Transformers

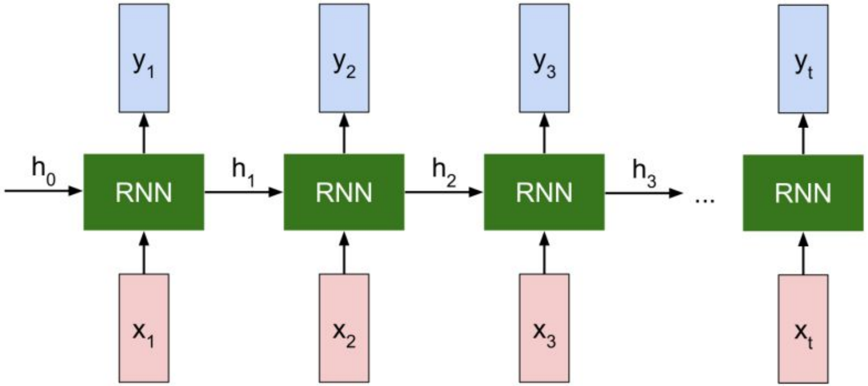
Ju Sun
Computer Science & Engineering

Nov 22, 2022

University of Minnesota
Driven to Discover℠

Recap

RNN: model sequences



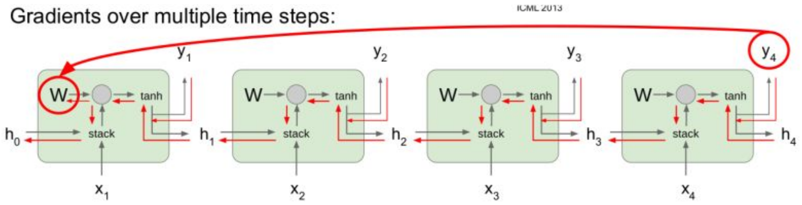
(Credit: Stanford CS231N)

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_t\right)$$

$$\boldsymbol{y}_t = \boldsymbol{V}_y\boldsymbol{h}_t$$

$\boldsymbol{W}_{\boldsymbol{h}}, \boldsymbol{W}_{\boldsymbol{x}}$ and $\boldsymbol{V}_{\boldsymbol{y}}$ are shared across the sequence

Vanishing/exploding gradient issue



(Credit: Stanford CS231N)

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}\cdots\frac{\partial \boldsymbol{h}_1}{\boldsymbol{W}} = \frac{\partial L_t}{\partial \boldsymbol{h}_t}\left(\prod_{k=2}^{t}\frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}}\right)\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

$$= \frac{\partial L_t}{\partial \boldsymbol{h}_t}\left(\prod_{k=2}^{t}\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\boldsymbol{W}_h\right)\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

* when $\|\boldsymbol{W}_h\| > 1$, gradient **explodes** if t large
* when $\|\boldsymbol{W}_h\| < 1$, gradient **vanishes** if t large

$$\left\|\prod_{k=2}^{t}\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\boldsymbol{W}_h\right\|$$

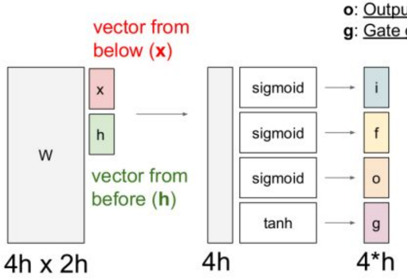
$$\leq \prod_{k=2}^{t}\left\|\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\right\|\left\|\boldsymbol{W}_h\right\|$$

$$\leq \prod_{k=2}^{t}\left\|\operatorname{diag}\left(\tanh'\left(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{h}_{k-1} + \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x}_k\right)\right)\right\|\left\|\boldsymbol{W}_h\right\|^{t-1}$$

Gated RNNs

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

i: <u>Input gate</u>, whether to write to cell
f: <u>Forget gate</u>, Whether to erase cell
o: <u>Output gate</u>, How much to reveal cell
g: <u>Gate gate</u> (?), How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

(Credit: Stanford CS231N)

$\boldsymbol{u}$: **update gate**, control state update

$\boldsymbol{r}$: **reset gate**, control how previous state affects new content

$\boldsymbol{g}$: new content

Gated recurrent unit (GRU)

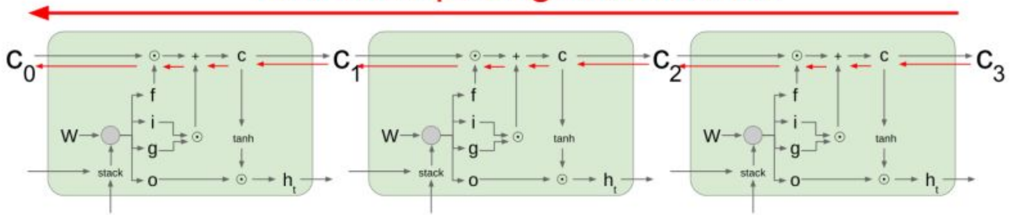
$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{r} \\ \boldsymbol{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \end{bmatrix} \left(\boldsymbol{W} \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{bmatrix} \right)$$

$$\boldsymbol{g} = \tanh\left(\boldsymbol{W}_h (\boldsymbol{r} \odot \boldsymbol{h}_{t-1}) + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_g\right)$$

$$\boldsymbol{h}_t = \boldsymbol{u} \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}) \odot \boldsymbol{g}$$

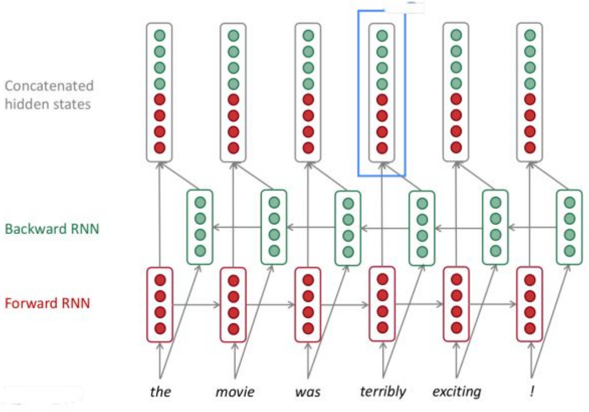
$\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{o}$ are merged

Uninterrupted gradient flow!



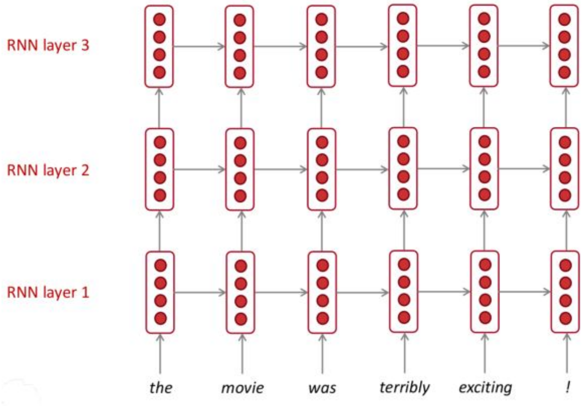
(Credit: Stanford CS231N)

Modern RNNs



(Credit: Stanford CS224N)

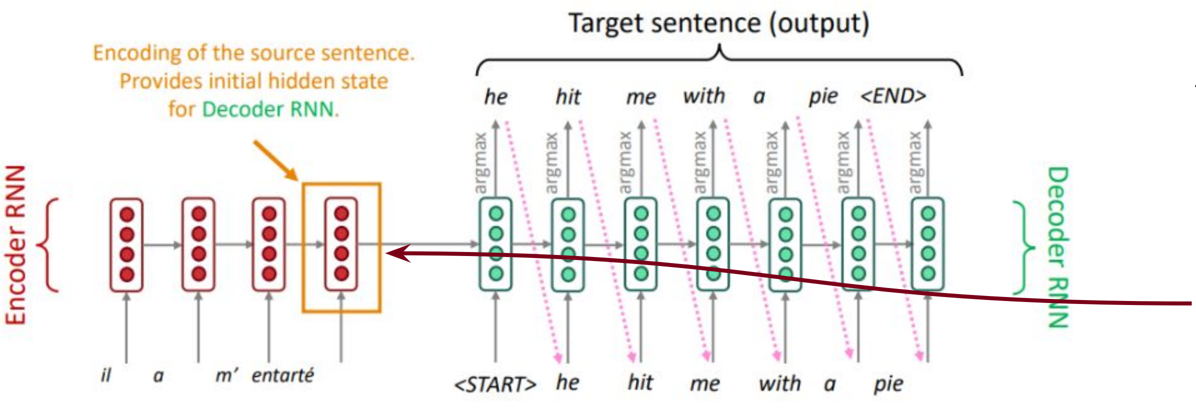
Bidirectional RNN



(Credit: Stanford CS231N)

Deep RNN

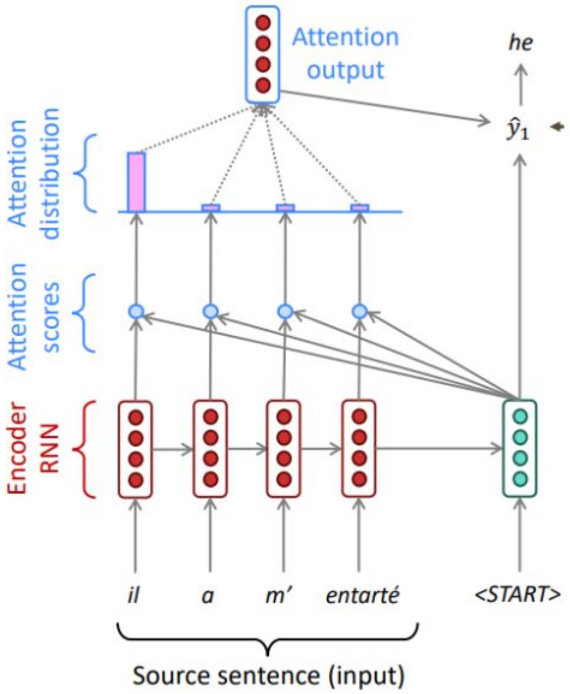
Seq2Seq model



Bottleneck problem

(Credit: Stanford CS231N)

Attention mechanism



(Credit: Stanford CS231N)

Input: source vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in \mathbb{R}^h$, and target vector $\boldsymbol{t}$

Output: weighted summation

$$\sum_{j=1}^{N} w_j \boldsymbol{s}_j \quad \text{where } w_j = \text{similarity}(\boldsymbol{s}_j, \boldsymbol{t})$$

Many possibilities:

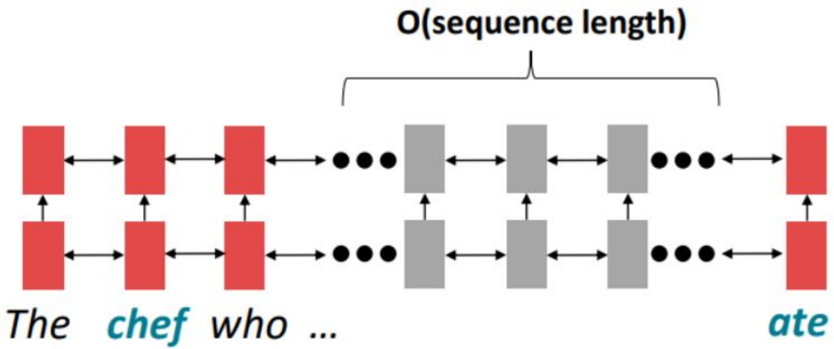
Attention scores

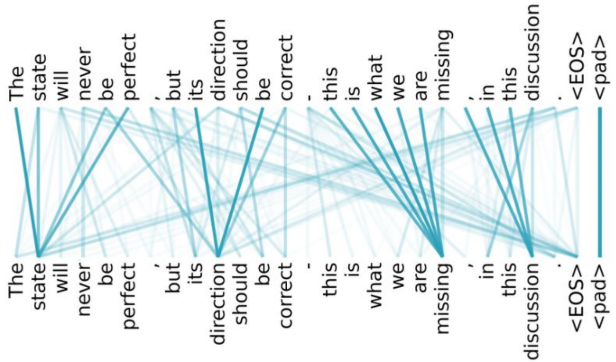
- dot-product attention: $\widehat{w_j} = \langle \boldsymbol{s}_j, \boldsymbol{t} \rangle$ (Is is better to normalize this or rescale it by the dimension factor?)
- multiplicative attention: $\widehat{w_j} = \langle \boldsymbol{s}_j, \boldsymbol{W}\boldsymbol{t} \rangle$
- "additive attention": $\widehat{w_j} = \boldsymbol{v}^\intercal \sigma \left(\boldsymbol{W}_1 \boldsymbol{s}_j + \boldsymbol{W}_2 \boldsymbol{t} \right)$

The actual weights are attention scores passed through **softmax**

$$w_j = \frac{\exp\left(\widehat{w_j}\right)}{\sum_k \exp\left(\widehat{w_k}\right)}$$

Self-attention





RNN

- Long interaction distance
- Resistant to parallelization

Self-attention

- O(1) interaction distance
- Highly parallelizable

Each token gets a selective summary of information from all others

Self-attention

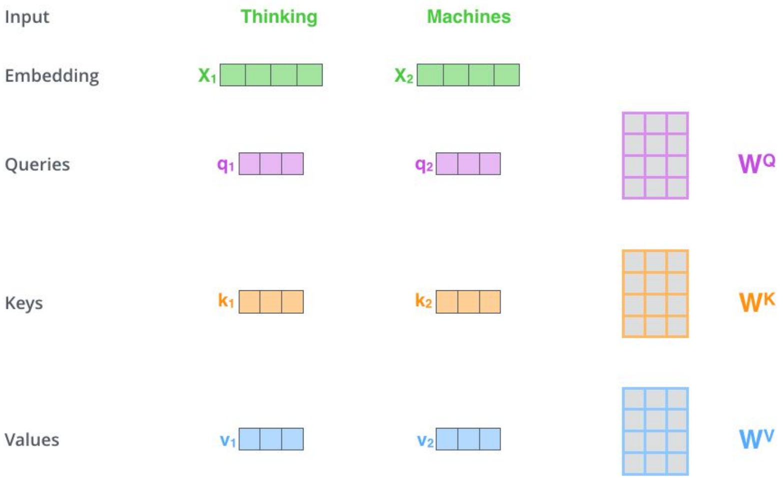


Image credit: https://jalammar.github.io/illustrated-transformer/

- Each word now encoded as (query, key, value) triple
- For an input $\boldsymbol{x}_i$, we have:

$$\boldsymbol{q}_i = (\boldsymbol{W}^Q)^\intercal \boldsymbol{x}_i, \quad \boldsymbol{k}_i = (\boldsymbol{W}^K)^\intercal \boldsymbol{x}_i, \quad \boldsymbol{v}_i = (\boldsymbol{W}^V)^\intercal \boldsymbol{x}_i$$

- Calculate attention scores between query and all keys: $e_{ij} = \langle \boldsymbol{q}_i, \boldsymbol{k}_j \rangle$
- softmax normalization $w_{ij} = \exp(e_{ij}) / \sum_k \exp(e_{ik})$
- output the weighted sum of values $\sum_j w_{ij} \boldsymbol{v}_j$

In matrix notation

- Compute queries, keys, and values

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

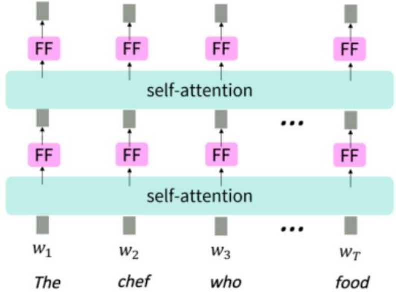
- Calculate attention scores between query and all keys: $\boldsymbol{E} = \boldsymbol{Q}\boldsymbol{K}^\intercal$
- softmax normalization $\boldsymbol{A} = \mathrm{softmax}(\boldsymbol{E})$
- output the weighted sum of values $\boldsymbol{A}\boldsymbol{V}$

$$\mathrm{output} = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal)\boldsymbol{V}$$

Question: why we need both query and key?

Equation for Feed Forward Layer

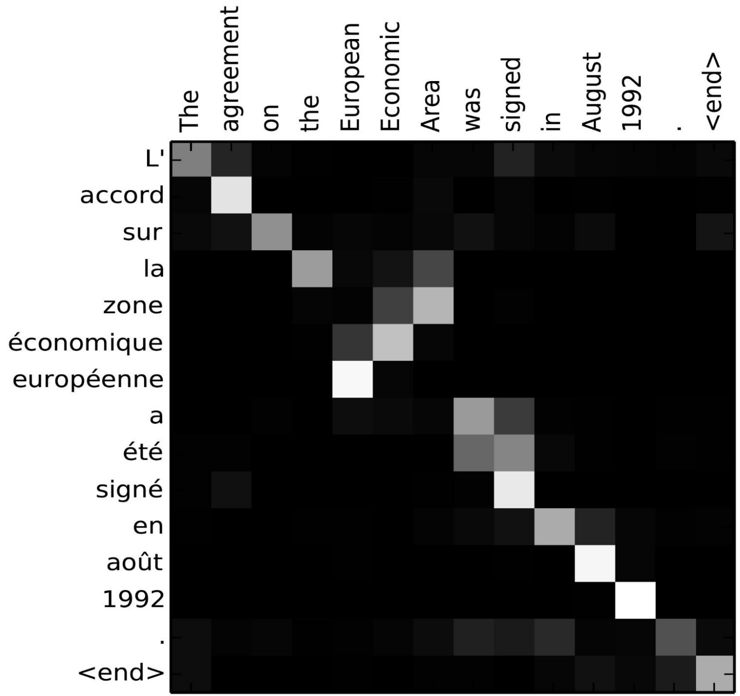
$$\begin{aligned} m_i &= MLP(\mathrm{output}_i) \\ &= W_2 * \mathrm{ReLU}(W_1 \times \mathrm{output}_i + b_1) + b_2 \end{aligned}$$



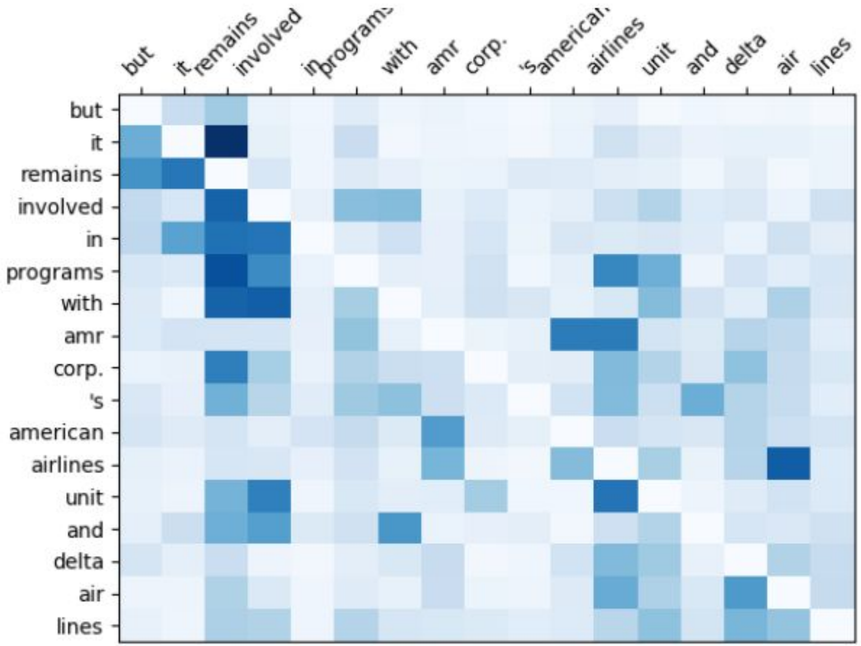
Adding in nonlinearity!

First step toward Transformers!

Attention matrices—visualizing correlations



General attention



Self-attention A

Transformers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

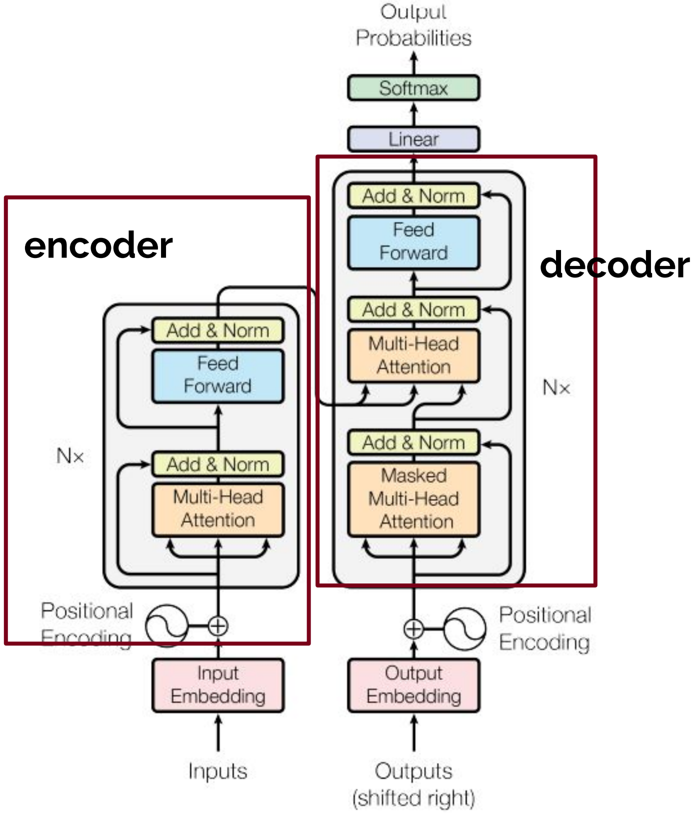


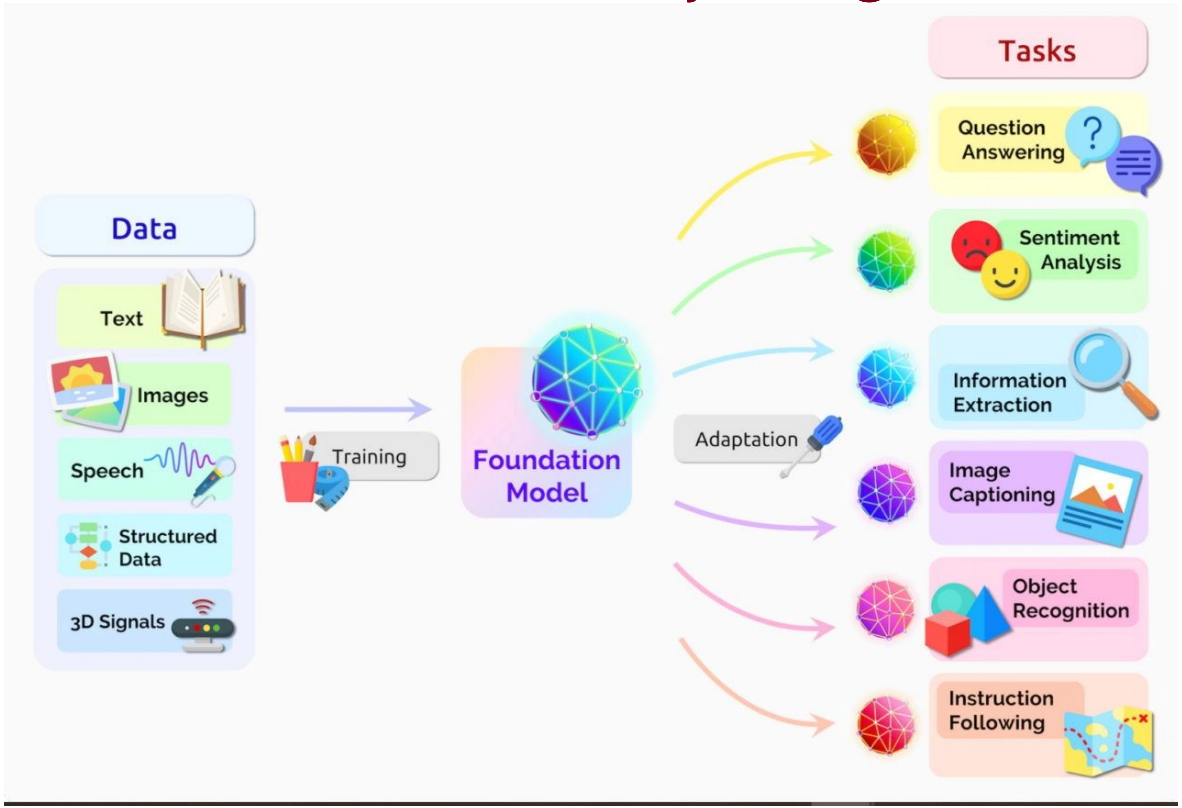
Figure 1: The Transformer - model architecture.

NIPS 2017; https://arxiv.org/abs/1706.03762

Transformers reign in NLP!

Image credit: https://medium.com/mlearning-ai/evolution-of-transformers-part-1-faac3f19d780

Transformers for everything!



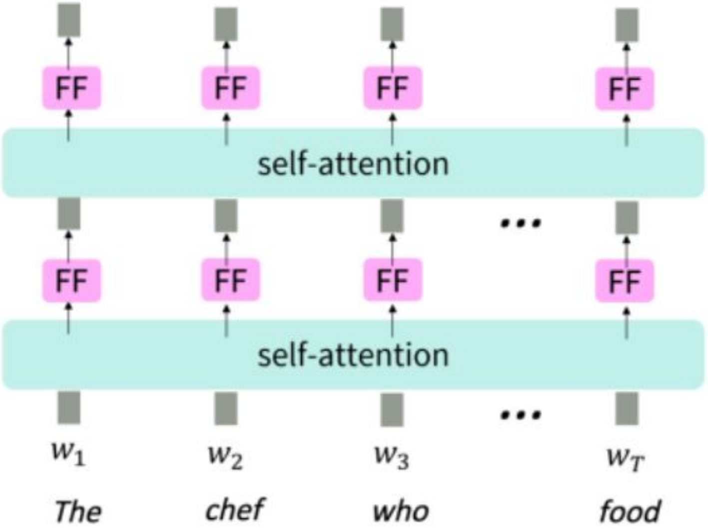
- Transformers have been modified to deal with **almost all** kinds of structured and **unstructured** data
- Enable multimodal data integration and interaction

Image credit: https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/

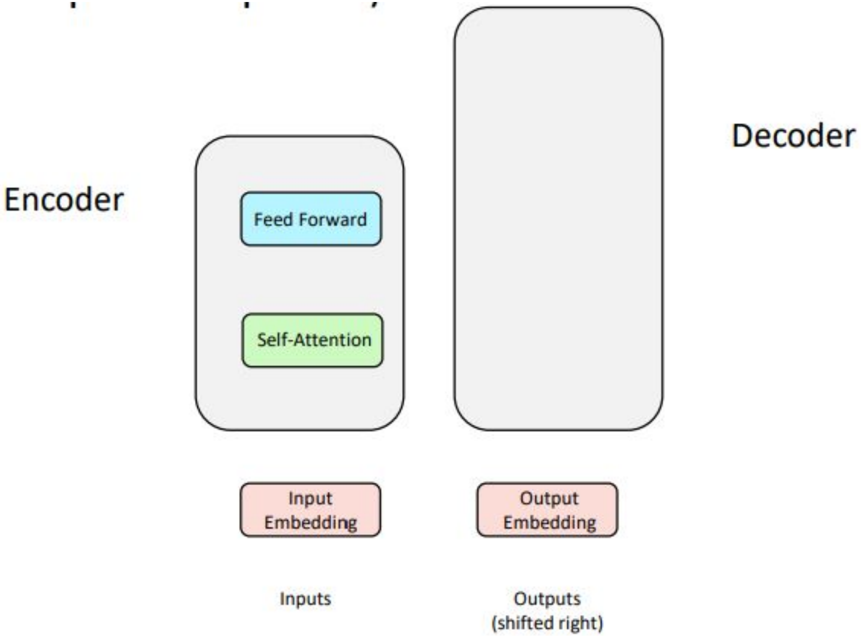
Starting from self-attention

Equation for Feed Forward Layer

$$
\begin{aligned}
m_i &= MLP(\text{output}_i) \\
&= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2
\end{aligned}
$$



(Credit: Stanford CS231N)

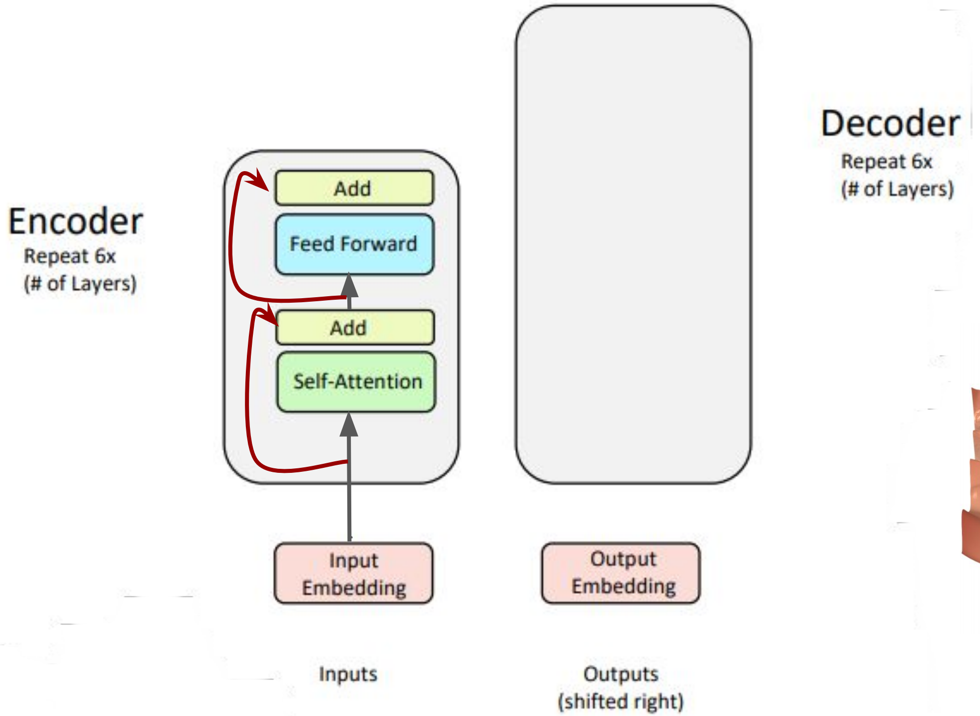


(Credit: Stanford CS231N)

Three tricks to build in depth:

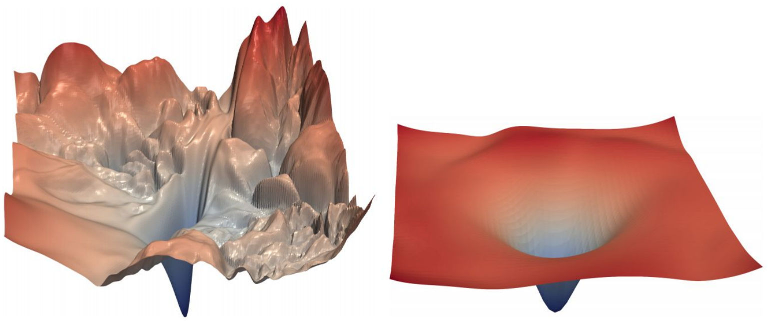
- Residual connection
- Layer normalization
- Scaled inner product attention

Trick 1: Residual connection



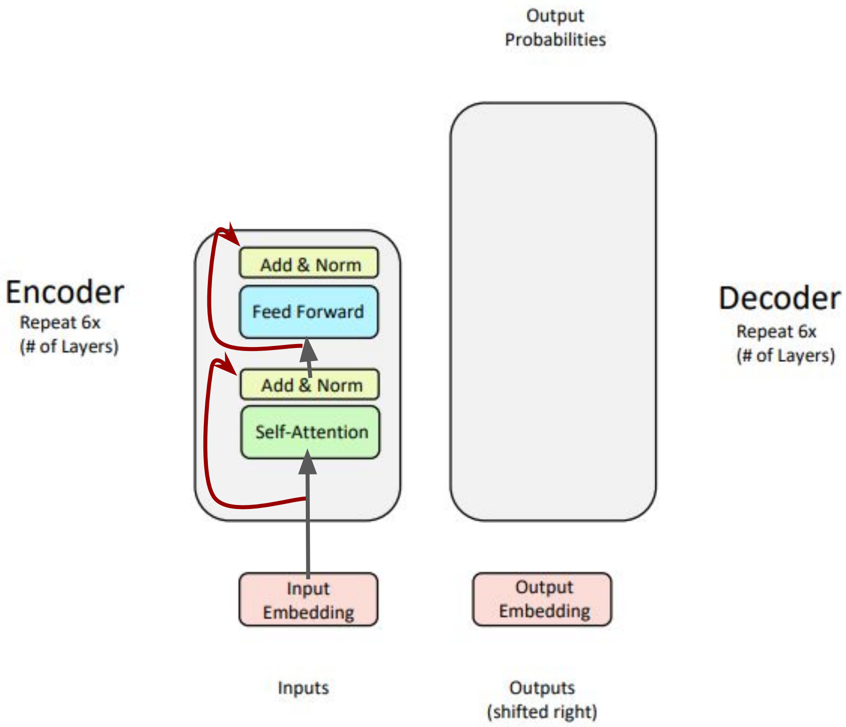
$$\boldsymbol{x}_k = F(\boldsymbol{x}_{k-1}) + \boldsymbol{x}_{k-1}$$

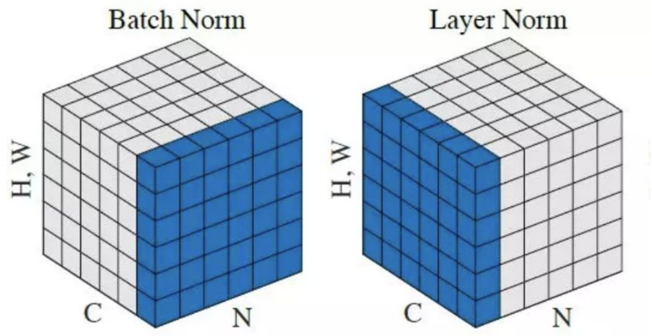
- Mitigating vanishing gradient
- Smoothing out landscape



https://arxiv.org/abs/1712.09913

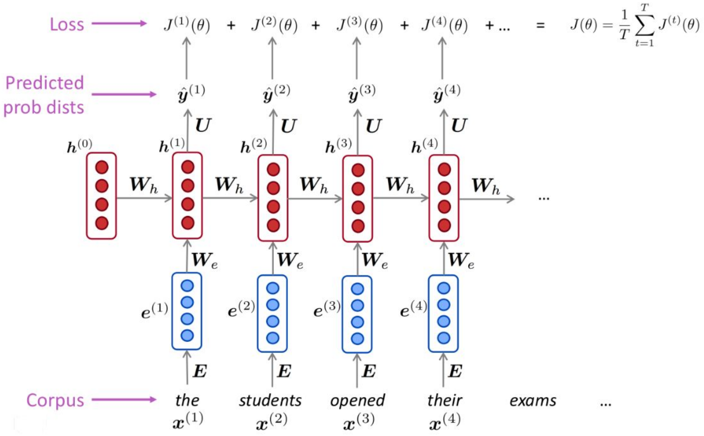
Trick 2: Layer normalization



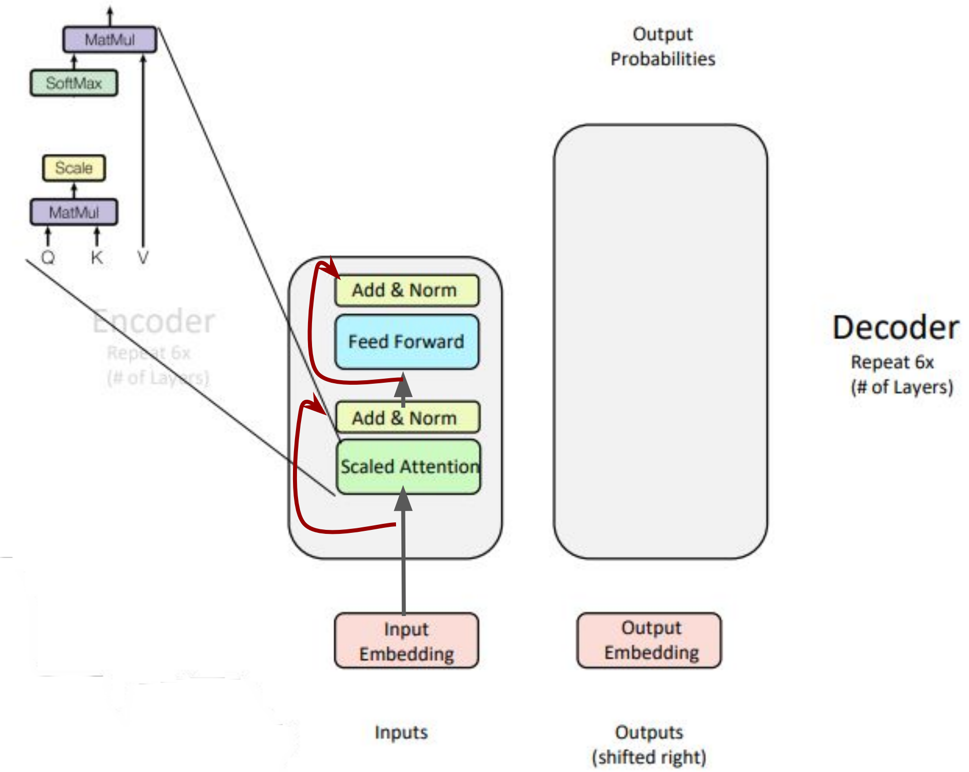


$$x^{\ell\prime} = \frac{x^{\ell} - \mu^{\ell}}{\sigma^{\ell} + \epsilon}$$

Why not batchnorm?



Trick 3: Scaled inner product attention



$$\text{output} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal)\boldsymbol{V}$$

- After Layernorm, entries of Q and K behaves like IID zero-mean, unit variance
- $\mathbb{E}\langle \boldsymbol{q}^i, \boldsymbol{k}^j \rangle = 0$ but

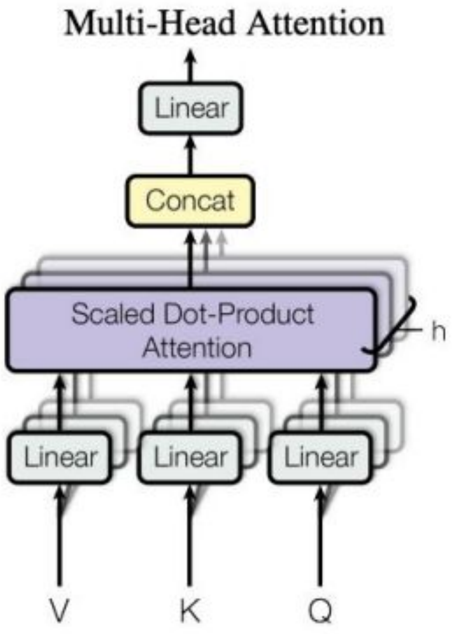
$$\mathrm{Var}\langle \boldsymbol{q}^i, \boldsymbol{k}^j \rangle = d_k$$

This can blow up exp computation in the softmax normalization for large d_k!

Solution: normalize by standard deviation

$$\text{output} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal/\sqrt{d_k})\boldsymbol{V}$$

Multi-head attention



[Vaswani et al. 2017]

Multiple, independent self-attention blocks in parallel

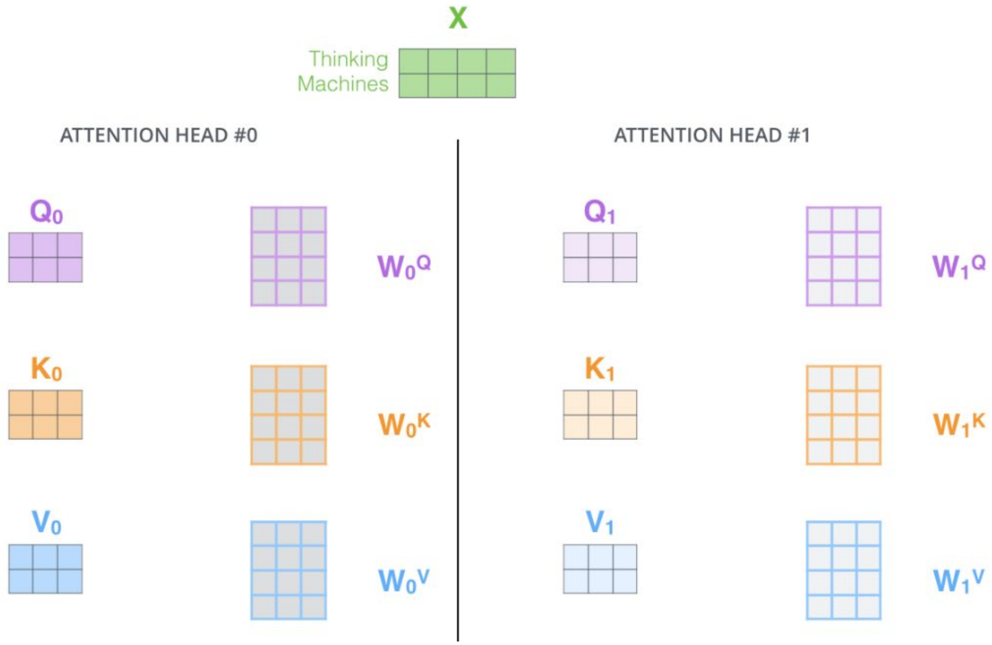


Image credit: https://jalammar.github.io/illustrated-transformer/

Intuition: allow the flexibility of capturing different kinds of "relevance"/correlations

Multi-head attention

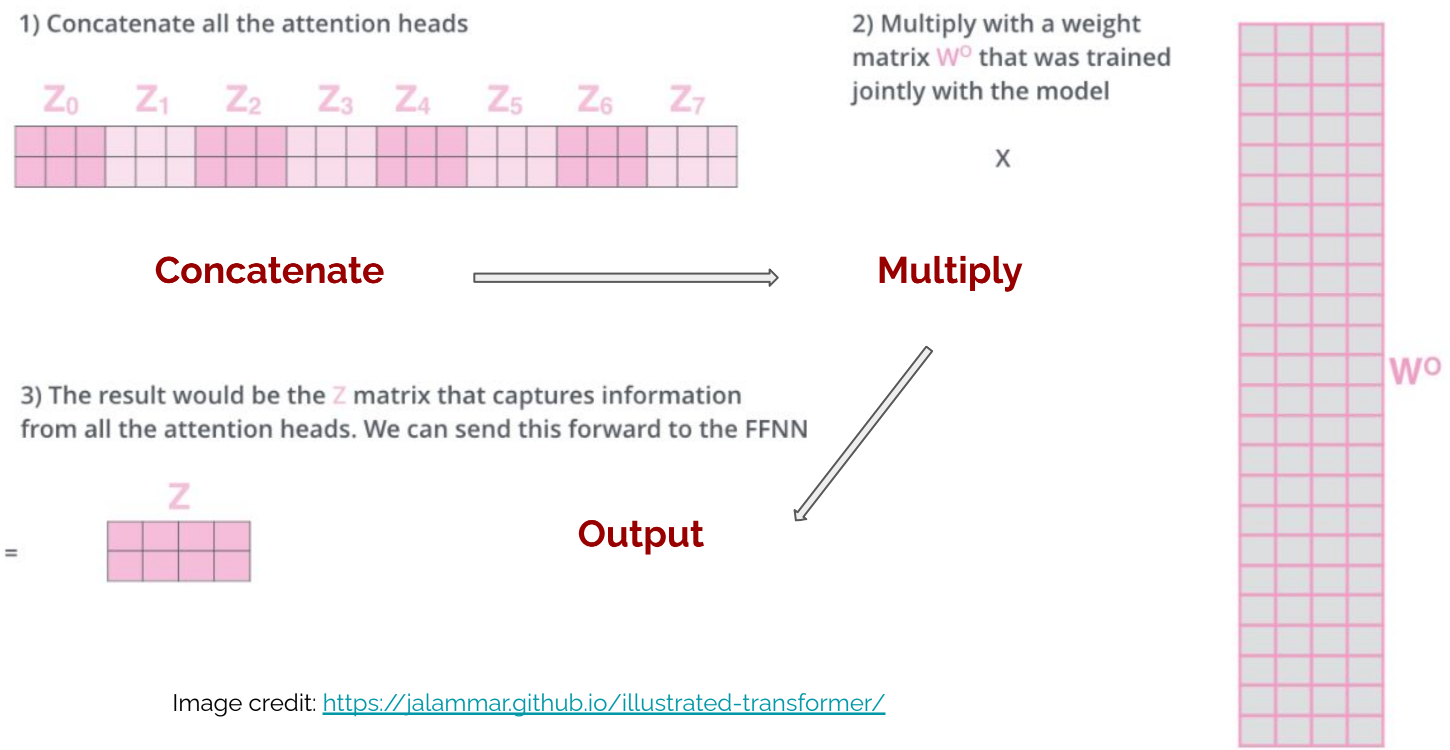


Image credit: https://jalammar.github.io/illustrated-transformer/

Multi-head attention

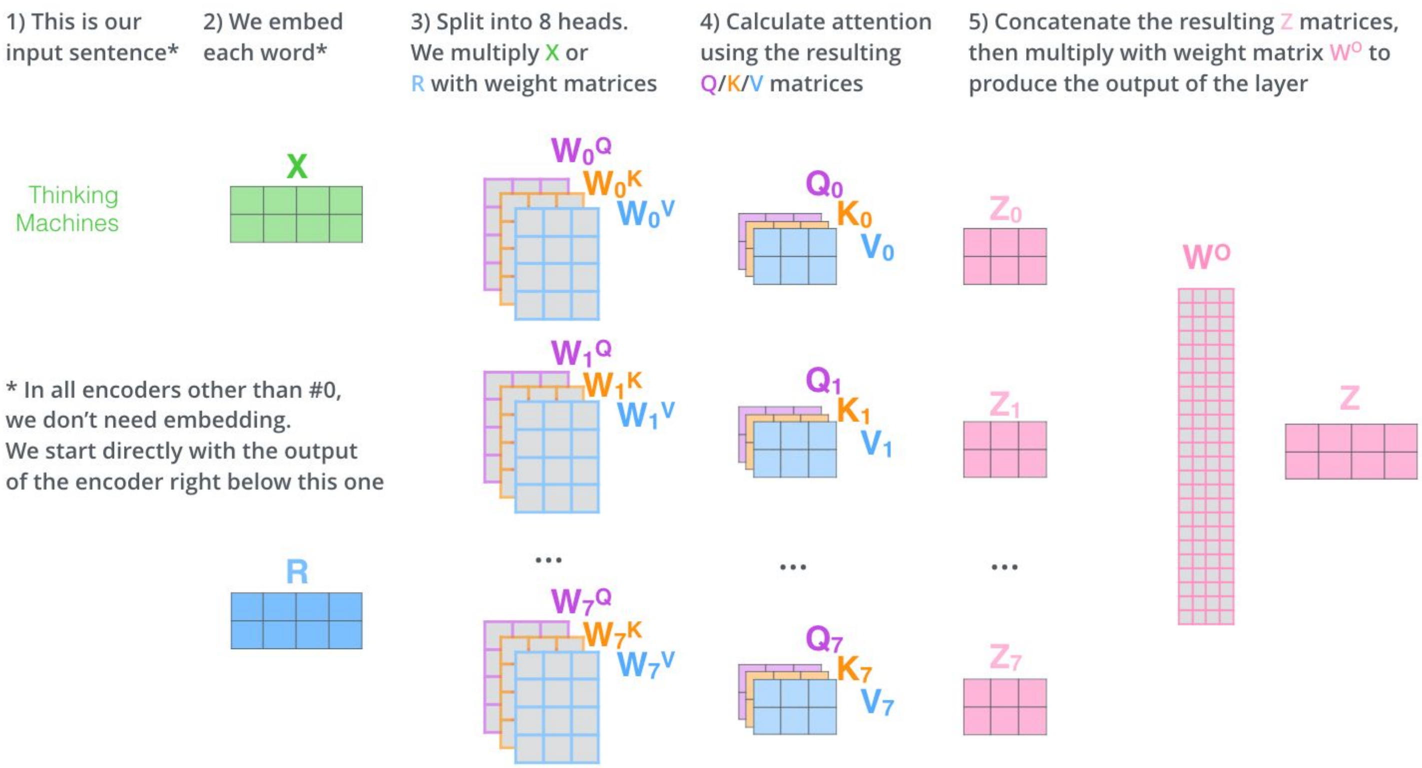
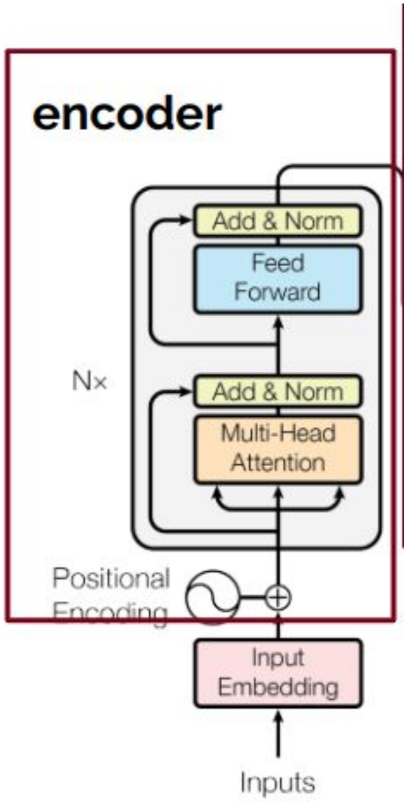


Image credit: https://jalammar.github.io/illustrated-transformer/

Positional encoding



Does the input order matter or not?

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$$\text{output} = \text{softmax}(QK^\top / \sqrt{d_k})V$$

Positional encoding to break the **order invariance**

- Idea: a positional vector to (hopefully) encode the position information

  E.g., $\quad X_p = X + P, \text{ or } X_p = [X, P]$

- P can be pre-defined, or made learnable

Sinusoidal positional encoding

L: sequence length
d: embedding dimension

$$\mathrm{PE}(i, \delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

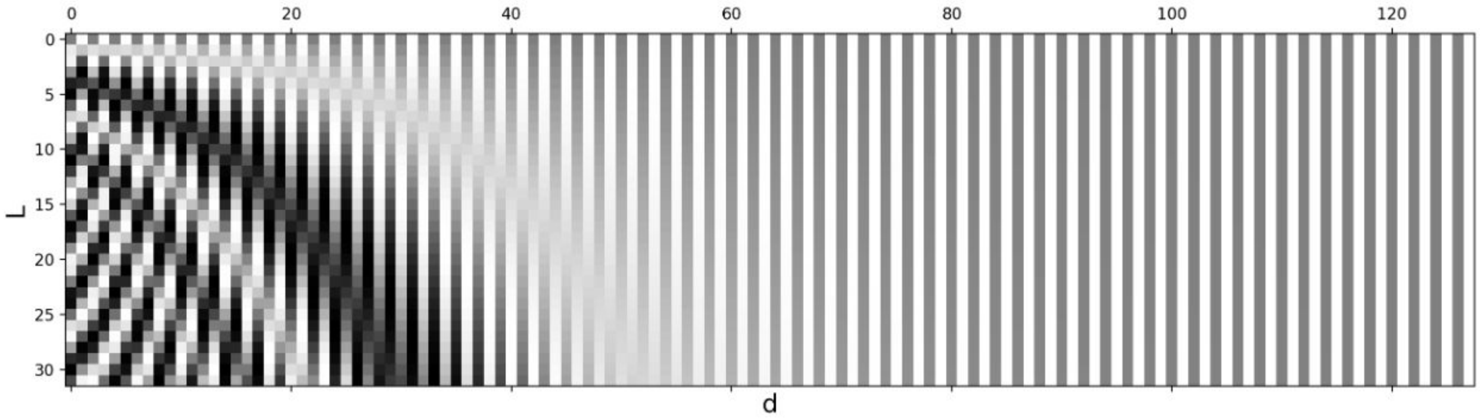


Image credit: https://lilianweng.github.io/posts/2020-04-07-the-transformer-family/

Decoder

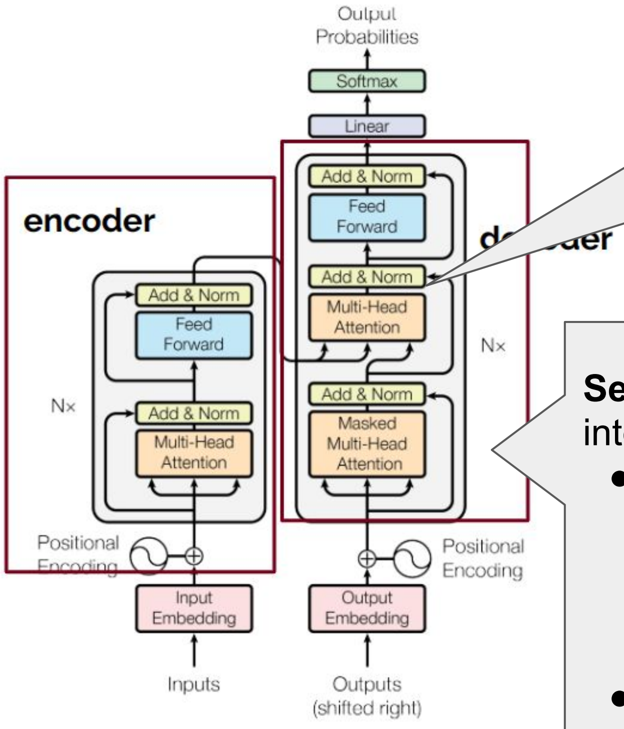
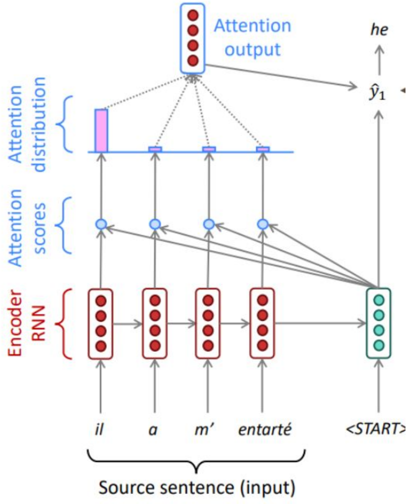


Figure 1: The Transformer - model architecture.

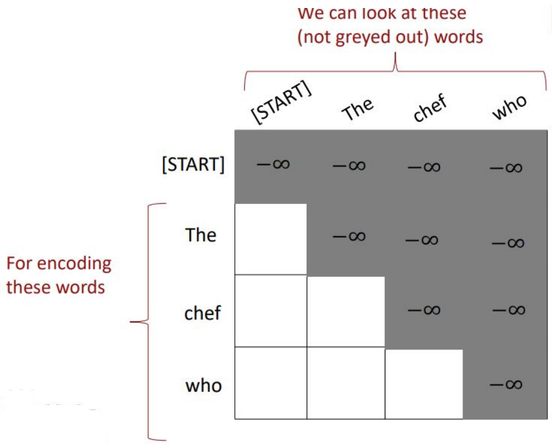
Cross-attention (to model the interaction between the encoder key-values and the current decoder query)

Self-attention (to model the interaction within itself)

- Respect the sequential nature (e.g., language modeling, assuming access to the future is cheating!)
- Masked out future tokens



(Credit: Stanford CS231N)



Attention matrix

Computation

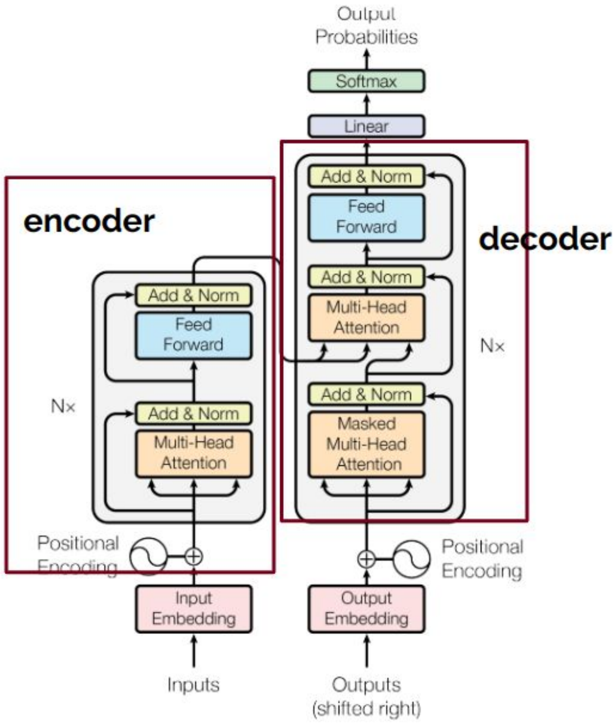


Figure 1: The Transformer - model architecture.

What's the total computation?

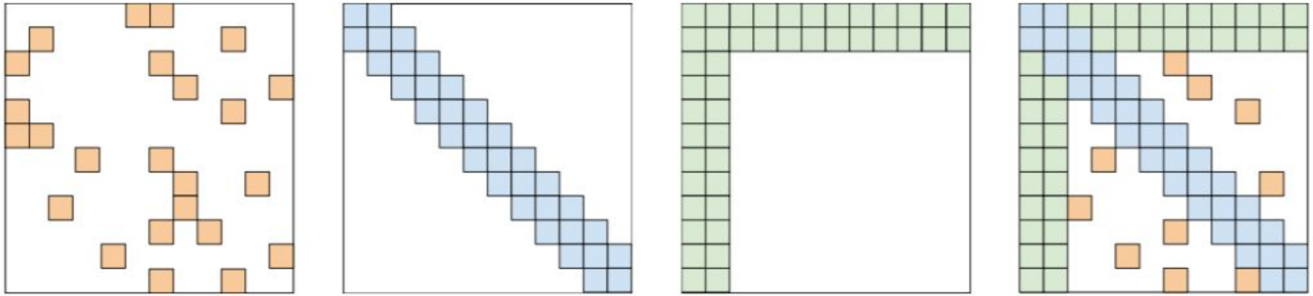
$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

$$\text{output} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\intercal/\sqrt{d_k})\boldsymbol{V}$$

$$O(T^2 d)$$

Quadratic computation vs. linear computation in RNNs

Computation



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Idea; building in sparsity https://arxiv.org/abs/2007.14062

Do Transformer Modifications Transfer Across Implementations and Applications?

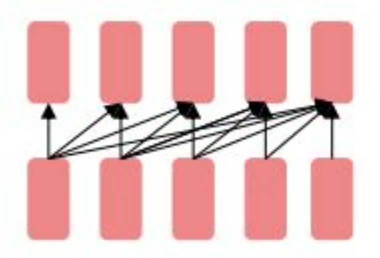
Sharan Narang* Hyung Won Chung Yi Tay William Fedus
Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel
Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li
Nan Ding Jake Marcus Adam Roberts Colin Raffel†

But not much consistent improvement so far
https://arxiv.org/abs/2102.11972

Pretraining

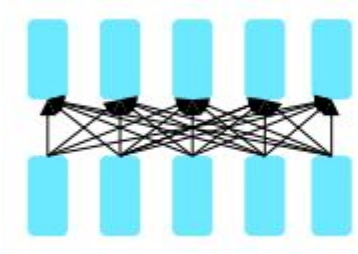
Pretraining + finetuning pipeline is standard in modern NLP/CV and many applied areas

Decoder



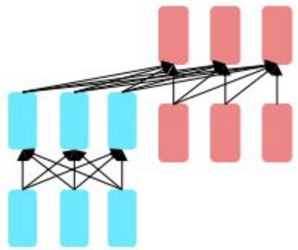
Good for content generation
(e.g., GPT-2, GPT-3)

Encoder



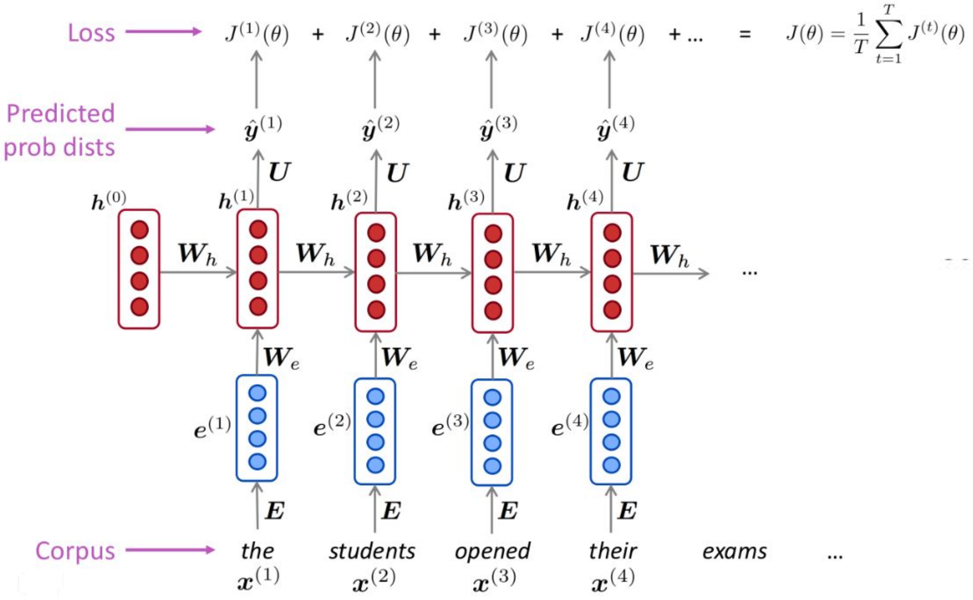
Good for feature extraction
(e.g., X-**BERT**)

Encoder-decoder

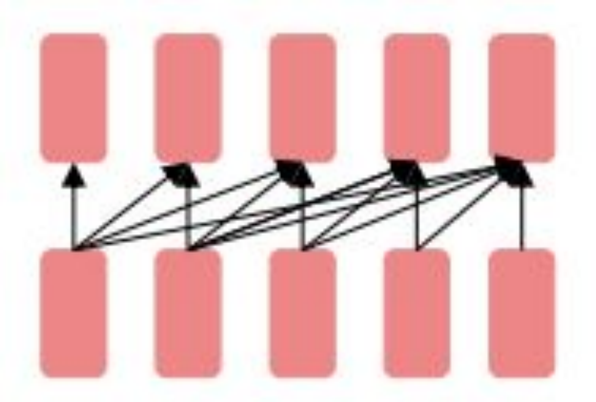


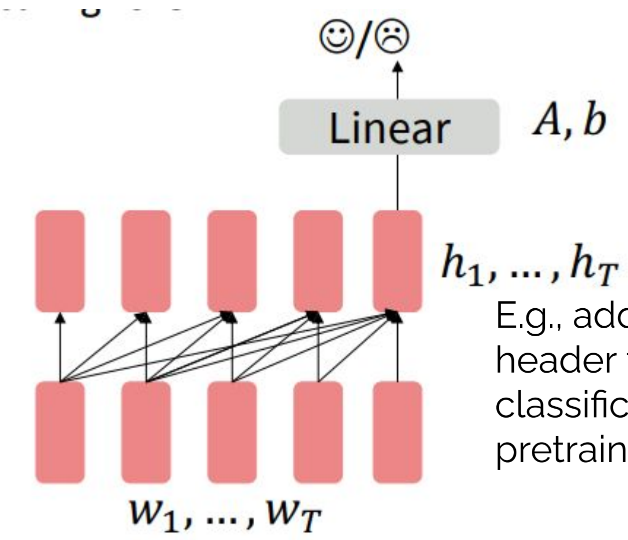
Good for everything?
(e.g., Transformer, T5)

Decoder pretraining



(Credit: Stanford CS224N)

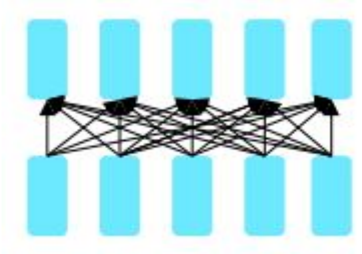




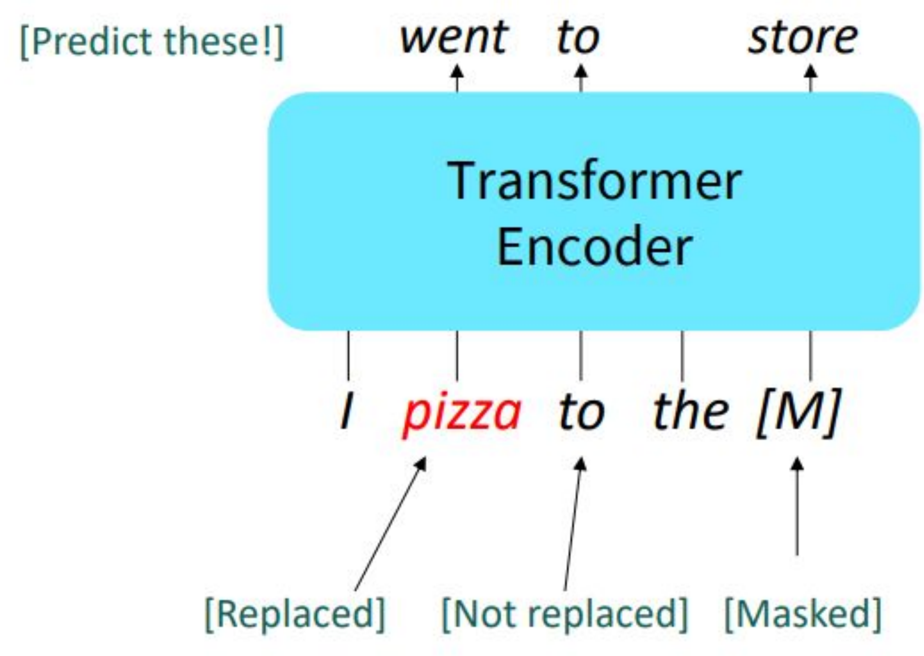
E.g., add prediction header for sentiment classification after pretraining

Can be pretrained on language modeling, i.e., model $\mathbb{P}\left[x^{(1)}, \ldots, x^{(T)}\right]$ or $\mathbb{P}\left[x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}\right]$

Encoder pretraining



Both past and future information available due to the bi-direction modeling; not ideal for language modeling



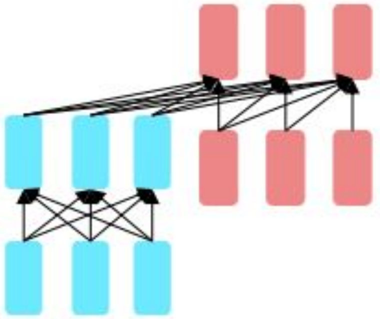
Idea: create corruption and predict the right things

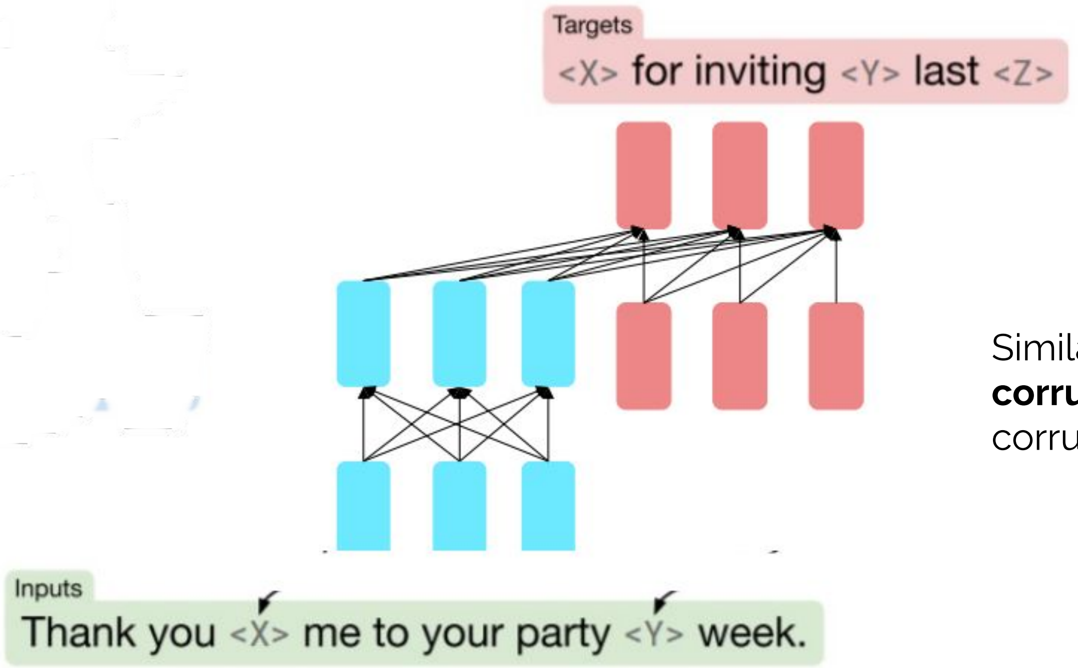
- Masked out words (i.e., missing words)
- Noisy words (randomly replaced)

Self-supervised learning! (pretext tasks)

https://arxiv.org/abs/1810.04805

Encoder-decoder pretraining





Similar to pretraining encoder, **corruption removal**! (called span corruption)

https://arxiv.org/abs/1910.10683