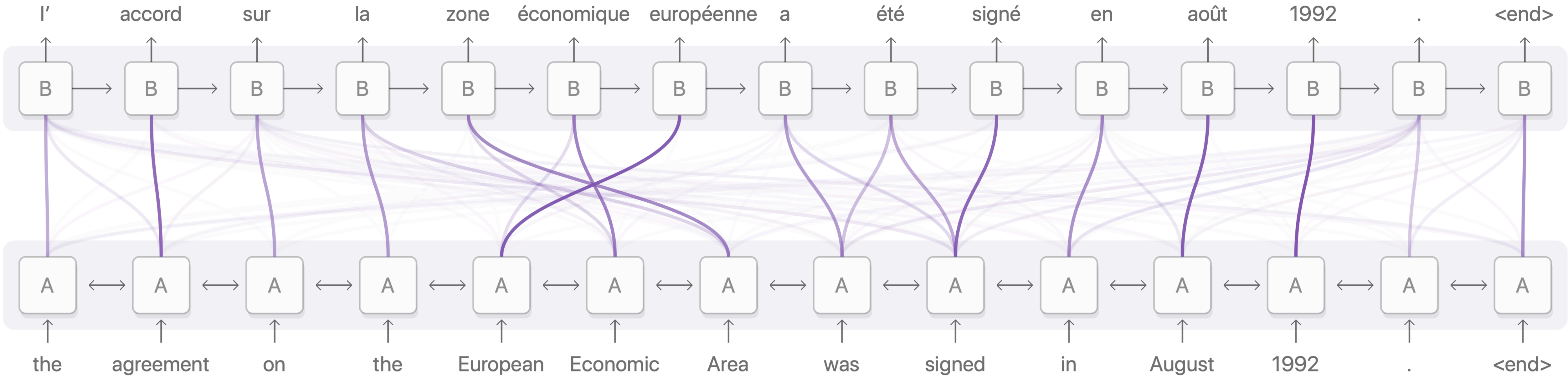
Encoder-Decoder with attention



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

- Attention layer enriches token-level information
- Alternative setup: attention layer using cell state and enriching input information to the RNN instead of output information

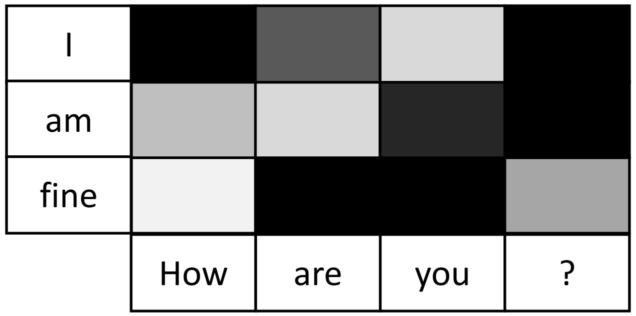
Applications – Machine Translation



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

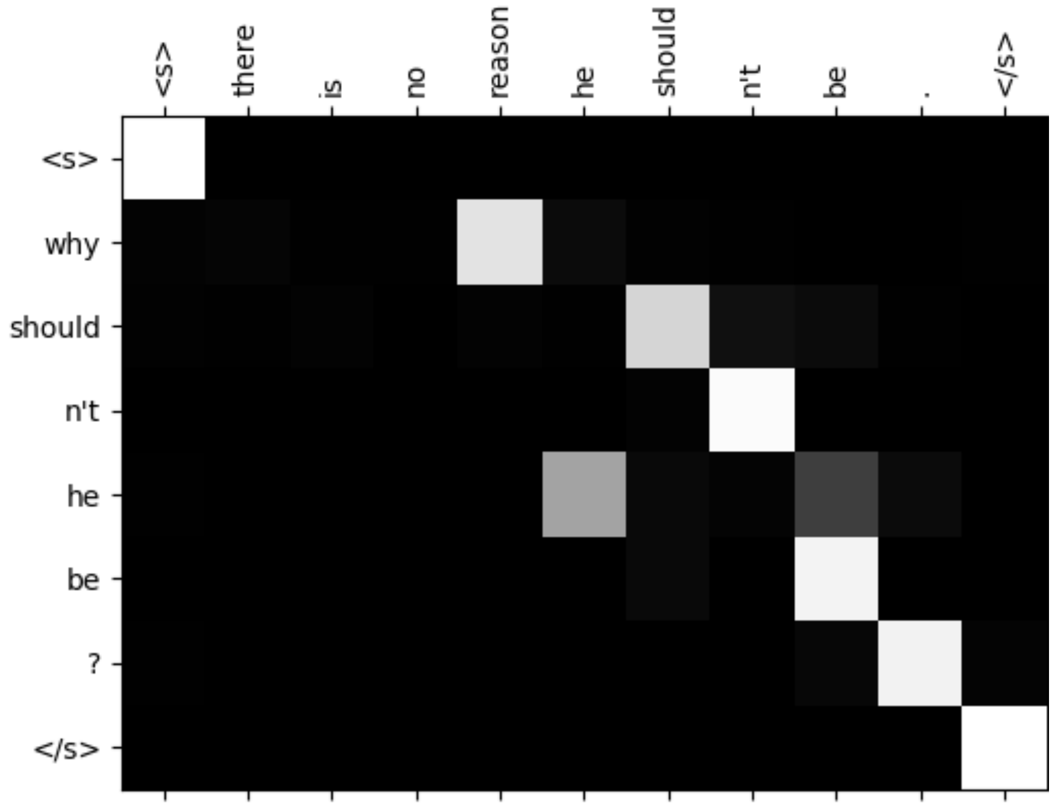
Cross-Attention

- General setup
- Applications



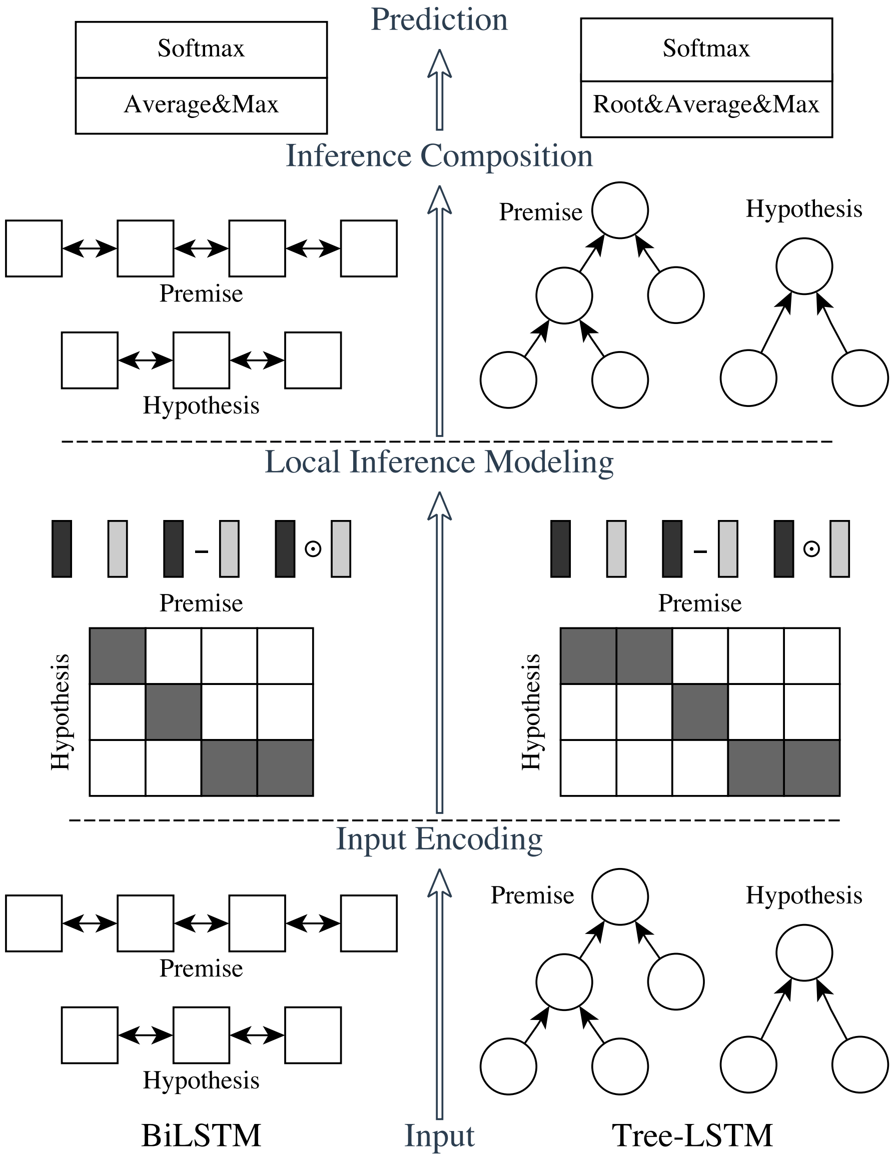
Cross-Attention

- 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



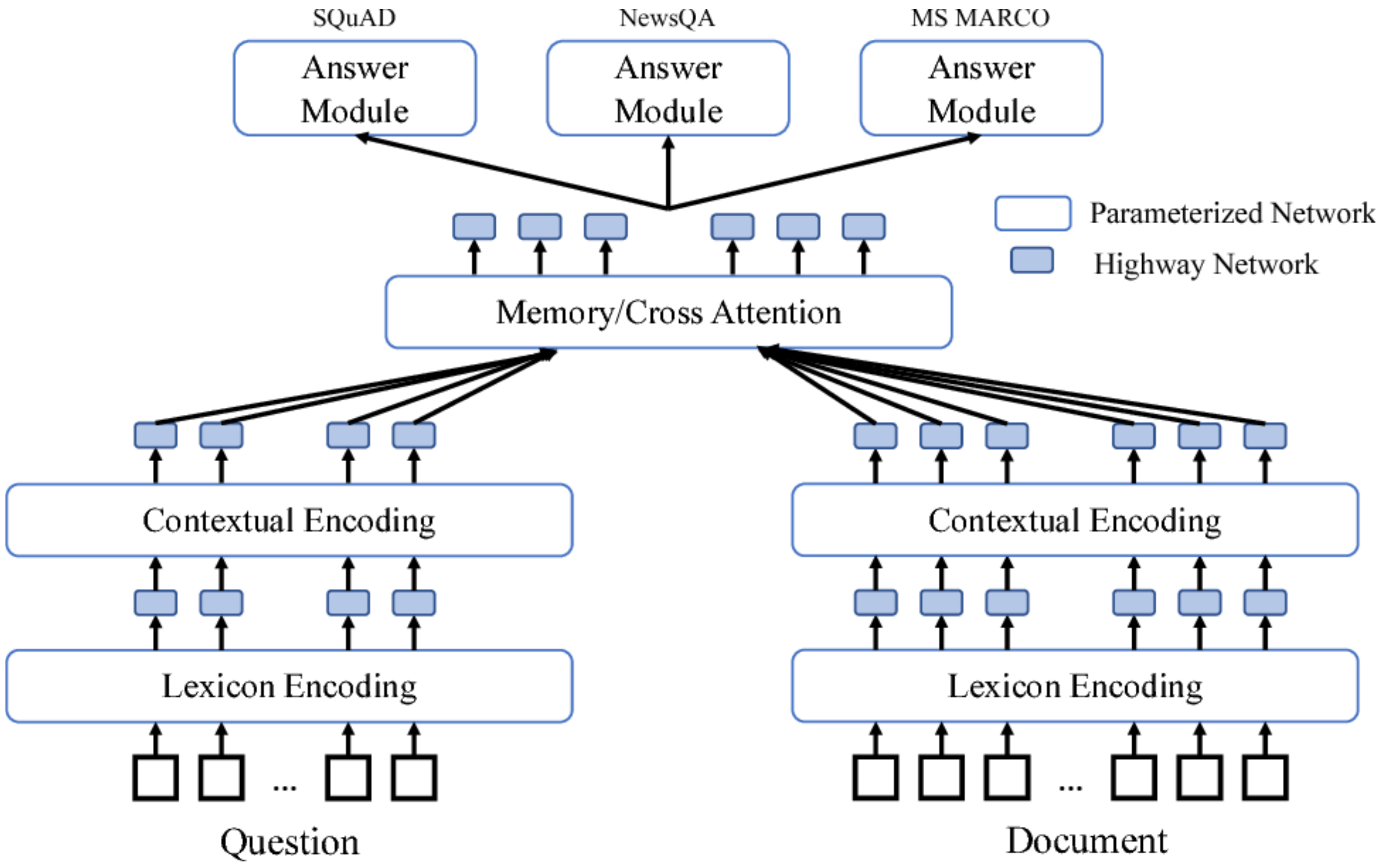
Applications – NLI

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



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

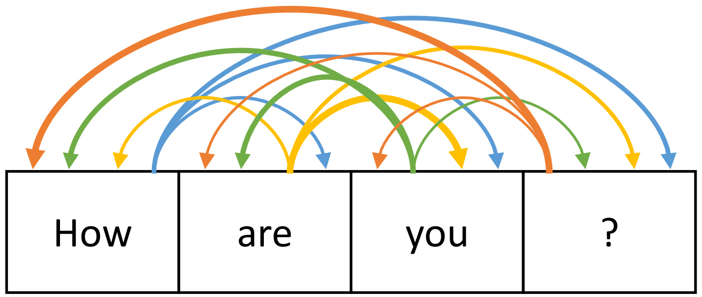
Applications – Question-Answering



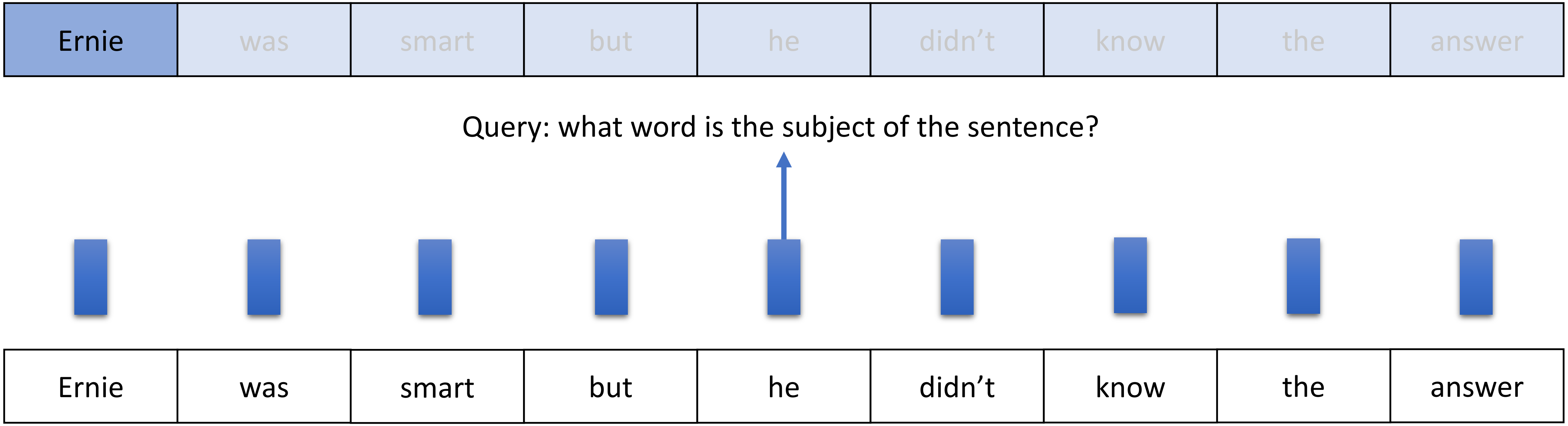
Credit: Xu et al. „Multi-Task Learning for Machine Reading Comprehension.“ (2018)

Self-attention

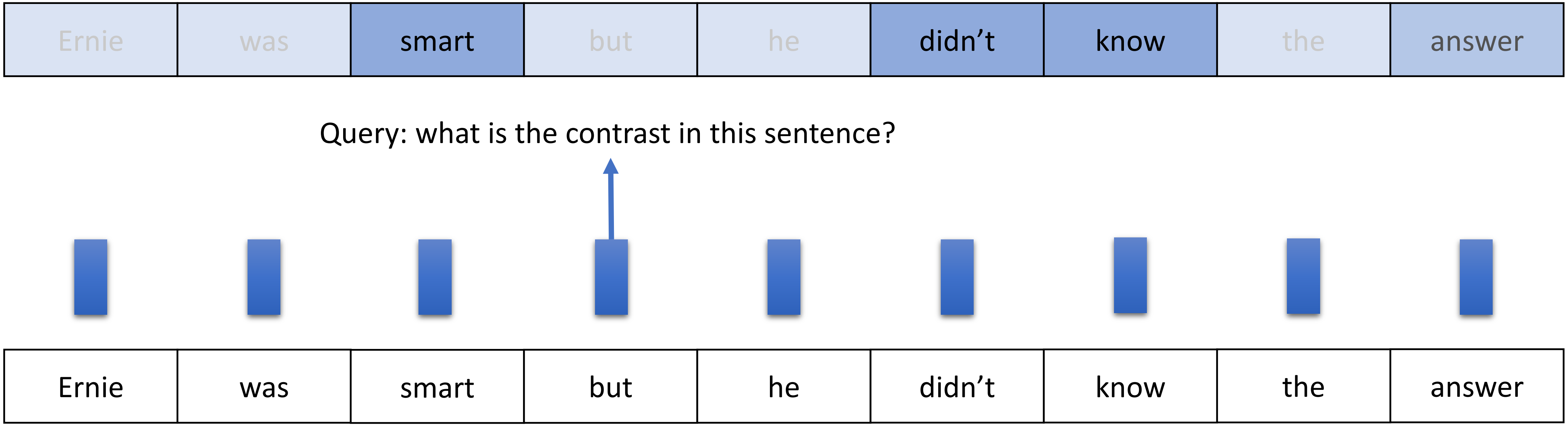
- Intuition and Motivation
- Self-attention layer
- Transformer architecture
- (Optional) Optimization issues and training tips
- (Optional) Transformers as Graph Neural Network



Intuition

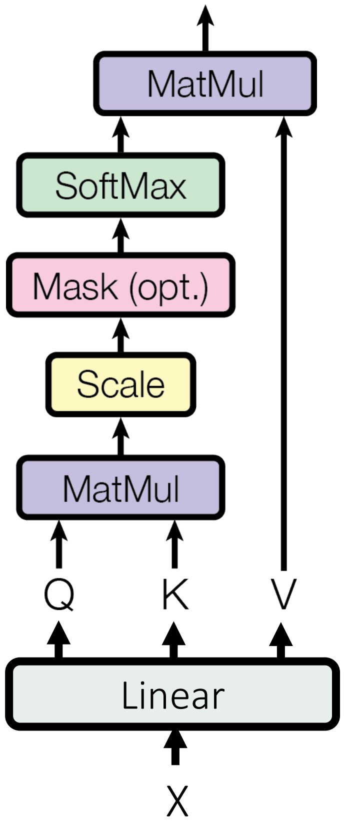

Ernie
was
smart
but
he
didn't
know
the
answer
Query: what word is the subject of the sentence?
Ernie
was
smart
but
he
didn't
know
the
answer

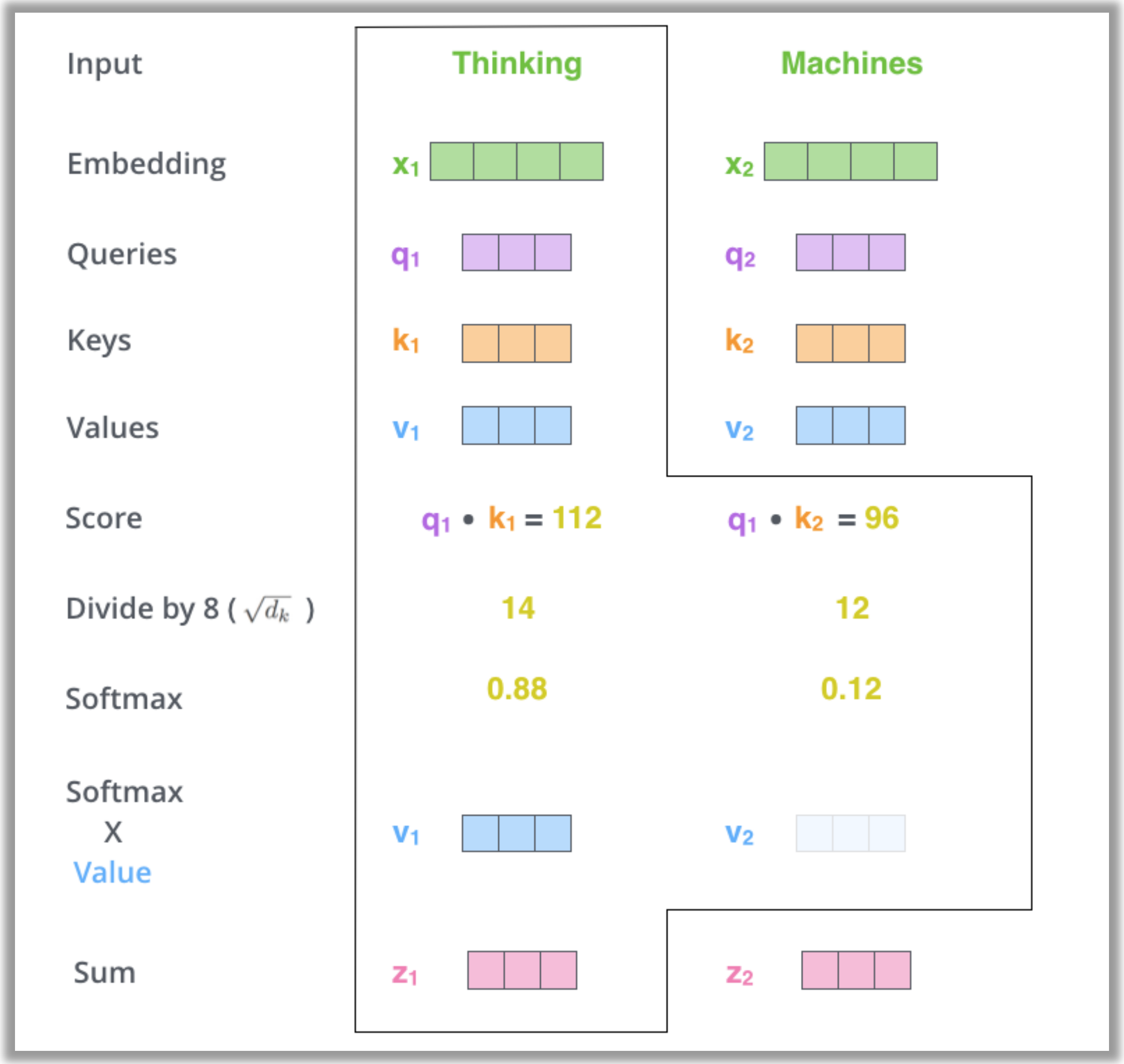
Intuition


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 layer

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



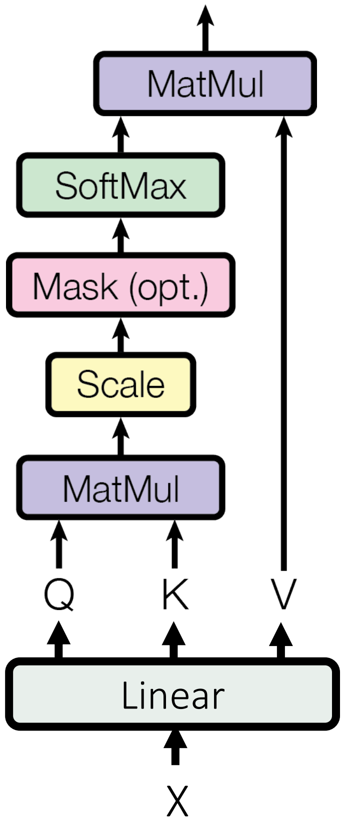


Credit: Alammar, Jay: The Illustrated Transformer, http://jalammar.github.io/illustrated-transformer/

Formula legend
d_k - hidden size of key/query

Self-attention layer

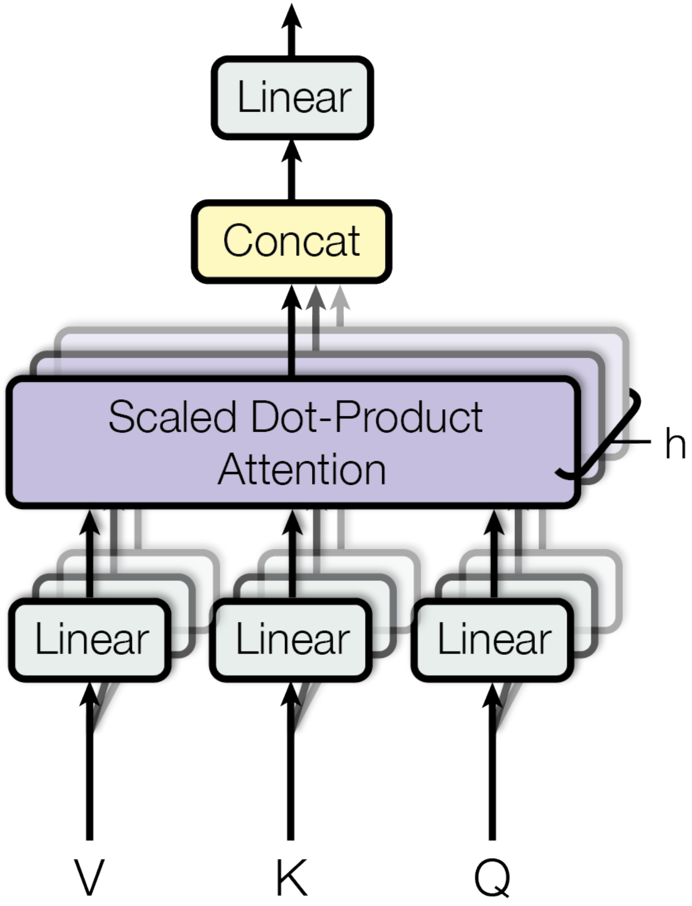
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



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
- High initial values significantly harm gradient flow

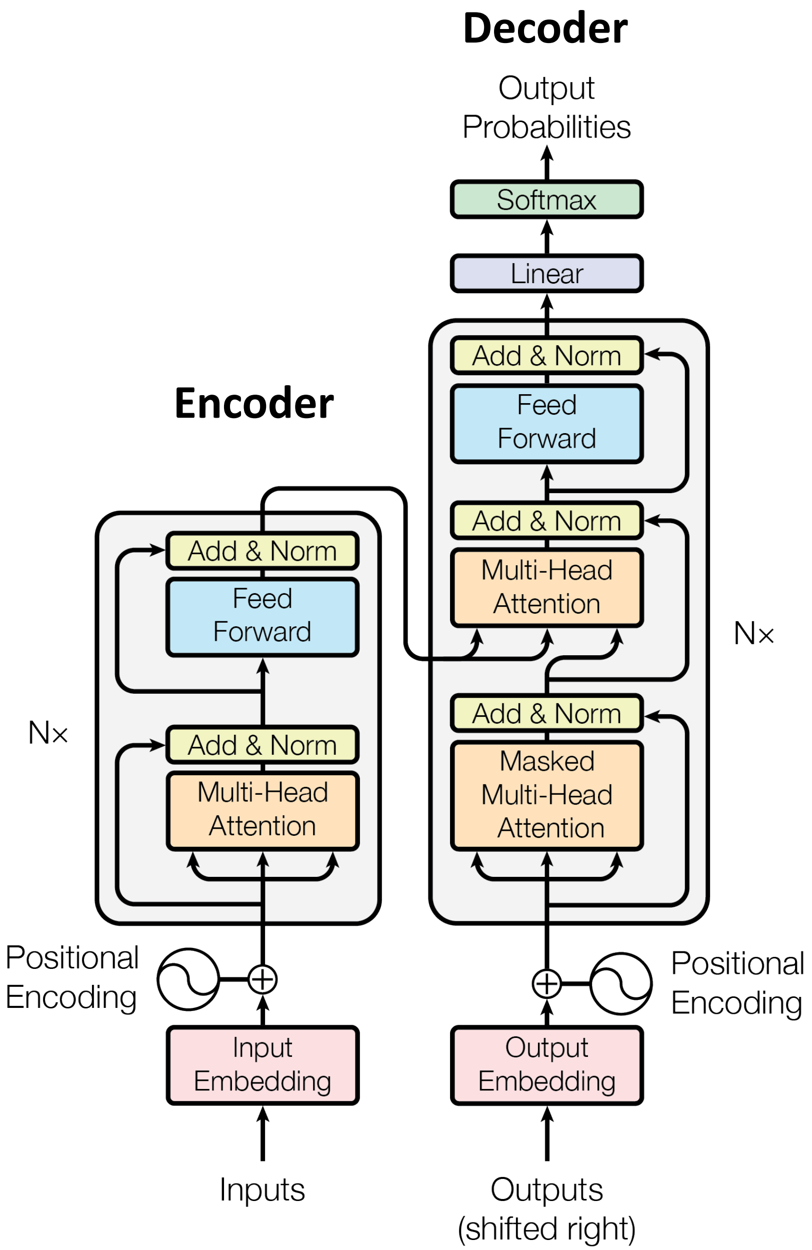
Multi-Head self-attention



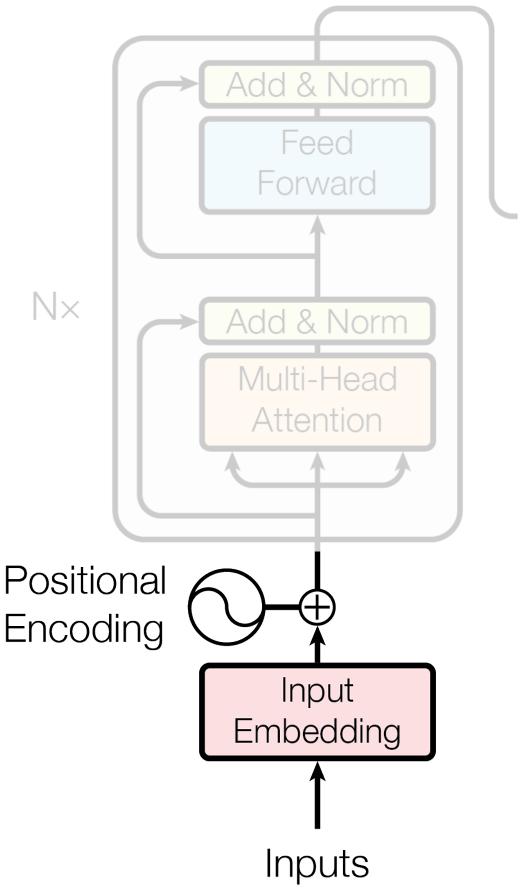
- Single head offers only one perspective on the data
  ⇒ Often not enough, can harm gradients again
- Performing several self-attentions in parallel increases flexibility and non-linearity/complexity
- Output projection to scale down the concatenation if necessary

Transformer architecture

- Transformer has an encoder-decoder structure
- Both parts consists 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

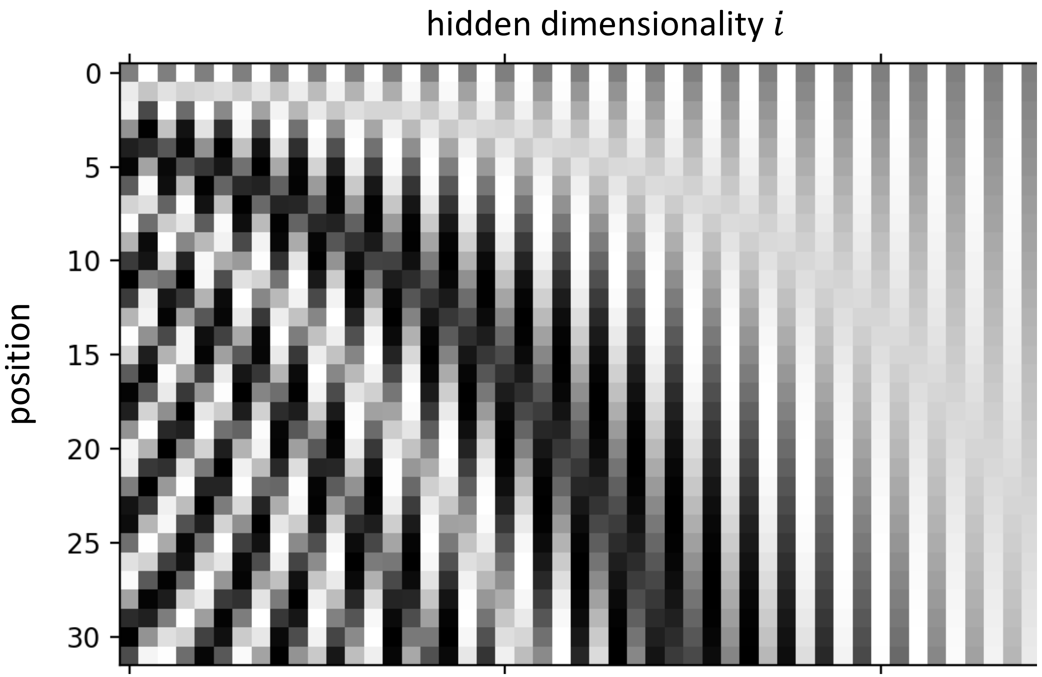
- Easier adaptation to unseen words in the training corpus
- Sharing of common word parts ("-ing", "re-", etc.)

Positional embeddings

- Self-attention layers do not encode position, but view the input as **set** (permutation invariant).
- Sinusoidal positional encoding added to embeddings

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

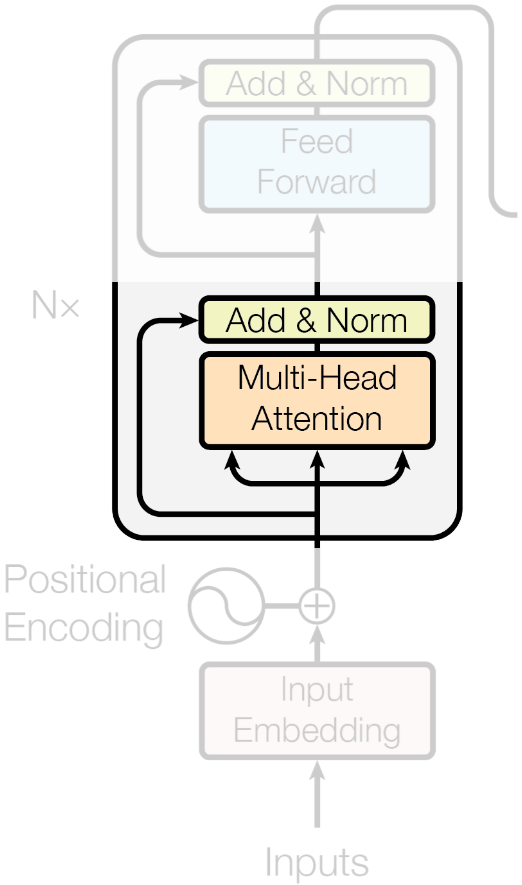
- Scales to unseen lengths
- Encodes distance between positions



Credit: Weng, Lilian: The Transformer family

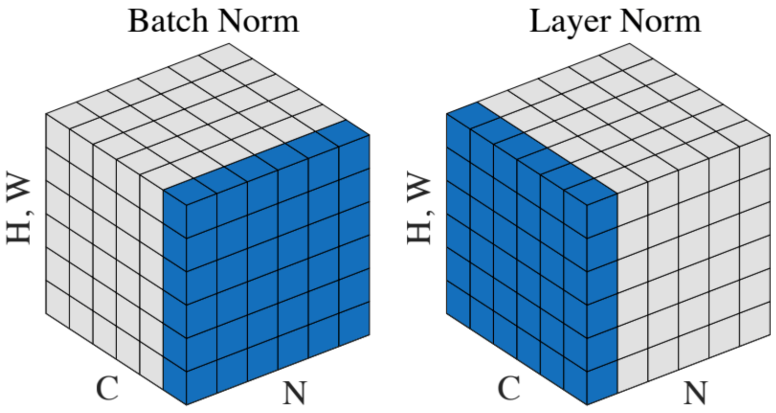
Formula legend
d_{model} - hidden size of embedding
i – index over the hidden dimension
pos – position of word in sentence

Transformer - Encoder



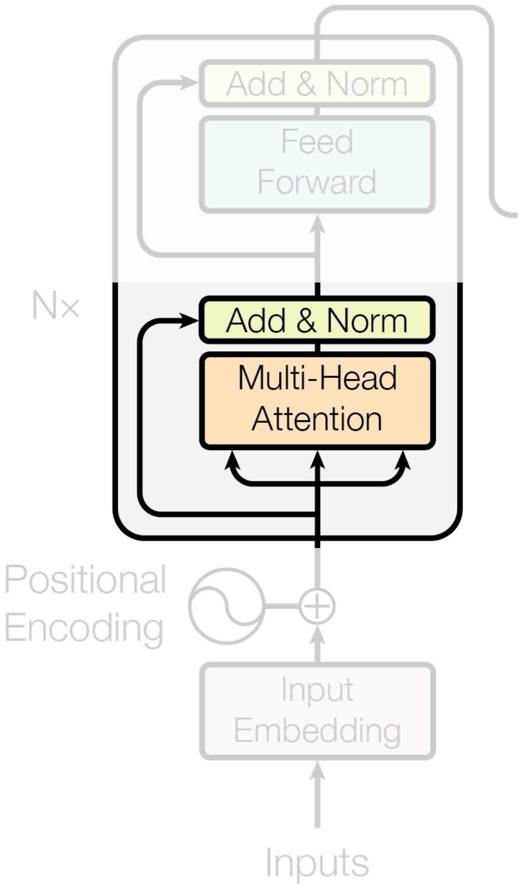
- Residual connection combined with Layer normalization

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



Credit: Kurita, Keita, [An Overview of Normalization Methods]

Transformer - Encoder



- Residual connection combined with Layer normalization

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

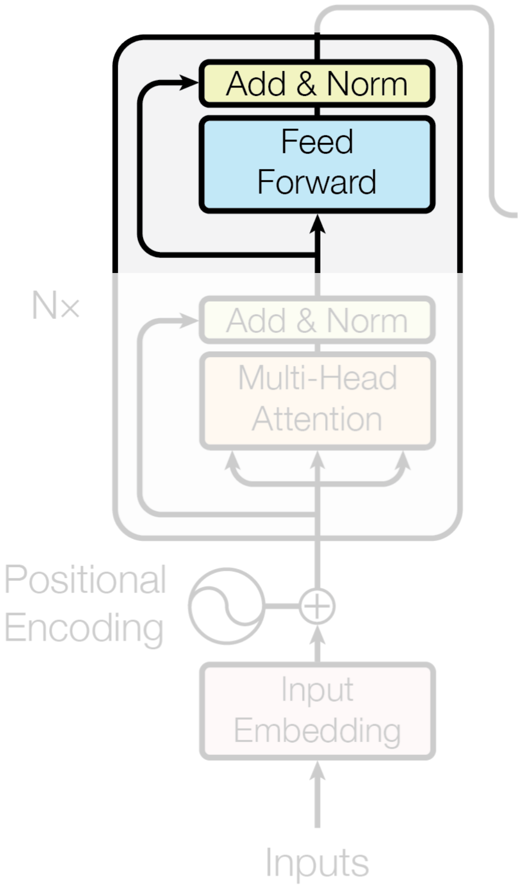
Why do we need residual connections?

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

Why do we need Layer normalization?

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

Transformer - Encoder



- Point-wise feed-forward network with ReLU activation

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

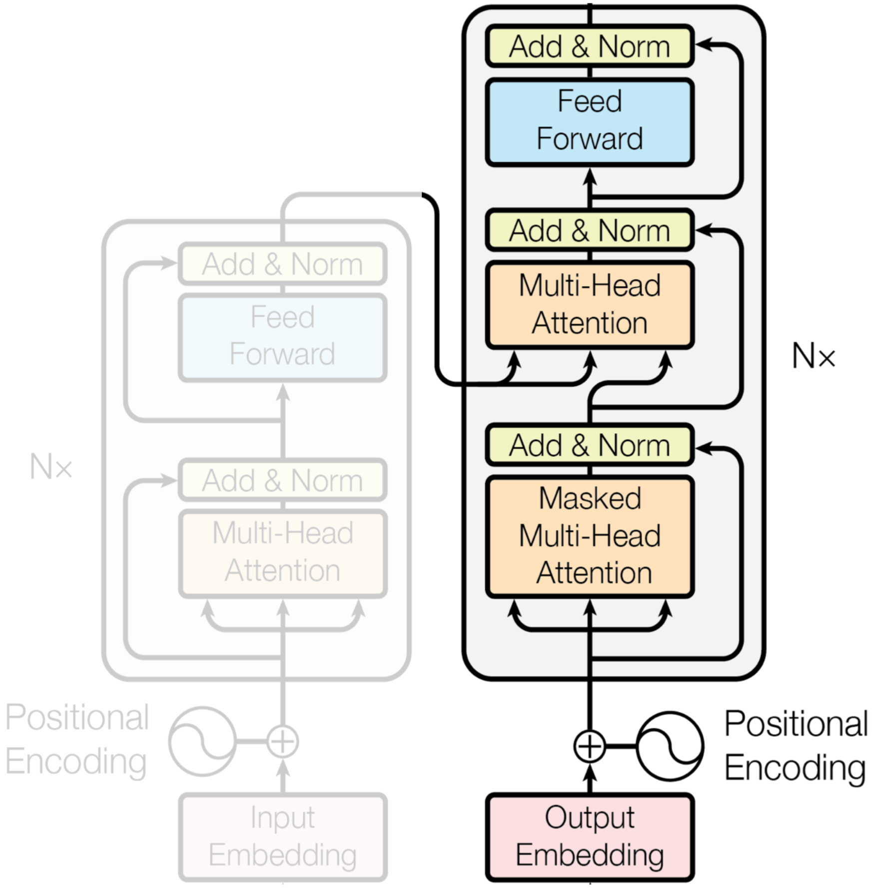
- Adds complexity with classical non-linearity to network
- Inner hidden dimensionality commonly 4-8x larger

Why larger hidden dimensionality instead of deeper MLP?

- Faster computation (can be run in parallel)
- Less parameters
- 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**	$2.3 \cdot 10^{19}$	

Is attention all we need?

Transformers

+ State-of-the-art on most benchmarks
+ Scalable to billions of parameters (Turing-NLG – 17 billion params)
+ Computation in parallel (feedforward network)

– Recurrence needs to be learned
⇒ lots of data required or autoregressive task
– 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

– Does not scale well beyond 5 layers
– Slower to run for long sequences
– Long-term dependencies problematic

Transformers vs RNNs

When to use Transformers? If you…

- have a lot of data
- have a challenging problem
- finetune a pretrained language model
- have strong GPUs with a lot of memory

When to use RNNs? If you…

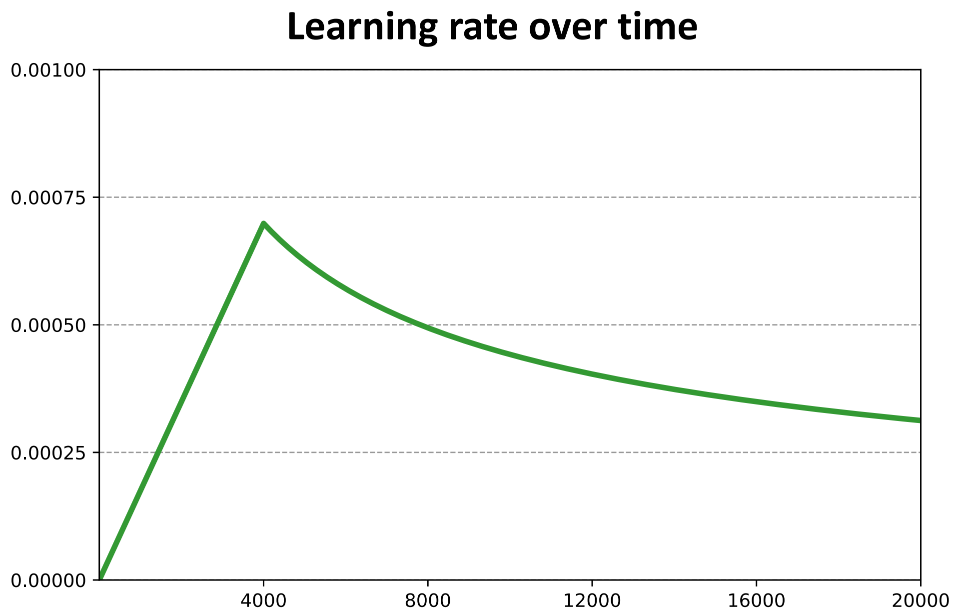
- have limited data
- can make use of pretrained embeddings
- have a strong recurrent bias in the data (i.e. position is important)

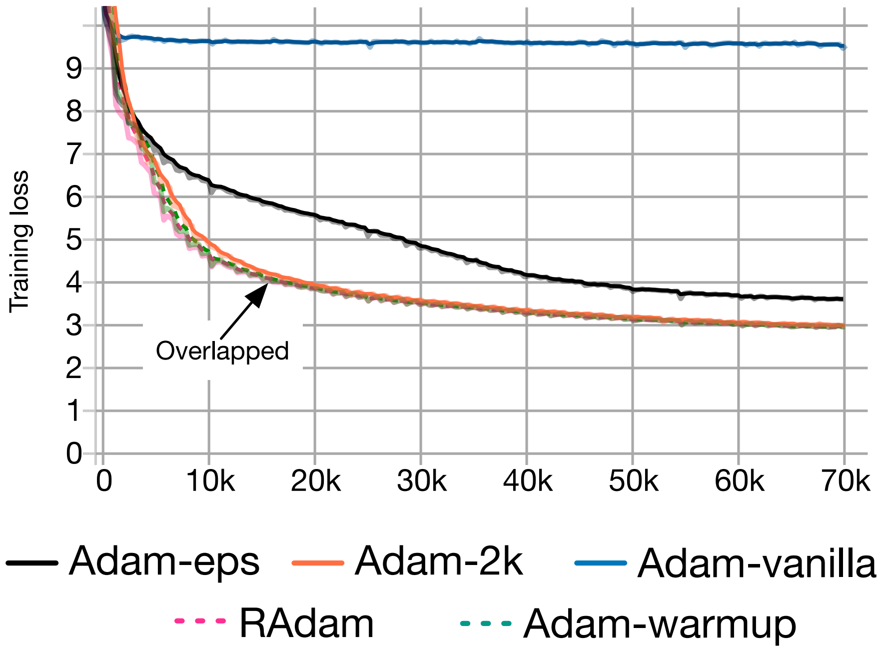
Transformers – Training tips

- Training Transformers can be painful on a single small GPU…
- Use many heads, but not too many. Commonly, 4-16 heads work well
- 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
  - Don't make sentence lengths too different
  - 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, Microsoft or NVIDIA (Lamb, ZeRO)
- BPE vocabulary must be trained on sufficient data. Otherwise it easily overfits

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?

(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

Formula legend

g^t - gradient at iteration t

m – momentum

v – second-order momentum (adaptive lr)

w – weight parameters

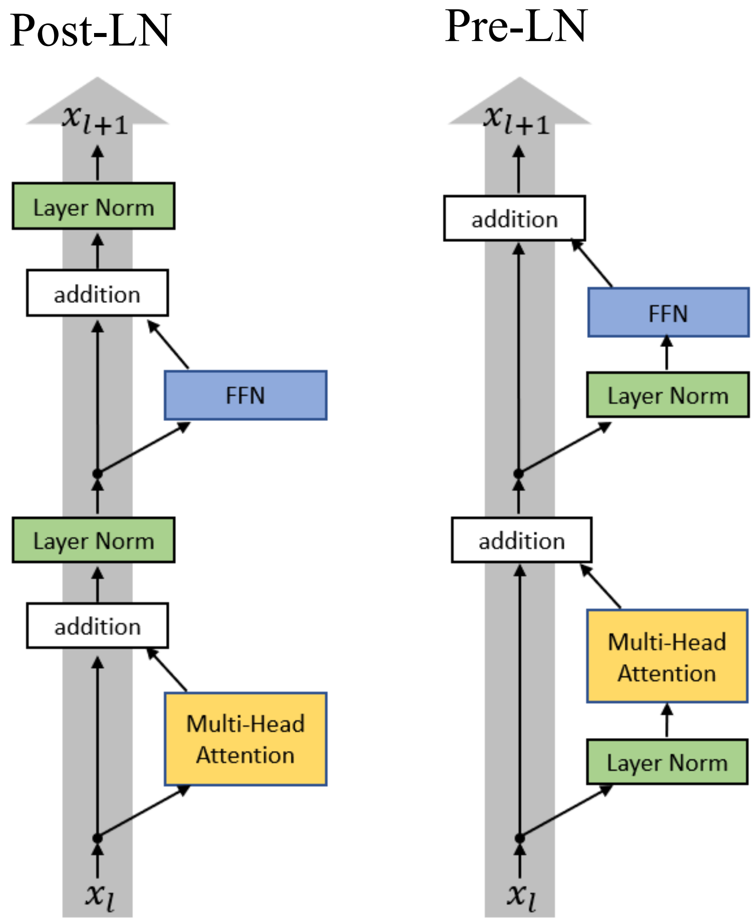
β_1, β_2 - Adam hyperparameters

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
- In short: last FFN and Multi-head attention layer have gradients independent of number of layers, making them sensitive for deep transformers
- Better: use Pre-Layer Normalization
- Even better: use different normalization
  ⇒ Adaptive Normalization
  ⇒ Power Normalization



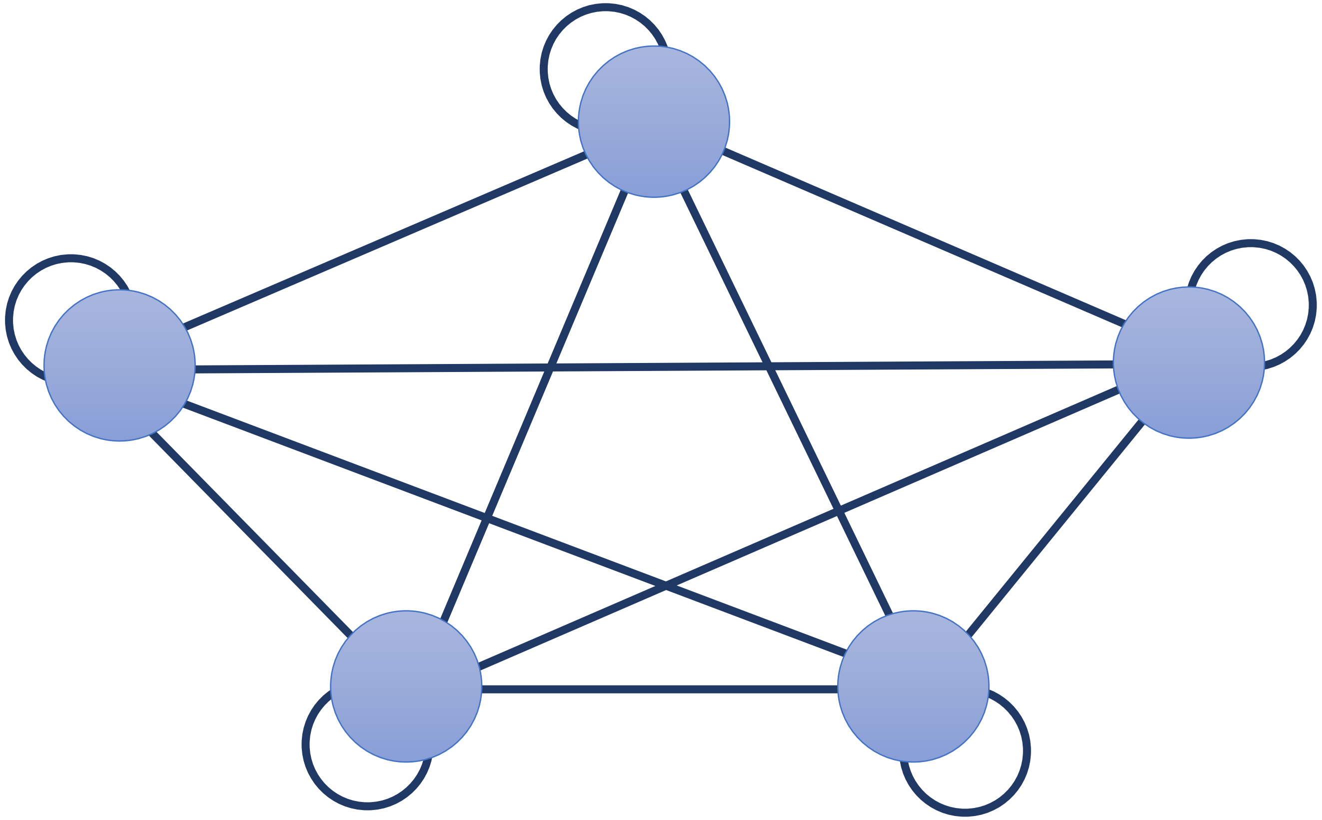
Credit: Xiong et al., "On Layer Normalization in the Transformer Architecture" (2020)

Transformers – Finetune

- Many state-of-the-art performances can be achieved by finetuning large pre-trained language models such as BERT
- If you want to finetune yourself, use libraries such as Hugging Face
- 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 BERT but only the last few layers to prevent overfitting and reduce memory
- Regularization like weight decay or dropout often helps

Transformers as Graph NN

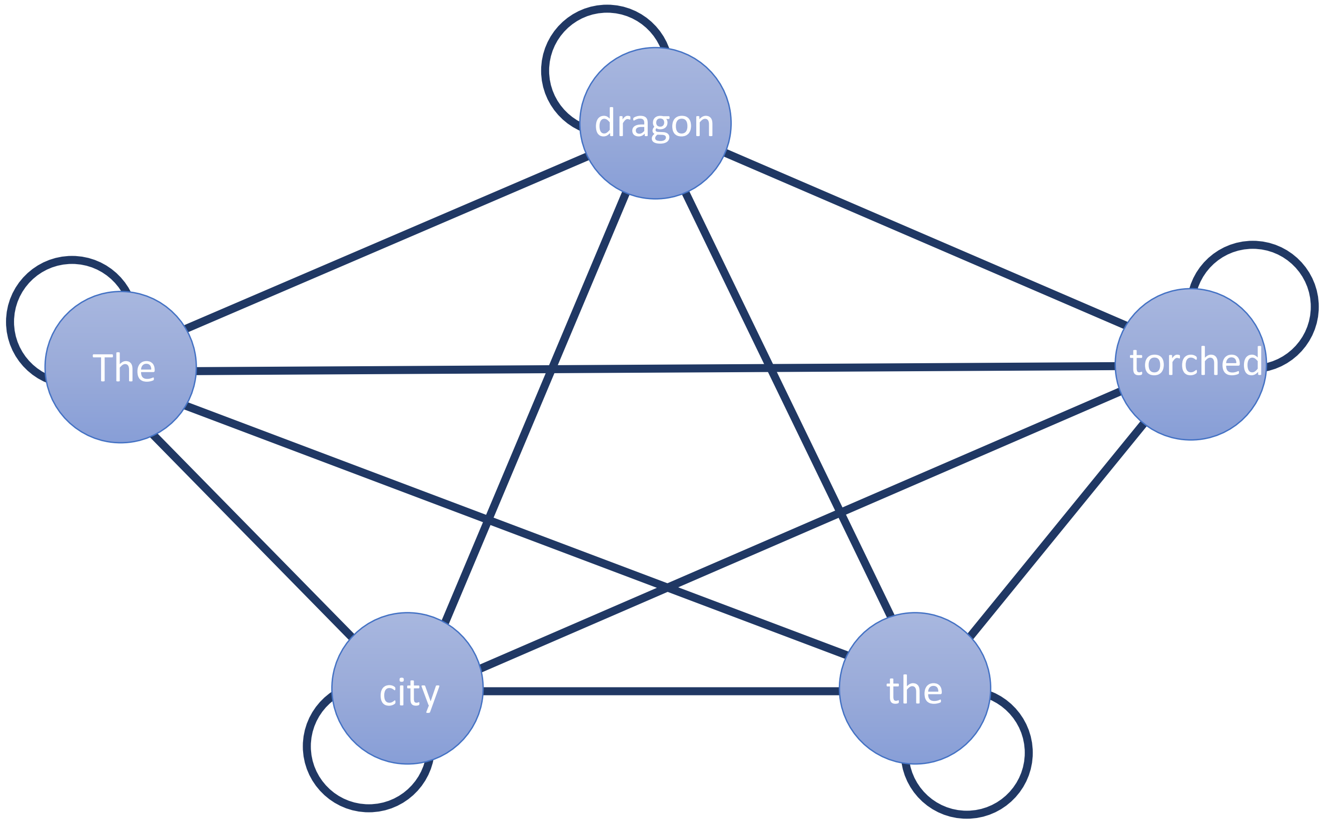
Claim: Transformers are just graph convolutions over dense graphs



- 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 NN

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 NN

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
  $\Rightarrow$ Graph sparsification based on syntax trees etc. corresponds to masking
- Self-attention can be used for permutation-invariant tasks
  - Data like sets, graphs, etc.

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

- 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.

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*

Q&A

BERT

Bidirectional Encoder Representations from Transformers

Presented by Omar Elbaghdadi

Word embeddings

- One word, one representation
- Problem: word’s meaning depends on context

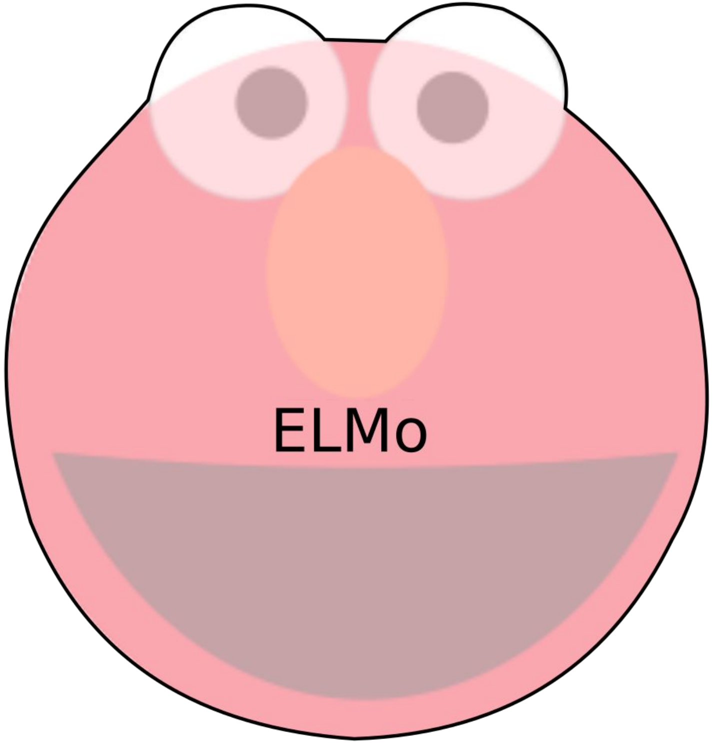
“**Stick** to the plan, dude.”

vs

“If you don’t pay attention to my presentation, I’ll hit you with a **stick**.”

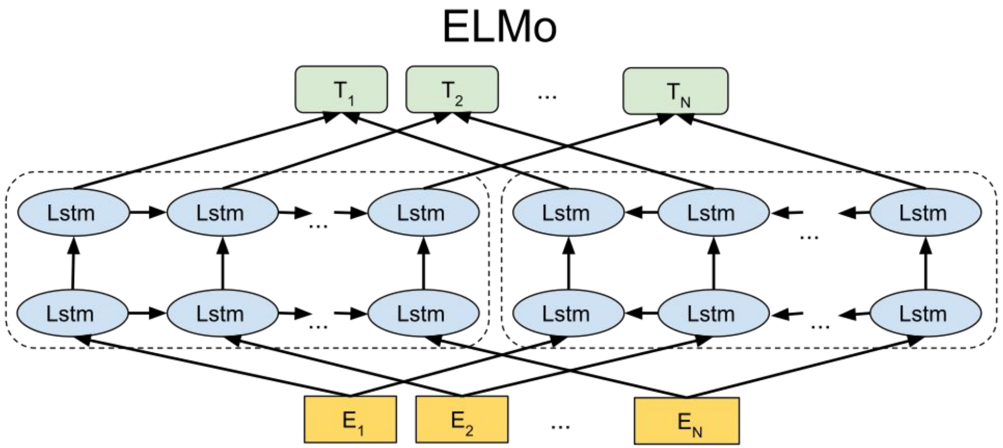
Contextualized Embeddings

Deep contextualized word representations



Deep contextualized word representations: ELMo

- Embeddings computed from bidirectional LSTM

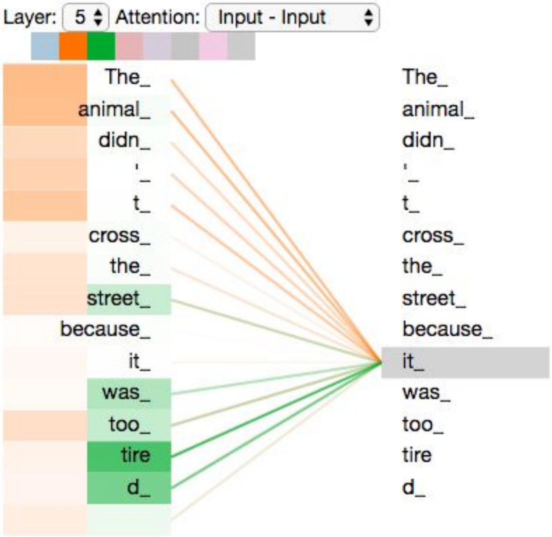


- So, embeddings now depend on context
- Pre-Train on Language Modelling (LM) task

All you need is attention

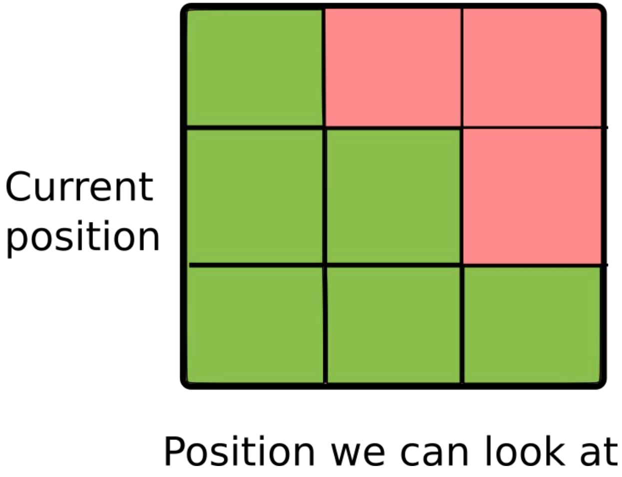
Transformers for language modelling

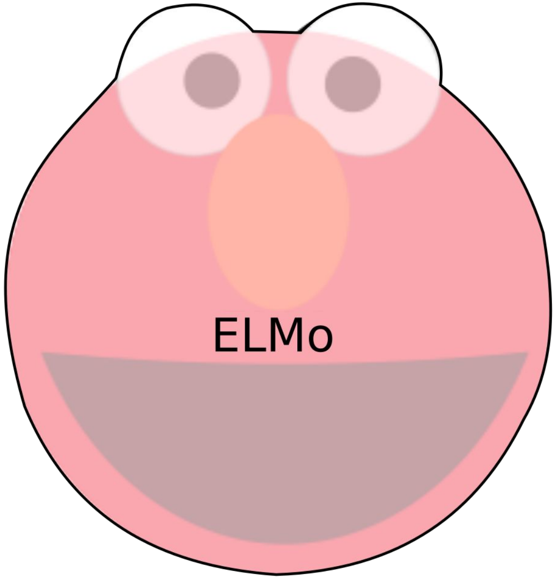
- Instead of recurrent model, use a Transformer
- **Self-attention**: condition on **all** other words



Transformers for language modelling

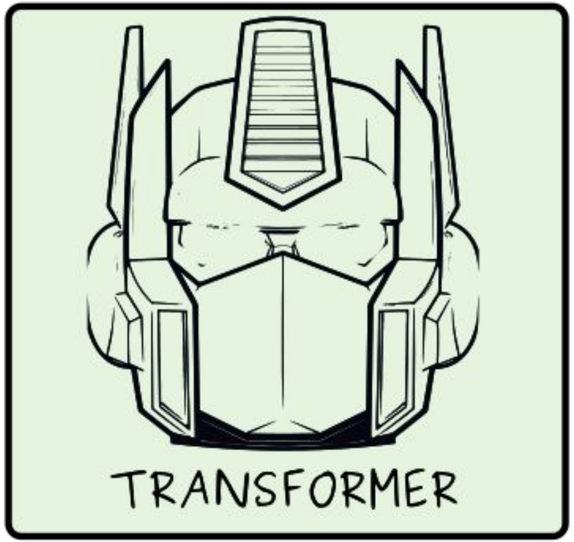
- LM task: predict next word
- Problem: self-attention uses **all** words
- Solution: mask words to the right





Contextual
Embeddings

+

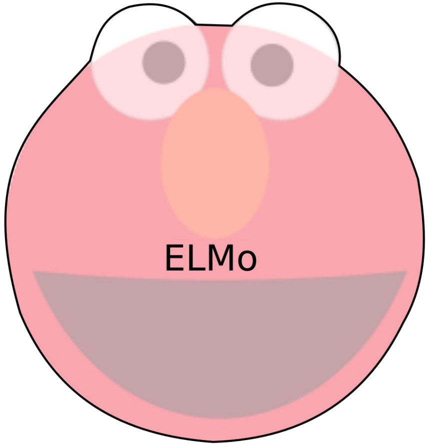


Left-to-right

Fine-tuning

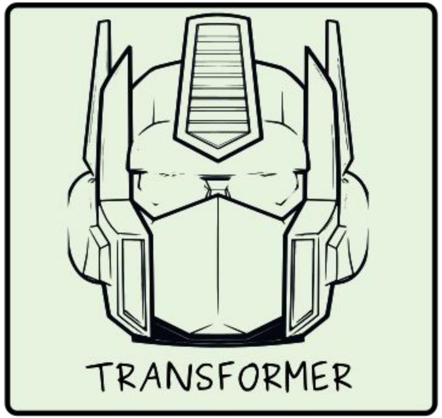
From feature-based to fine-tuning

- **Feature-based**: pre-trained representations as features
- Problems:
  - harder to generalize
  - embeddings not optimal for downstream task
- Solution: fine-tune pre-trained weights
- Finetuning: ULM-FiT



Contextual Embeddings

+

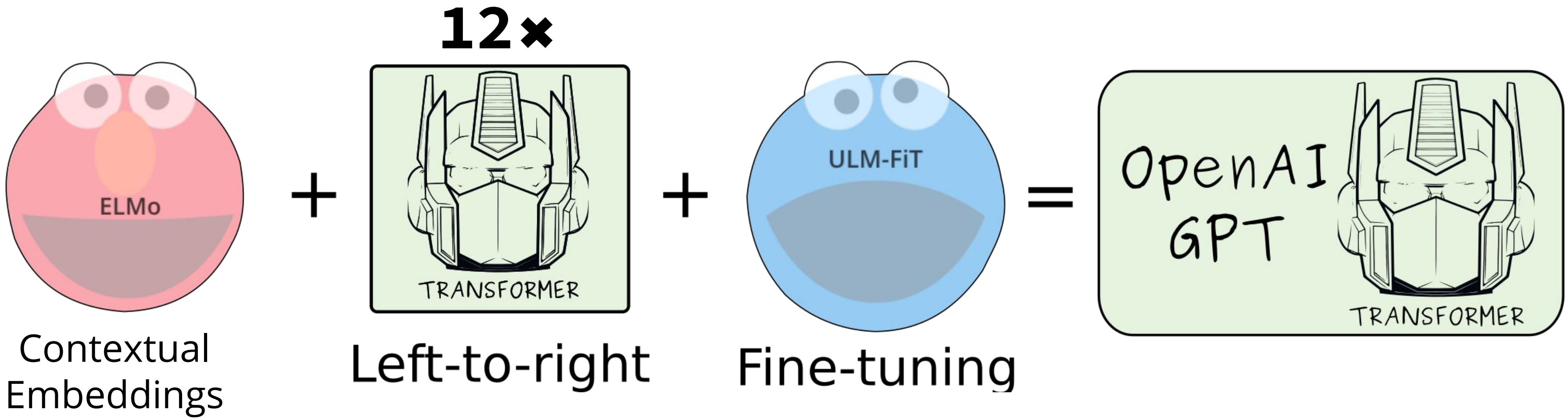


Left-to-right

+

Fine-tuning

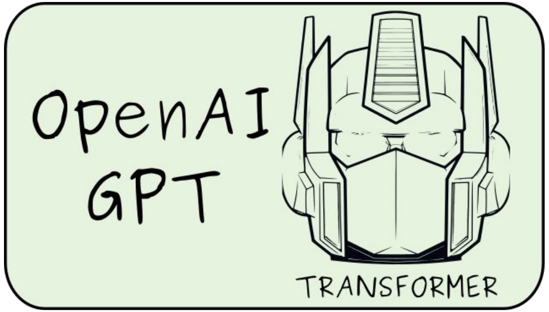
Model arithmetic



From GPT to BERT

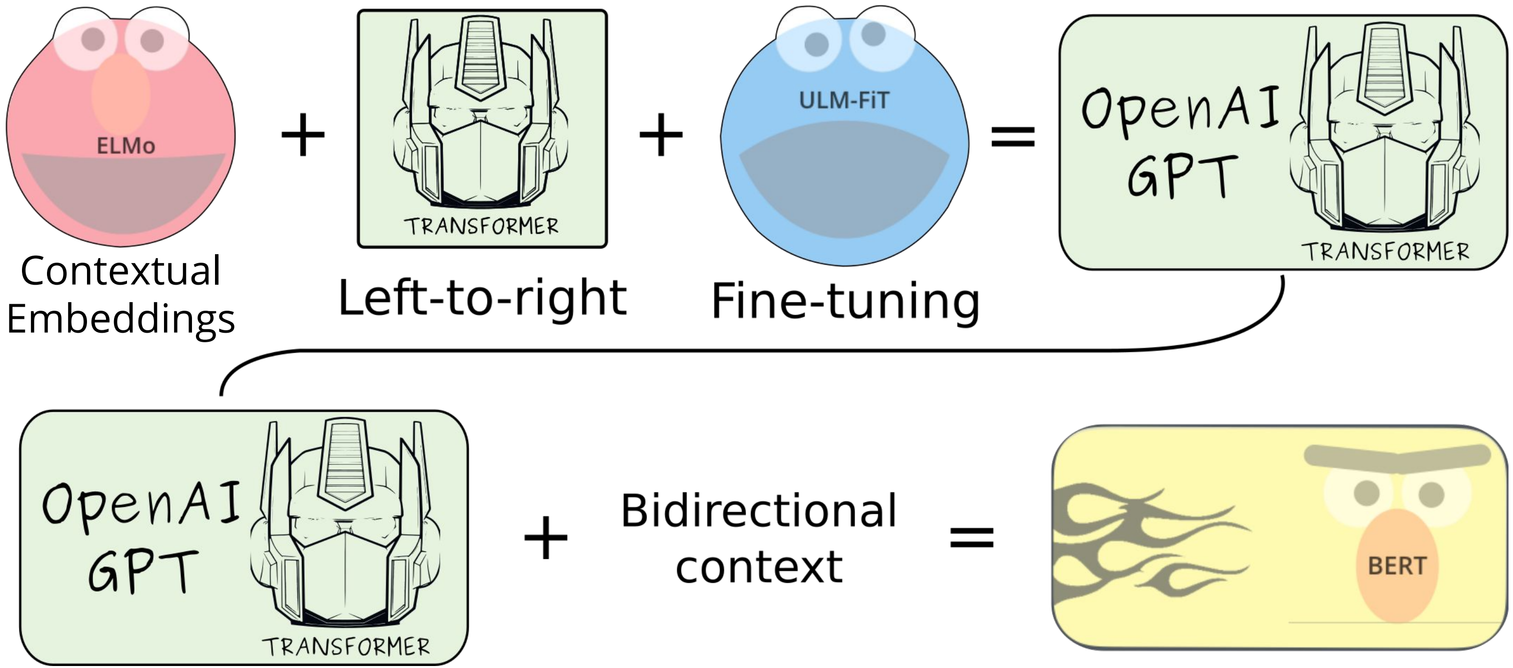
- GPT uses **left-to-right** (LTR) representations
- Intuitively, bidirectional representations more powerful
- BERT’s **main contribution:**

> ***How to do bidirectional context modelling with Transformers.***



+ Bidirectional context =

From GPT to BERT



Bidirectional Context Modelling: How?

Use special pre-training Tasks

Pre-training: masked Language Model (MLM)

The cat ___ on the mat

Pre-training: masked Language Model (MLM)

`The cat sat on the mat`

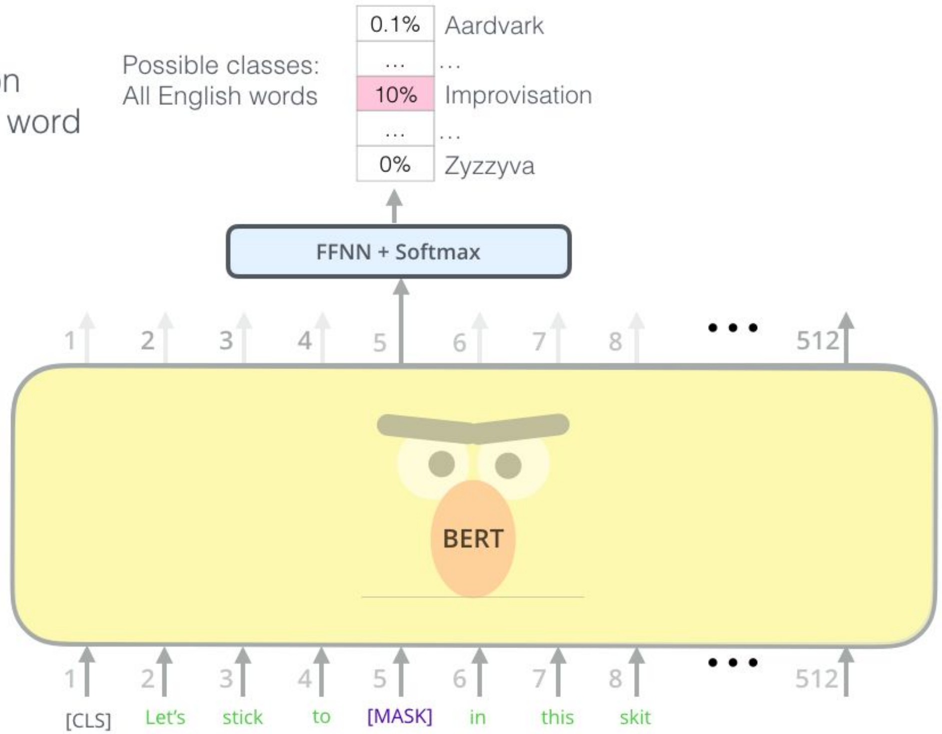
Pre-training: masked Language Model (MLM)

Randomly mask
15% of tokens

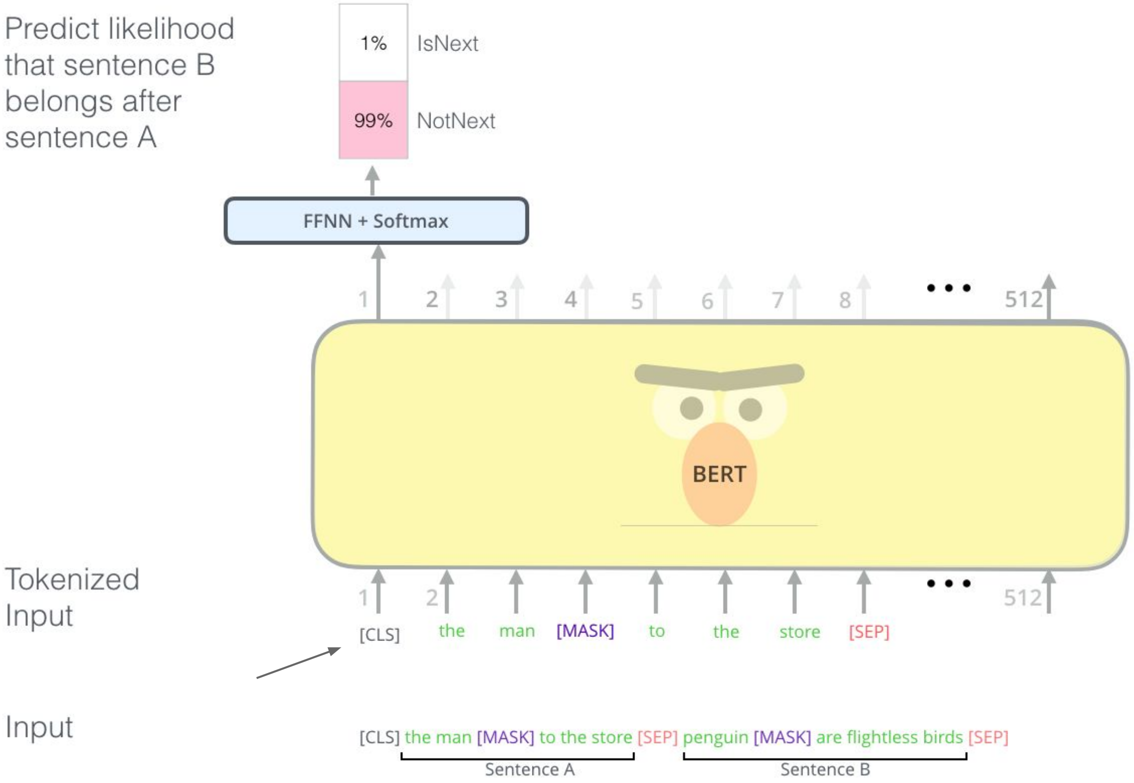
Pre-training: masked Language Model (MLM)

Use the output of the masked word's position to predict the masked word

Randomly mask 15% of tokens



Pre-training: next sentence prediction (NSP)

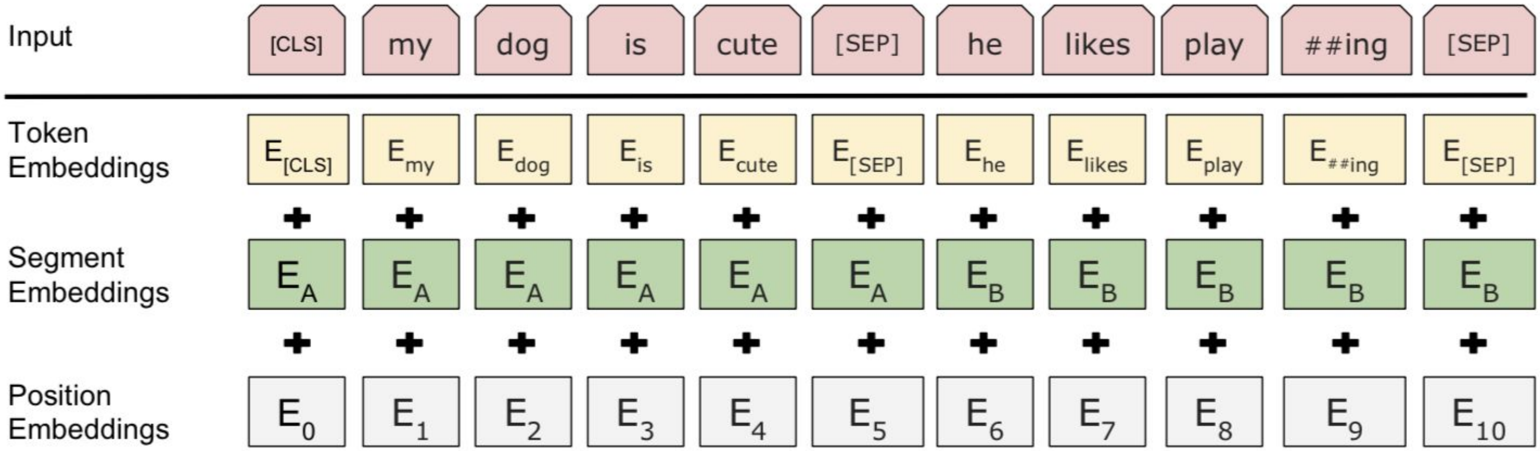


Pre-training: data

- English wikipedia (2,500M words)
- BooksCorpus (800M words)
- Document-level corpus critical

  (as opposed to shuffled sentence-level)

Input processing

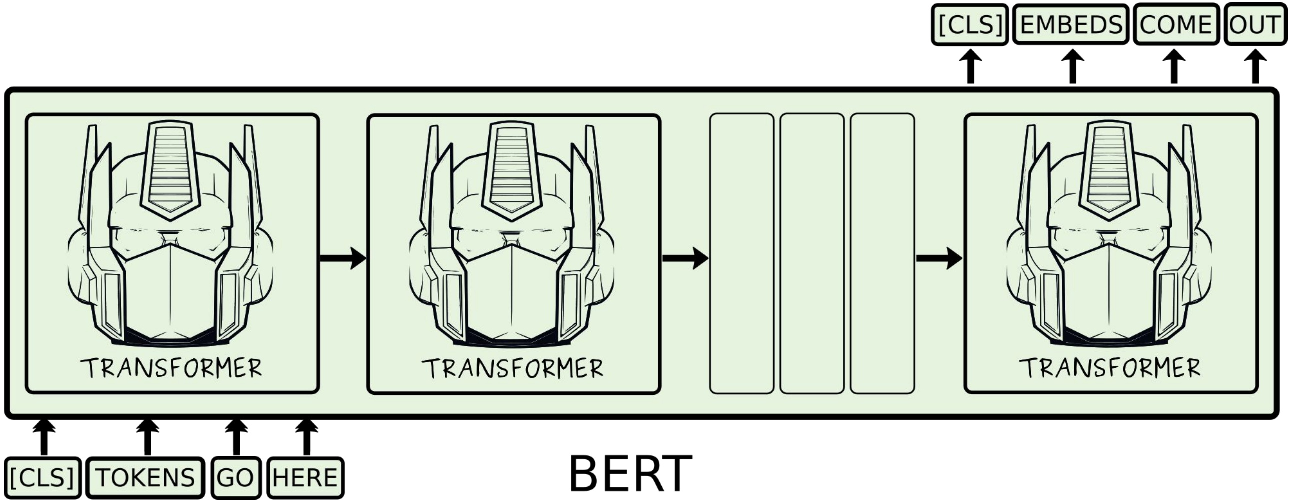


Fine-Tuning

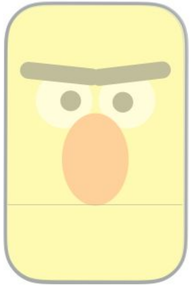
- Straightforward. Only need to adapt inputs/outputs.
- No need to encode text pairs explicitly
- Relatively inexpensive compared to pre-training

Architecture

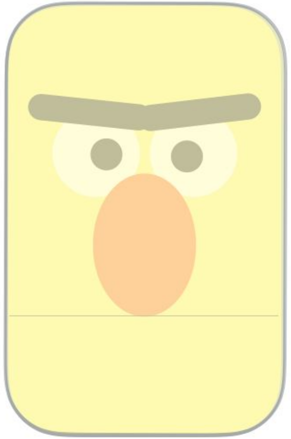
- Like GPT: stack of Transformer blocks



Bert: 2 sizes



BERT BASE



BERT LARGE

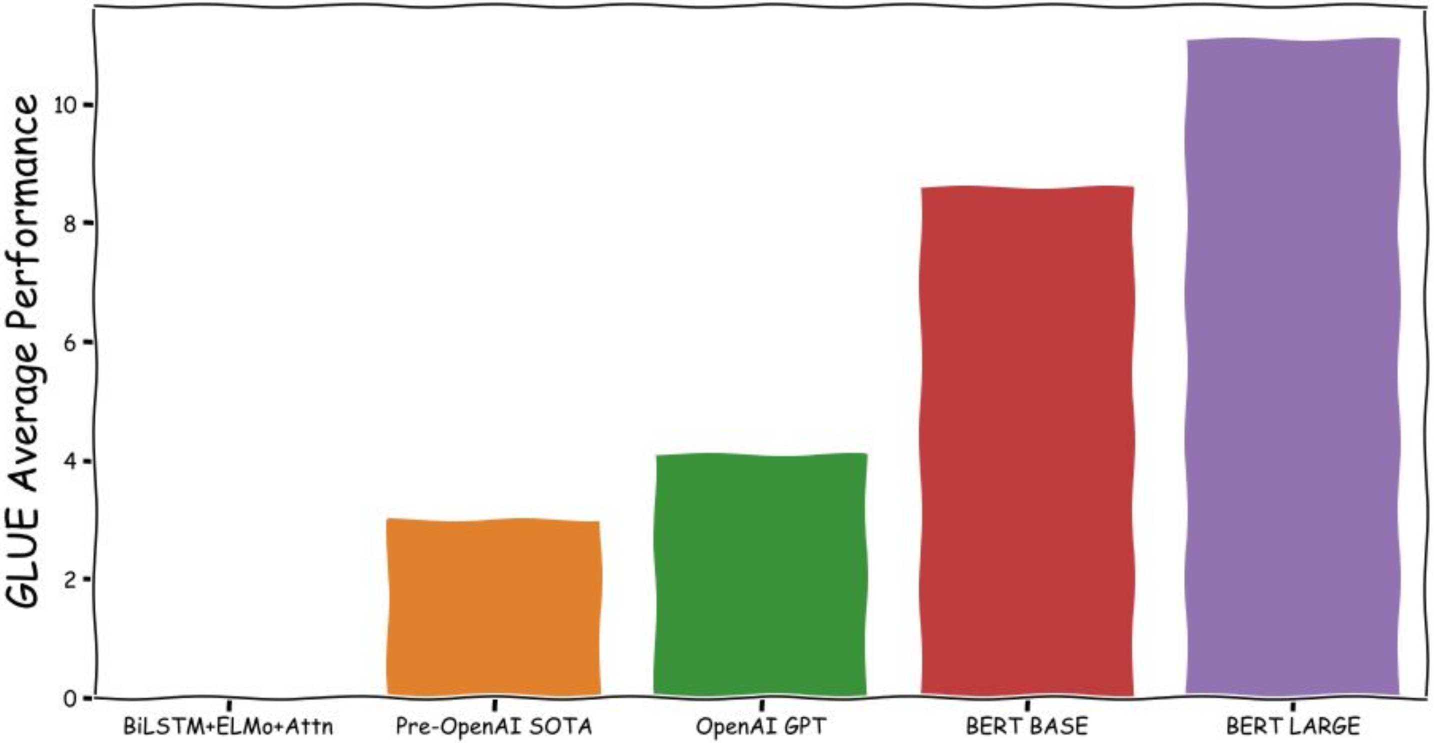
- Smaller model: 12 Transformer blocks
- Same size as GPT for comparison
- BERT-large: 24 blocks

Experiments And Results

Performance benchmarks

- GLUE: 11 NLP tasks
- Some other tasks
- A lot of tasks, basically
- Importantly, architecture stays same over most tasks

Performance benchmarks



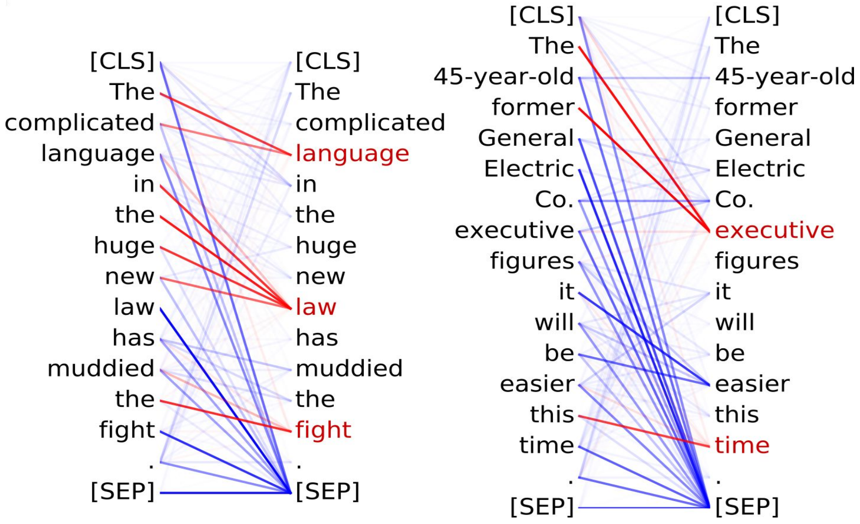
What makes it perform *so* well?

- Effect of pre-training tasks
  - Removing NSP hurts performance significantly
  - LTR model worse than MLM model on all tasks
  - Conlusion: bidirectionality is important
- Effect of model size
  - Bigger is better
  - Show that extreme model sizes improve even small scale tasks
- Feature-based approach:
  - Worse but not much
  - Concat Last Four Hidden works best in experiment

What does BERT learn?

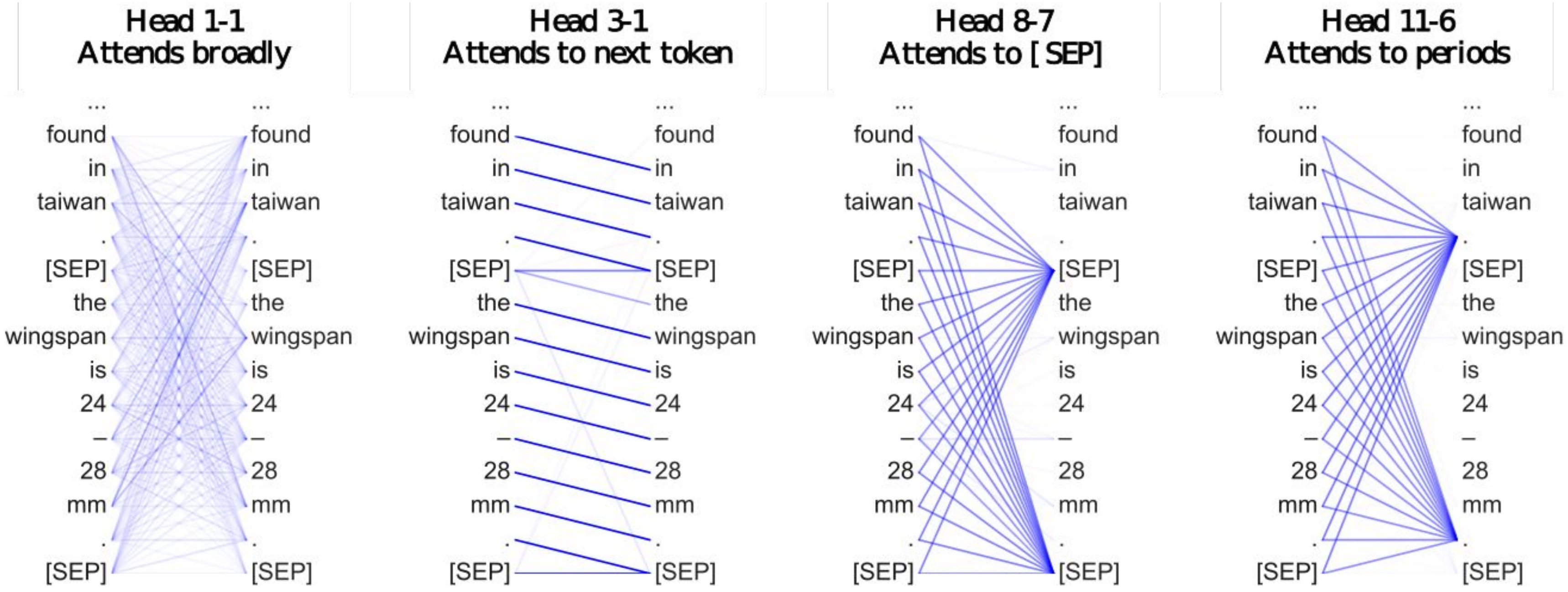
Head 8-11

- Noun modifiers (e.g., determiners) attend to their noun
- 94.3% accuracy at the det relation



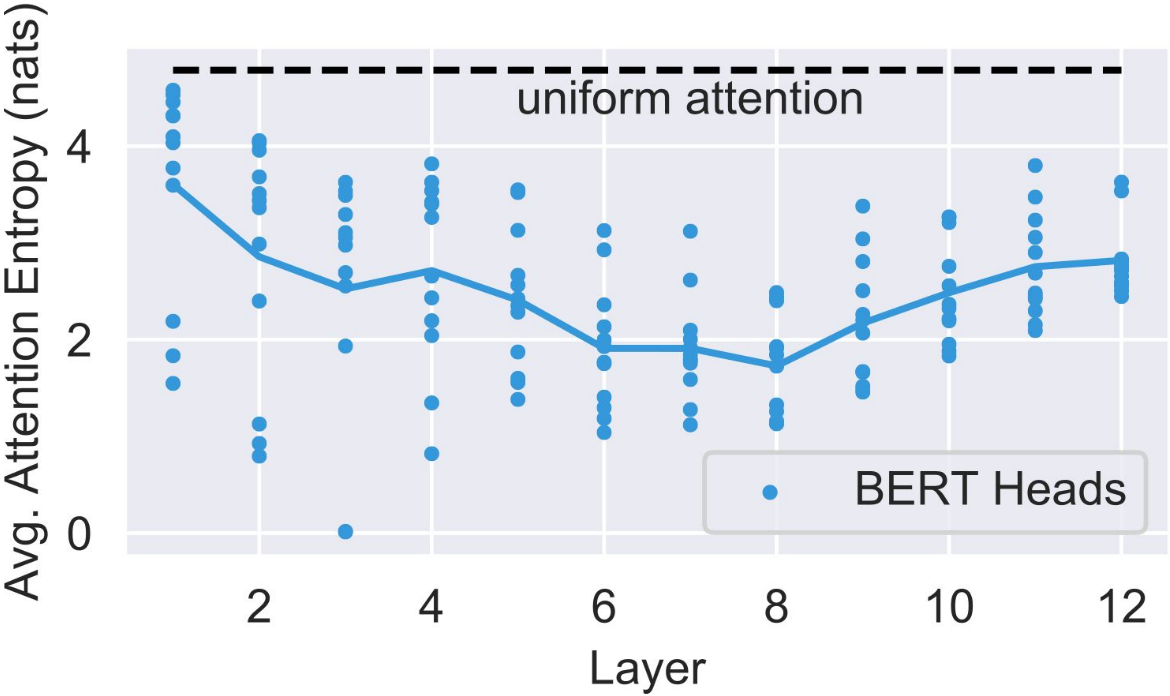
Source: Clark et al. 2019

What does BERT learn?



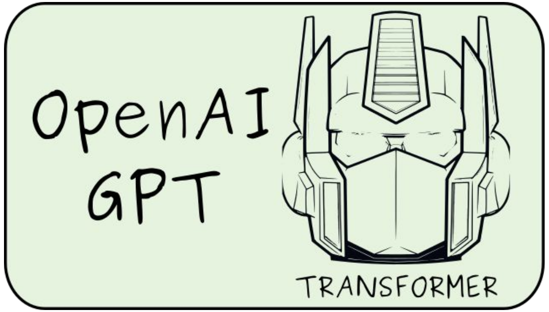
Source: Clark et al. 2019

What does BERT learn?



Source: Clark et al. 2019

Conclusion



+ Bidirectional context =

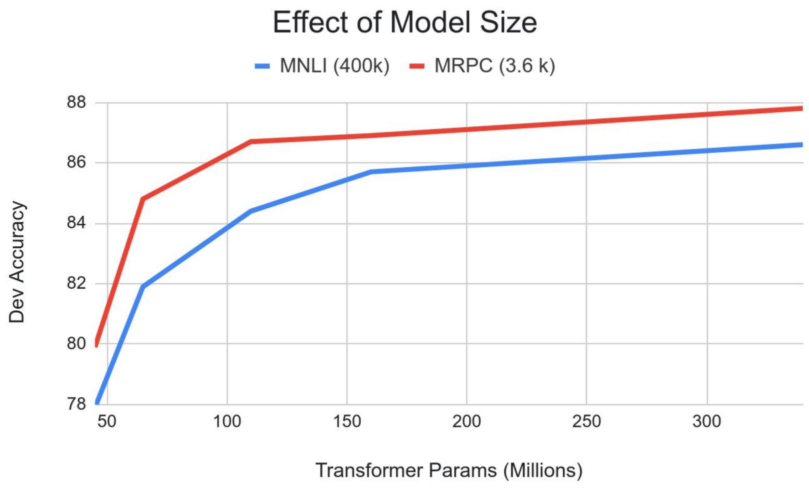
The **core argument:**

> ***Bi-directionality and the two pre-training tasks account for the majority of the empirical improvements***

Conclusion

Biggest impact on the field:

With pre-training, bigger == better, without clear limits (so far)



Opinion

- Good methodological study of model aspects
- Comparison with GPT very well done
- Open-sourcing pre-trained models
- No *understanding* learned representations

Further research

- Hierarchical representations
- More speed up -- smaller models
- *Understanding* representations

The end

Credit and references

Images for BERT models, Elmo, and Cookie Monster were taken from the Illustrated BERT blog post.

The input architecture and BiLSTM figures come from the BERT paper.

What Does BERT Look At? An Analysis of BERT's Attention (Kevin Clark, Urvashi Khandelwal, Omer Levy & Christopher Manning)

Slides by BERT co-author J. Devlin.