Natural Language Processing with Deep Learning

CS224N/Ling284

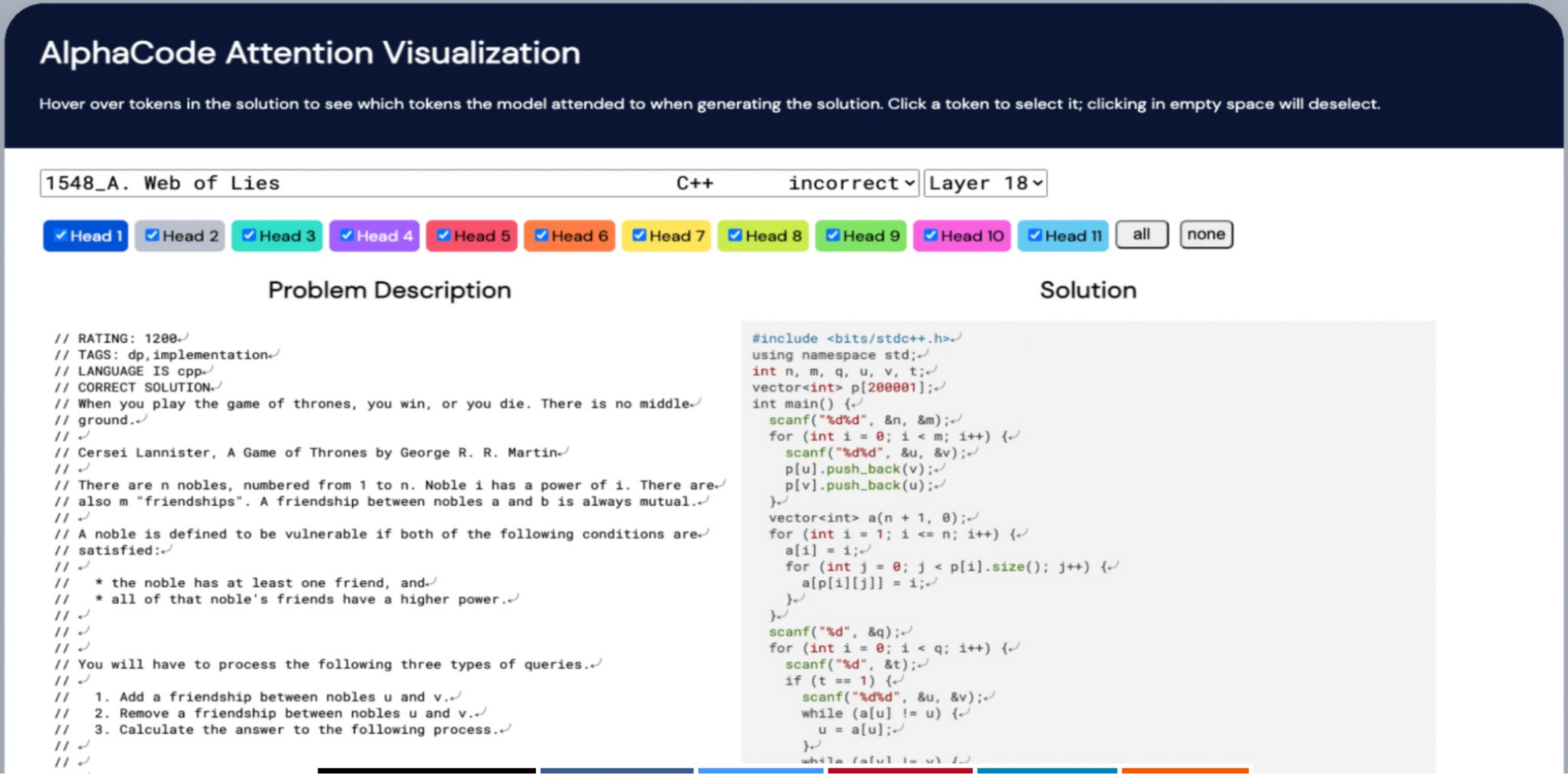
Anna Goldie

Lecture 10: Pretraining

Slides coauthored with John Hewitt

Breaking (Transformer) News!

AlphaCode (a pre-trained Transformer-based code generation model) achieved a top 54.3% rating on Codeforces programming competitions!



[Li et al., 2022]

More Breaking (Transformer) News!

Pre-Trained Transformer-Based theorem prover sets new state-of-the-art (41.2% vs. 29.3%) on a collection of challenging math Olympiad questions (miniF2F)!

PROBLEM 1

Adapted from AMC12 2000 Problem 5

Prove that if $|x-2|=p$, where $x<2$, then $x-p=2-2p$.

FORMAL INFORMAL

```
theorem amc12_2000_p5       -- ← theorem name
  (x p : ℝ)                 -- ← the statement we want
  (h₀ : x < 2)              --   to prove
  (h₁ : abs (x - 2) = p) :
  x - p = 2 - 2 * p :=
begin                       -- ← formal proof starts here
  -- This first tactic requires that the prover invent
  -- the term: `abs (x - 2) = -(x - 2)`.
  have h₂ : abs (x - 2) = -(x - 2), {
    apply abs_of_neg,
    linarith,
  },
  rw h₁ at h₂,
  -- At this stage the remaining goal to prove is:
  -- `x - p = 2 - 2 * p` knowing that `p = -(x - 2)`.
  linarith,
end
```

[Polu et al., 2022]

Lecture Plan

1. Quick review of Transformer model
2. Brief note on subword modeling
3. Motivating model pretraining from word embeddings
4. Model pretraining three ways
    1. Decoders
    2. Encoders
    3. Encoder-Decoders
5. Very large models and in-context learning

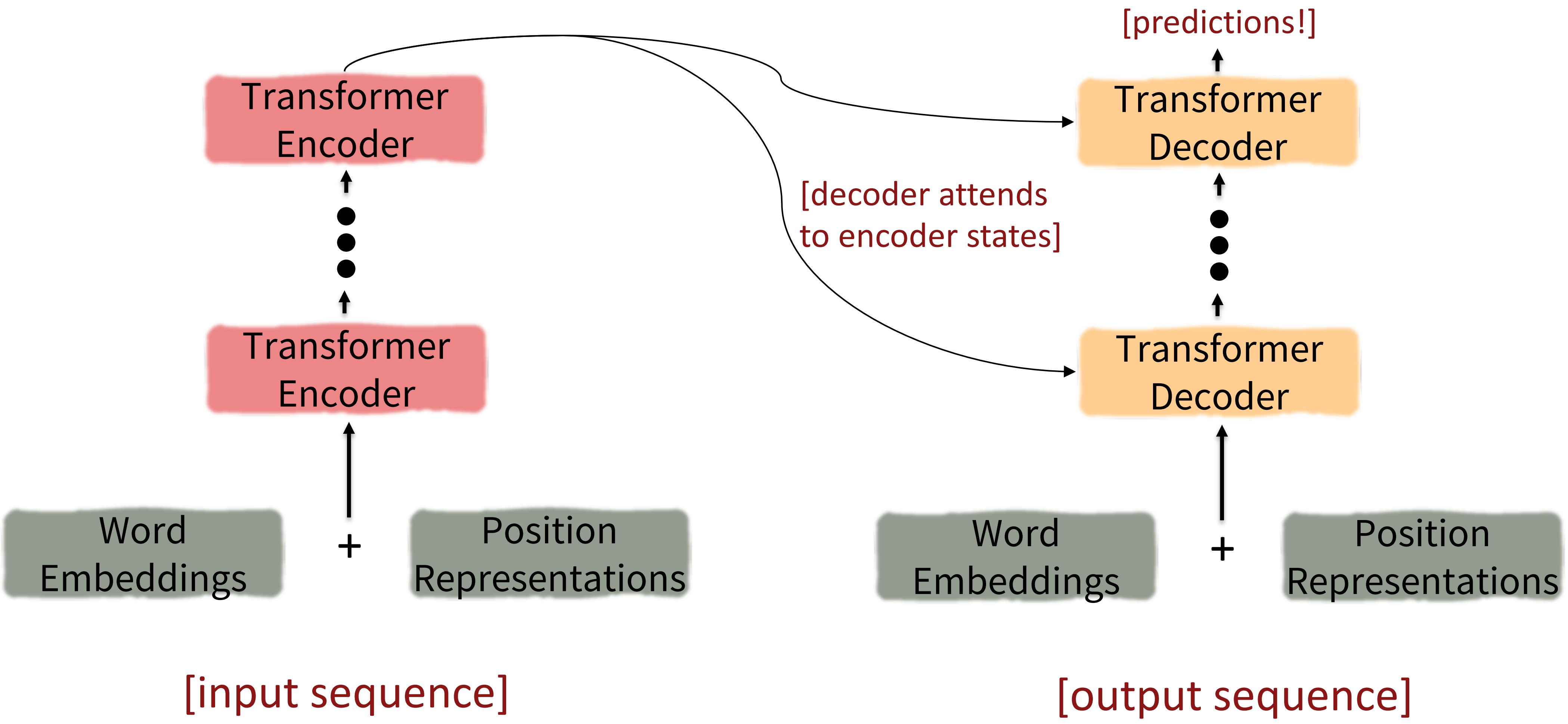
Reminders:

Assignment 5 is out today! It covers Lecture 9 (Tuesday) and Lecture 10 (Today)!

Hugging Face Transformers Tutorial Session on Friday 1:30-2:30pm (Thornton 102 and recorded)!

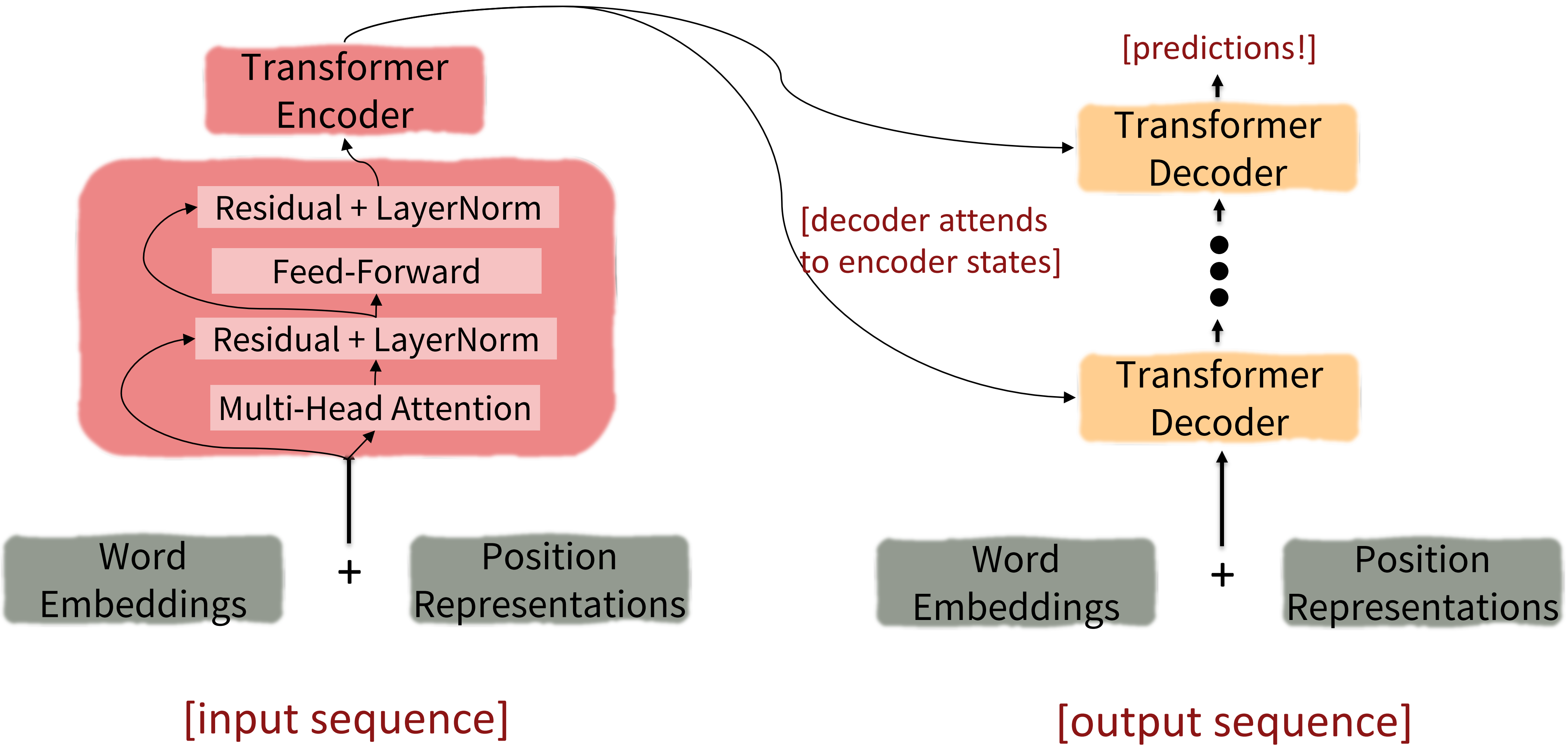
The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



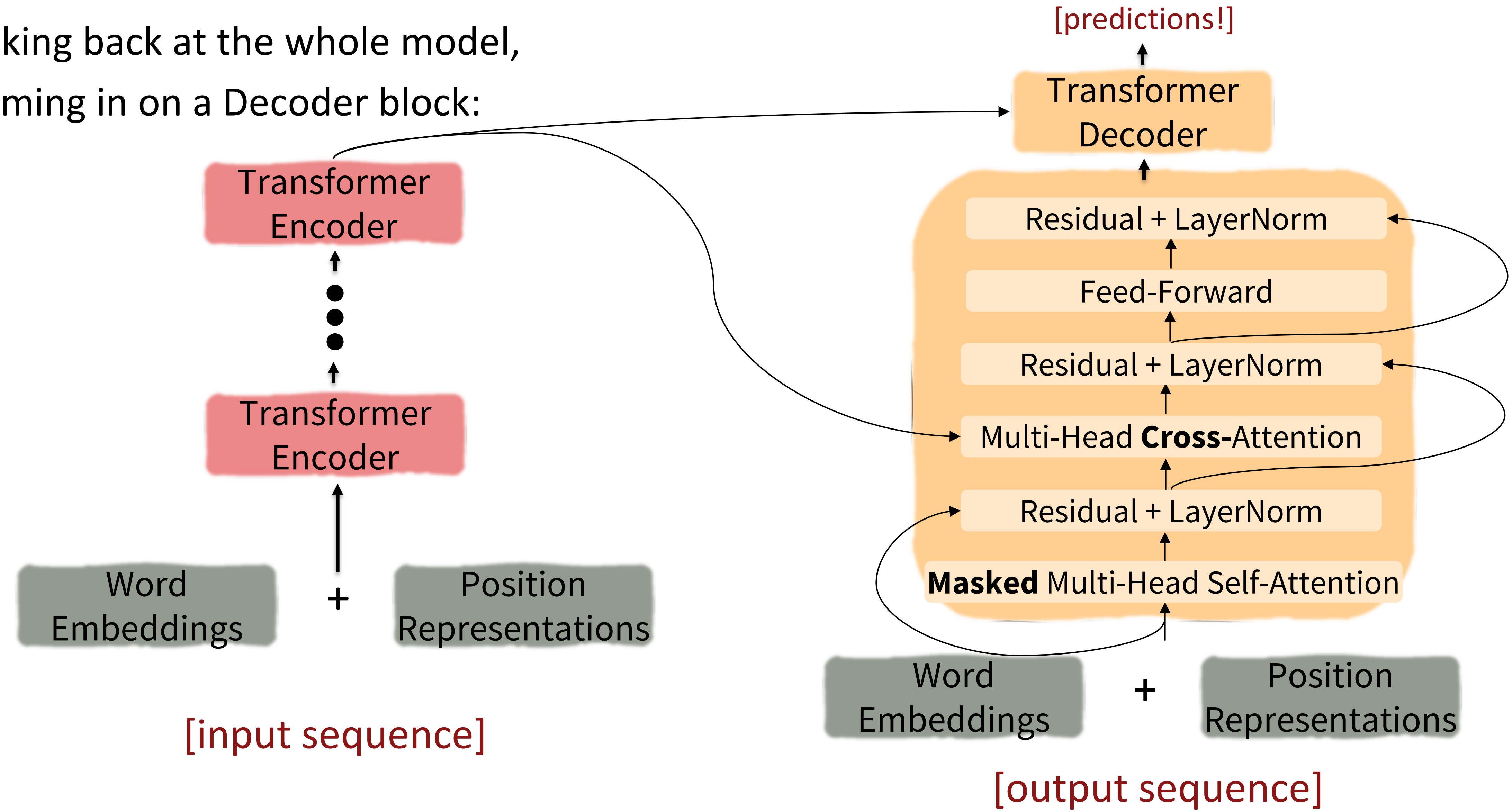
The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model,
zooming in on a Decoder block:



Lecture Plan

1. Quick review of Transformer model
2. Brief note on subword modeling
3. Motivating model pretraining from word embeddings
4. Model pretraining three ways
    1. Decoders
    2. Encoders
    3. Encoder-Decoders
5. Very large models and in-context learning

Word structure and subword models

Let’s take a look at the assumptions we’ve made about a language’s vocabulary.

We assume a fixed vocab of tens of thousands of words, built from the training set.
All *novel* words seen at test time are mapped to a single UNK.

	word		vocab mapping	embedding
Common words	hat	→	hat	
	learn	→	learn	
Variations	taaaaasty	→	UNK	
misspellings	laern	→	UNK	
novel items	Transformerify	→	UNK	

Word structure and subword models

Finite vocabulary assumptions make even *less* sense in many languages.

- Many languages exhibit complex **morphology**, or word structure.
  - The effect is more word types, each occurring fewer times.

Example: Swahili verbs can have hundreds of conjugations, each encoding a wide variety of information. (Tense, mood, definiteness, negation, information about the object, ++)

Here's a small fraction of the conjugations for *ambia* – to tell.

Conjugation of *-ambia* [less ▲]

Non-finite forms		
Form	**Positive**	**Negative**
Infinitive	kuambia	kutoambia
Simple finite forms		
Positive form	**Singular**	**Plural**
Imperative	ambia	ambieni
Habitual	huambia	

Complex finite forms

Polarity	Persons				Persons / Classes		Classes											
	1st		2nd		3rd / M-wa		M-mi		Ma		Ki-vi		N		U	Ku	Pa	Mu
	Sg.	Pl.	Sg.	Pl.	Sg. / 1	Pl. / 2	3	4	5	6	7	8	9	10	11 / 14	15 / 17	16	18
Past [less ▲]																		
Positive	niliambia naliambia	tuliambia twaliambia	uliambia waliambia	mliambia mwaliambia	aliambia	waliambia	uliambia	iliambia	liliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	muliambia
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambia	haukuambia	haikuambia	halikuambia	hayakuambia	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambia	hapakuambia	hamukuambia
Present [less ▲]																		
Positive	ninaambia naambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	haliambii	hayaambii	hakiambii	haviambii	haiambii	haziambii	hauambii	hakuambii	hapaambii	hamuambii
Future [less ▲]																		
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	litaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambia	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambia
Subjunctive [less ▲]																		
Positive	niambie	tuambie	uambie	mambie	aambie	waambie	uambie	iambie	liambie	yaambie	kiambie	viambie	iambie	ziambie	uambie	kuambie	paambie	muambie
Negative	nisiambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	lisiambie	yasiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie
Present Conditional [less ▲]																		
Positive	ningeambia	tungeambia	ungeambia	mngeambia	angeambia	wangeambia	ungeambia	ingeambia	lingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia
Negative	nisingeambia singeambia	tusingeambia hatungeambia	usingeambia hungeambia	msingeambia hamngeambia	asingeambia hangeambia	wasingeambia hawangeambia	usingeambia haungeambia	isingeambia haingeambia	lisingeambia halingeambia	yasingeambia hayangeambia	kisingeambia hakingeambia	visingeambia havingeambia	isingeambia haingeambia	zisingeambia hazingeambia	usingeambia haungeambia	kusingeambia hakungeambia	pasingeambia hapangeambia	musingeambia hamungeambia
Past Conditional [less ▲]																		
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia	ungaliambia	ingaliambia	lingaliambia	yangaliambia	kingaliambia	vingaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia	pangaliambia	mungaliambia
Negative	nisingaliambia singaliambia	tusingaliambia hatungaliambia	usingaliambia hungaliambia	msingaliambia hamngaliambia	asingaliambia hangaliambia	wasingaliambia hawangaliambia	usingaliambia haungaliambia	isingaliambia haingaliambia	lisingaliambia halingaliambia	yasingaliambia hayangaliambia	kisingaliambia hakingaliambia	visingaliambia havingaliambia	isingaliambia haingaliambia	zisingaliambia hazingaliambia	usingaliambia haungaliambia	kusingaliambia hakungaliambia	pasingaliambia hapangaliambia	musingaliambia hamungaliambia
Conditional Contrary to Fact [less ▲]																		
Positive	ningeliambia	tungeliambia	ungeliambia	mngeliambia	angeliambia	wangeliambia	ungeliambia	ingeliambia	lingeliambia	yangeliambia	kingeliambia	vingeliambia	ingeliambia	zingeliambia	ungeliambia	kungeliambia	pangeliambia	mungeliambia
Gnomic [less ▲]																		
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia
Perfect [less ▲]																		

[Wiktionary]

The byte-pair encoding algorithm

Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of **parts of words (subword tokens).**
- At training and testing time, each word is split into a sequence of known subwords.

Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary.

1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
2. Using a corpus of text, find the most common pair of adjacent characters "a,b"; add subword "ab" to the vocab.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

[Sennrich et al., 2016, Wu et al., 2016]

Word structure and subword models

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.

	word		vocab mapping	embedding
Common words	hat	→	hat	
	learn	→	learn	
Variations	taaaaasty	→	taa## aaa## sty	
misspellings	laern	→	la## ern	
novel items	Transformerify	→	Transformer## ify	

Outline

1. Quick review of Transformer models
2. Brief note on subword modeling
3. Motivating model pretraining from word embeddings
4. Model pretraining three ways
    1. Decoders
    2. Encoders
    3. Encoder-Decoders
5. Very large models and in-context learning

Motivating word meaning and context

Recall the adage we mentioned at the beginning of the course:

"You shall know a word by the company it keeps" (J. R. Firth 1957: 11)

This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:

"… the complete meaning of a word is always contextual,
and no study of meaning apart from a complete context
can be taken seriously." (J. R. Firth 1935)

Consider *I **record** the **record***: the two instances of ***record*** mean different things.

[Thanks to Yoav Goldberg on Twitter for pointing out the 1935 Firth quote.]

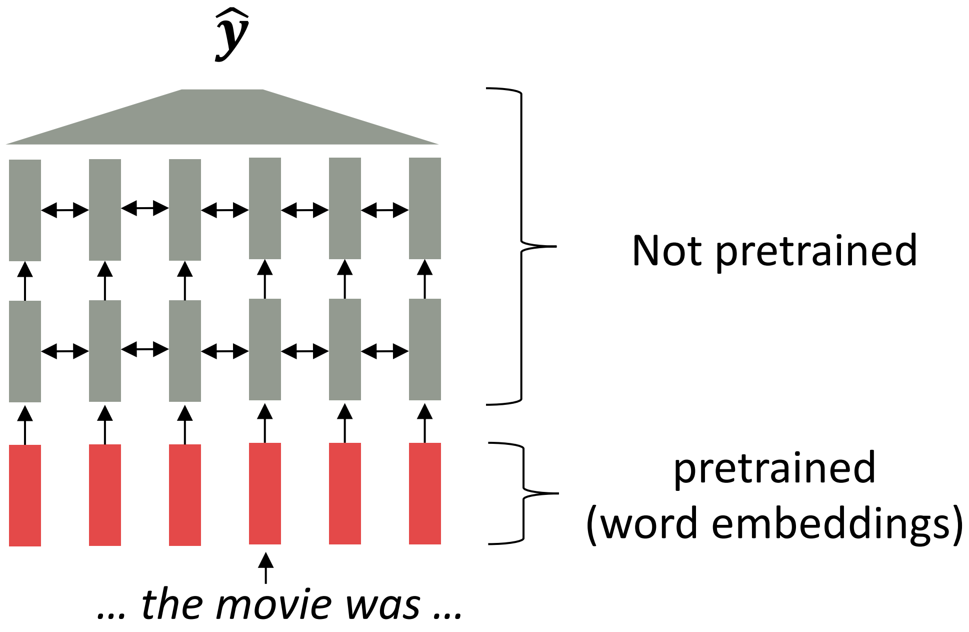
Where we were: **pretrained word embeddings**

Circa 2017:

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.

Some issues to think about:

- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomly initialized!



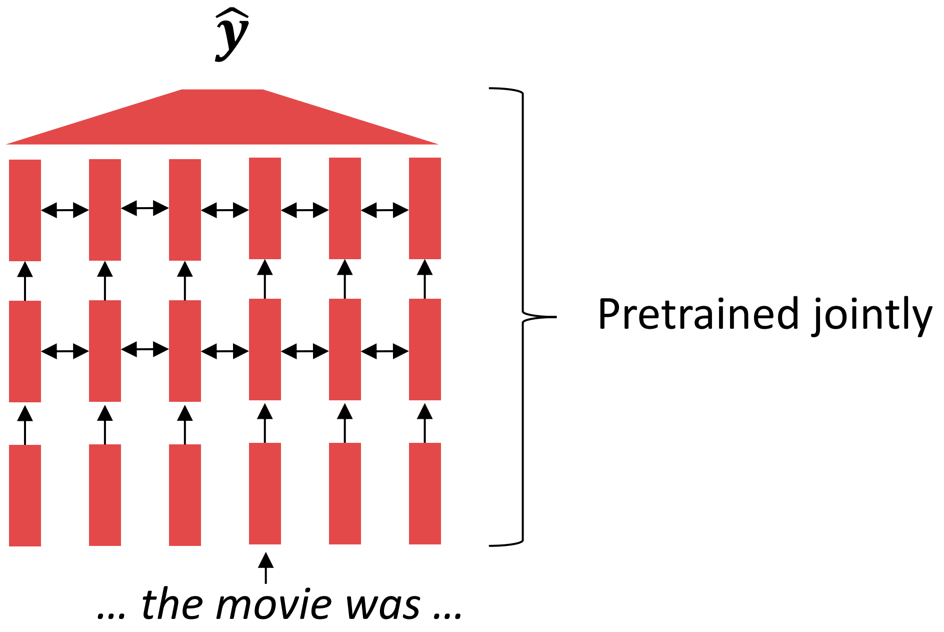
[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

Where we're going: **pretraining whole models**

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]

What can we learn from reconstructing the input?

Stanford University is located in __________, California.

What can we learn from reconstructing the input?

I put ___ fork down on the table.

What can we learn from reconstructing the input?

The woman walked across the street,
checking for traffic over ___ shoulder.

What can we learn from reconstructing the input?

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

What can we learn from reconstructing the input?

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

What can we learn from reconstructing the input?

Iroh went into the kitchen to make some tea.

Standing next to Iroh, Zuko pondered his destiny.

Zuko left the _______.

What can we learn from reconstructing the input?

I was thinking about the sequence that goes

1, 1, 2, 3, 5, 8, 13, 21, ____

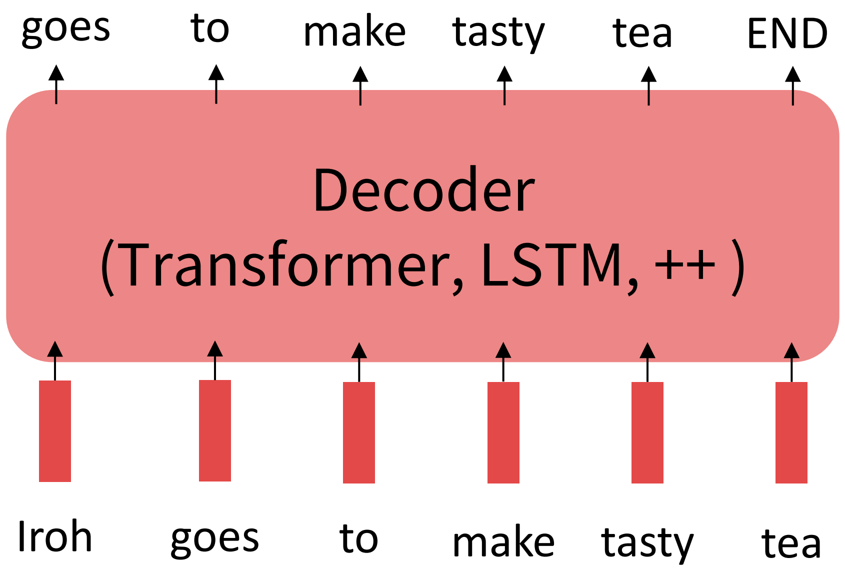
Pretraining through language modeling [Dai and Le, 2015]

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.

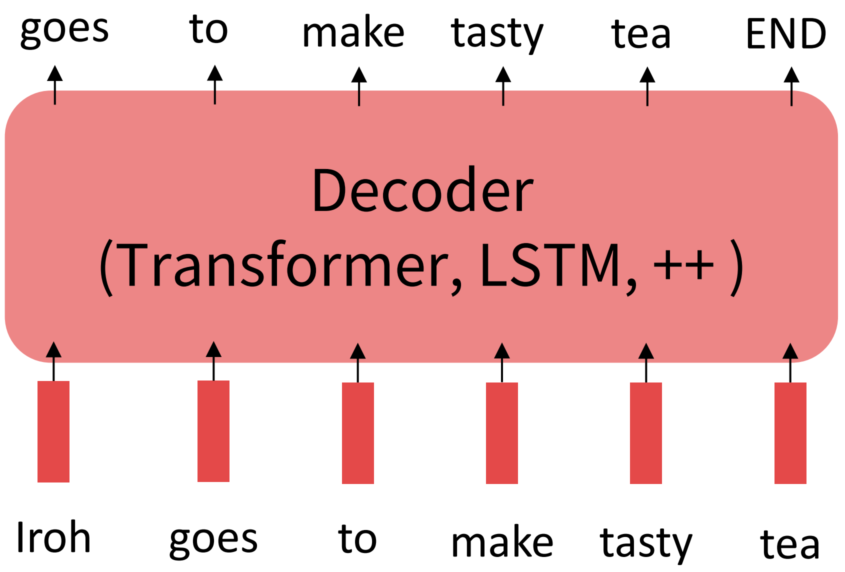


The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

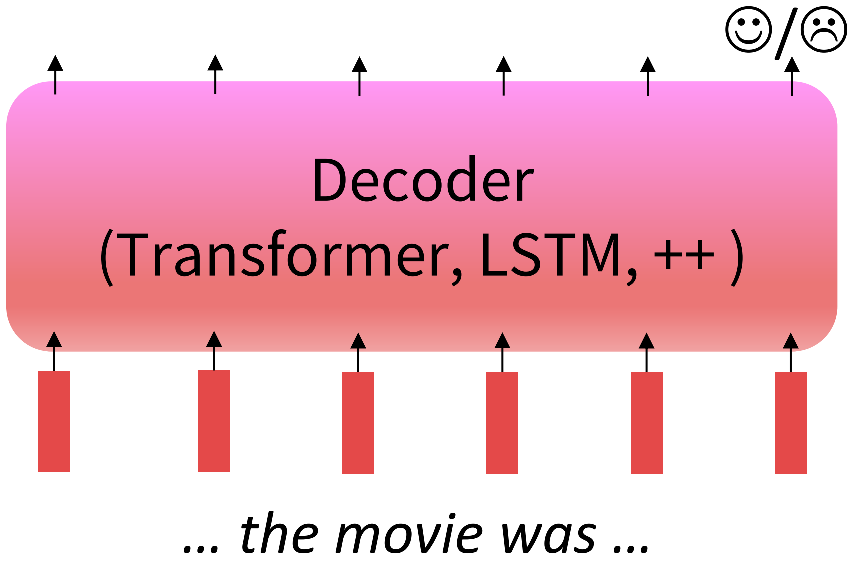
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!

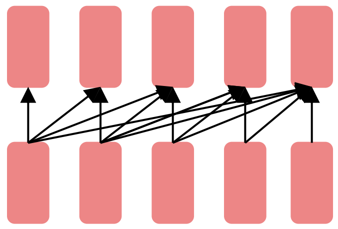


Lecture Plan

1. A brief note on subword modeling
2. Motivating model pretraining from word embeddings
3. Model pretraining three ways
    1. Decoders
    2. Encoders
    3. Encoder-Decoders
4. Very large models and in-context learning

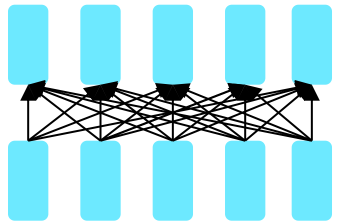
Pretraining for three types of architectures

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



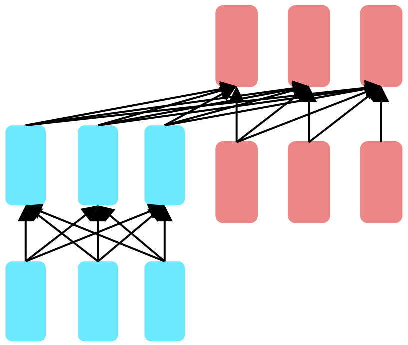
Decoders

- 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

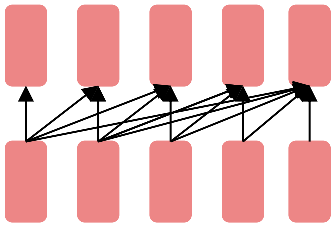


Encoder-Decoders

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

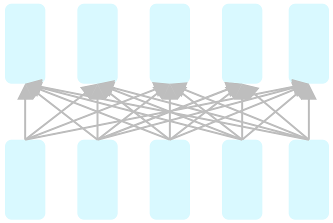
Pretraining for three types of architectures

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



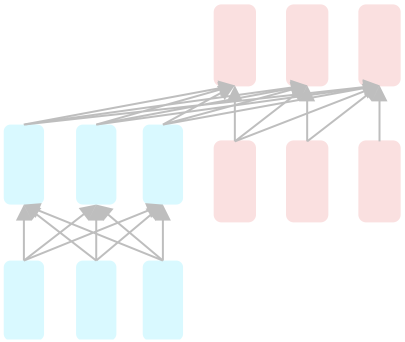
Decoders

- 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

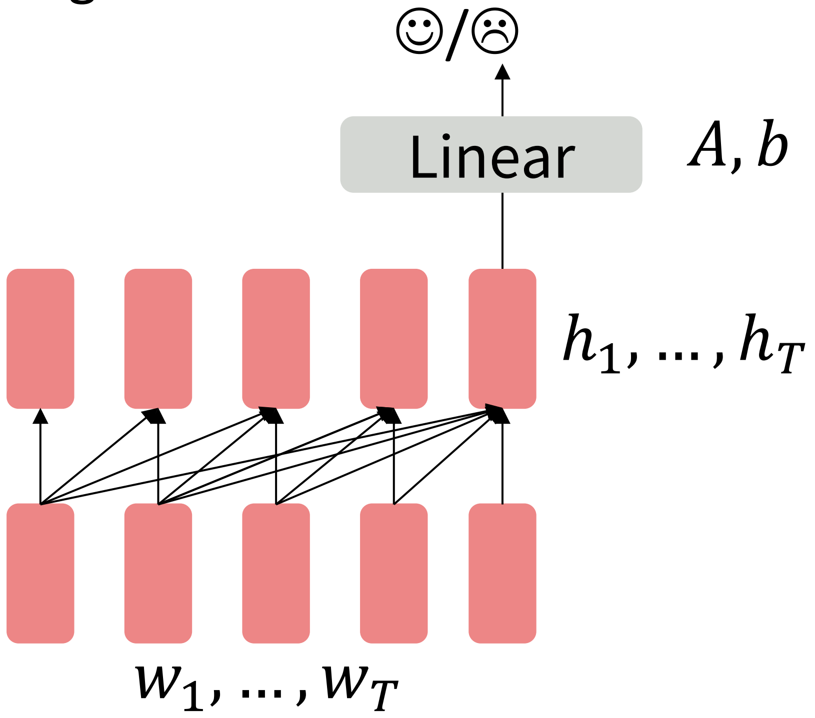
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$.

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

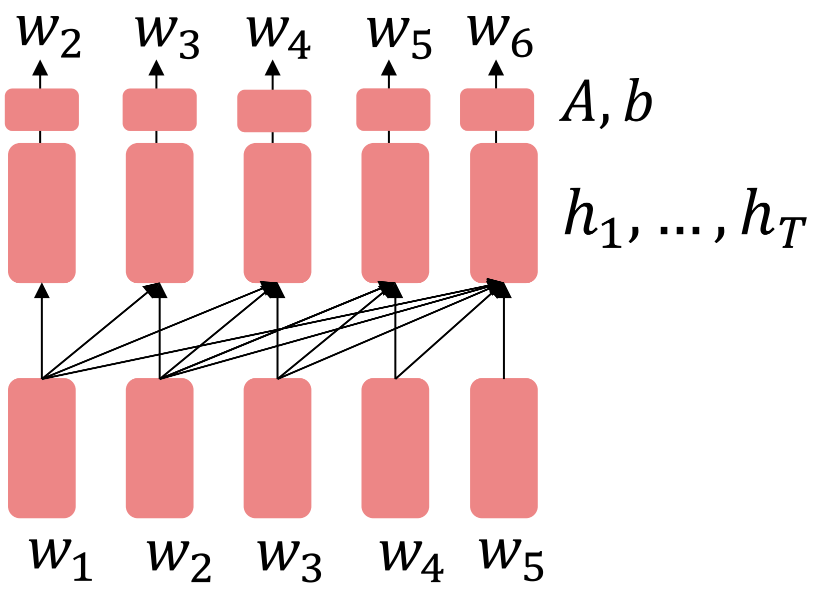
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

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)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim A h_{t-1} + b$$

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]

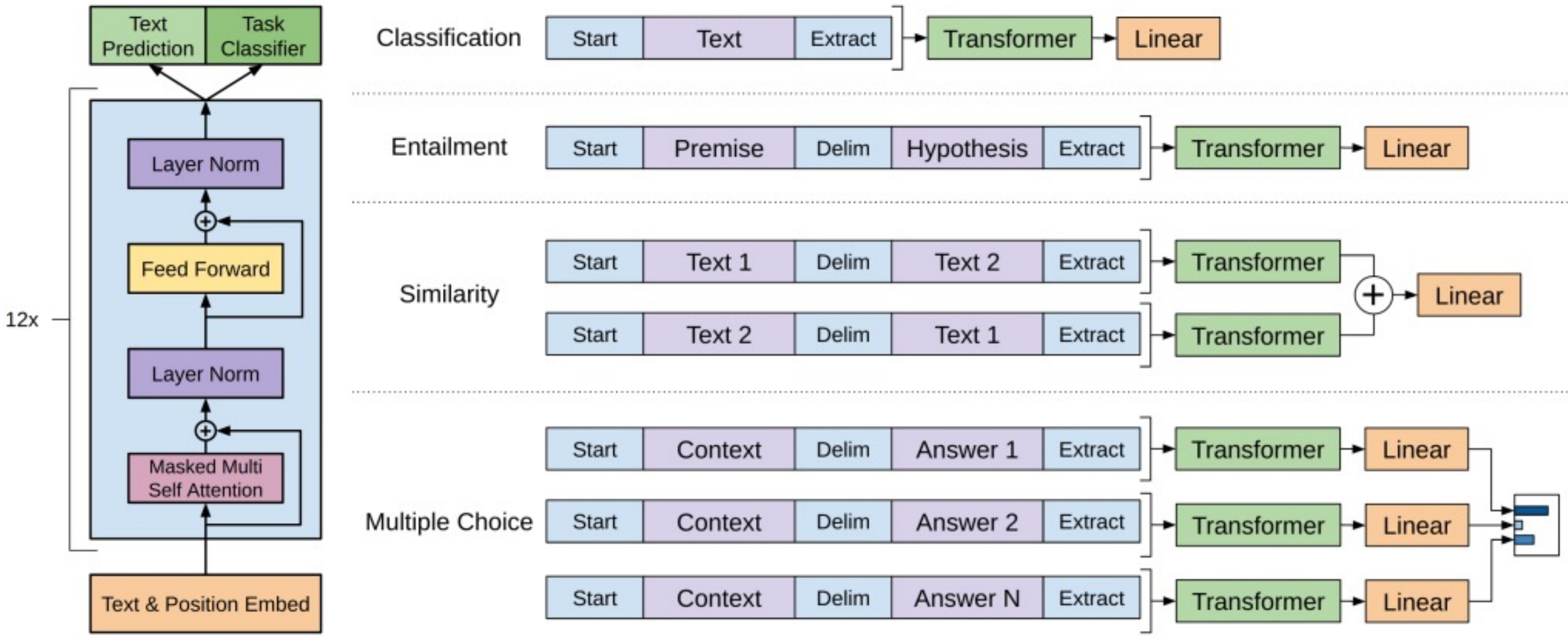
2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.

[Devlin et al., 2018]

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

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



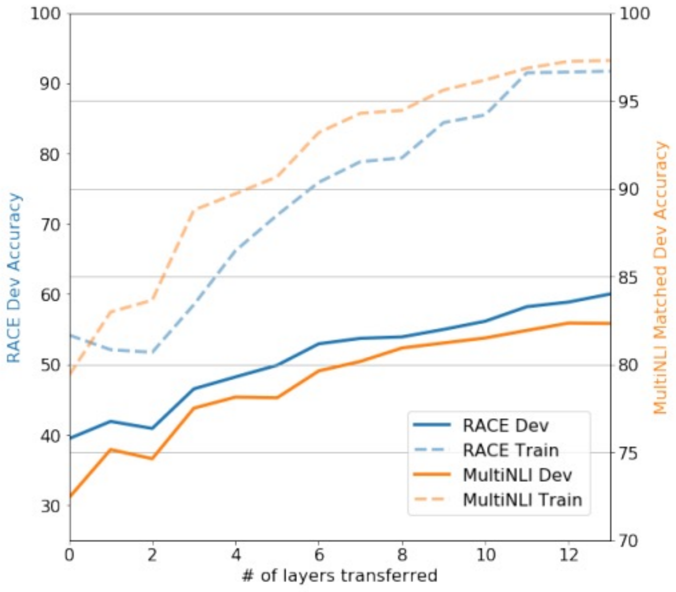
The linear classifier is applied to the representation of the [EXTRACT] token.

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

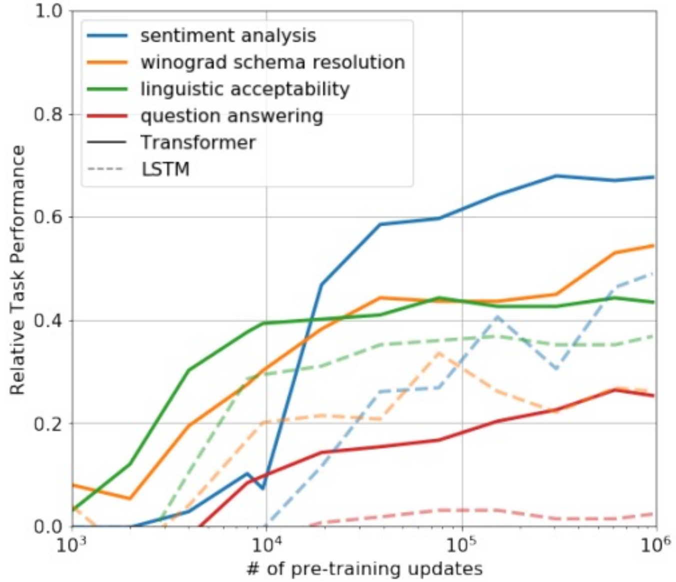
GPT results on various *natural language inference* datasets.

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

Examining the Effect of Pretraining in GPT [Radford et al., 2018]



As more layers are transferred, performance improves on RACE (a large-scale reading comprehension dataset) and MultiNLI.



Zero-shot performance of Transformer vs. LSTM as a function of the # of pre-training updates.

Increasingly convincing generations (GPT2) [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.

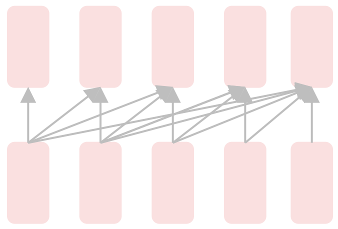
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

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.

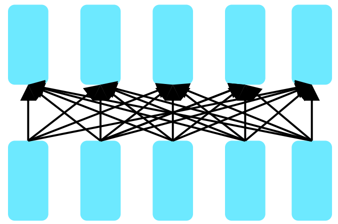
Pretraining for three types of architectures

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



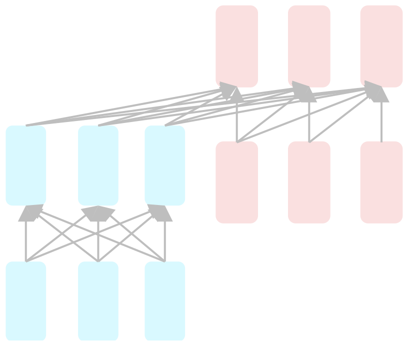
Decoders

- 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

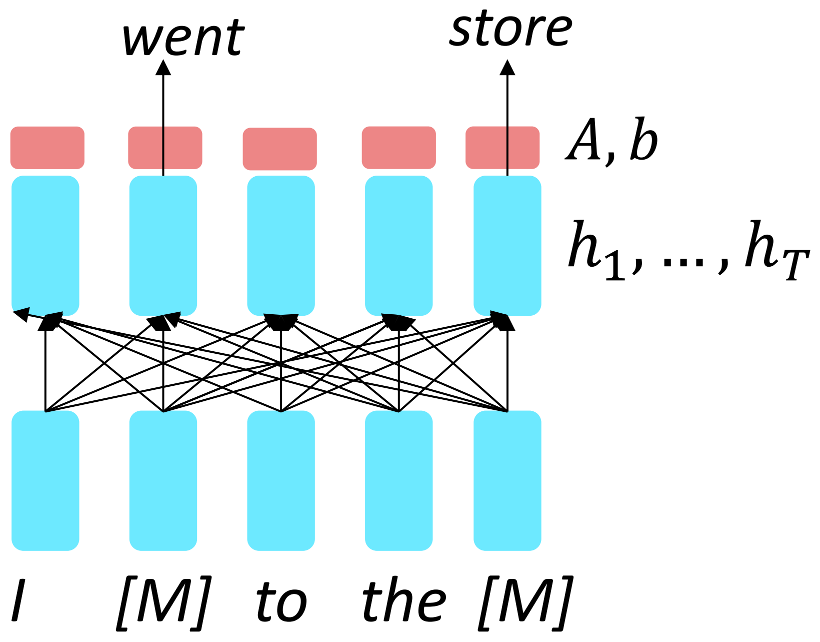
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

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**.



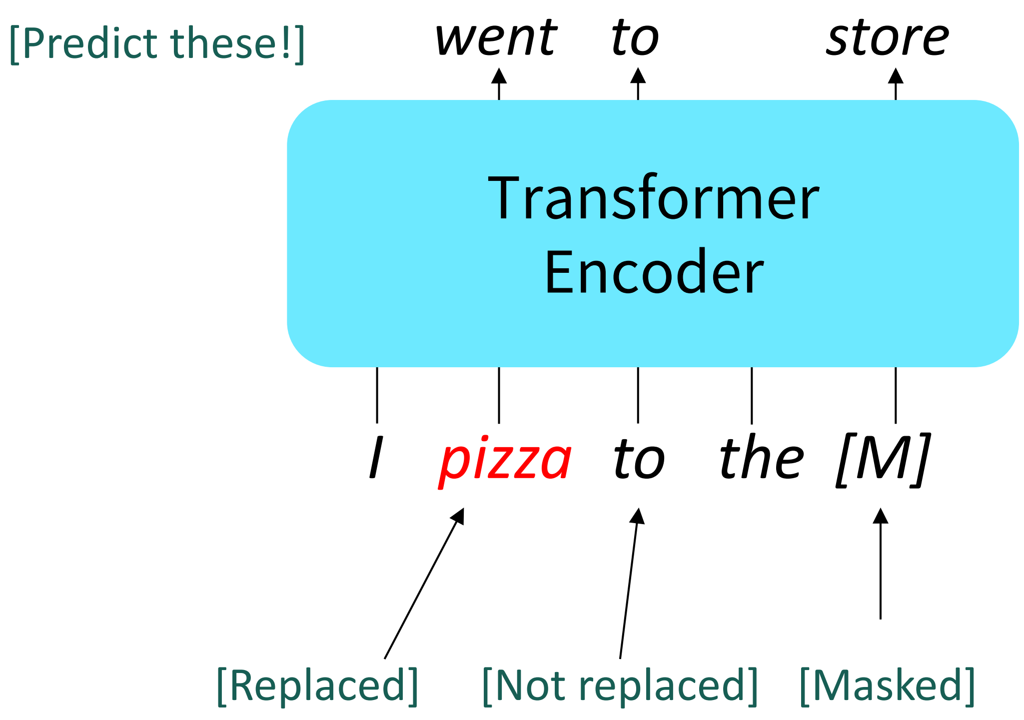
[Devlin et al., 2018]

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:

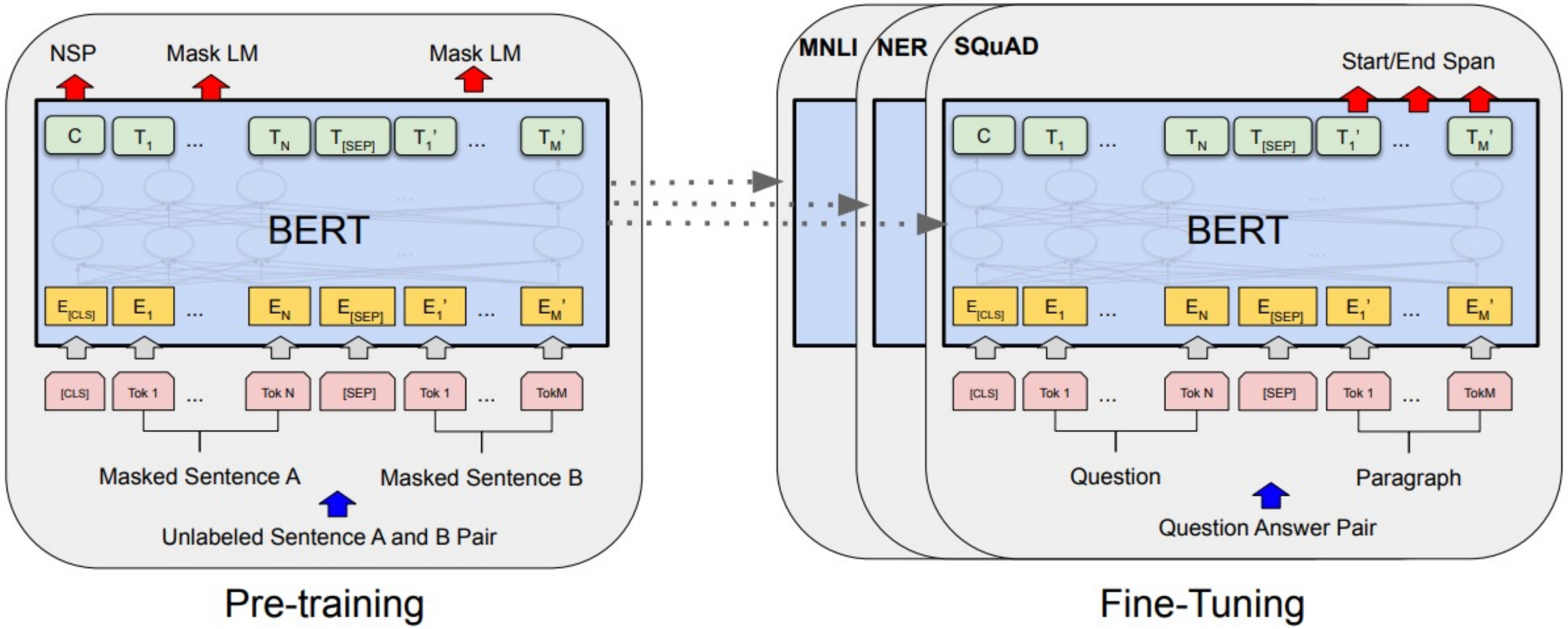
- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn’t let the model get complacent and not build strong representations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



[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

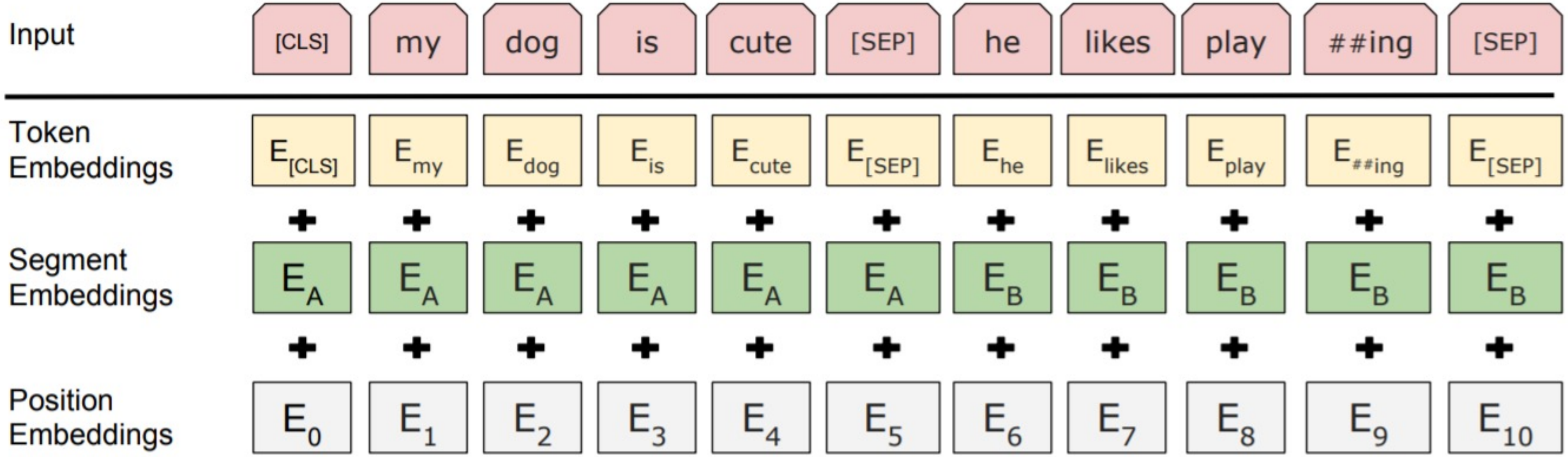
- **Unified Architecture:** As shown below, there are minimal differences between the pre-training architecture and the fine-tuned version for each downstream task.



[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

- The pretraining input to BERT was two separate contiguous chunks of text:



- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
  - Later work has argued this "next sentence prediction" is not necessary.

[Devlin et al., 2018, Liu et al., 2019]

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.”

[Devlin et al., 2018]

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) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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**

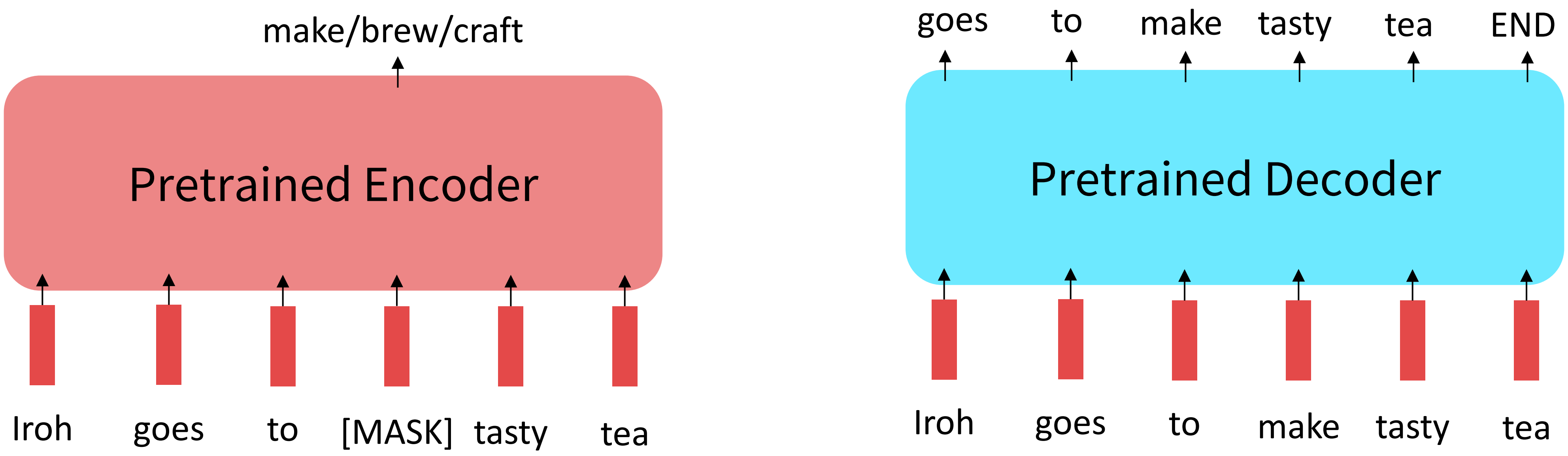
Note that $\text{BERT}_{\text{BASE}}$ was chosen to have the same number of parameters as OpenAI GPT.

[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.

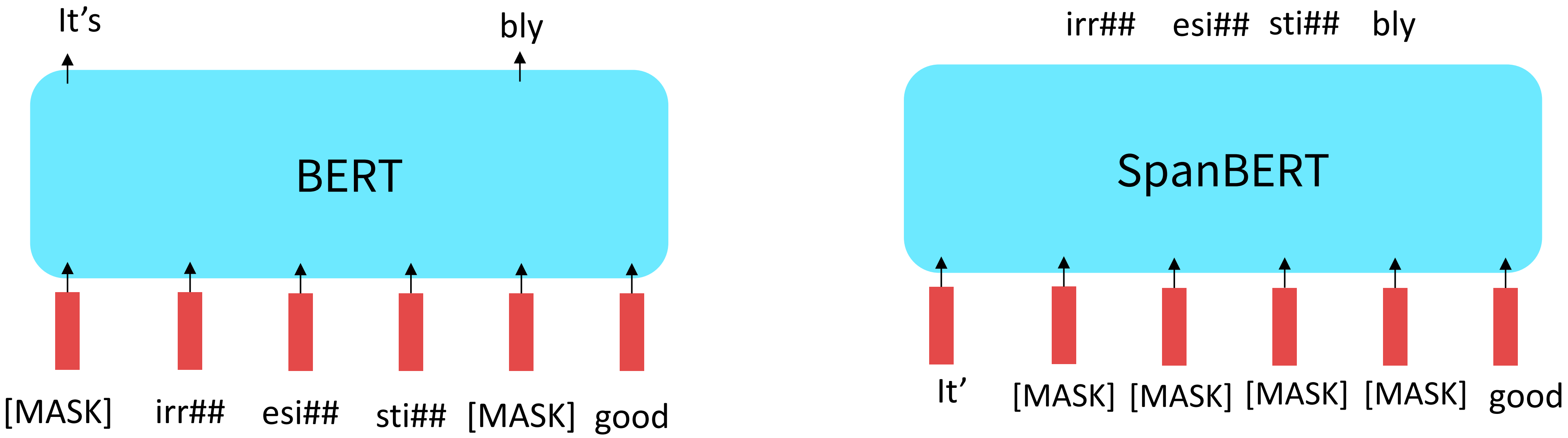


Extensions of BERT

You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++

Some generally accepted improvements to the BERT pretraining formula:

- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task



[Liu et al., 2019; Joshi et al., 2020]

Extensions of BERT

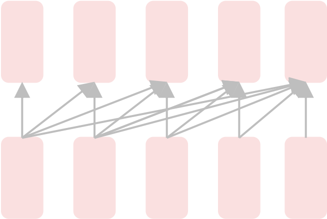
A takeaway from the RoBERTa paper: more compute, more data can improve pretraining even when not changing the underlying Transformer encoder.

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

[Liu et al., 2019; Joshi et al., 2020]

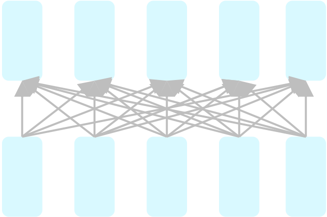
Pretraining for three types of architectures

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



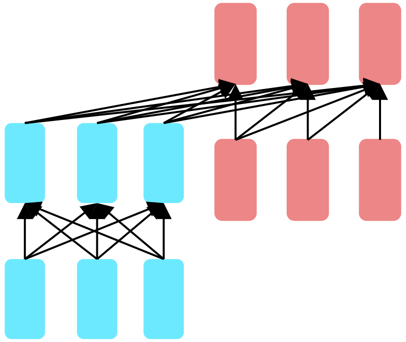
Decoders

- 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

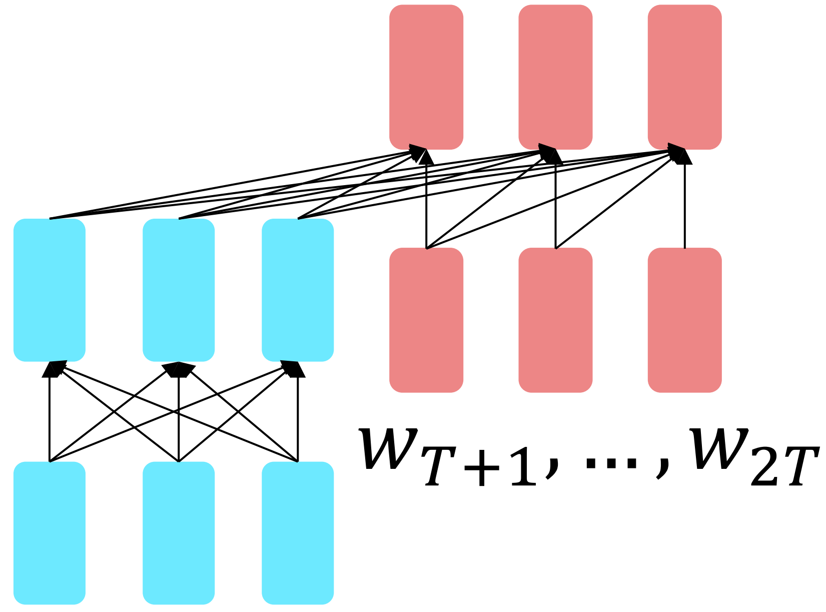
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

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.

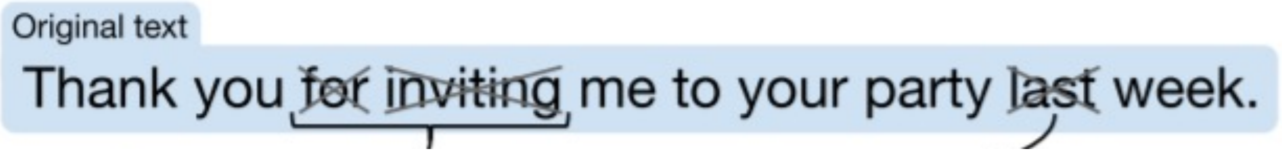


[Raffel et al., 2018]

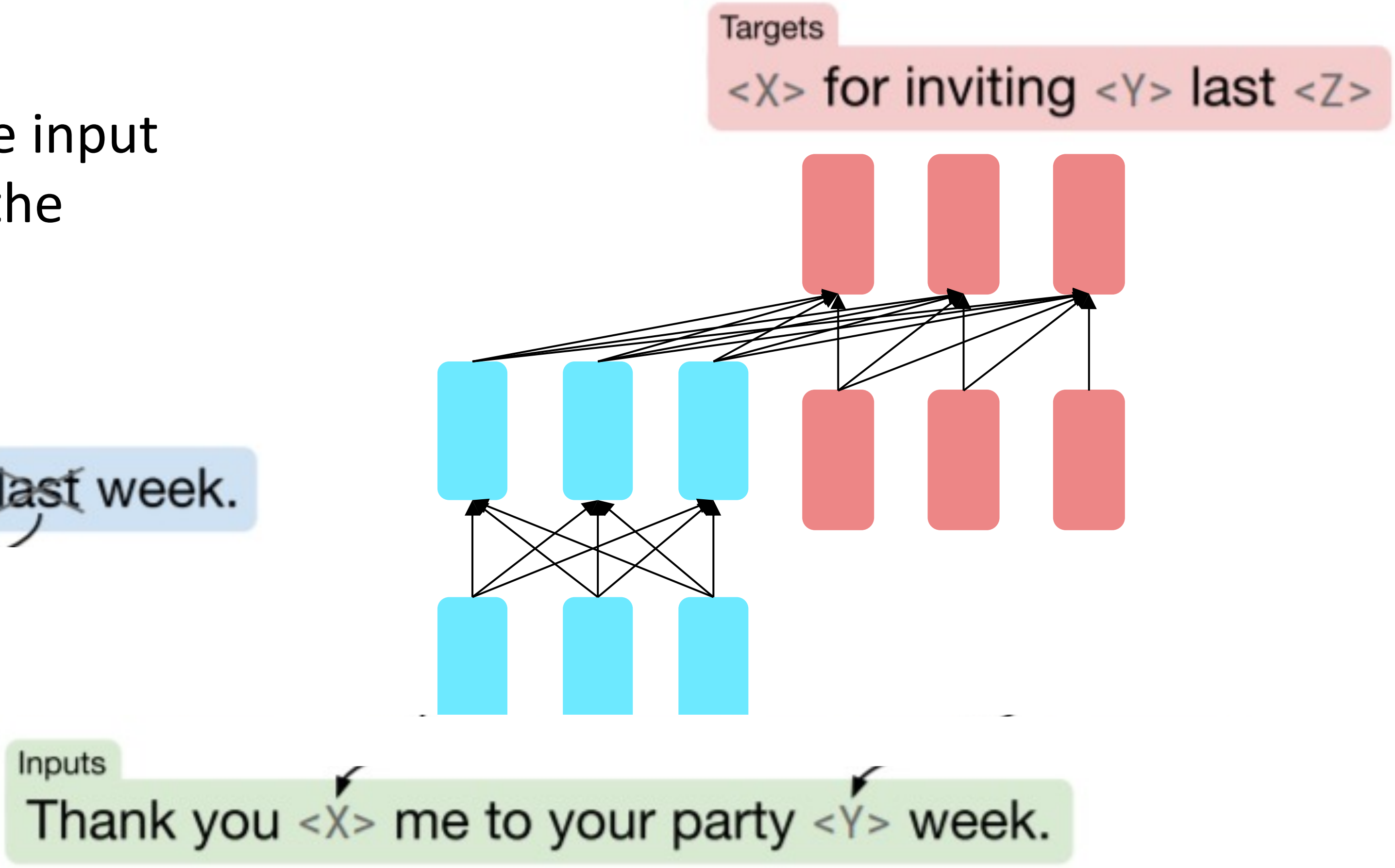
Pretraining encoder-decoders: what pretraining objective to use?

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!



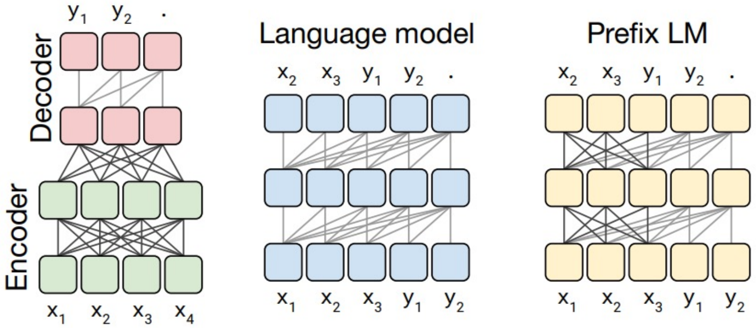
This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



[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

[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

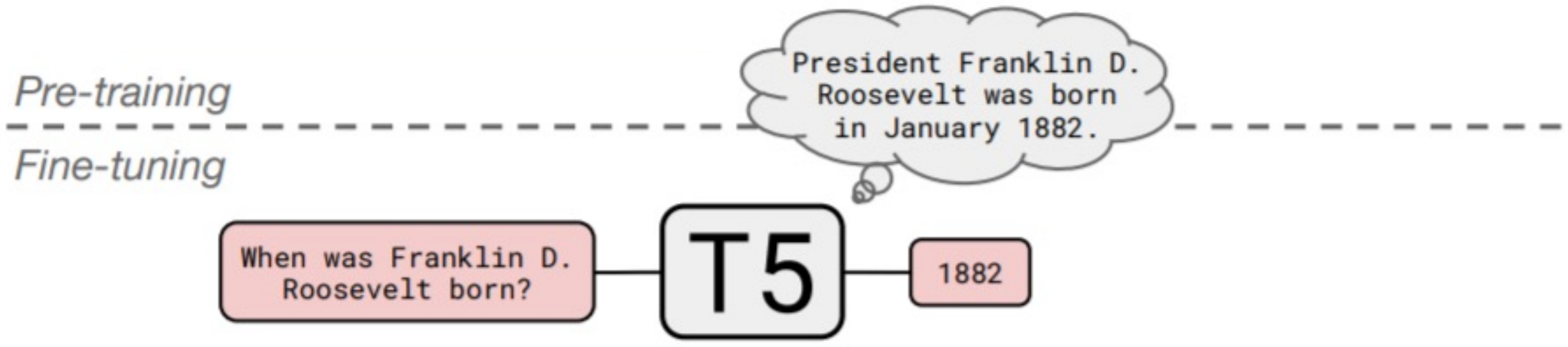
A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.

NQ: Natural Questions

WQ: WebQuestions

TQA: Trivia QA

All “open-domain” versions



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

[Raffel et al., 2018]

Outline

1. Prelude: A brief note on subword modeling
2. Motivating model pretraining from word embeddings
3. Model pretraining three ways
    1. Decoders
    2. Encoders
    3. Encoder-Decoders
4. Very large models and in-context learning

GPT-3, In-context learning, and very large models

So far, we've interacted with pretrained models in two ways:

- Sample from the distributions they define (maybe providing a prompt)
- Fine-tune them on a task we care about, and then take their predictions.

Emergent behavior: Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.

GPT-3 has 175 billion parameters.

GPT-3, In-context learning, and very large models

Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

Input (prefix within a single Transformer decoder context):

“ thanks -> merci

hello -> bonjour

mint -> menthe

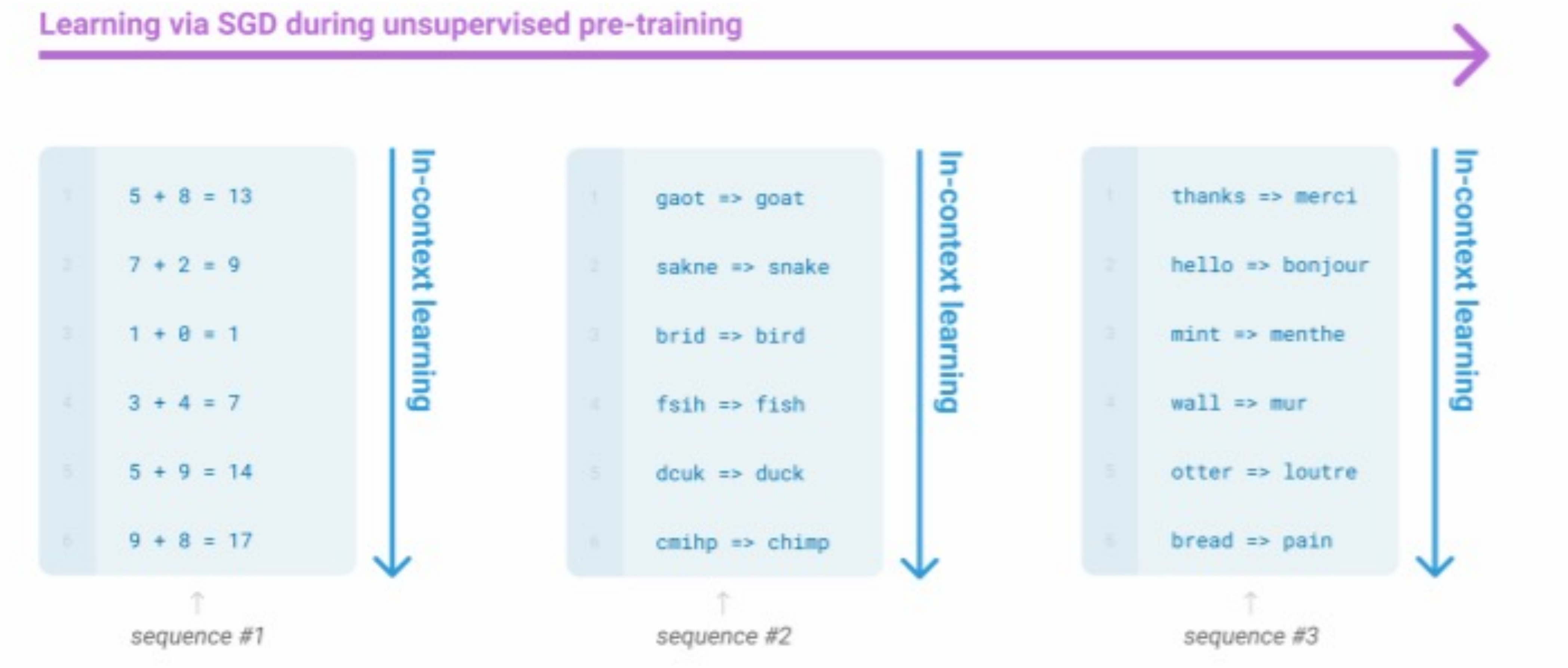
otter -> ”

Output (conditional generations):

loutre...”

GPT-3, In-context learning, and very large models

Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.



Parting remarks

- We learned about GPT-X, BERT, T5 and other large pre-trained language models
- Emergent in-context learning is not yet well-understood!
- “Small” models like BERT have become general tools in a wide range of settings.
- Many issues left to explore!
  - Bias, toxicity, and fairness (Guest Lecturer: Maarten Sap)
  - Retrieval Augmented Language Models + Knowledge (Guest Lecturer: Kelvin Guu)
  - Scaling Laws (Guest Lecturer: Jared Kaplan)
- Assignment 5 out today! It covers material from Tuesday’s and today’s lectures.
- Hugging Face Transformers Tutorial Session on Friday 1:30-2:30pm (Thornton 102 and recorded)!