EECS 487: Introduction to Natural Language Processing

Instructor: Prof. Lu Wang

Computer Science and Engineering

University of Michigan

Webpage: web.eecs.umich.edu/~wangluxy

Today's Outline

- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

[Some slides are adopted from Stanford's cs224n]

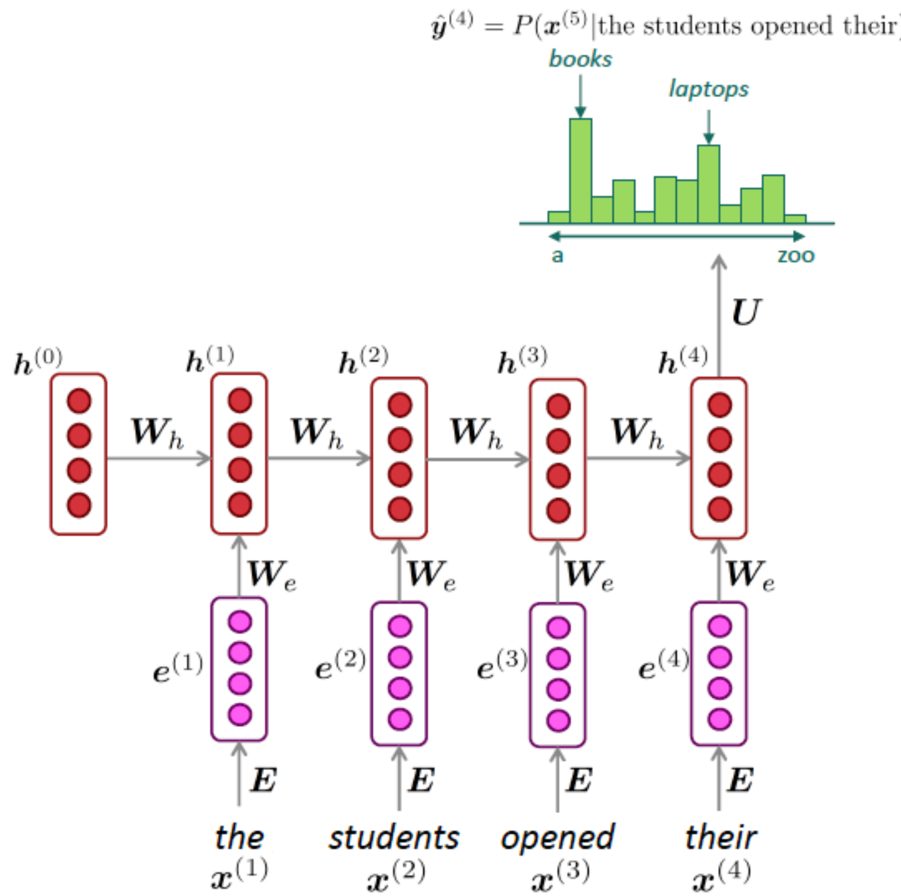
RNN Pros and Cons

RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

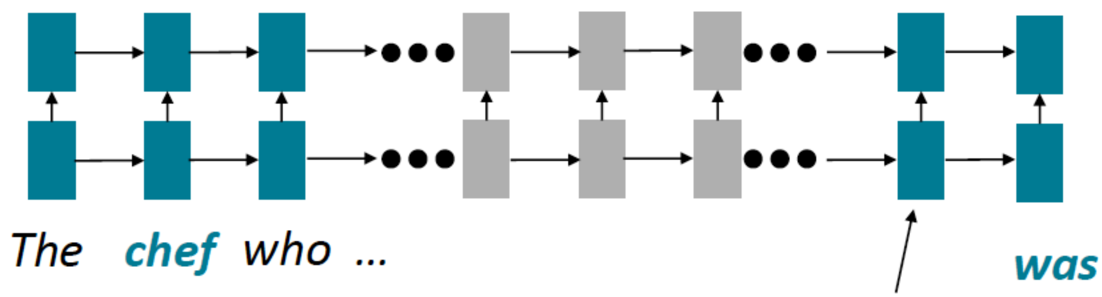
- Recurrent computation is slow
- In practice, difficult to access information from many steps back



RNN: Linear interaction distance

O(sequence length) steps for distant word pairs to interact means:

- Hard to learn long-distance dependencies (because gradient problems!)
- Linear order of words is “baked in”; we already know linear order isn’t the right way to think about sentences…

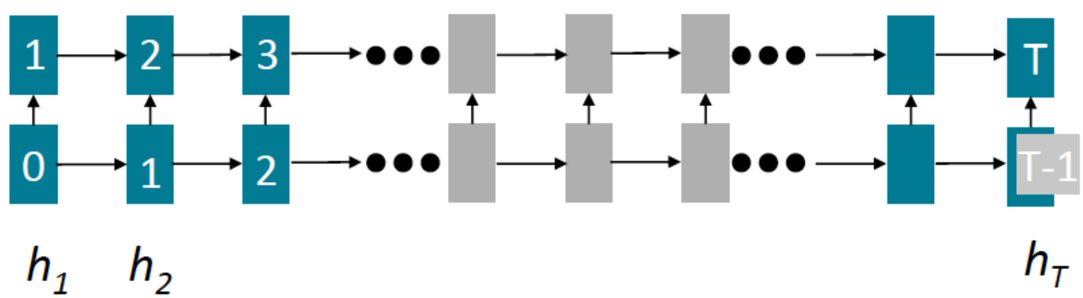


Info of ***chef*** has gone through O(sequence length) many layers!

RNN: Lack of parallelizability

Forward and backward passes have **O(sequence length)** unparallelizable operations

- GPUs can perform a bunch of independent computations at once!
- But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
- Inhibits training on very large datasets!



Numbers indicate min # of steps before a state can be computed

Today’s Outline

- State-of-the-art model: Transformer
    - Self-attentions
    - Position embeddings and adding nonlinearities
    - Tricks: Residual connections and layer normalization
    - Other aspects: multi-head attentions, cross-attention
    - Variants of Transformers
- Pretrained large Transformers
    - Pretraining → finetuning
    - Pretraining for three types of architectures

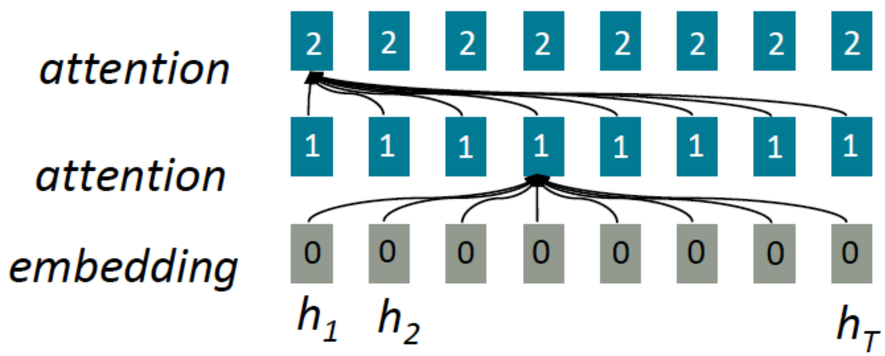
Attentions

Attention operates on **queries**, **keys**, and **values**.

Attentions

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**

- Number of unparallelizable operations does not increase sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!



All words attend to all words in previous layer; most arrows here are omitted

Self-attention

- We have some **queries** $q_1, q_2, \dots, q_T$. Each query is $q_i \in \mathbb{R}^d$
- We have some **keys** $k_1, k_2, \dots, k_T$. Each key is $k_i \in \mathbb{R}^d$
- We have some **values** $v_1, v_2, \dots, v_T$. Each value is $v_i \in \mathbb{R}^d$

In **self-attention**, the queries, keys, and values are drawn from the same source.

- For example, if the output of the previous layer is $x_1, \dots, x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)

The (dot product) self-attention operation is as follows:

$$e_{ij} = q_i^{\top} k_j$$

Compute **key-query** affinities

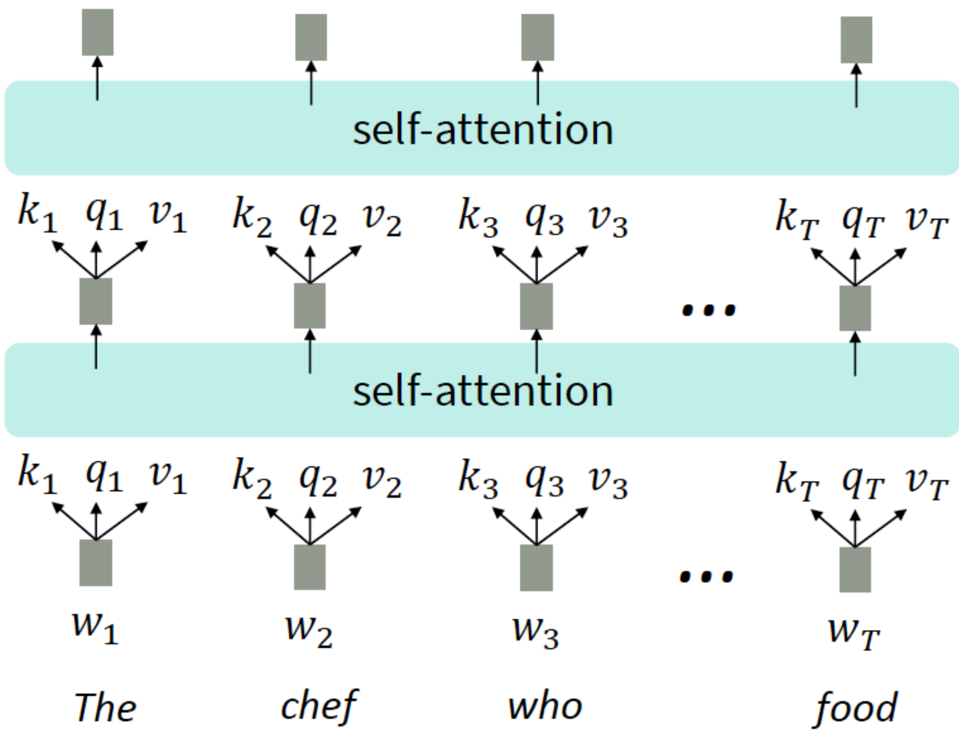
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

Compute attention weights from affinities (softmax)

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute outputs as weighted sum of **values**

Self-attention as an NLP building block



Stacked self attention blocks

Self-attention doesn't know the order of its inputs.

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn’t have an inherent notion of order!
- No nonlinearities for deep learning! It’s all just weighted averages

Today’s Outline

- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

Fixing the first self-attention problem: **sequence order**

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\ldots,T\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i$ $\tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

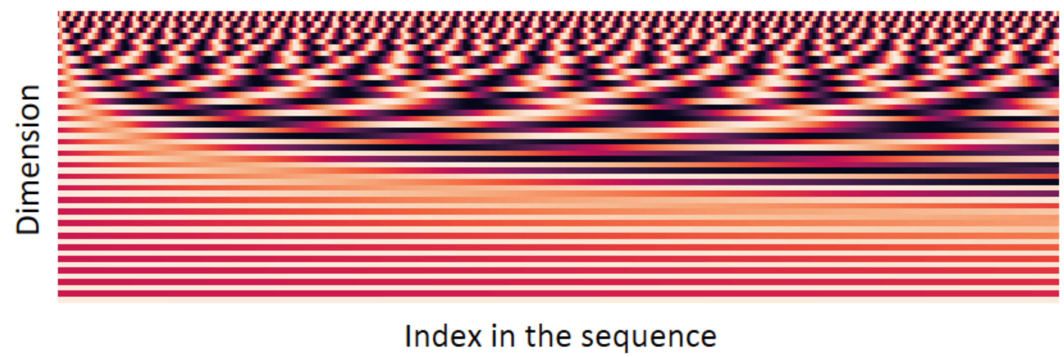
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Position representation vectors through sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros:
  - Periodicity indicates that maybe "absolute position" isn't as important
  - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
  - Not learnable; also the extrapolation doesn't really work!

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!

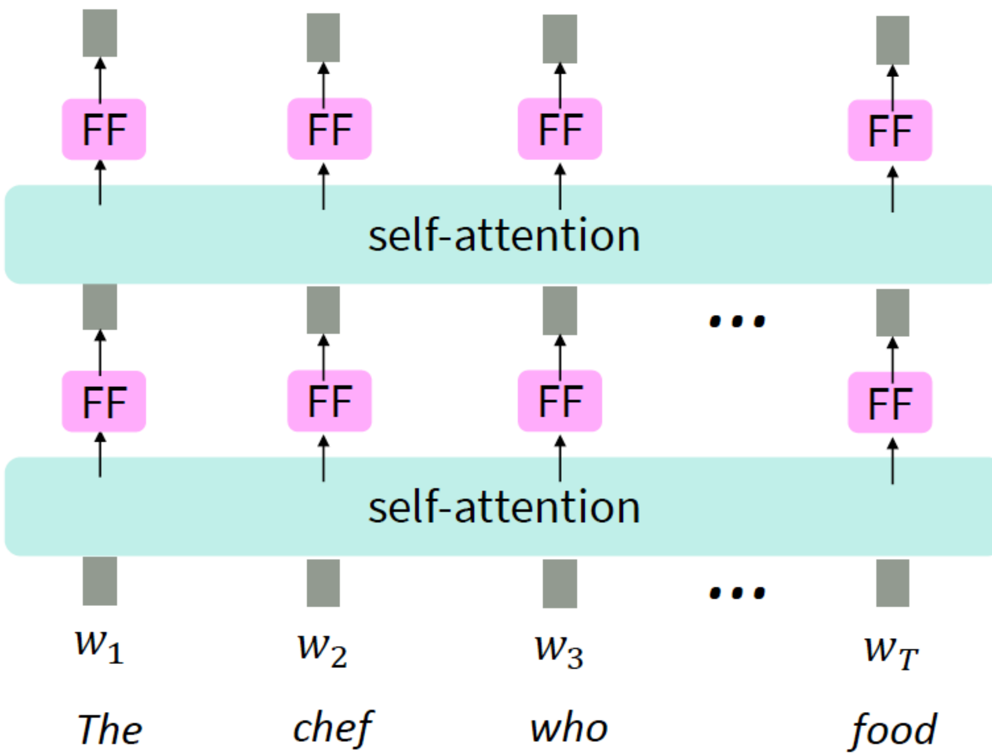
- No nonlinearities for deep learning! It's all just weighted averages

Fixing the second self-attention problem: **Adding nonlinearities in self-attention**

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors

- Easy fix: add a **feed-forward network** to post-process each output vector.

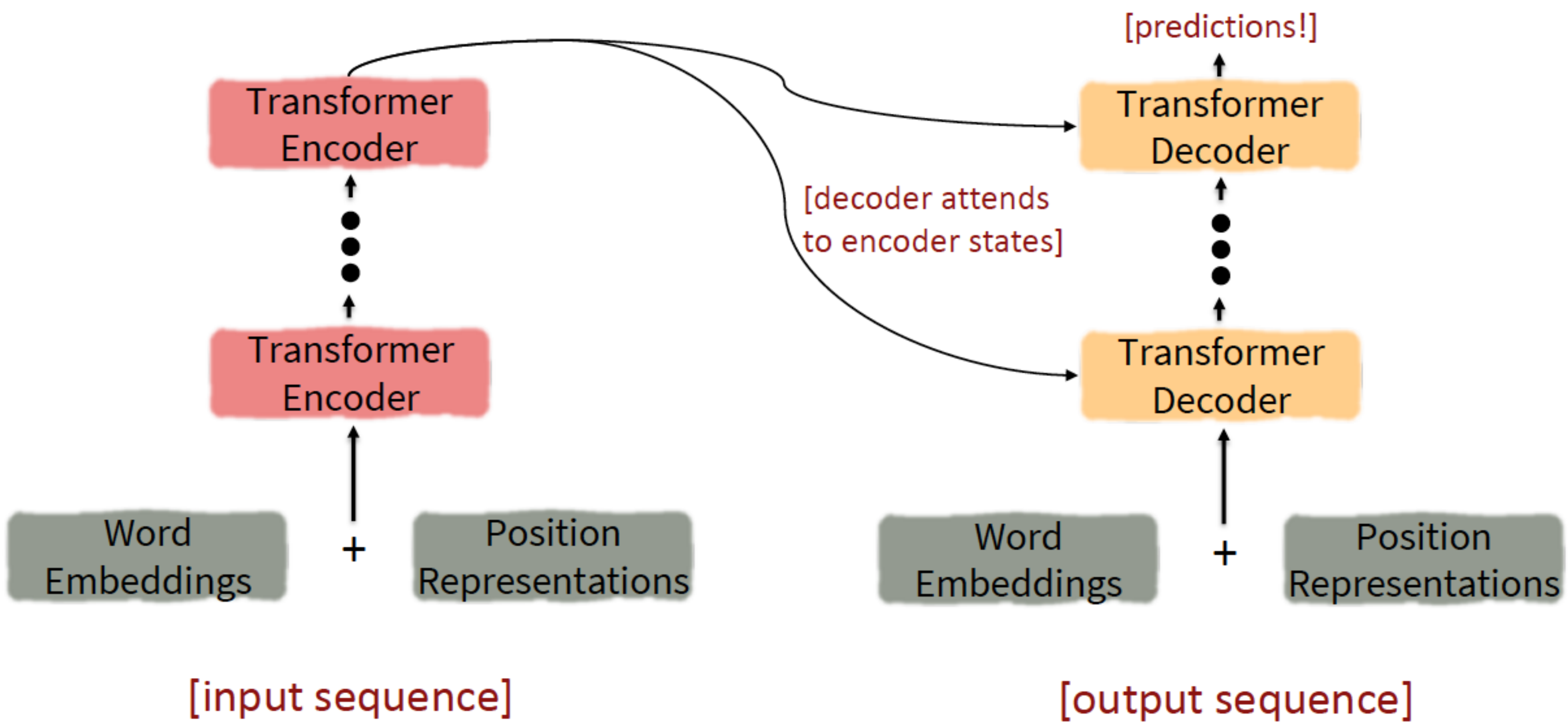
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



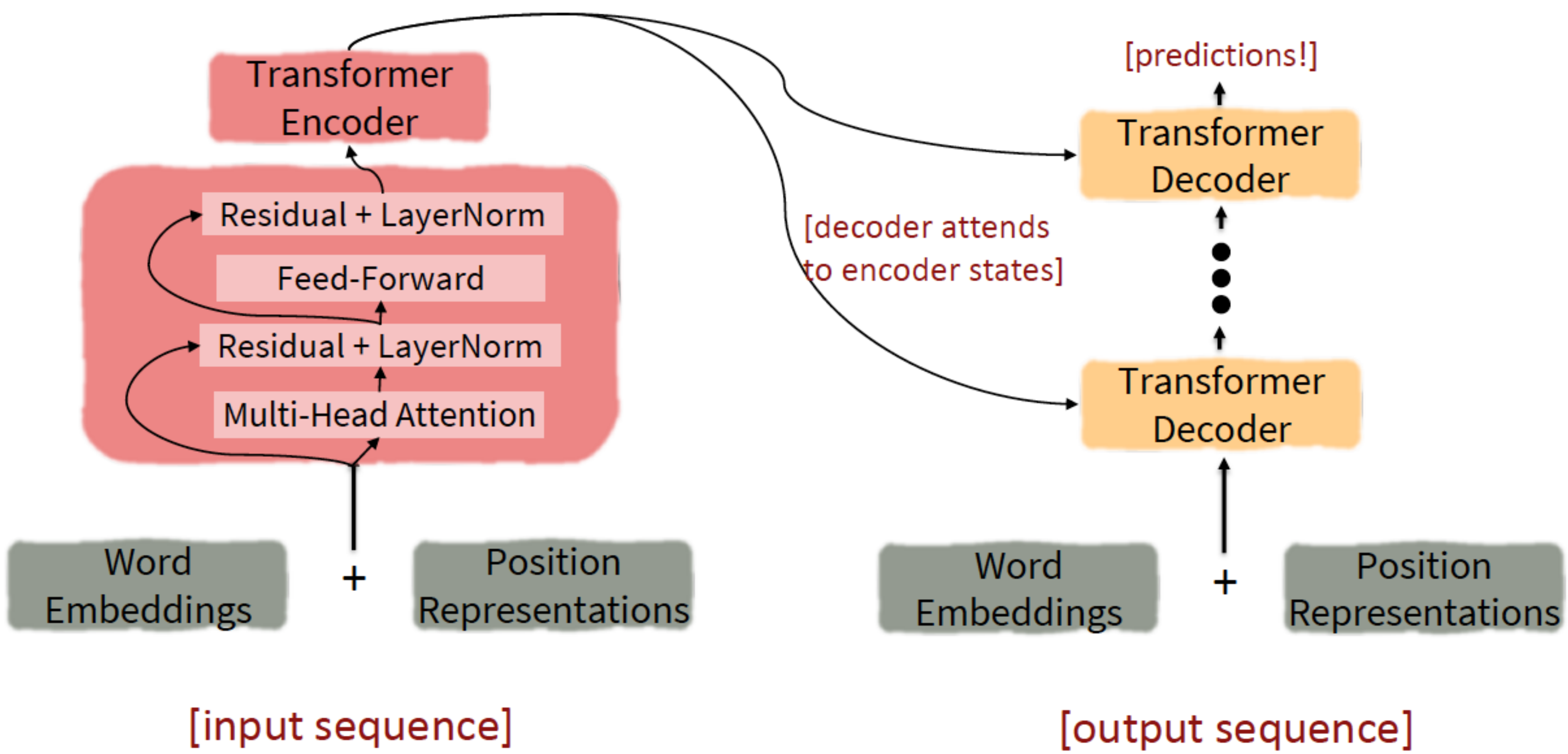
Barriers and solutions for Self-Attention as a building block

Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	→	• Easy fix: apply the same feedforward network to each self-attention output.

The Transformer Encoder-Decoder



The Transformer Encoder-Decoder

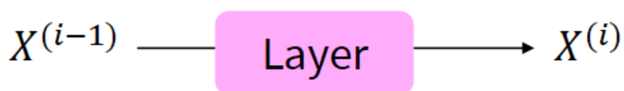


Today’s Outline

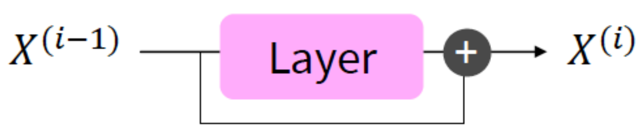
- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

The Transformer Encoder: **Residual connections**

- **Residual connections** are a trick to help models train better.
    - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)



- We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn “the residual” from the previous layer)



The Transformer Encoder: **Layer normalization**

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} \left(x_j - \mu\right)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

Today's Outline

- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

The Transformer Encoder: **Multi-headed attention**

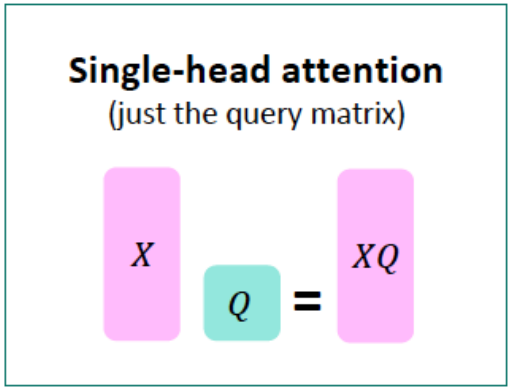
What if we want to look in multiple places in the sentence at once?

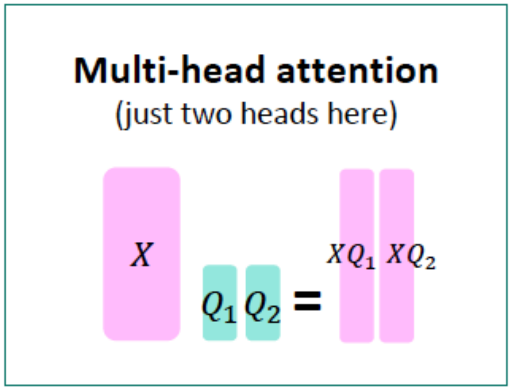
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(X Q_\ell K_\ell^\top X^\top\right) * X V_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = Y[\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$

- Each head gets to "look" at different things, and construct value vectors differently.

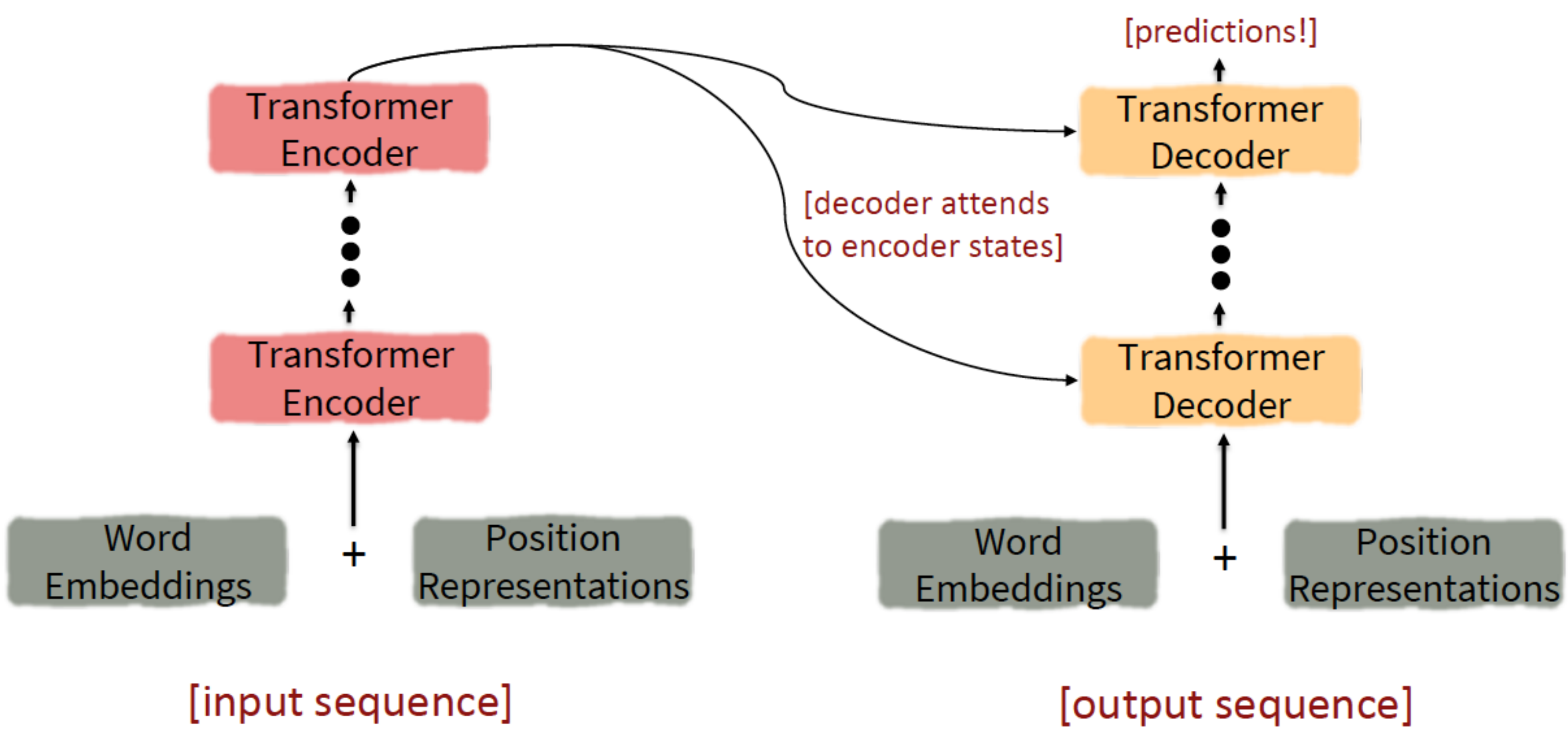
The Transformer Encoder: **Multi-headed attention**

What if we want to look in multiple places in the sentence at once?

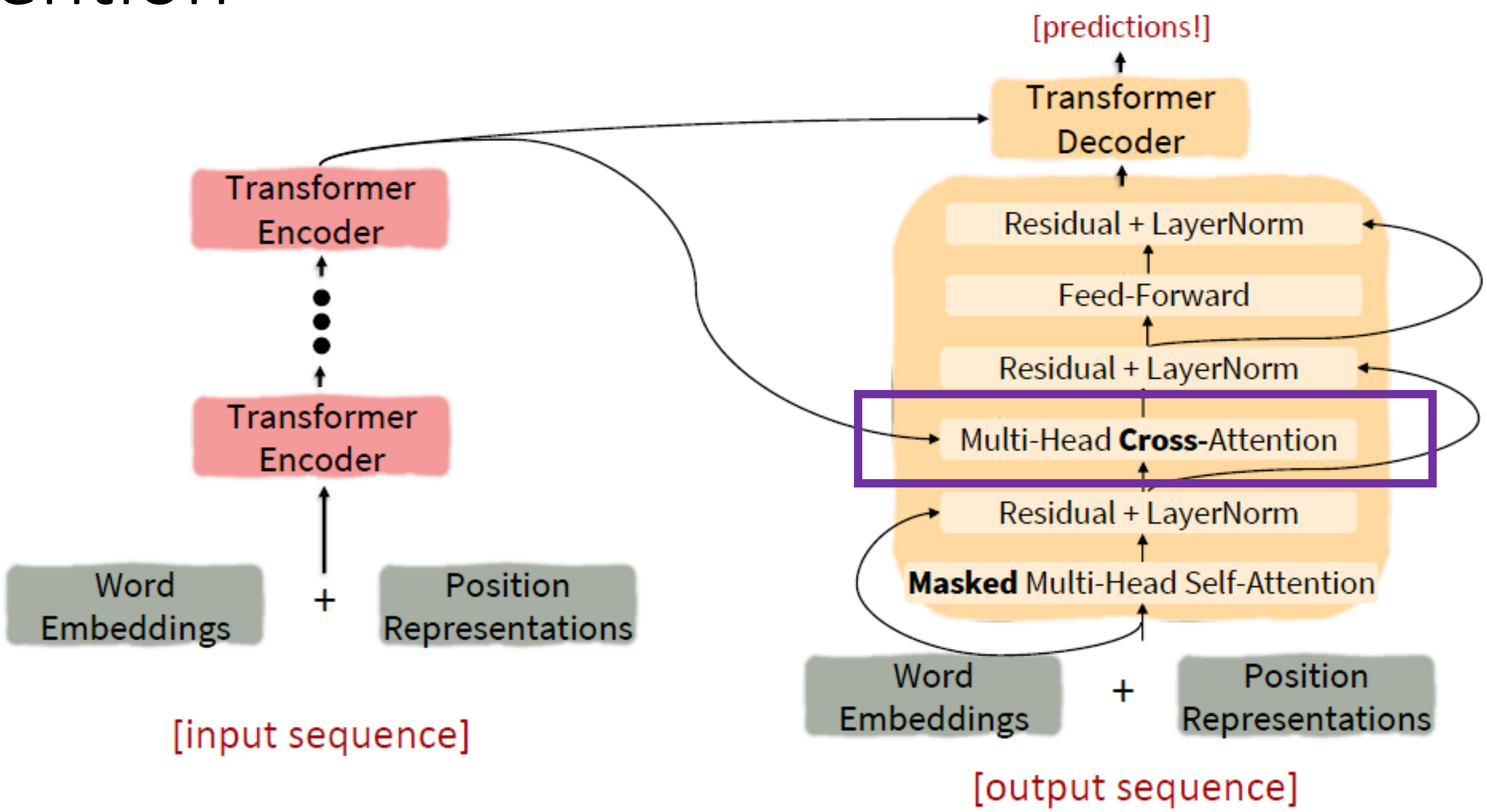




The Transformer Encoder-Decoder



The Transformer Encoder-Decoder: Cross-attention



Today's Outline

- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

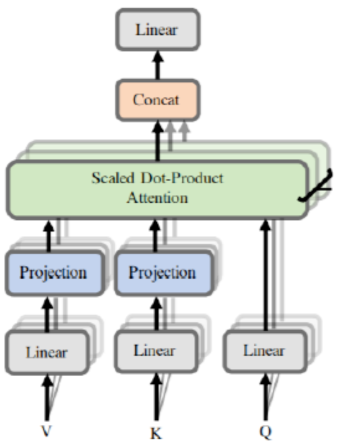
Drawback 1

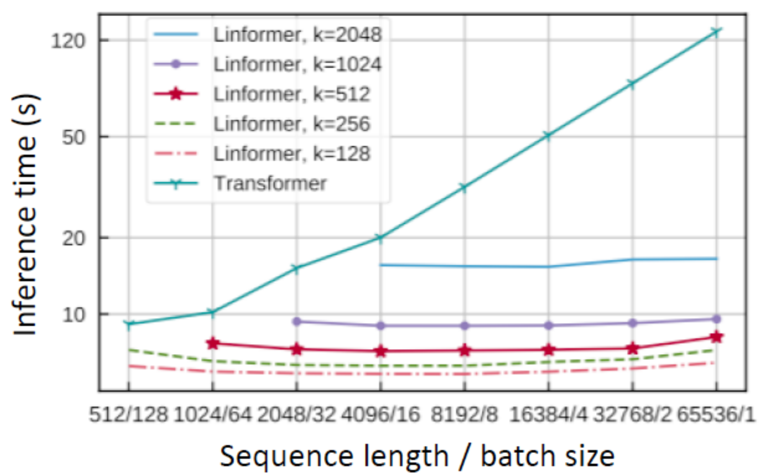
- Quadratic compute in self-attention
  - Computing all pairs of interactions means our computation grows quadratically with the sequence length!

Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*
- For example, **Linformer** [Wang et al., 2020]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys

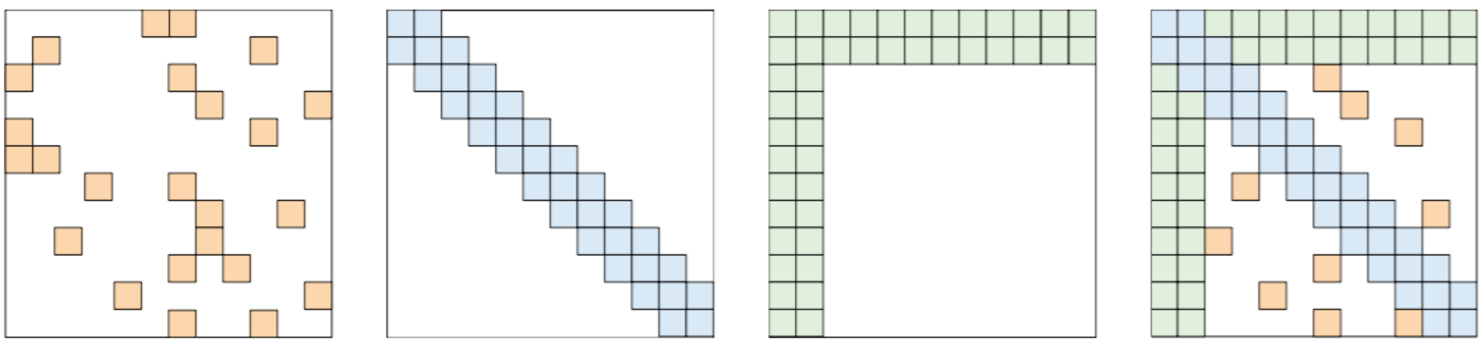




Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the* $O(T^2)$ *all-pairs self-attention cost?*
- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



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

Drawback 2

- Position representations
  - Relative linear position attention [Shaw et al. 2018]
  - Dependency syntax based position [Wang et al. 2019]

Today's Outline

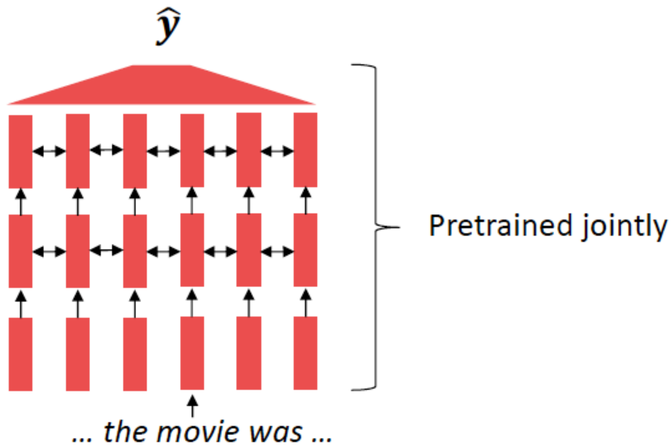
- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

Pretraining

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and then train the model to reconstruct those parts.

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **probability distributions** over language that we can sample from



[This model has learned how to represent entire sentences through pretraining]

Pretraining with reconstruction

I put ___ fork down on the table.

I went to the ocean to see the fish, turtles, seals, and _____.

Overall, the value I got from the two hours watching
it was the sum total of the popcorn and the drink.
The movie was ___.

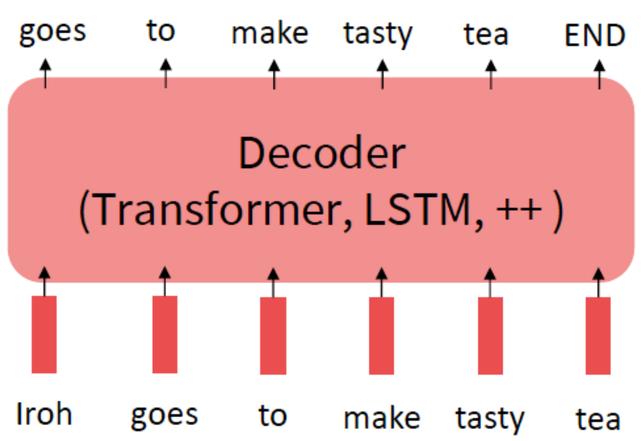
Pretraining through language modeling [Dai and Le, 2015]

Recall the **language modeling** task:

- Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

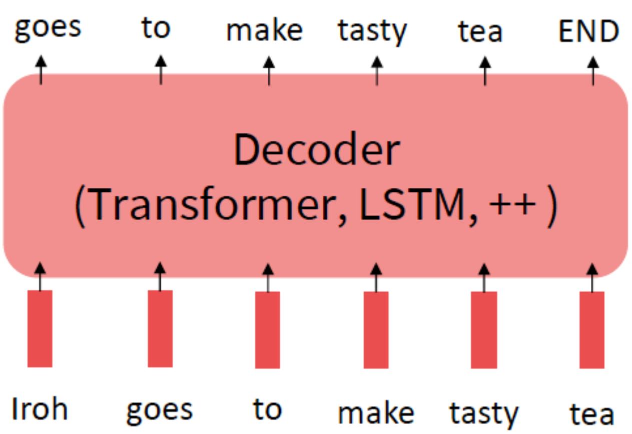
- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.



The Pretraining → Finetuning Paradigm

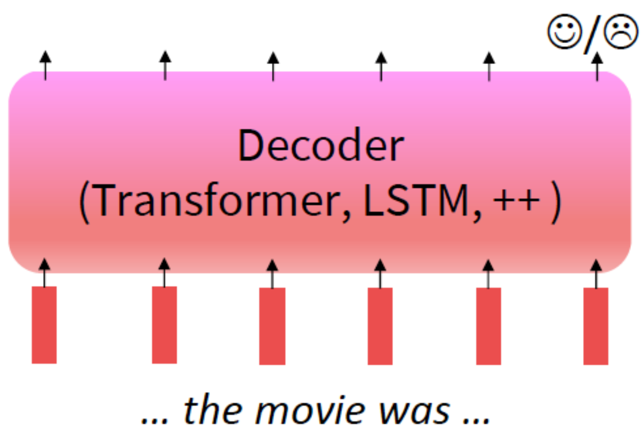
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!

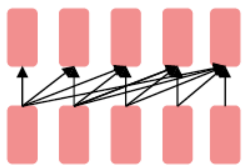


Today's Outline

- State-of-the-art model: Transformer
  - Self-attentions
  - Position embeddings and adding nonlinearities
  - Tricks: Residual connections and layer normalization
  - Other aspects: multi-head attentions, cross-attention
  - Variants of Transformers
- Pretrained large Transformers
  - Pretraining → finetuning
  - Pretraining for three types of architectures

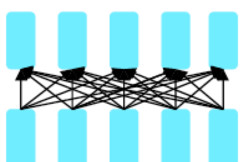
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



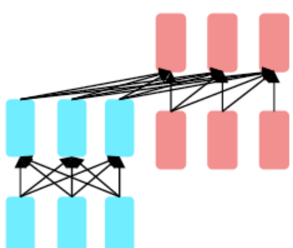
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining decoders

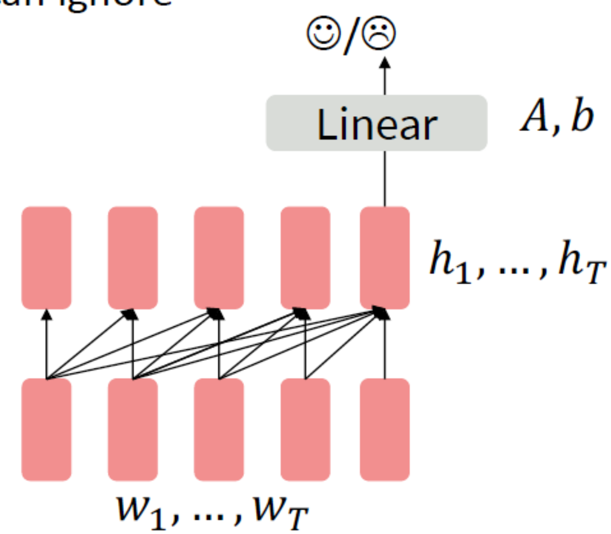
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$.

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$y \sim Ah_T + b$$

Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

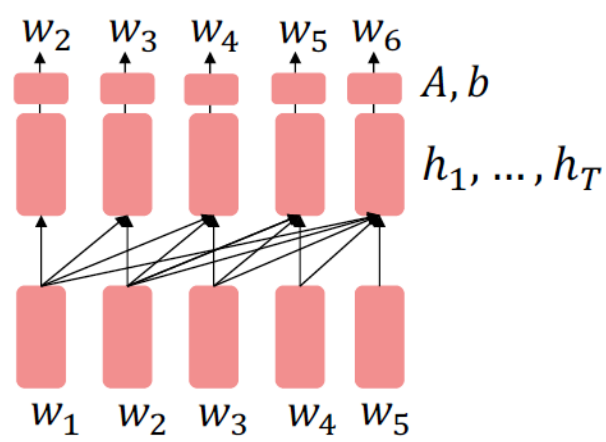
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$
\begin{aligned}
h_1, \dots, h_T &= \text{Decoder}(w_1, \dots, w_T) \\
w_t &\sim Ah_{t-1} + b
\end{aligned}
$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

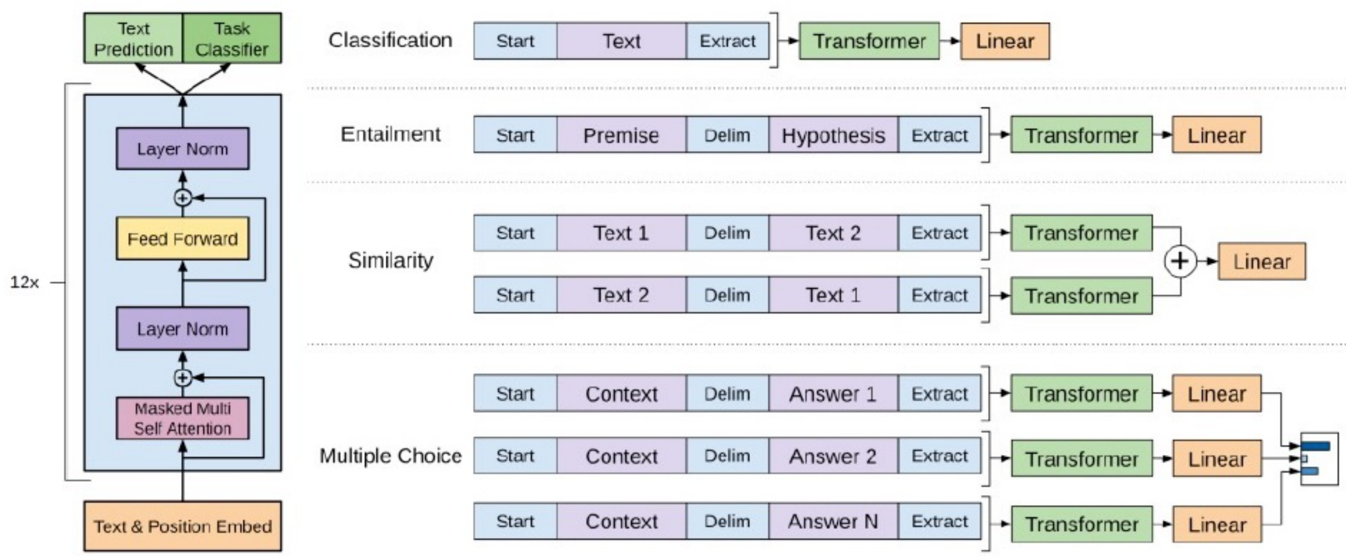
Generative Pretrained Transformer (GPT) [Radford et al., 2018]

2018’s GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

How do we format inputs to our decoder for **finetuning tasks?**



The linear classifier is applied to the representation of the [EXTRACT] token.

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

GPT results on various *natural language inference* datasets.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

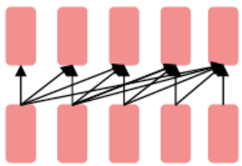
We mentioned how pretrained decoders can be used **in their capacities as language models.**

GPT-2, a larger version of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

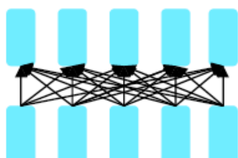
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



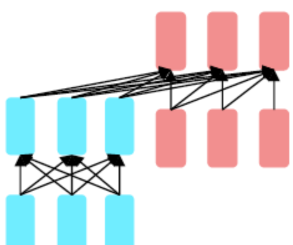
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

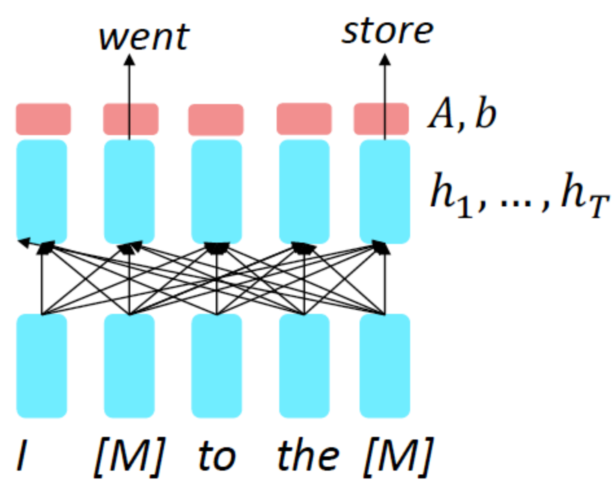
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining encoders: what pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM**.

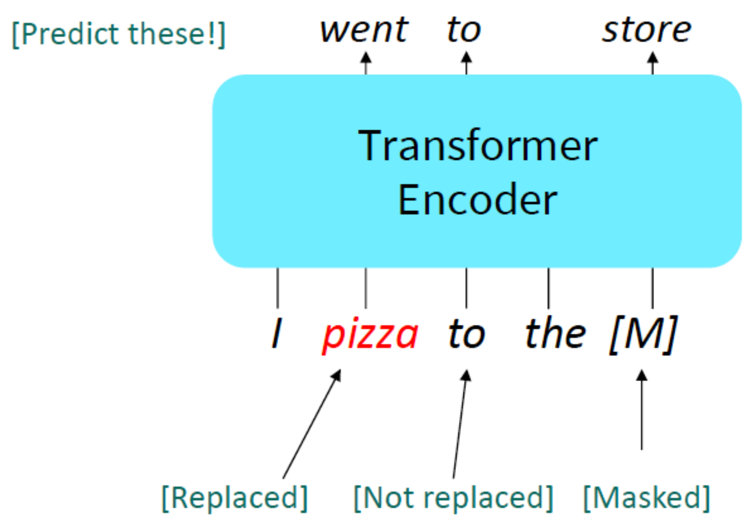


BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the “Masked LM” objective, open-sourced their model as the tensor2tensor library, and **released the weights of their pretrained Transformer (BERT)**.

Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn’t let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



BERT: Bidirectional Encoder Representations from Transformers

Details about BERT

- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”

BERT: Bidirectional Encoder Representations from Transformers

BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

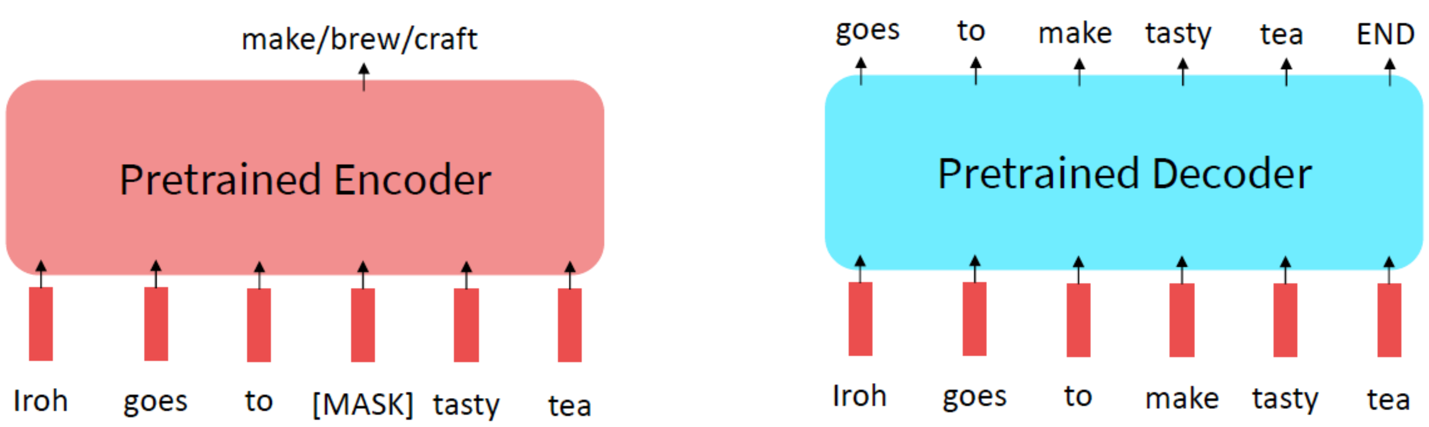
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Note that $\text{BERT}_{\text{BASE}}$ was chosen to have the same number of parameters as OpenAI GPT. [Devlin et al., 2018]

Limitations of pretrained encoders

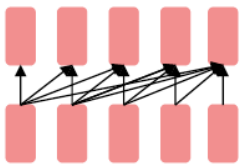
Those results looked great! Why not used pretrained encoders for everything?

If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don’t naturally lead to nice autoregressive (1-word-at-a-time) generation methods.



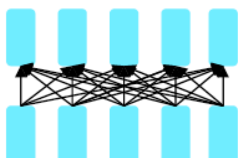
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



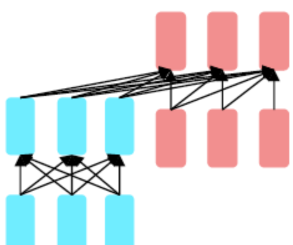
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

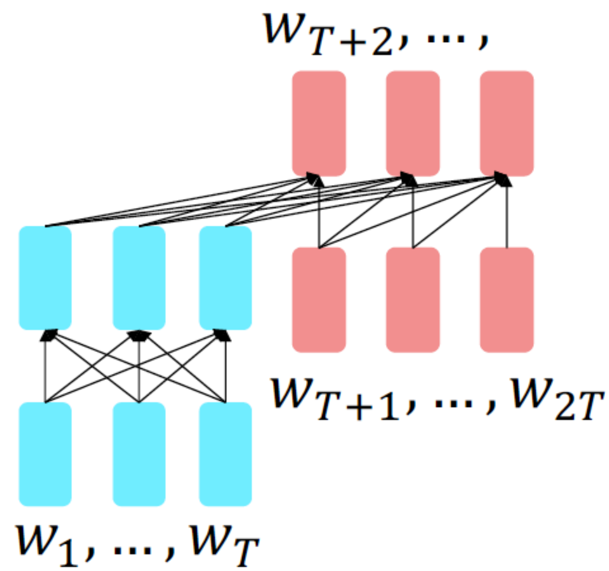
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining encoder-decoders: what pretraining objective to use?

For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$h_{T+1}, \ldots, h_2 = Decoder(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_i \sim Aw_i + b, i > T$$

The **encoder** portion benefits from bidirectional context; the **decoder** portion is used to train the whole model through language modeling.

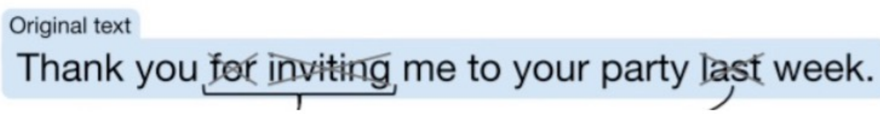


[Raffel et al., 2018]

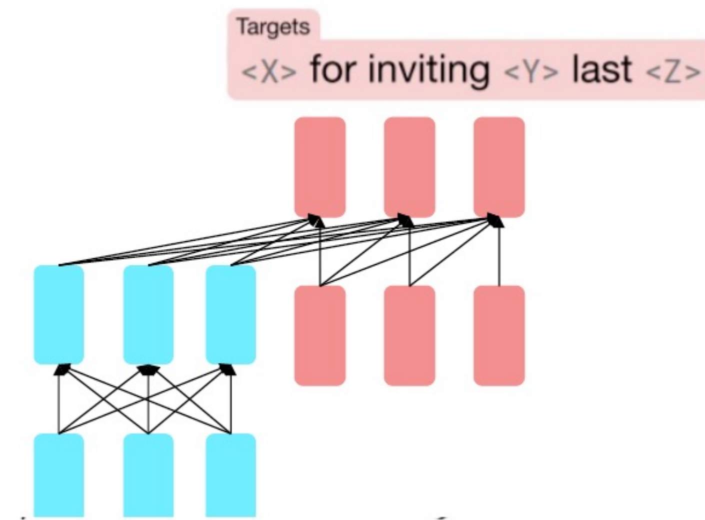
Pretraining encoder-decoders: what pretraining objective to use?

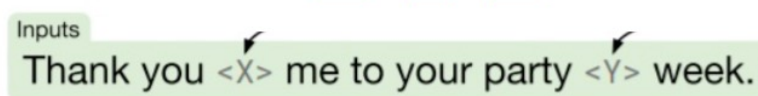
What Raffel et al., 2018 found to work best was **span corruption.** Their model: **T5**.

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!



This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.

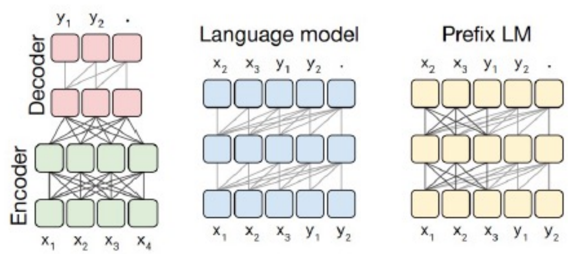




[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

Raffel et al., 2018 found encoder-decoders to work better than decoders for their tasks, and span corruption (denoising) to work better than language modeling.



Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

What to explore next?

- How to deal with their “black-box” nature?
  - Model probing, interpretability, etc
- How to improve reasoning with large models?
  - Prompt-based or instruction-based methods
- How to effectively store and retrieve knowledge from large models?
  - Output is more factual and reliable
- How to ensure the models are safe and trustworthy?
- What else these large models can do?