

# Natural Language Processing with Deep Learning

## CS224N/Ling284



John Hewitt

Lecture 9: Pretraining

*Adapted from slides by Anna Goldie, John Hewitt*

# 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. Interlude: what do we think pretraining is teaching?
5. Very large models and in-context learning

## Reminders:

Assignment 5 is out on Thursday! It covers lecture 8 and lecture 9(Today)!






It has ~pedagogically relevant math~

# 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	→	pizza (index)	
	learn	→	tasty (index)	
Variations	taaaaasty	→	UNK (index)	
misspellings	laern	→	UNK (index)	
novel items	Transformerify	→	UNK (index)	

Unknown

# 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.

One word many variations

Conjugation of -ambia																							[less ▲]	
Form Infinitive		Non-finite forms																		Negative kutoambia				
		Simple finite forms																						
Positive form Imperative Habitual		Singular ambia																		Plural ambieni				
		huambia																						
Polarity	Persons						Complex finite forms																	
	Persons / Classes						Classes																	
	Sg.	1st	Pl.	Sg.	2nd	Pl.	Sg. 3rd / 1 / 1	Pl. 2 / 2	3	M-mi	4	5	Ma	6	7	Ki-vi	8	9	N	10	11 / 14	Ku 15 / 17	Pa 16	Mu 18
	Past																							[less ▲]
Positive	niliambia	tuliambia	uliambia	waliambia	aliambia	waliambia	uliambia	iliambia	iliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	mulambia						
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambi	haukuambia	haikuambia	halikuambia	hayakuambi	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambi	hapakuambi	hamukuambi						
	Present																							[less ▲]
Positive	ninaambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia						
Negative	siambia	hatuambi	huambi	hamambi	haambi	hawaambi	hauambi	haiambi	haliambi	hayaambi	hakiambi	haviambi	haiambi	haziambi	hauambi	hakuambi	hapaambi	hamuambi						
	Future																							[less ▲]
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	itaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia						
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambi	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambi						
	Subjunctive																							[less ▲]
Positive	niambia	tuambia	uambia	mambia	aambia	waambia	uambia	lambia	liambia	yaambia	kiambia	viambia	lambia	ziambia	uambia	kuambia	paambia	muambia						
Negative	niambia	tusiambia	usiambia	msiambia	asiambia	wasiambia	usiambia	isambia	lisambia	yasambia	kisambia	visambia	isambia	zisambia	usiambia	kusiambia	pasiambia	musiambia						
	Present Conditional																							[less ▲]
Positive	ningeambia	tungeambi	ungeambi	mngeambi	angeambi	wangeambi	ungeambi	ingeambi	lingeambi	yangeambi	kingeambi	vingeambi	ingeambi	zingeambi	ungeambi	kungeambi	pangeambi	mungeambi						
Negative	nisingeambi	tusingeambi	usingeambi	msingeambi	asingeambi	wasingeambi	usingeambi	isingeambi	lingeambi	yasingeambi	kingeambi	vingeambi	isingeambi	zingeambi	usingeambi	kusingeambi	pasingeambi	musingeambi						
	Past Conditional																							[less ▲]
Positive	ningaliambia	tungaliambi	ungaliambi	mngaliambi	angaliambi	wangaliambi	ungaliambi	ingaliambi	lingaliambi	yangaliambi	kingaliambi	vingaliambi	ingaliambi	zingaliambi	ungaliambi	kungaliambi	pangaliambi	mungaliambi						
Negative	nisingaliambi	tungaliambi	ungaliambi	msingaliambi	asingaliambi	wasingaliambi	ungaliambi	isingaliambi	lingaliambi	yasingaliambi	kingaliambi	vingaliambi	isingaliambi	zingaliambi	ungaliambi	kusingaliambi	pasingaliambi	musingaliambi						
	Conditional Contrary to Fact																							[less ▲]
Positive	ningeliambi	tungeliambi	ungeliambi	mngeliambi	angeliambi	wangeliambi	ungeliambi	ingeliambi	lingeliambi	yangeliambi	kingeliambi	vingeliambi	ingeliambi	zingeliambi	ungeliambi	kungeliambi	pangeliambi	mungeliambi						
	Gnomic																							[less ▲]
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaaambia						
	Perfect																							[less ▲]

# 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.

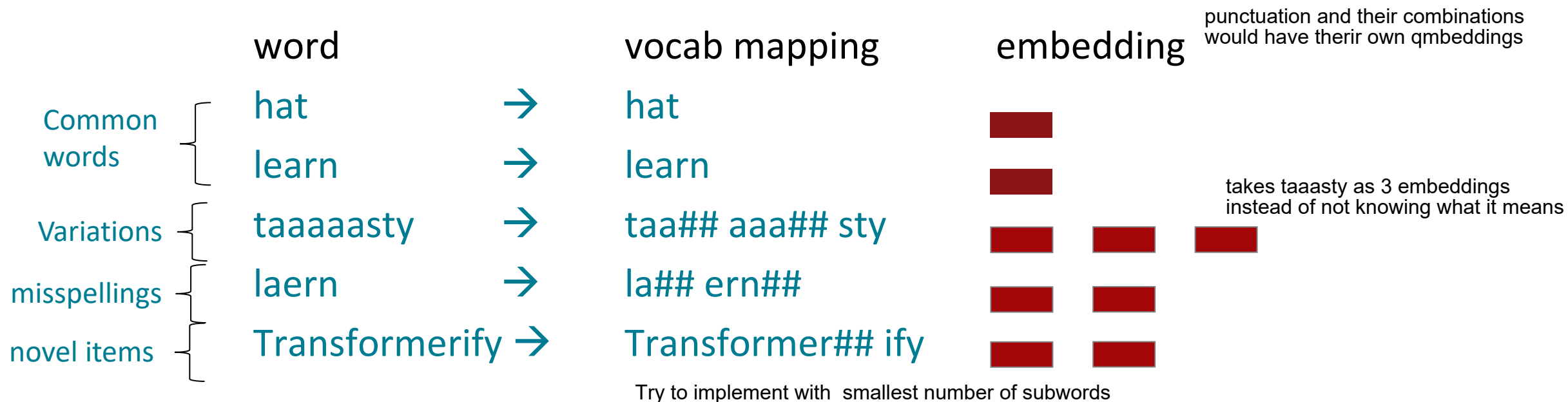
1. Start with a vocabulary containing **only characters** and an “end-of-word” symbol.
2. Using a corpus of text, find the most **common adjacent characters** “a,b”; add “ab” as a subword.
3. Replace instances of the **character pair with the new subword**; repeat until desired vocab size.

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

# 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.



# Outline

1. A brief note on subword modeling
2. Motivating model pretraining from word embeddings
3. Model pretraining three ways
  1. Encoders
  2. Encoder-Decoders
  3. Decoders
4. What do we think pretraining is teaching?

# 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.



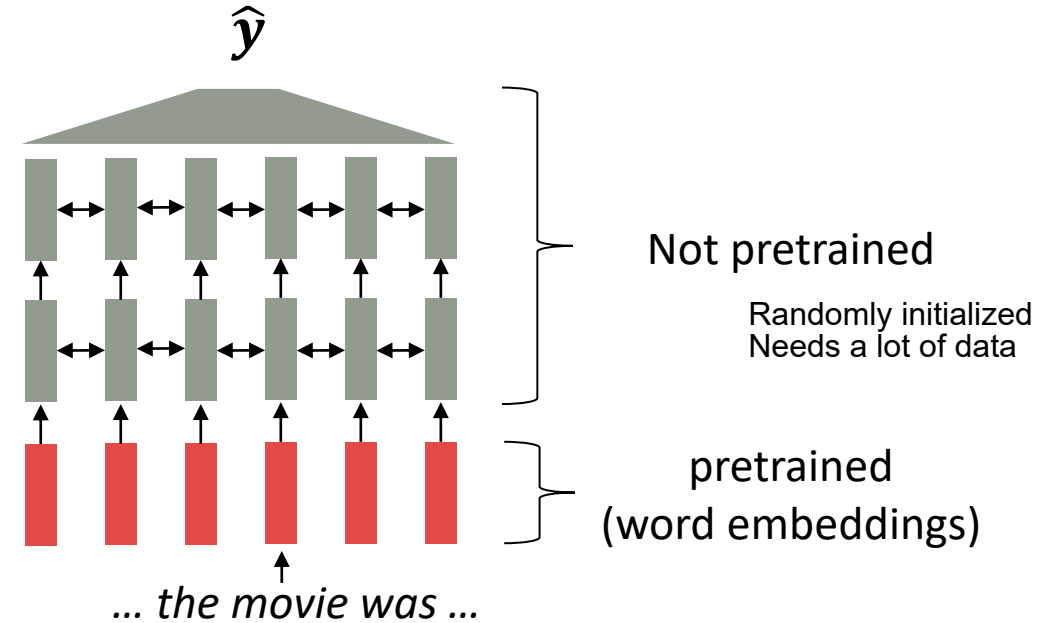
# Where we were: pretrained word embeddings

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!

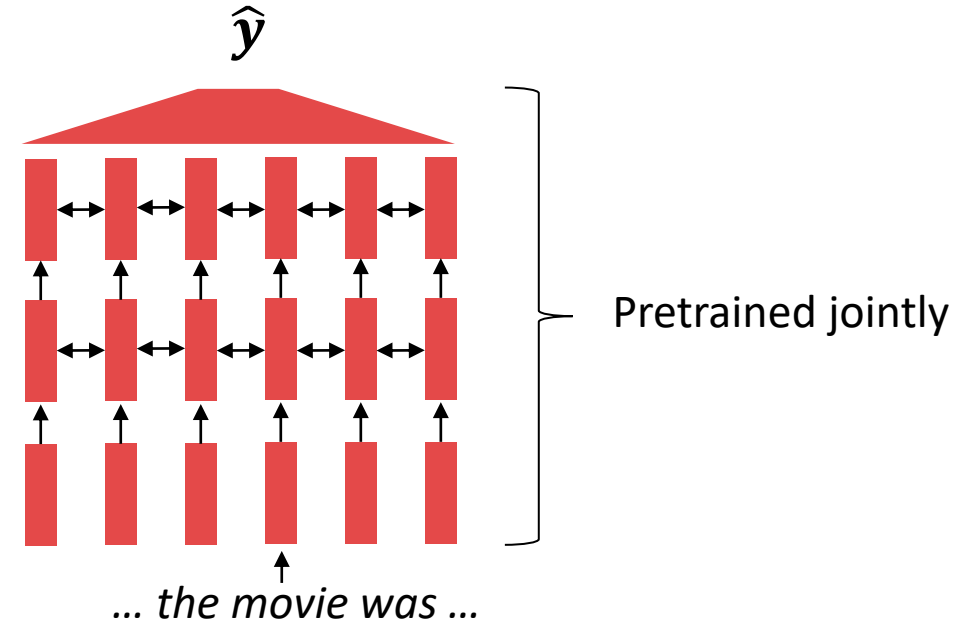


[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

# 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 train the model to reconstruct those parts.
- This has been exceptionally effective at building strong:
  - **representations of language** record and Record have different representations
  - **parameter initializations** for strong NLP models. weights from pretraining
  - **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?

can learn facts: locations etc

Stanford University is located in \_\_\_\_\_, California.

# What can we learn from reconstructing the input?

I put \_\_\_\_ fork down on the table.

syntax: a , and the are the only possible ones due to syntactic constraints

# What can we learn from reconstructing the input?

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

Context

# What can we learn from reconstructing the input?

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

semantics

# 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 \_\_\_\_.

sentiment

# 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 \_\_\_\_\_.

how the world works, location etc



# What can we learn from reconstructing the input?

I was thinking about the sequence that goes  
1, 1, 2, 3, 5, 8, 13, 21, \_\_\_\_\_

Fibonacci sequence

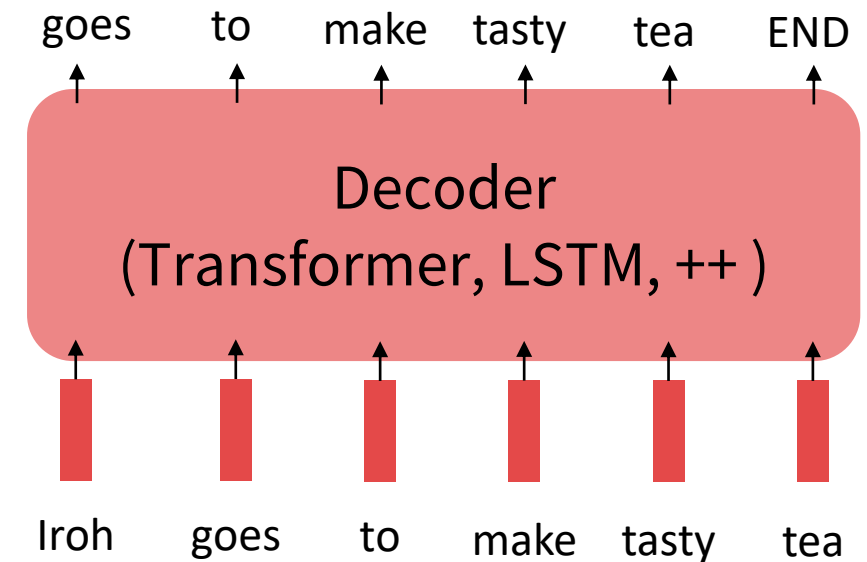
# Pretraining through language modeling [[Dai and Le, 2015](#)]

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.**

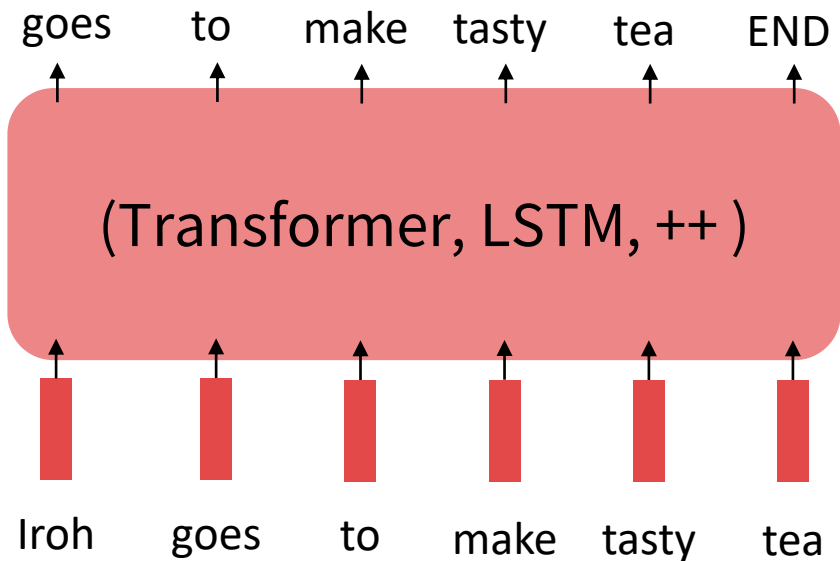


# The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter **initialization**.

## Step 1: Pretrain (on language modeling)

Lots of text; learn general things!

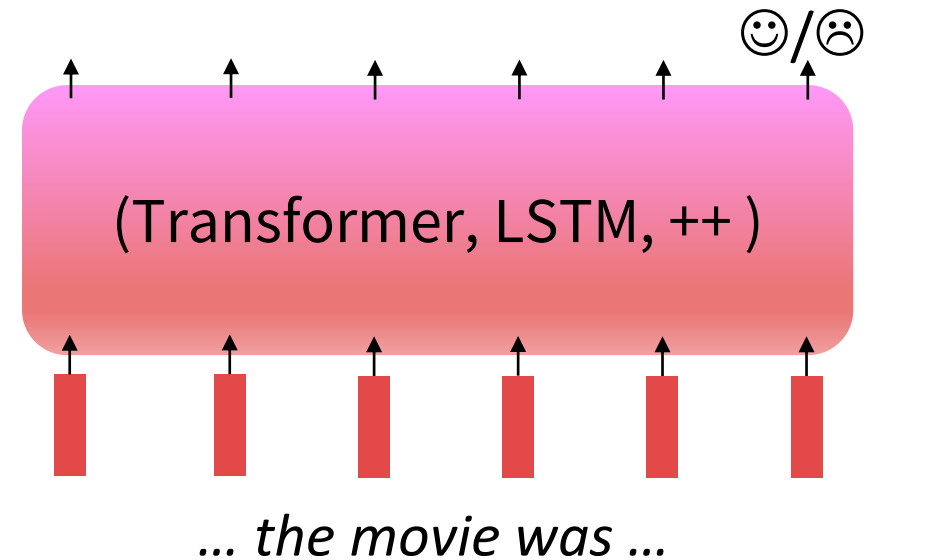


## Step 2: Finetune (on your task)

Not many labels; adapt to the task!

less data required

sentiment analysis



# Stochastic gradient descent and pretrain/finetune

Why should pretraining and finetuning help, from a “training neural nets” perspective?

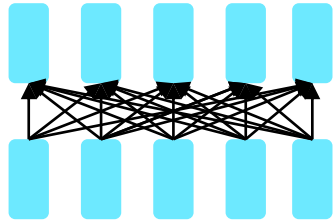
- Consider, provides parameters  $\hat{\theta}$  by approximating  $\min_{\theta} \mathcal{L}_{\text{pretrain}}(\theta)$ .
  - (The pretraining loss.)
- Then, finetuning approximates  $\min_{\theta} \mathcal{L}_{\text{finetune}}(\theta)$ , starting at  $\hat{\theta}$ .
  - (The finetuning loss)
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to  $\hat{\theta}$  during finetuning.
  - So, maybe the finetuning local minima near  $\hat{\theta}$  tend to generalize well!
  - And/or, maybe the gradients of finetuning loss near  $\hat{\theta}$  propagate nicely!  
pretraining simply sees a lot more variety than the finetuning task ever will

# Lecture Plan

1. A brief note on subword modeling
2. Motivating model pretraining from word embeddings
3. Model pretraining three ways
  1. Encoders
  2. Encoder-Decoders
  3. Decoders
4. What do we think pretraining is teaching?

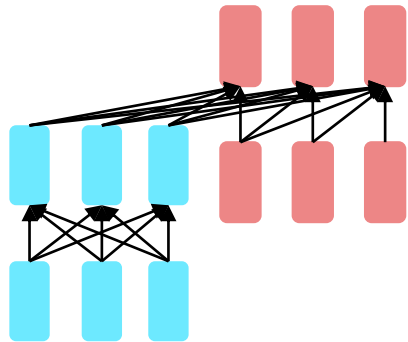
# Pretraining for three types of architectures

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



**Encoders**

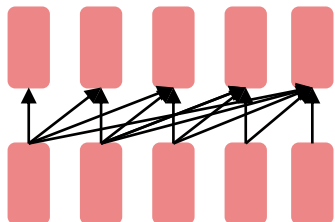
- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



**Encoder-  
Decoders**

Encoder gets the bidirectional context, decoder only unidirectional

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

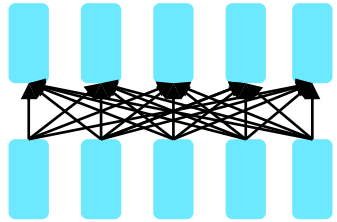


**Decoders**

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

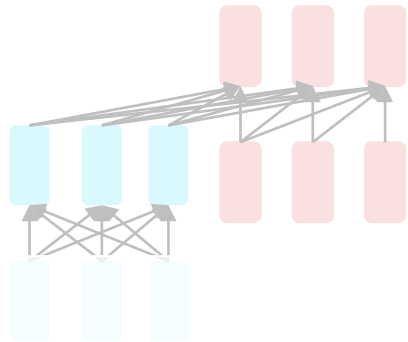
# Pretraining for three types of architectures

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



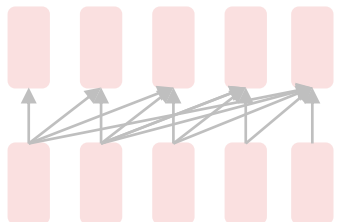
**Encoders**

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



**Encoder-  
Decoders**

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



**Decoders**

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

# 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.

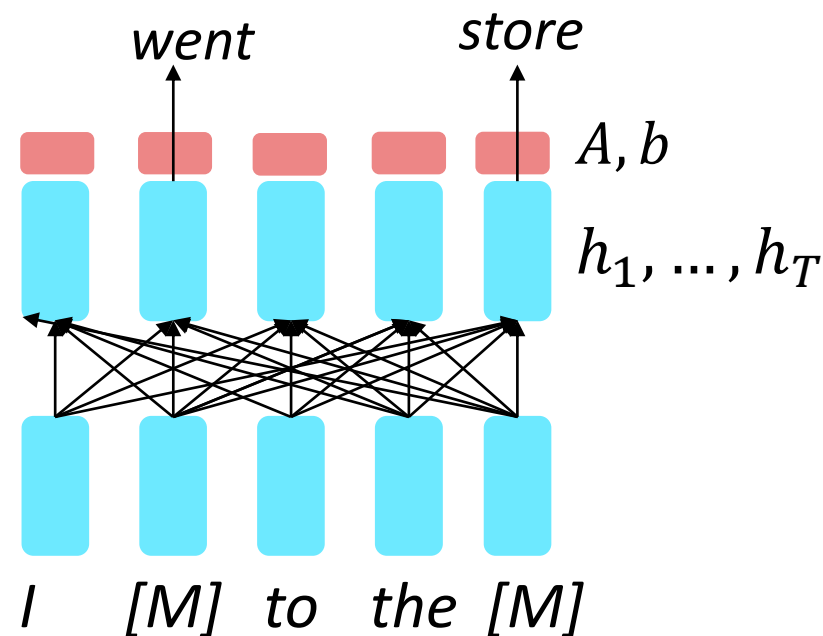
$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$

$$y_i \sim Aw_i + b$$

vocab y      Ahi+b

hidden state  $\rightarrow$  vocab

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]

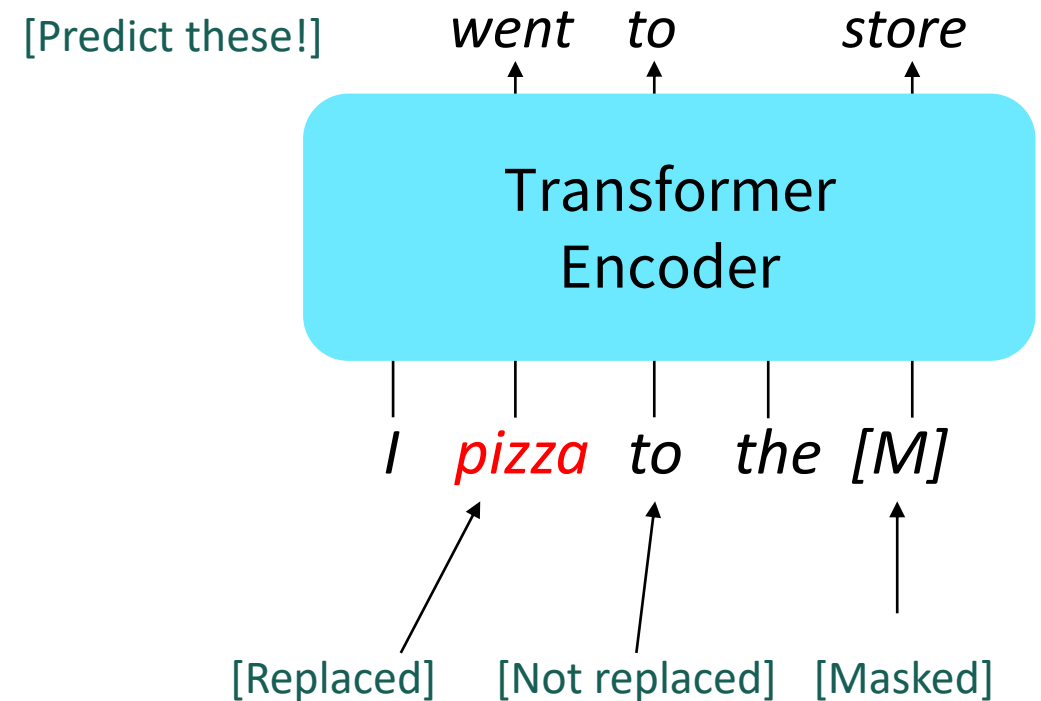


# BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the “Masked LM” objective and **released the weights of a pretrained Transformer**, a model they labeled BERT.

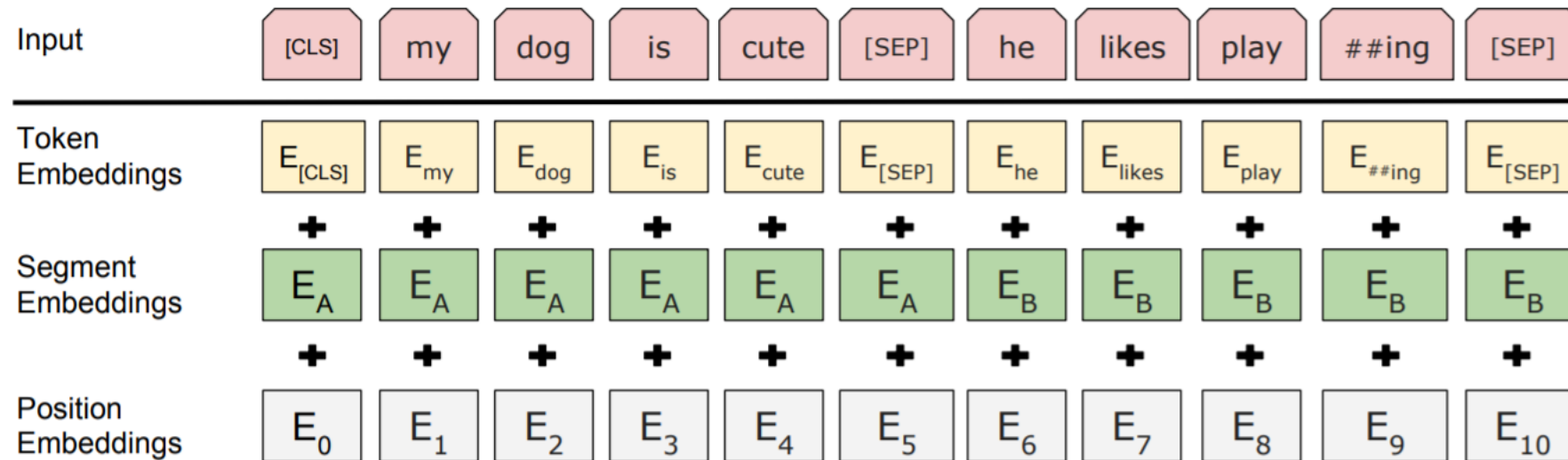
Some more details about Masked LM for BERT:

- 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  
80% of 15%
  - 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

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



segment embeddings were used to encode the info

- 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.

# 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

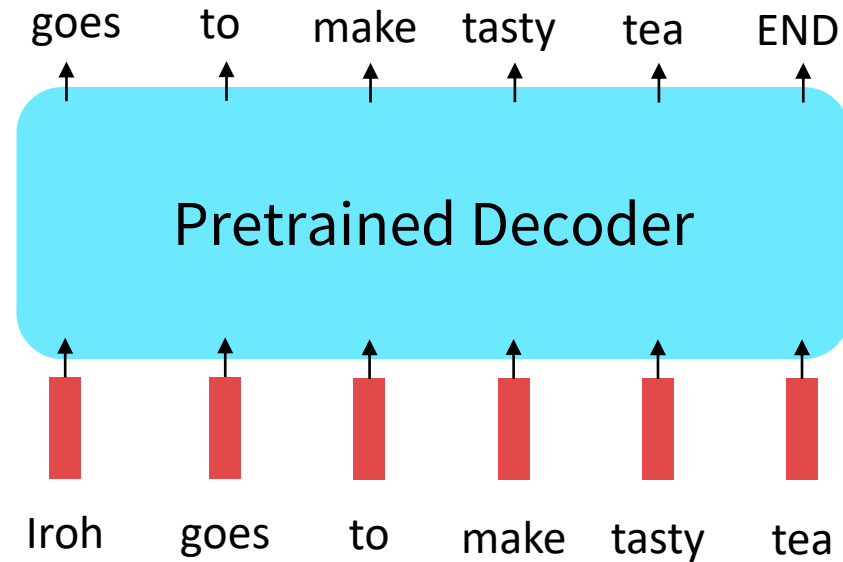
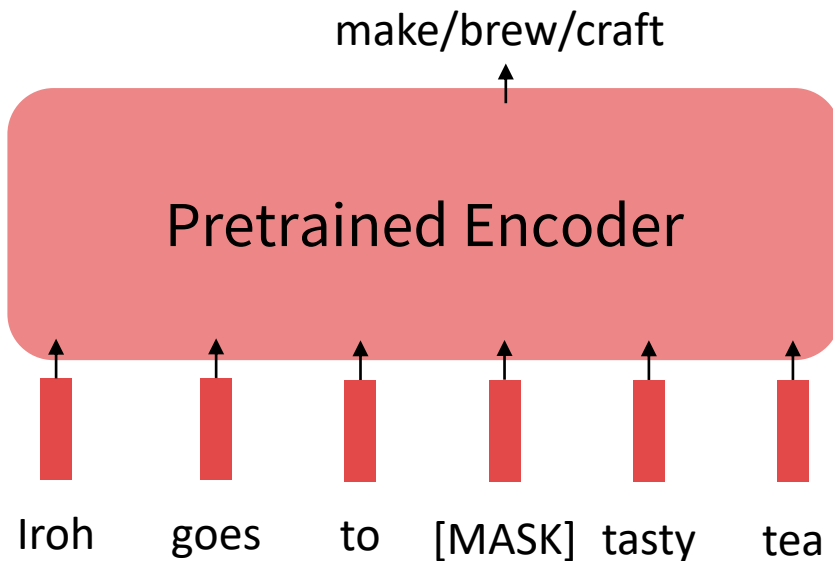
Evaluation task datasets

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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

# Limitations of pretrained encoders

Those results looked great! Why not use 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.



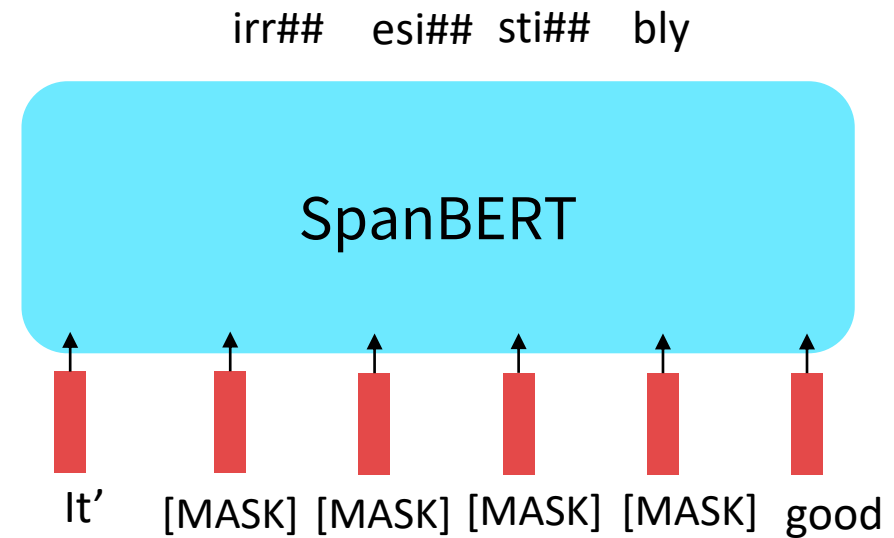
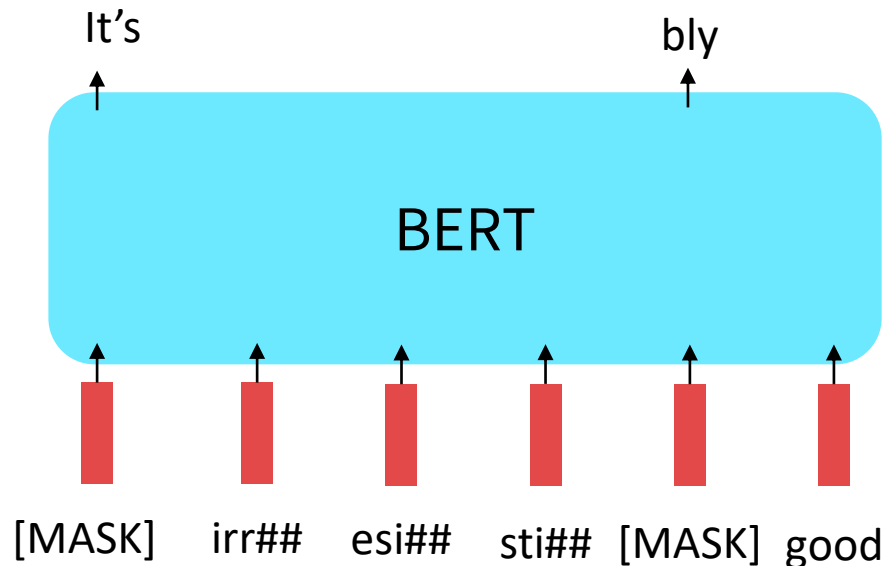
BERT would not be good for summarization and other sequence generation tasks but will be good for sentiment analysis, masked prediction, etc.

# 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



# Extensions of BERT

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

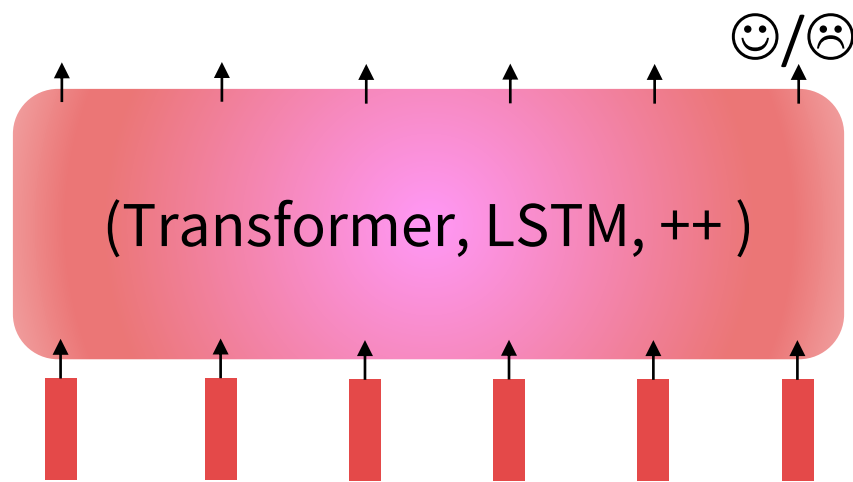
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	<b>94.6/89.4</b>	<b>90.2</b>	<b>96.4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

# Full Finetuning vs. Parameter-Efficient Finetuning

Finetuning every parameter in a pretrained model works well, but is memory-intensive. But **lightweight** finetuning methods adapt pretrained models in a constrained way. Leads to **less overfitting** and/or **more efficient finetuning and inference**.

## Full Finetuning

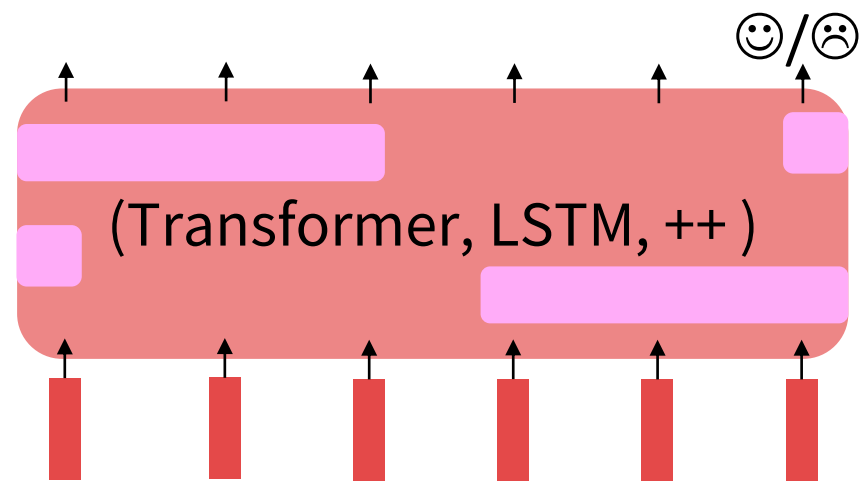
Adapt all parameters



*... the movie was ...*

## Lightweight Finetuning

Train a few existing or new parameters



*... the movie was ...*

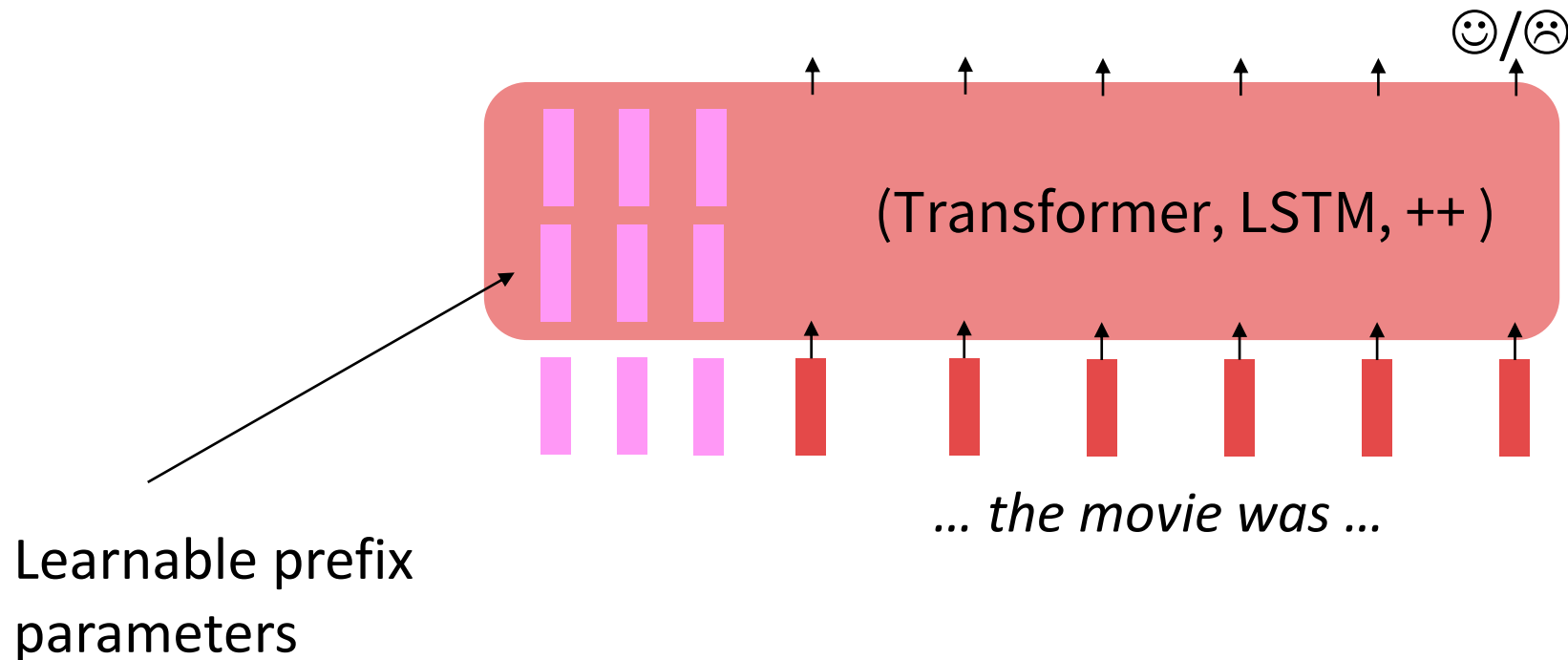


# Parameter-Efficient Finetuning: Prefix-Tuning, Prompt tuning

Prefix-Tuning adds a **prefix** of parameters, and **freezes all pretrained parameters**.

The prefix is processed by the model just like real words would be.

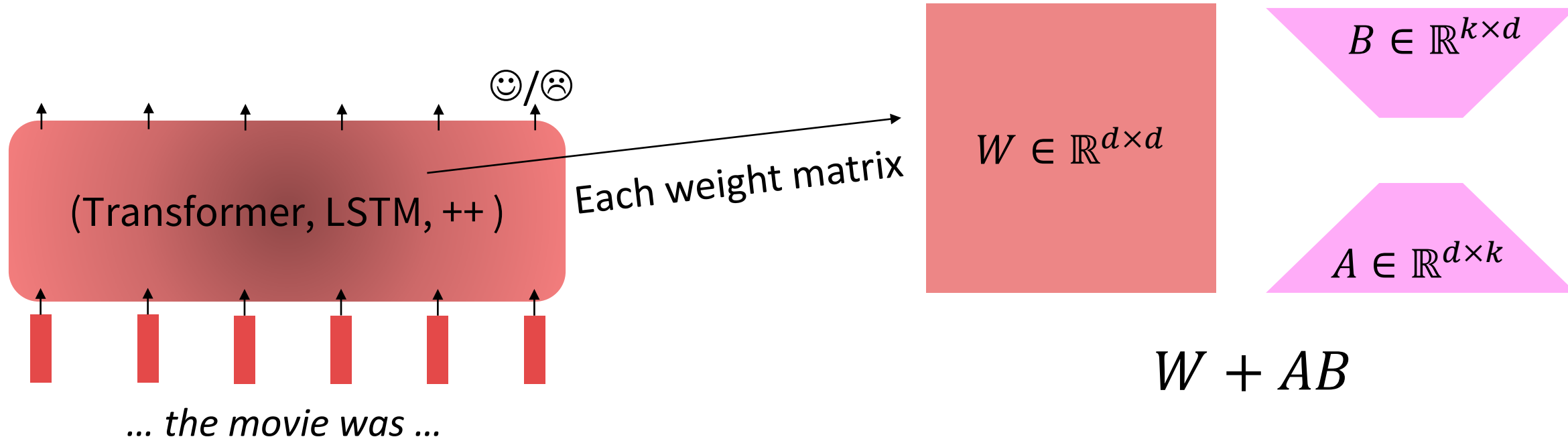
Advantage: each element of a batch at inference could run a different tuned model.



# Parameter-Efficient Finetuning: Low-Rank Adaptation

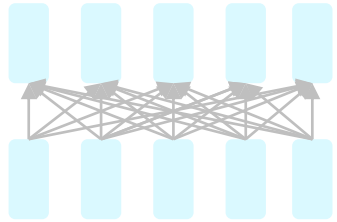
Low-Rank Adaptation Learns a low-rank “diff” between the pretrained and finetuned weight matrices.

Easier to learn than prefix-tuning.



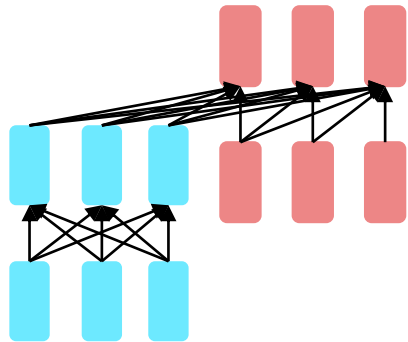
# Pretraining for three types of architectures

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



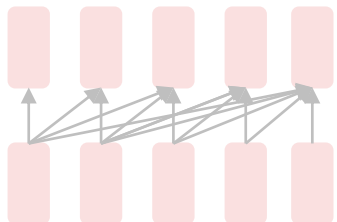
**Encoders**

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



**Encoder-  
Decoders**

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



**Decoders**

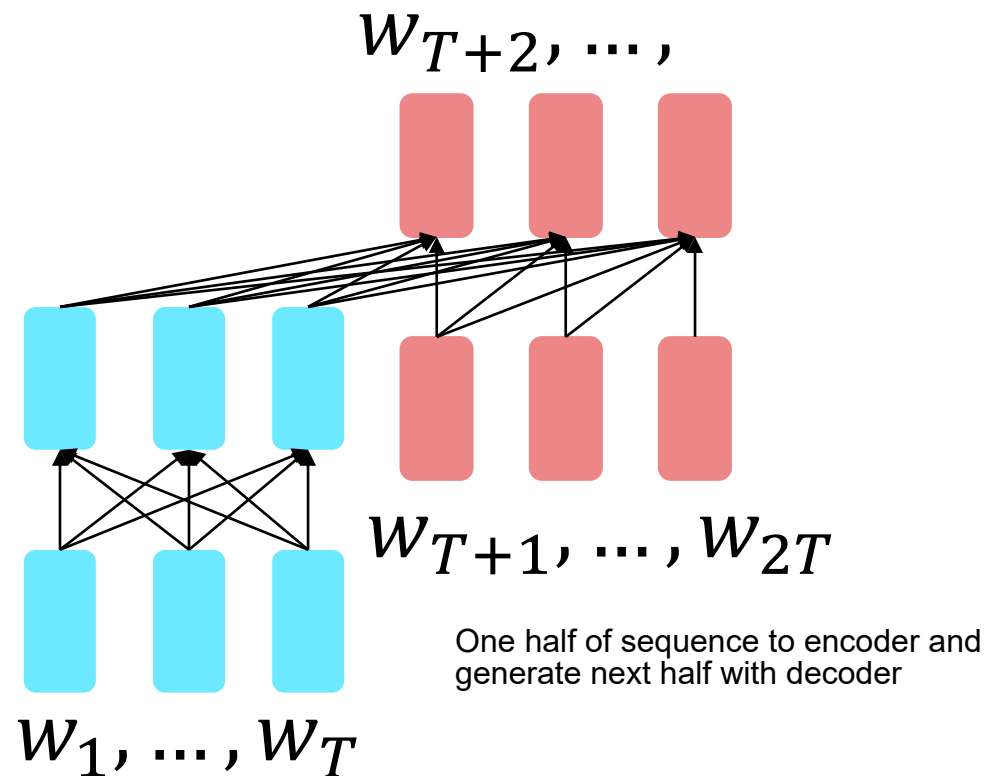
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

# 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.

$$\begin{aligned}h_1, \dots, h_T &= \text{Encoder}(w_1, \dots, w_T) \\h_{T+1}, \dots, h_{2T} &= \text{Decoder}(w_1, \dots, w_T, h_1, \dots, h_T) \\y_i &\sim Ah_i + b, i > T\end{aligned}$$

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!

Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.

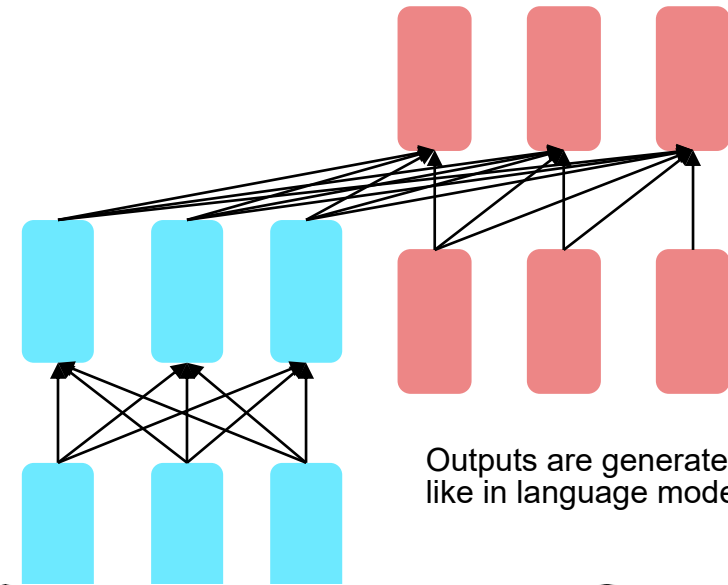
This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.

Inputs

Thank you **<X>** me to your party **<Y>** week.

Targets

**<X>** for inviting **<Y>** last **<Z>**



Outputs are generated in a sequence like in language modeling

After finetuning, Good for machine translation, input that can provide bidirectional context, and generate an output

# Pretraining encoder-decoders: what pretraining objective to use?

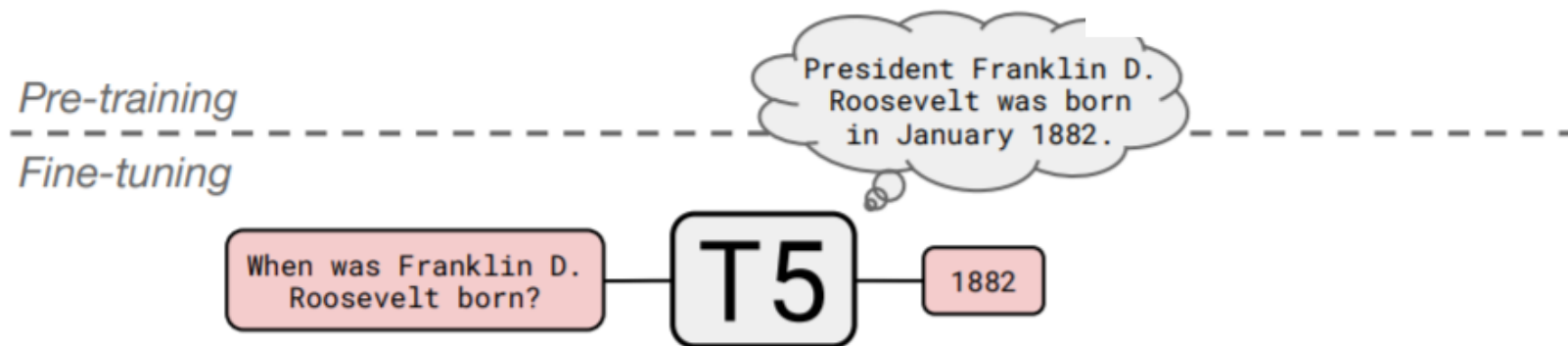
[Raffel et al., 2018](#) found encoder-decoders to work better than decoders for their tasks, and span corruption (denoising) to work better than language modeling.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
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

# Pretraining encoder-decoders: what pretraining objective to use?

After finetuning on some questions and answers, T5 could answer unseen questions based on the parameters learned during pretraining

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

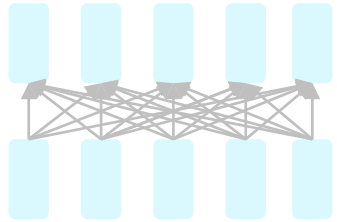
TQA: Trivia QA

All “open-domain”  
versions

	NQ	WQ	TQA		
			dev	test	
<u>Karpukhin et al. (2020)</u>	<b>41.5</b>	42.4	<b>57.9</b>	–	
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
<u>T5.1.1-XXL + SSM</u>	35.2	<b>42.8</b>	51.9	<b>61.6</b>	

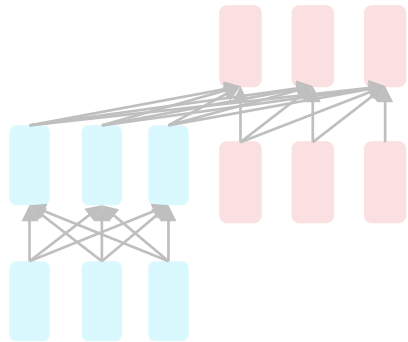
# Pretraining for three types of architectures

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



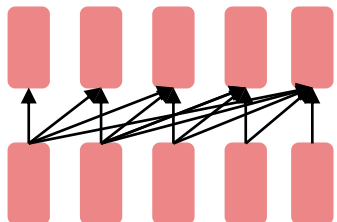
**Encoders**

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



**Encoder-  
Decoders**

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



**Decoders**

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- All the biggest pretrained models are Decoders.



# Pretraining decoders

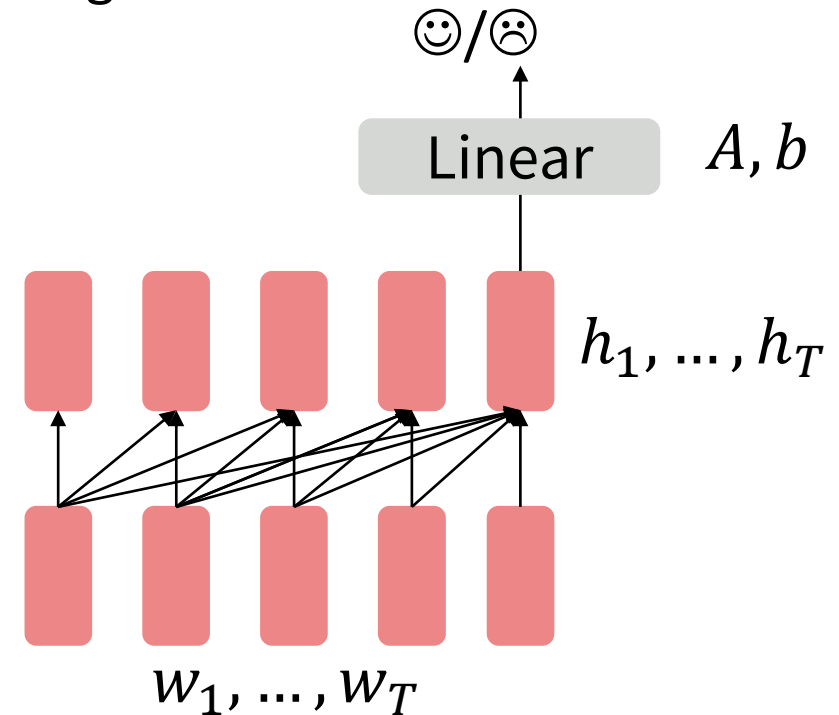
When using language model pretrained decoders, we can ignore that they were trained to model  $p(w_t|w_{1:t-1})$ .

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

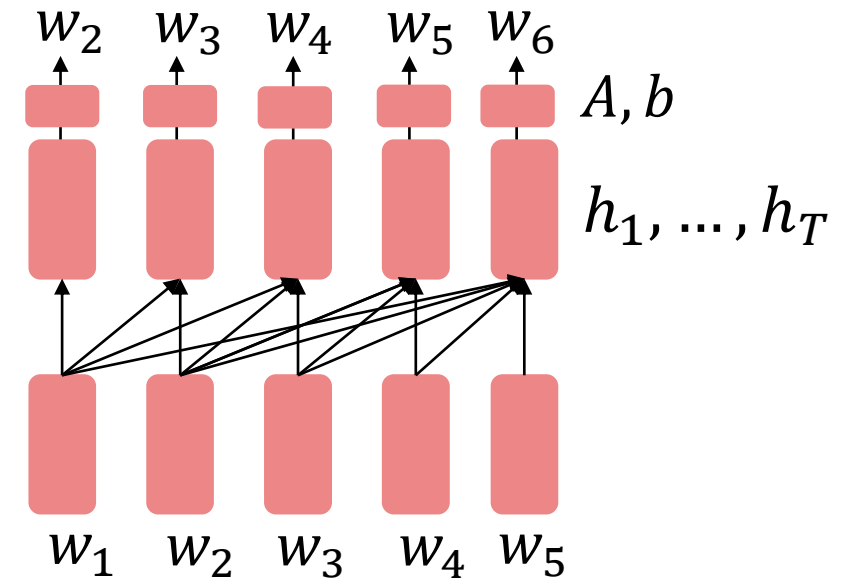
It's natural to pretrain decoders as language models and then use them as generators, finetuning their  $p_{\theta}(w_t|w_{1:t-1})$ !

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)

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

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, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges 40000, subwords / vocabulary
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym “GPT” never showed up in the original paper; it could stand for “Generative PreTraining” or “Generative Pretrained Transformer”

Encoder-only models: Good for tasks that require understanding of the input, such as sentence classification and named entity recognition.

Decoder-only models: Good for generative tasks such as text generation.

Encoder-decoder models or sequence-to-sequence models: Good for generative tasks that require an input, such as translation or summarization.

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

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

**Natural Language Inference:** Label pairs of sentences as *entailing/contradictory/neutral*

Premise: *The man is in the doorway*  
Hypothesis: *The person is near the door* } **entailment**

Radford et al., 2018 evaluate on natural language inference.

Here's roughly how the input was formatted, as a sequence of tokens for the decoder.

[START] *The man is in the doorway* [DELIM] *The person is near the door* [EXTRACT]

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)	-	-	<u>89.3</u>	-	-	-
CAFE [58] (5x)	80.2	79.0	<u>89.3</u>	-	-	-
Stochastic Answer Network [35] (3x)	<u>80.6</u>	<u>80.1</u>	-	-	-	-
CAFE [58]	78.7	77.9	88.5	<u>83.3</u>		
GenSen [64]	71.4	71.3	-	-	<u>82.3</u>	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	<b>61.7</b>
Finetuned Transformer LM (ours)	<b>82.1</b>	<b>81.4</b>	<b>89.9</b>	<b>88.3</b>	<b>88.1</b>	56.0

BERT was slightly better than GPT as it gave bidirectional context. BERT came after GPT.

# 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 (1.5B) 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.

# 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 take their predictions.

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):**

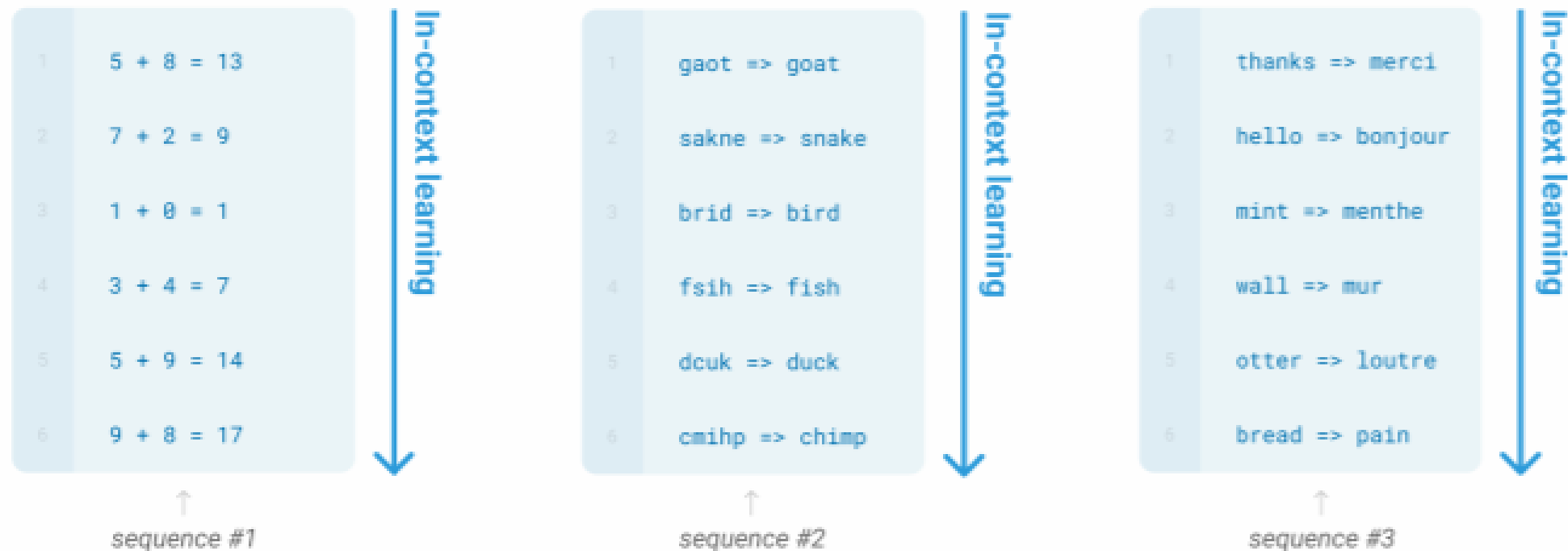
loutre...”



# 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.

Learning via SGD during unsupervised pre-training



# Scaling Efficiency: how do we best use our compute

GPT-3 was **175B parameters** and trained on **300B** tokens of text.

Roughly, the cost of training a large transformer scales as **parameters\*tokens**

Did OpenAI strike the right parameter-token data to get the best model? No.

Model	Size (# Parameters)	Training Tokens
LaMDA ( <a href="#">Thoppilan et al., 2022</a> )	137 Billion	168 Billion
GPT-3 ( <a href="#">Brown et al., 2020</a> )	175 Billion	300 Billion
Jurassic ( <a href="#">Lieber et al., 2021</a> )	178 Billion	300 Billion
<i>Gopher</i> ( <a href="#">Rae et al., 2021</a> )	280 Billion	300 Billion
MT-NLG 530B ( <a href="#">Smith et al., 2022</a> )	530 Billion	270 Billion
<i>Chinchilla</i>	70 Billion	1.4 Trillion

This 70B parameter model is better than the much larger other models!

# The prefix as task specification and scratch pad: chain-of-thought

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. ❌

## Chain-of-Thought Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✅

Giving model time to think

These models are very powerful but exceptionally hard to explain their actions. All we have are theories.

[[Wei et al., 2023](#)]

# Outline

1. A brief note on subword modeling
2. Motivating model pretraining from word embeddings
3. Model pretraining three ways
  1. Encoders
  2. Encoder-Decoders
  3. Decoders
4. What do we think pretraining is teaching?

# What kinds of things does pretraining teach?

There's increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language. Taking our examples from the start of class:

- *Stanford University is located in \_\_\_\_\_, California.* [Trivia]
- *I put \_\_\_\_ fork down on the table.* [syntax]
- *The woman walked across the street, checking for traffic over \_\_\_\_ shoulder.* [coreference]
- *I went to the ocean to see the fish, turtles, seals, and \_\_\_\_\_.* [lexical semantics/topic]
- *Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was \_\_\_\_.* [sentiment]
- *Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the \_\_\_\_\_.* [some reasoning – this is harder]
- *I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, \_\_\_\_* [some basic arithmetic; they don't learn the Fibonacci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.
- More on all this in the interpretability lecture!

# Parting remarks

These models are still not well-understood.

“Small” models like BERT have become general tools in a wide range of settings.

More on this in later lectures!

Assignment 5 out Thursday! Tuesday’s and today’s lectures in its subject matter.

Paper: Dont stop pretraining

Pretrain on general data, extend pretraining on your specific unlabelled data, finetune in labelled data

Certain large amount of specific data will be necessary on this training paradigm