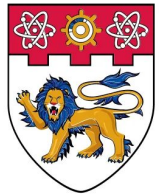


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Natural Language Processing

SC4002 / CE4045 / CZ4045

by Wang Wenya

Email: wangwy@ntu.edu.sg

Click here for Lecture 5

Modules we will cover

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Transformer

Attention mechanism, encoder/decoder

Sequence

Sequence modeling, seq2seq learning

Pretraining

Masking, natural language generation

Prompting

Prompts, in-context learning

Outline for today

01 Subword Modeling

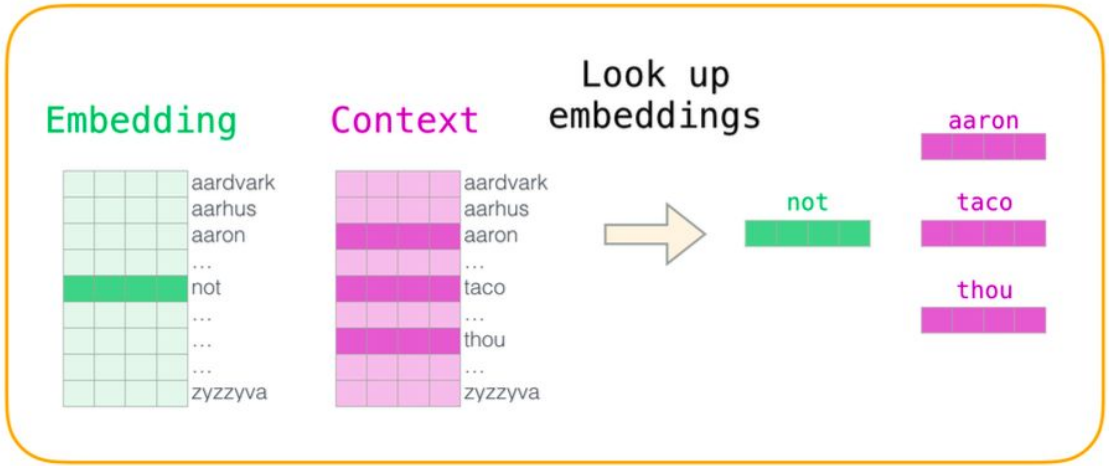
02 Knowledge Transfer

03 Pretraining vs. Finetuning

04 Natural Language Generation

Word2vec (Revisit)

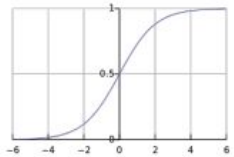
- Requires a word dictionary for look-up
- Same word vector with different contexts



Word2vec with End Tasks (Revisit)

$$J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

predicted model probability of class



$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

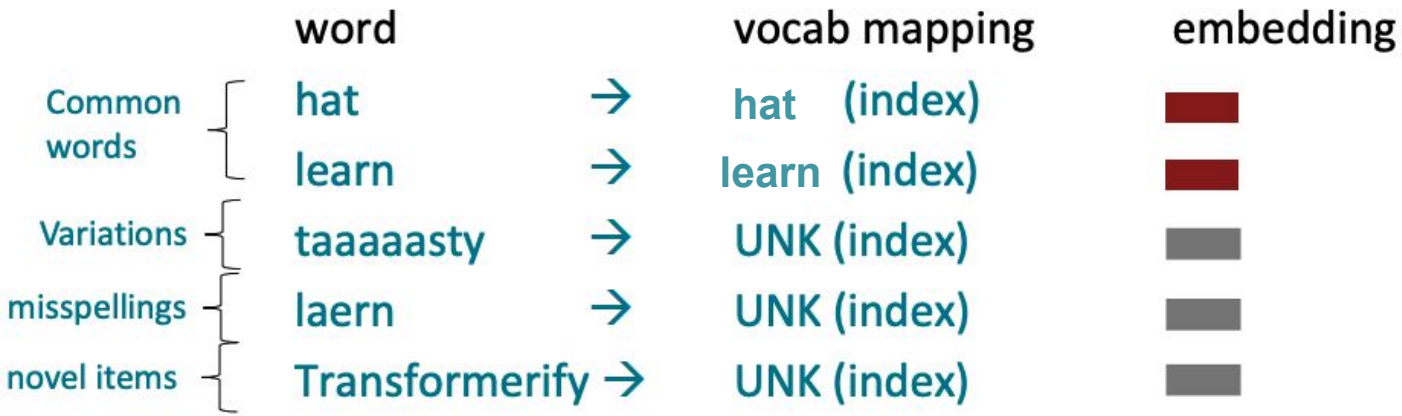
$$\boldsymbol{x} \quad \text{(input)}$$

f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

Problem with Word Dictionary

Source: stanford 224n

- 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.



Problem with Word Dictionary

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

Byte-Pair Encoding

Source: stanford 224n

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.

Byte-Pair Encoding

An unsupervised word segmentation algorithm:

- Start with a unigram vocabulary of all (Unicode) characters in data

Vocabulary

l, o, w, e, r, n, w, s, t, i, d

Dictionary

5 l o w
2 l o w e r
6 n e w e s t
3 w i d e s t

Dictionary of word frequency

Byte-Pair Encoding

An unsupervised word segmentation algorithm:

- Most frequent ngram pairs ↦ a new ngram

Dictionary

5 l o w
2 l o w e r
6 n e w **es** t
3 w i d **es** t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Add a pair (e, s) with freq 9

Dictionary of word frequency

Byte-Pair Encoding

An unsupervised word segmentation algorithm:

- Most frequent ngram pairs ↦ a new ngram

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

Dictionary

5	**lo** w
2	**lo** w e r
6	n e w est
3	w i d est

Add a pair (l, o) with freq 7

Dictionary of word frequency

Subword Models

Source: stanford 224n

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	

you can use as subwords, or process as a whole word

Problem with Identical Word Vectors

Source: stanford 224n

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)

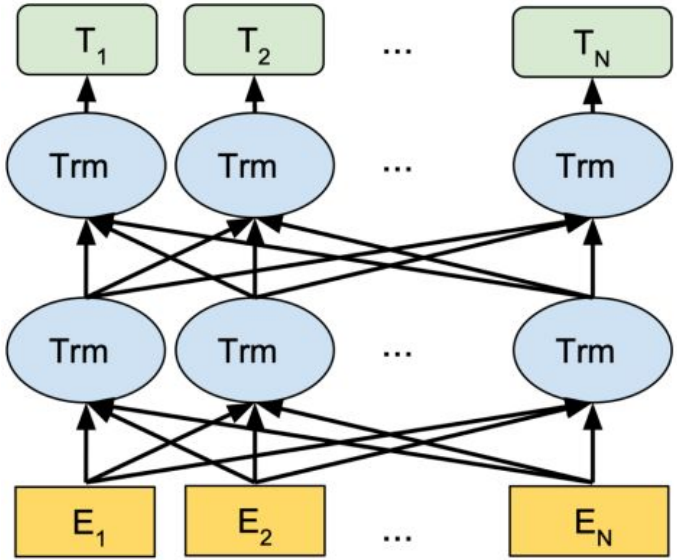
This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:

> "... 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.

Solution: BERT

- BERT: Bidirectional Encoder Representations from Transformers [Devlin et al., 2018]
- BERT uses Word Piece (similar mechanism as Byte-pair) tokenization
- BERT gives contextualized word representations via multi-layer self-attention (transformer)



Outline for today

01 Subword Modeling

02 Knowledge Transfer

03 Pretraining to Finetuning

04 Natural Language Generation

What is Transfer Learning

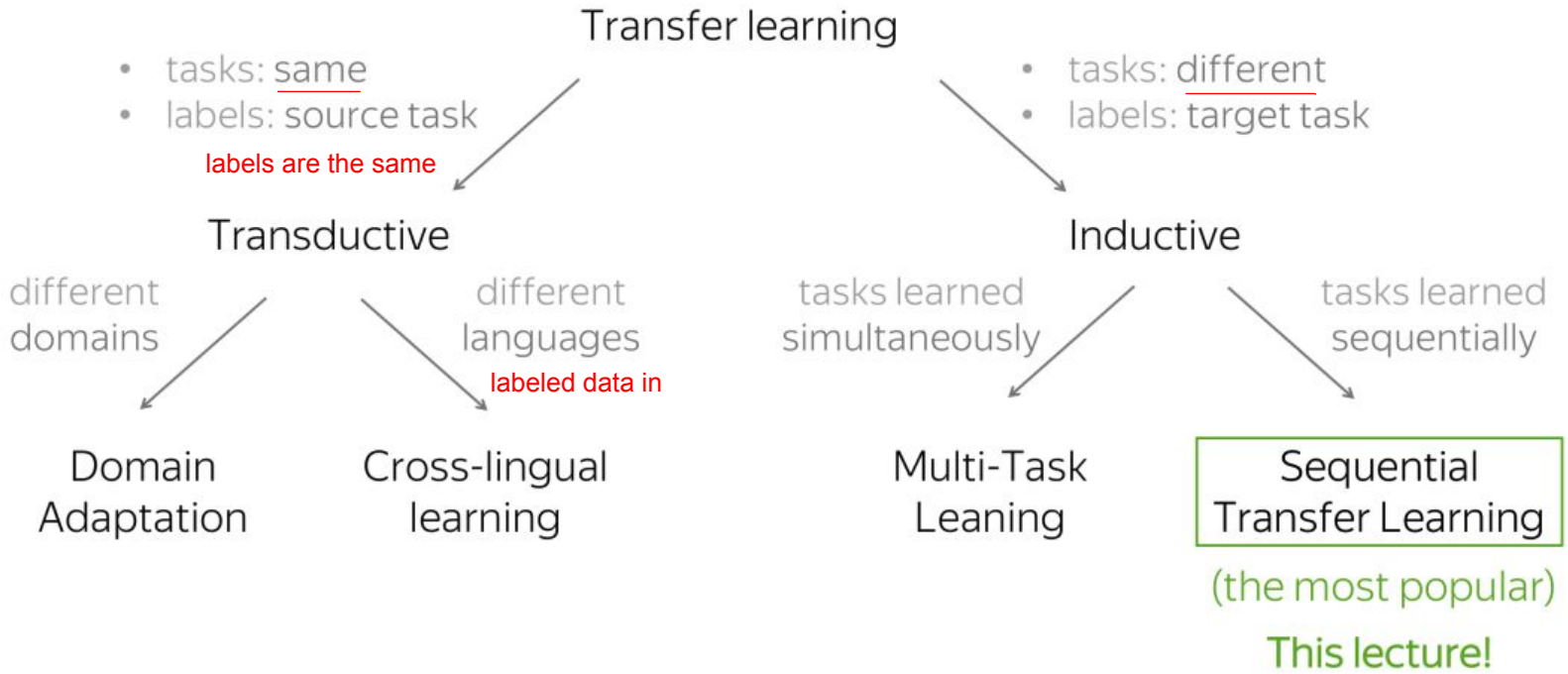
Typically, we do not have enough training data to estimate an accurate model on the data for the target task

Consider question answering:

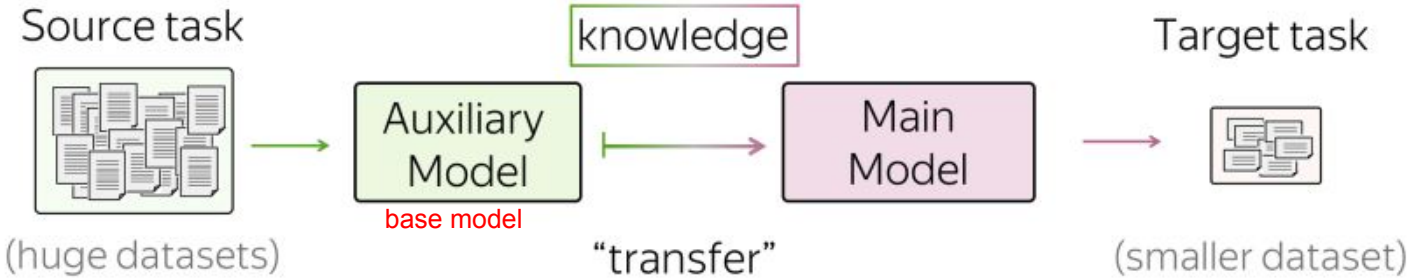
It is impossible to maintain a large and up-to-date collection of question-answer pairs for all possible questions, domains, languages cultures…

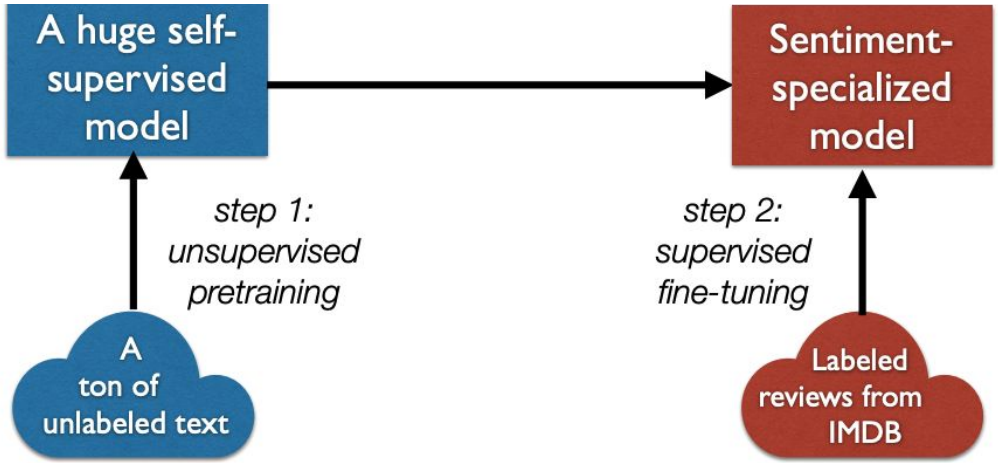
How can we benefit from data for other tasks? (including tasks for which data occurs 'naturally' such as language modeling = predicting next word)

Transfer Learning is a Broad Area



Transfer Learning in NLP





Pre-training and Fine-tuning

- Train from scratch

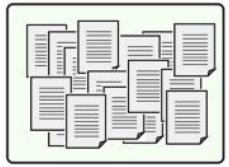
What they will know:

Domain-specific labeled data

May be not enough to learn relationships between words

- Take pretrained (Word2Vec, GloVe)

What they will know:

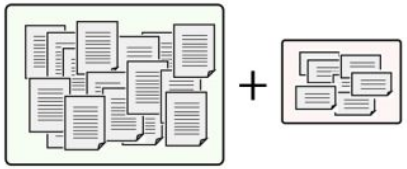


Huge unlabeled corpus

Know relationships between words, but are **not** specific to the task

- Initialize with pretrained, then fine-tune

What they will know:

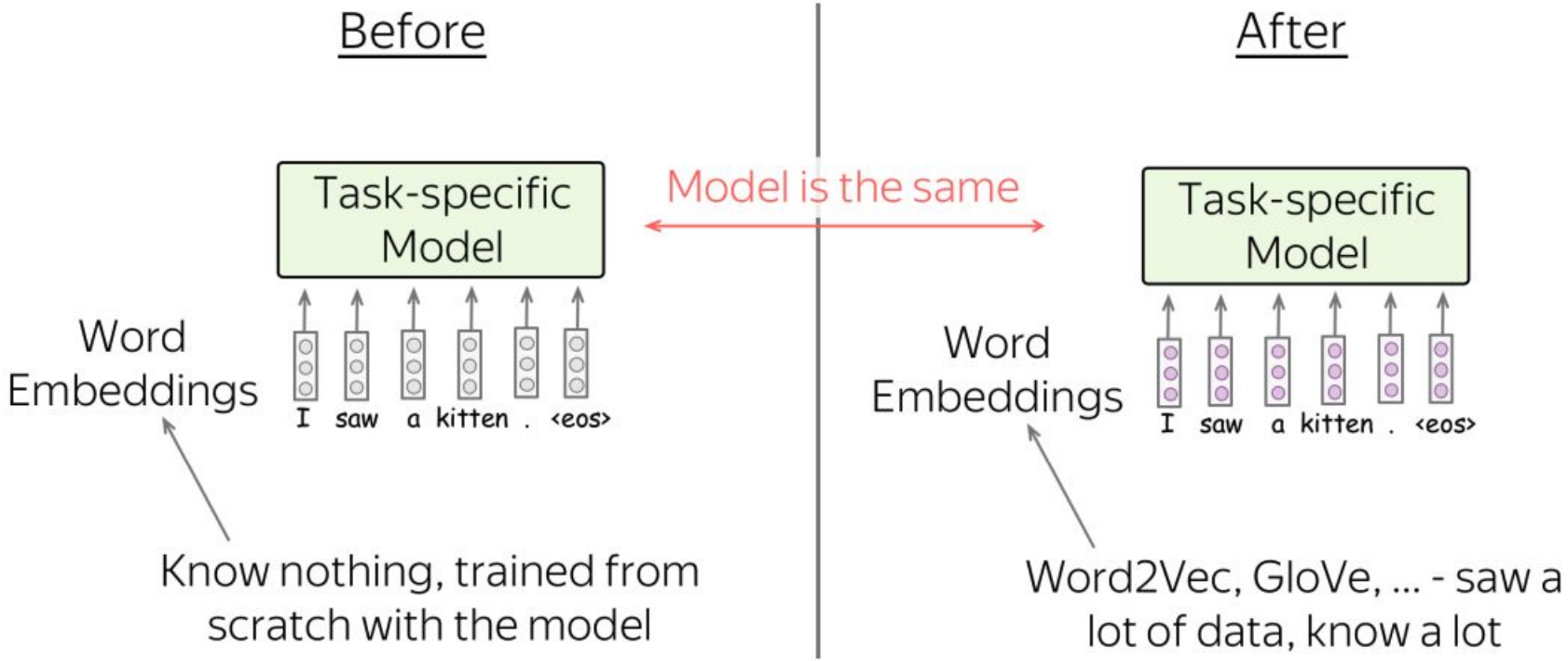


Know relationships between words and adapted for the task

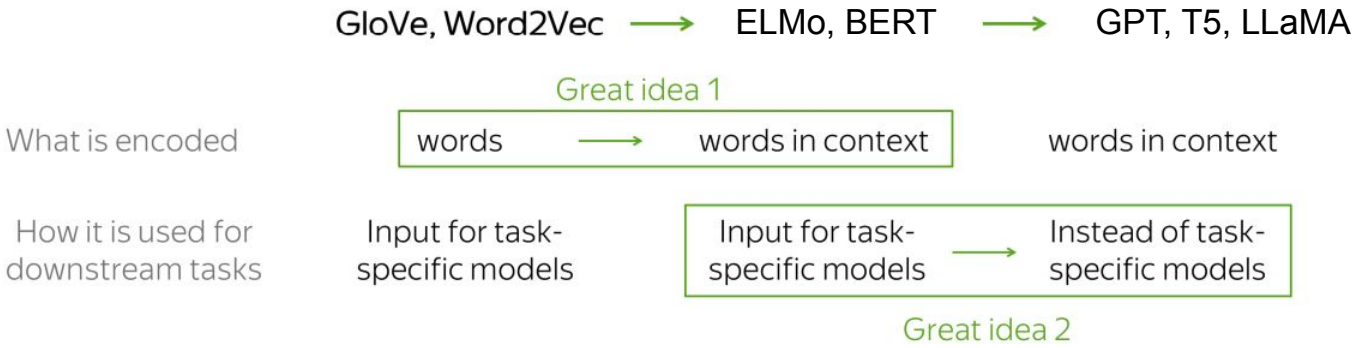
"**Transfer**" knowledge from a huge unlabeled corpus to your task-specific model

Starting with pretrained embeddings and then fine-tuning (= training) them on the specific task enables the transfer of knowledge from a vast dataset, while specializing the embedding to the target task and domain

Transfer through Word Embeddings



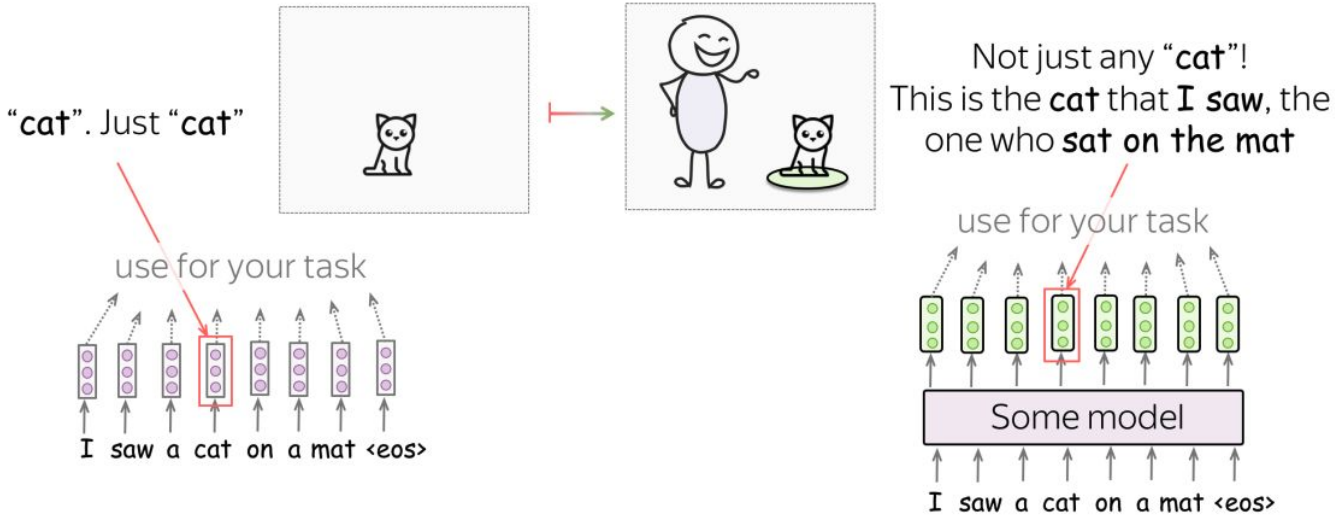
Limitations and Solutions



The two great ideas:

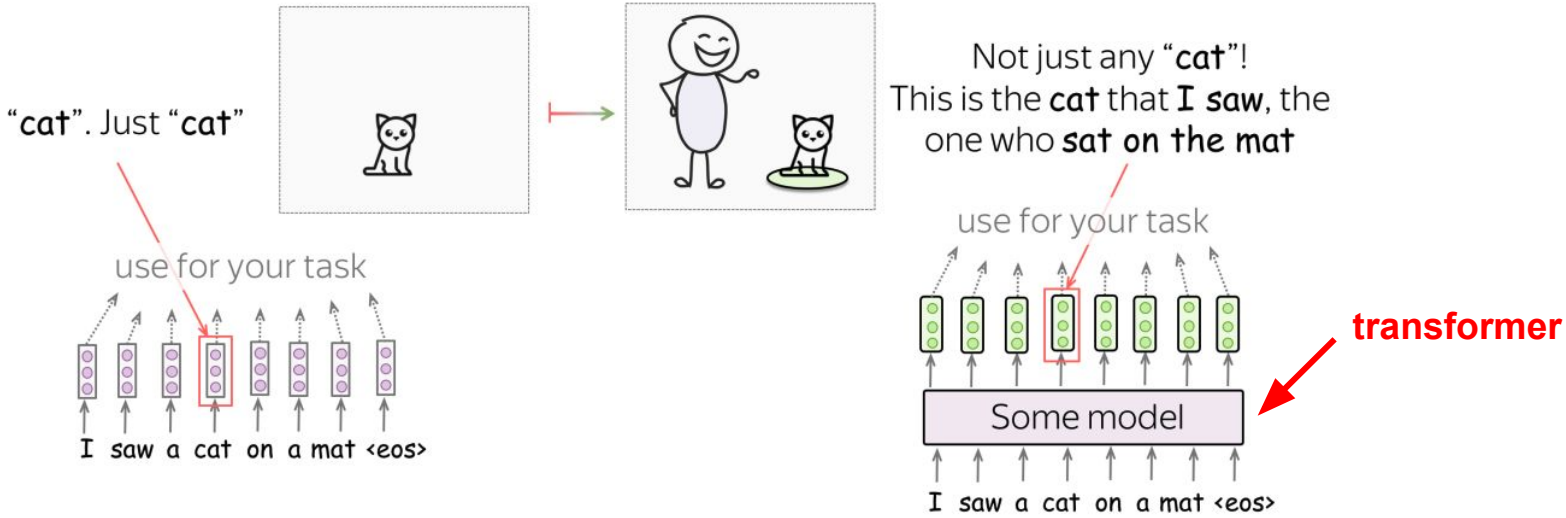
- what is encoded: from words to words-in-context (the transition from Word2Vec to ELMo)
- usage for downstream tasks: from task-specific models to task-unified models (the transition from BERT to GPT)

BERT: Transferring words-in-context



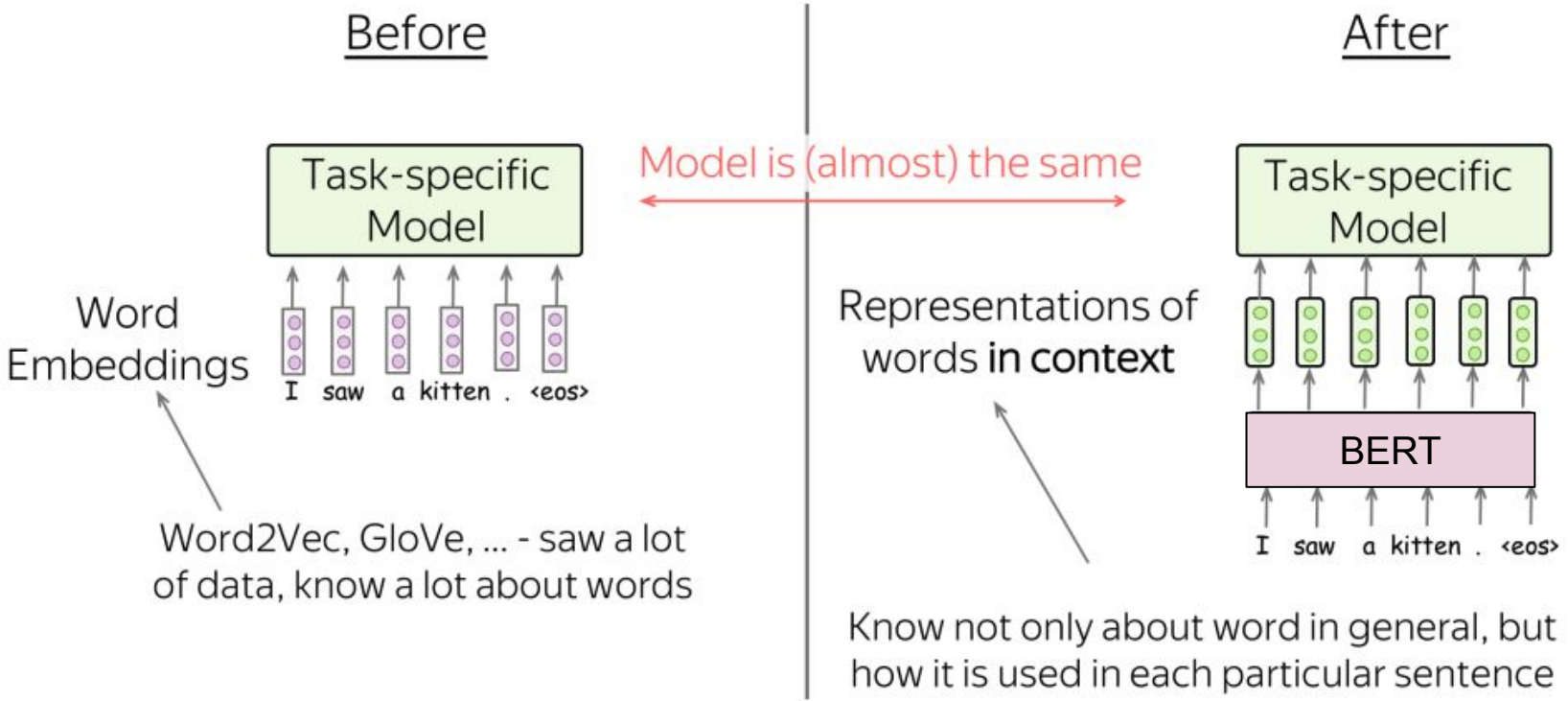
If successful, the representation of the word will both disambiguate the term and also specialize it for the given context

BERT: Transferring words-in-context

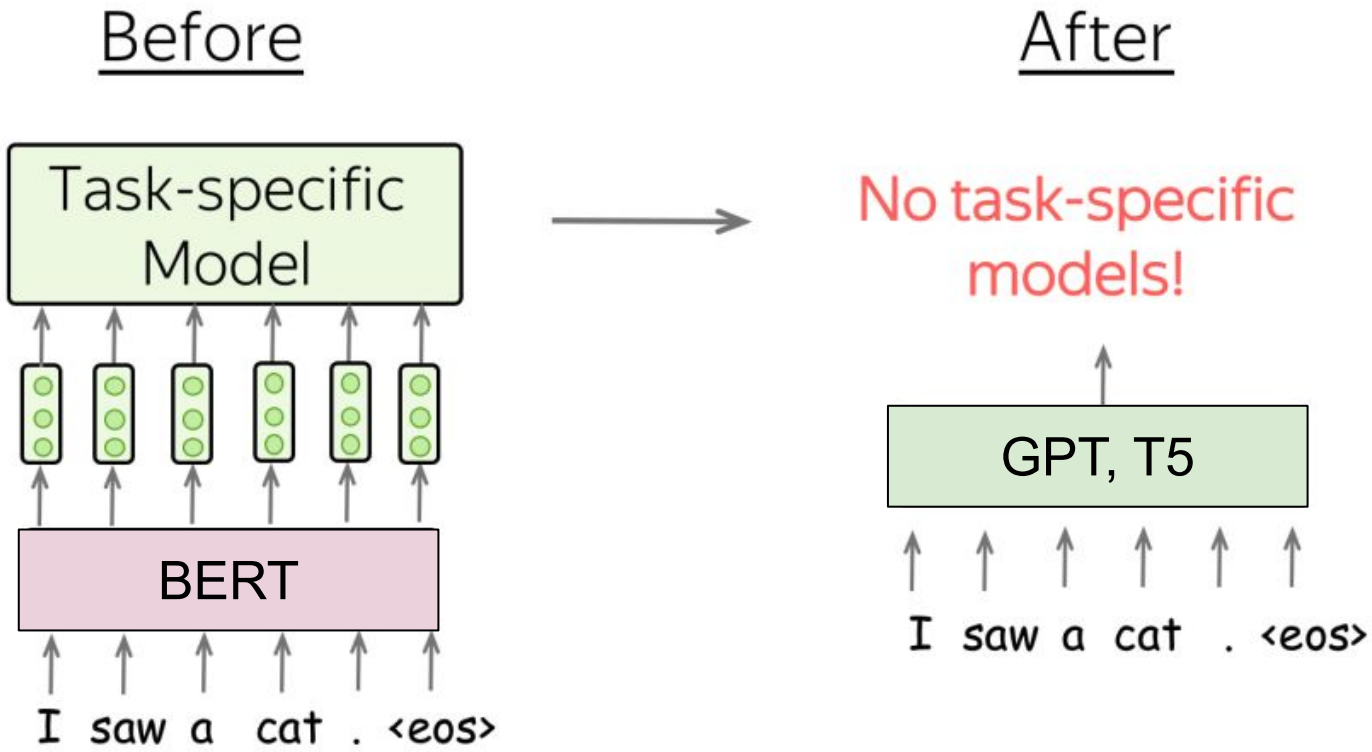


If successful, the representation of the word will both disambiguate the term and also specialize it for the given context

BERT: Transferring words-in-context



Going Further: Transferring Entire Models



Outline for today

01 Subword Modeling

02 Knowledge Transfer

03 Pretraining vs. Finetuning

04 Natural Language Generation

Where We Were: Pipeline

it used to be just word embedding that was

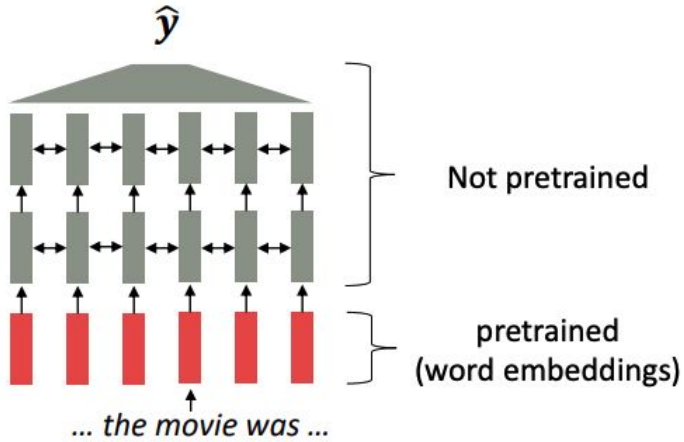
Source: stanford 224n

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 Are Going: Pretraining

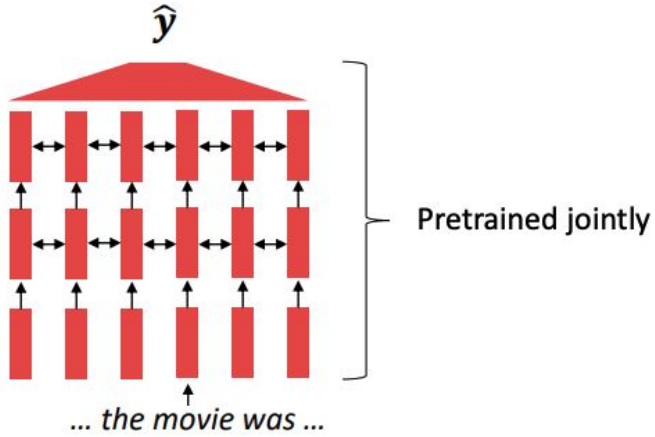
Source: stanford 224n

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**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from

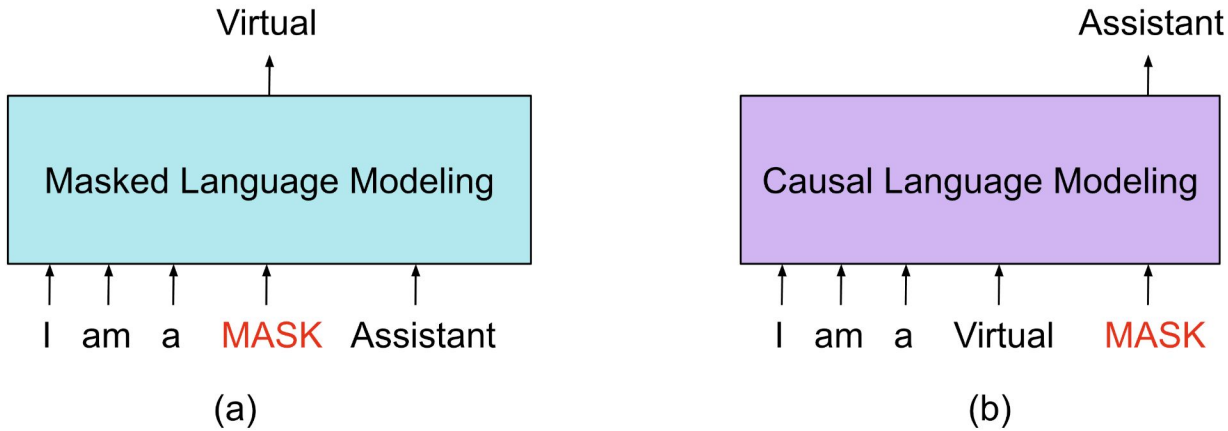
Transfer word-in-context, entire model is transferred



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

Pretraining Objective

- Masked Language Modeling (MLM)
- (Causal) Language Modeling (LM)



https://www.holisticai.com/blog/from-transformer-architecture-to-prompt-engineering

Masked Language Modeling

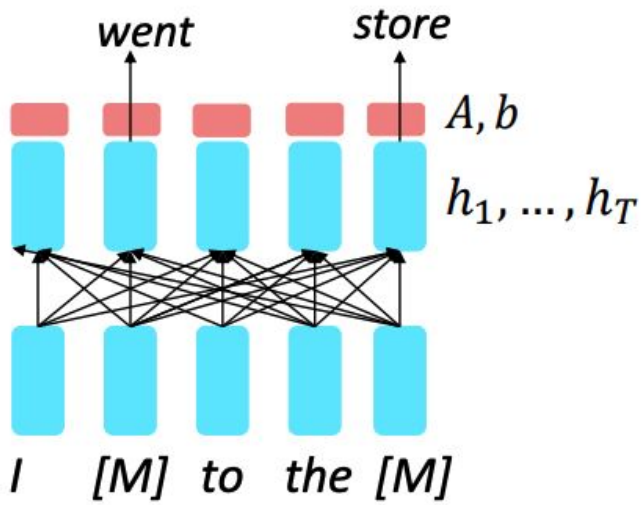
Source: stanford 224n

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

$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$y_i \sim Aw_i + b$$

softmax (A.h_i + b)

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 uses this model, where it does bi-directional handling

(Causal) Language Modeling (Revisit)

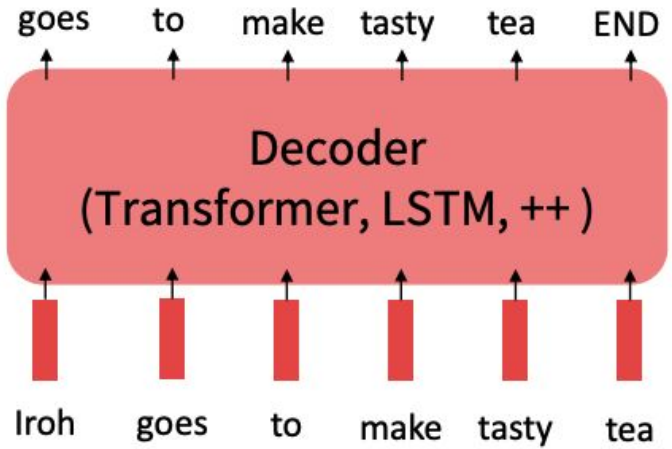
More formally: given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Pretraining via Language Modeling

- Train a neural network to perform language modeling on a large amount of text.
- Predict the next word given the past context



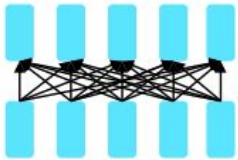
3 Pretraining Paradigms

- Encoders
- Encoder-Decoders
- Decoders

3 Pretraining Paradigms

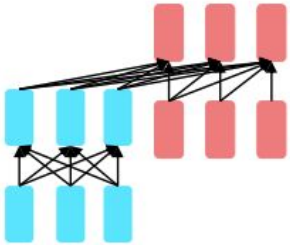
Source: stanford 224n

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



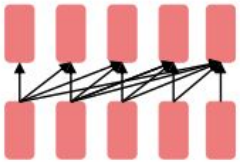
Encoders

- 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?



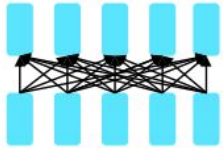
Decoders

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

Encoders

Source: stanford 224n

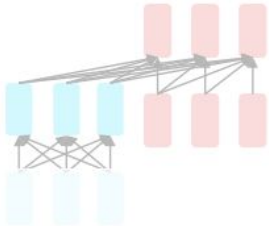
The neural architecture influences the type of pretraining, and natural use cases.



Encoders

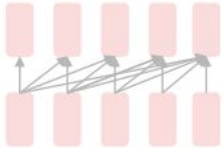
BERT models

- 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

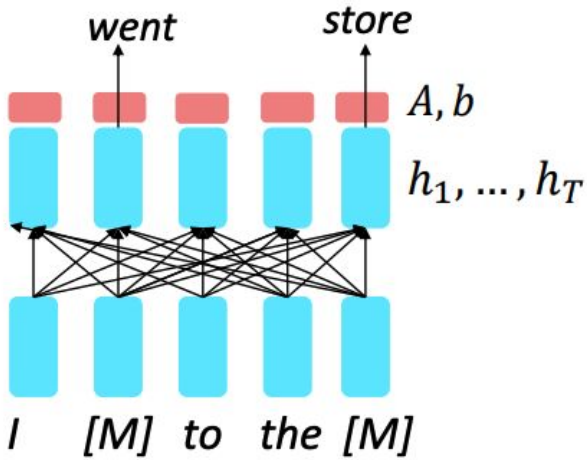
Encoders [Devlin et al., 2018]

Source: stanford 224n

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.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$



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**.

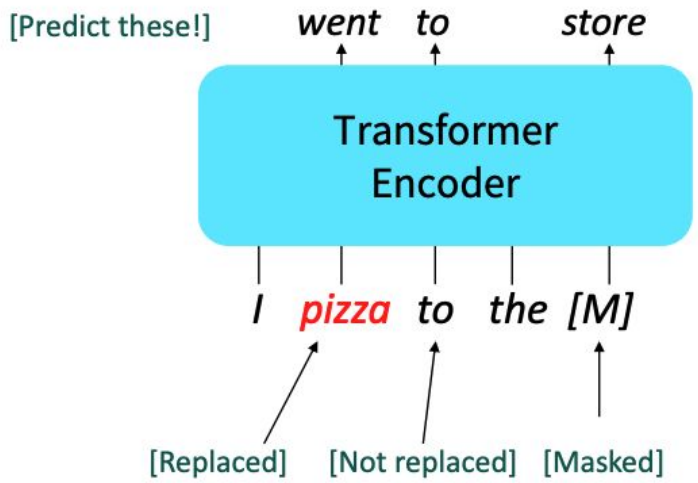
BERT [Devlin et al., 2018]

Source: stanford 224n

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:

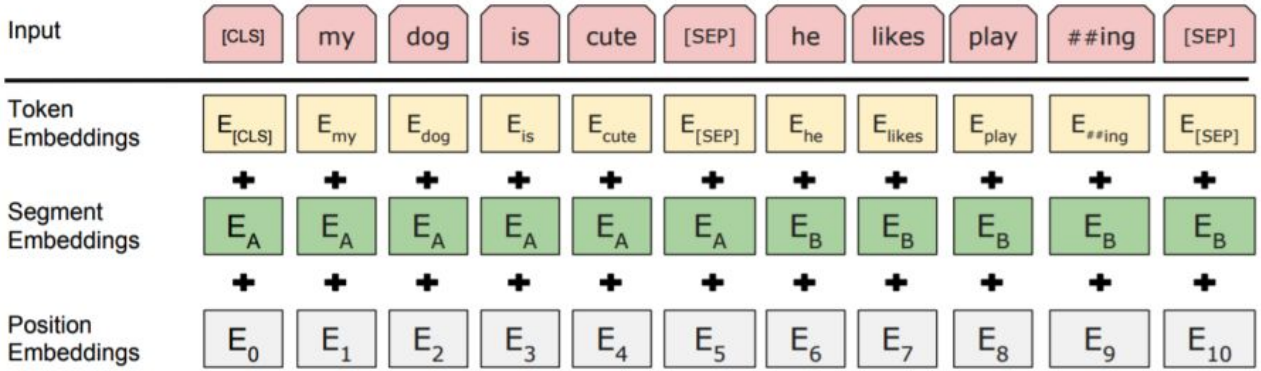
- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time add noise in model
  - Leave input word unchanged 10% of the time (but still predict it!) allow recovery of model on actual input
- Why? Doesn't let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



BERT [Devlin et al., 2018]

Source: stanford 224n

- 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.

BERT [Devlin et al., 2018]

Source: stanford 224n

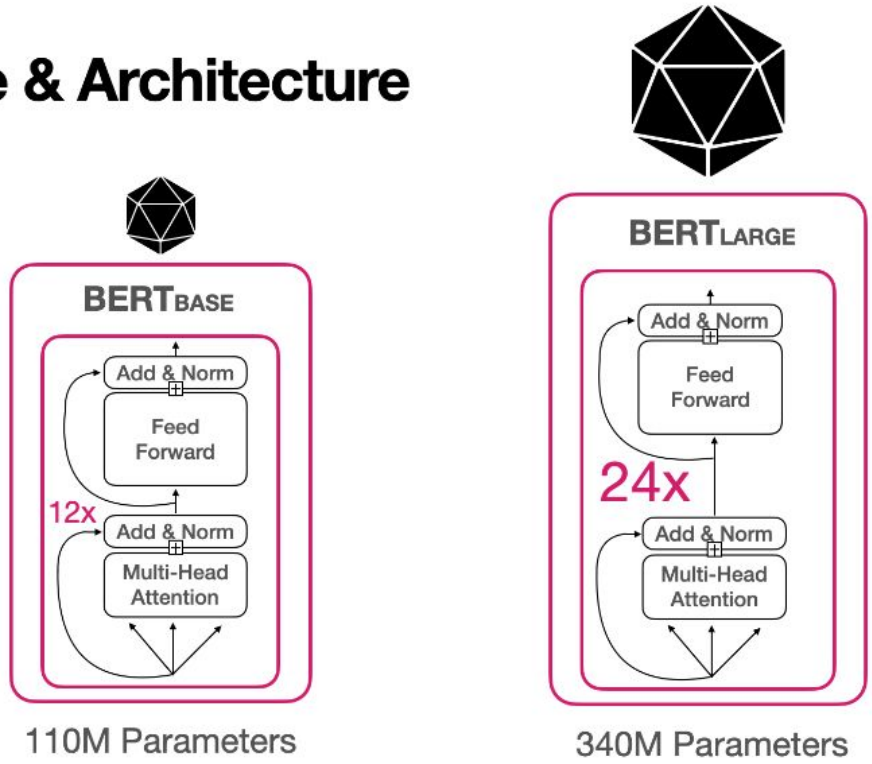
Details about BERT

- 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 [Devlin et al., 2018]

Source: stanford 224n

BERT Size & Architecture



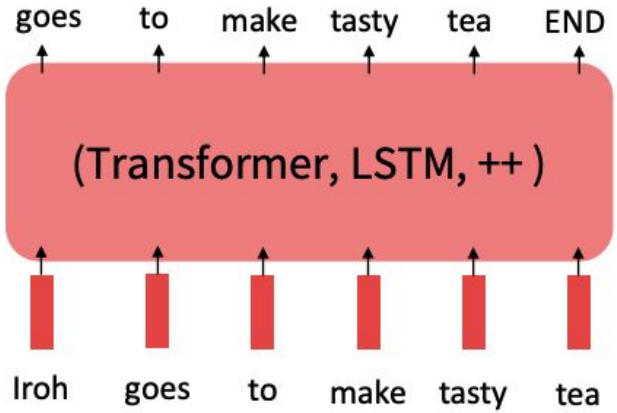
The Pretraining/Finetuning Paradigm

Source: stanford 224n

Pretraining can improve NLP applications by serving as parameter initialization.

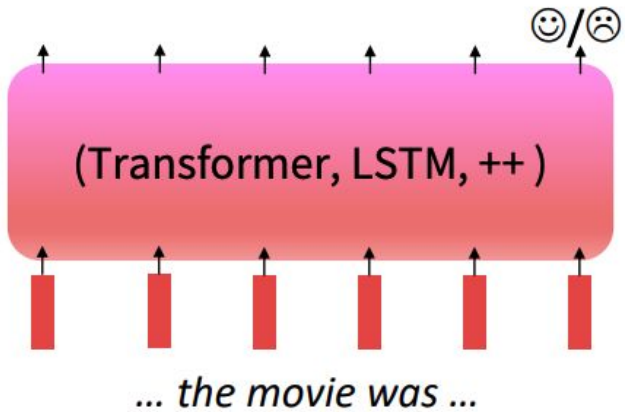
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



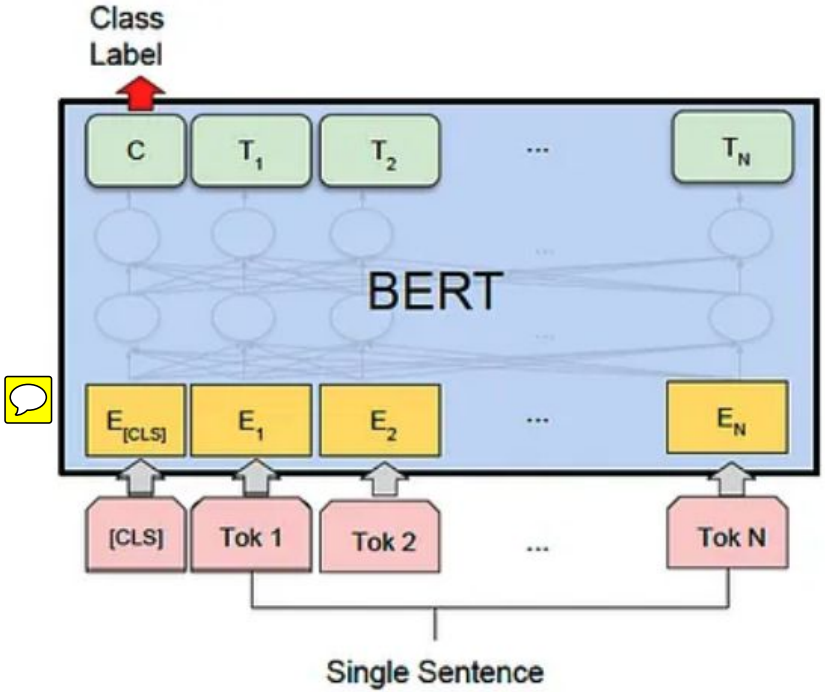
Step 2: Finetune (on your task)

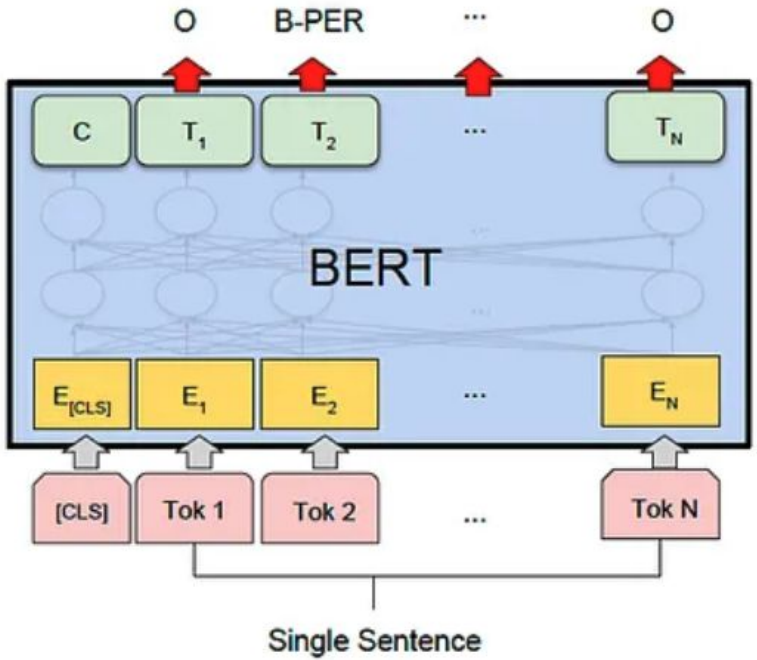
Not many labels; adapt to the task!



The Pretraining-Finetuning Paradigm

same as what we learn





The Pretraining-Finetuning Paradigm

Source: stanford 224n

- 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)

finetune loss learning is small, as the model \hat \theta is already optimal

- 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!

Parameter-Efficient Finetuning (Optional)

Finetuning every parameter in a pretrained model works well, but is memory-intensive.

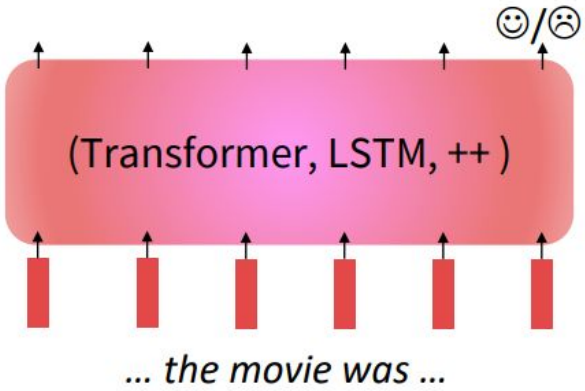
But **lightweight** finetuning methods adapt pretrained models in a constrained way.

Leads to **less overfitting** and/or **more efficient finetuning and inference.**

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

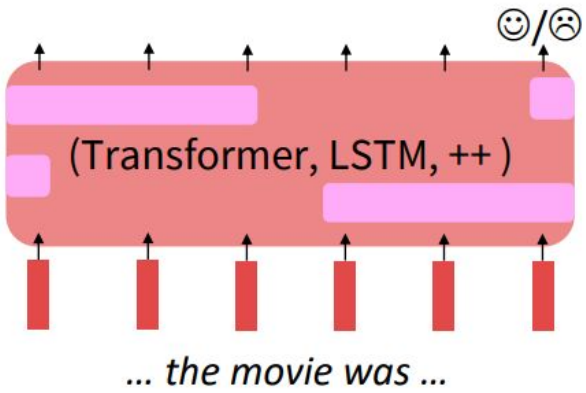
Full Finetuning

Adapt all parameters



Lightweight Finetuning

Train a few existing or new parameters



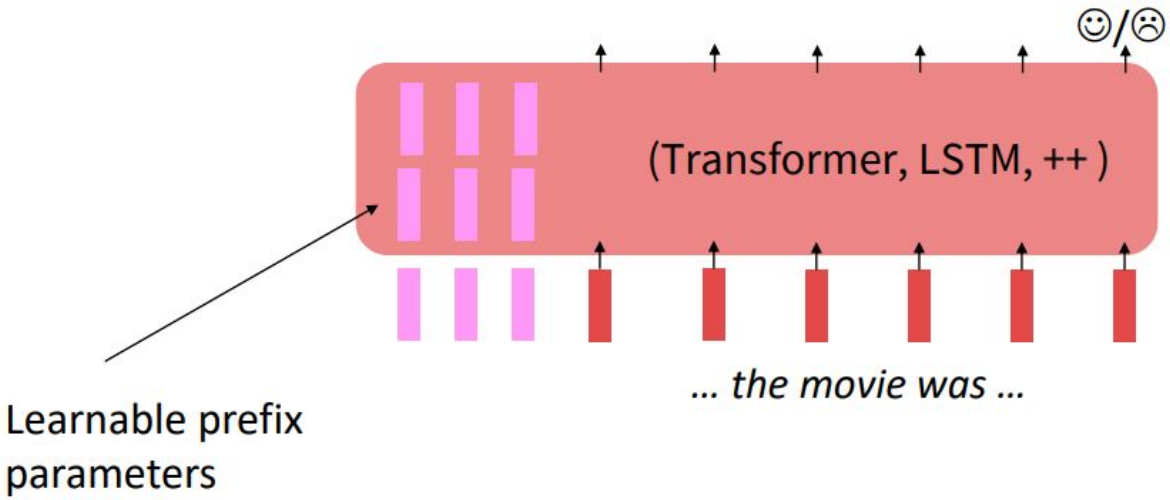
Prefix Tuning, Prompt Tuning (Optional)

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.

[Li and Liang et al., 2021]; [Lester et al., 2021]



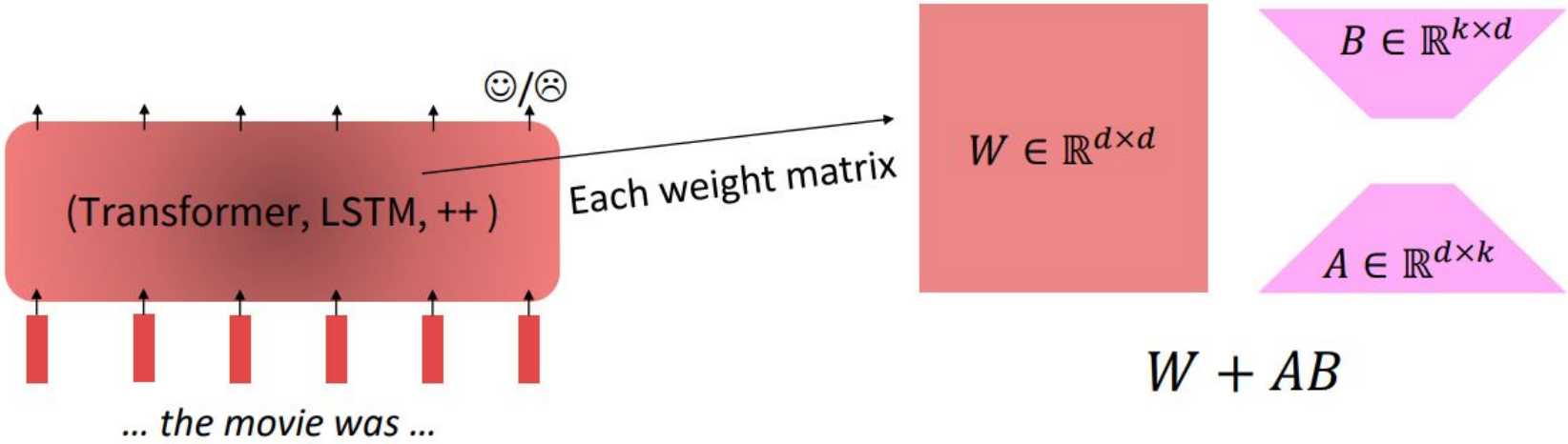
Low-Rank Adaptation (Optional)

Source: stanford 224n

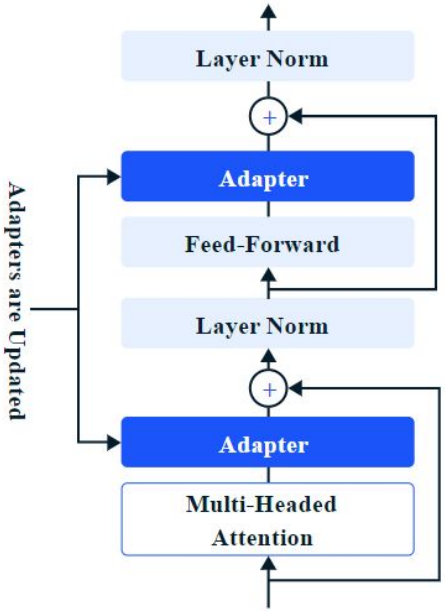
Low-Rank Adaptation Learns a low-rank "diff" between the pretrained and finetuned weight matrices.

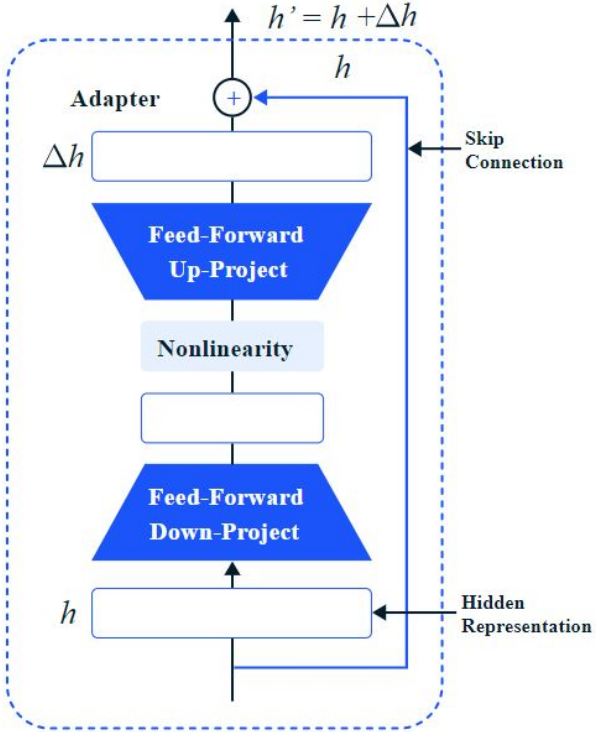
Easier to learn than prefix-tuning.

[Hu et al., 2021]



Adapter (Optional)





[Houlsby et al., 2019]

https://www.leewayhertz.com/parameter-efficient-fine-tuning/

BERT [Devlin et al., 2018]

Source: stanford 224n

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

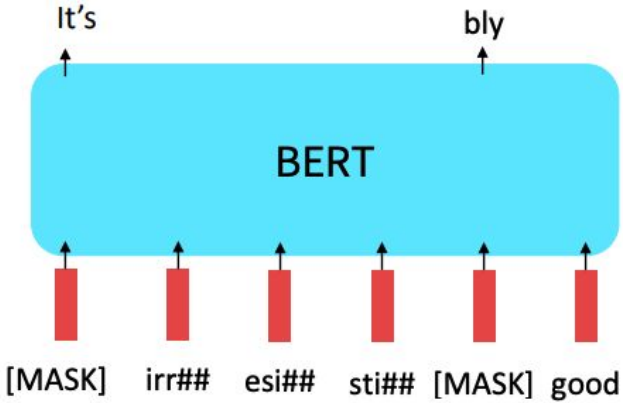
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	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**

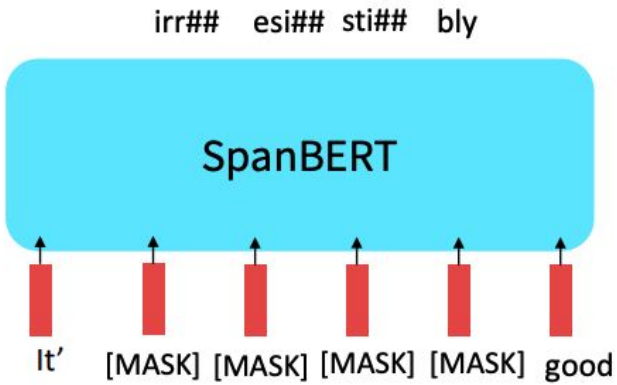
BERT Variants [Liu et al., 2019; Joshi et al., 2020]

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





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

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

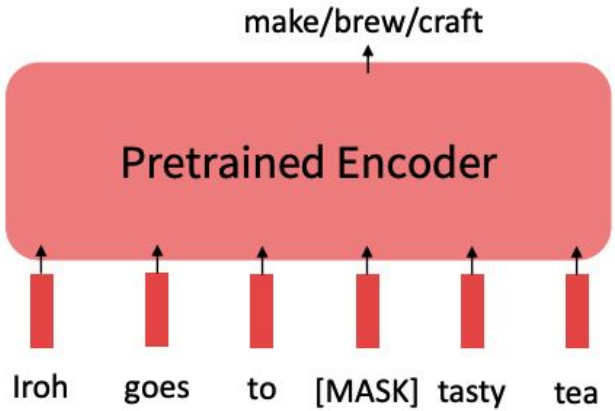
Limitation of Pretrained Encoders

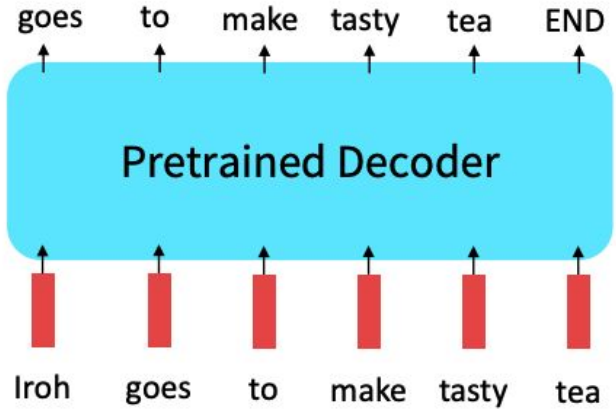
Source: stanford 224n

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.

BERT is bi-directional, so it cannot generate language

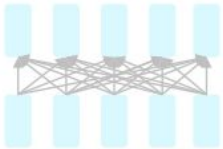




Decoders

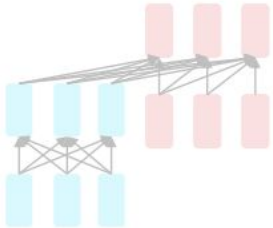
Source: stanford 224n

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



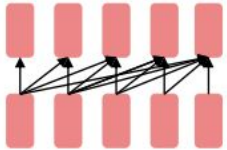
Encoders

- 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?

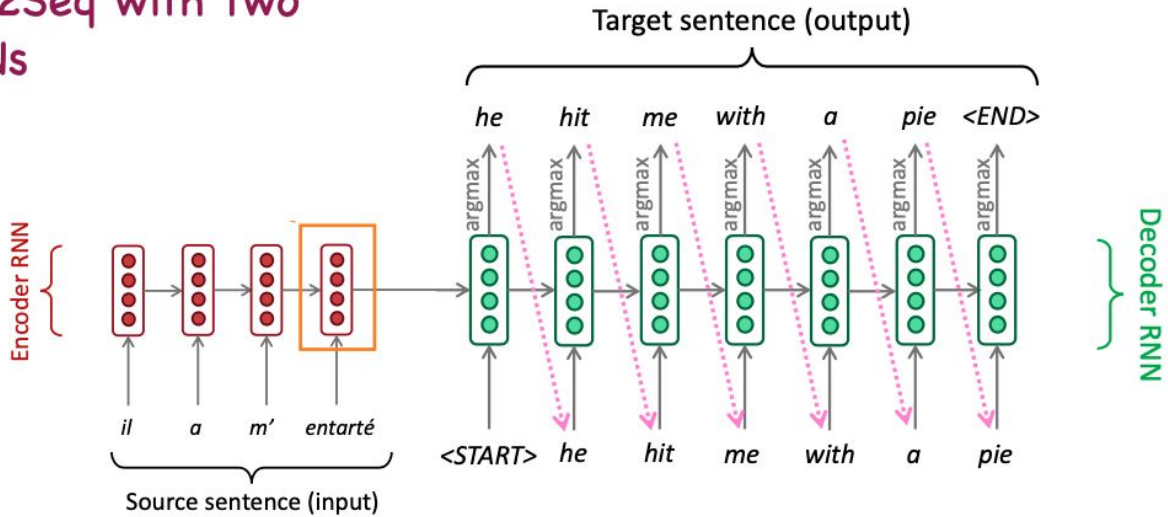


Decoders

- 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.

Seq2Seq Modeling with RNNs (Revisit)

- Seq2Seq with Two RNNs

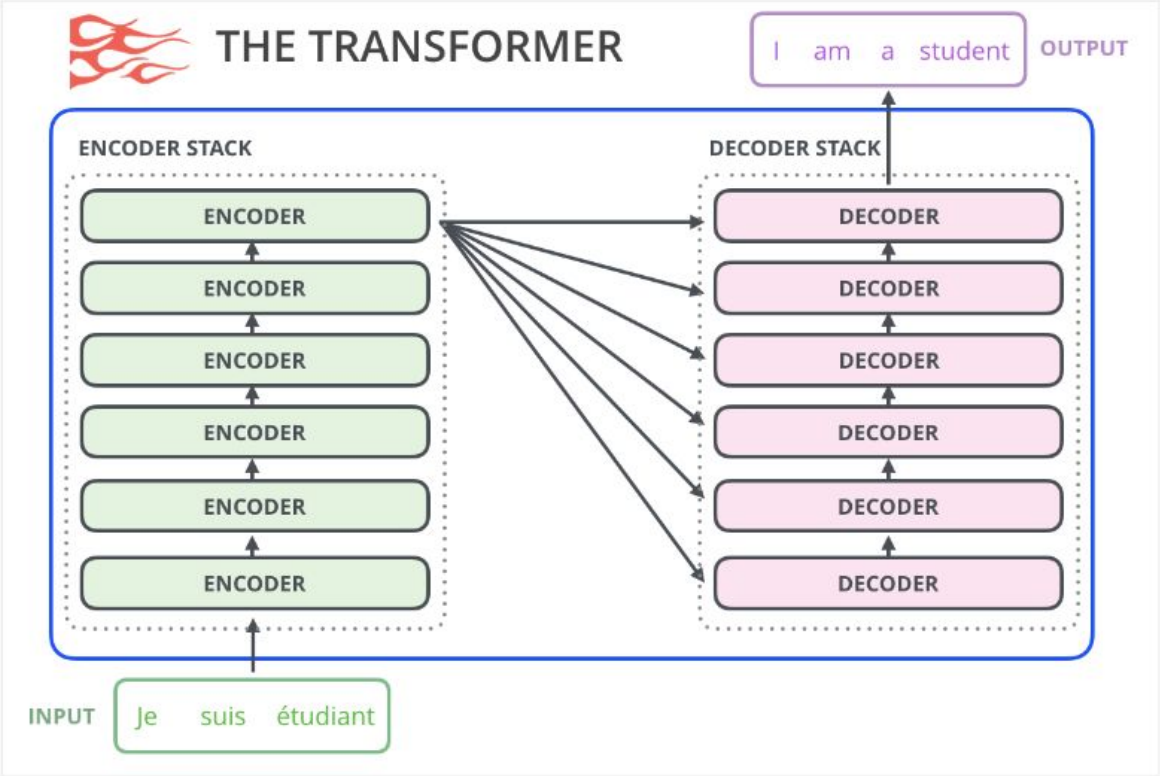


Encoder produces an encoding of the source sentence.

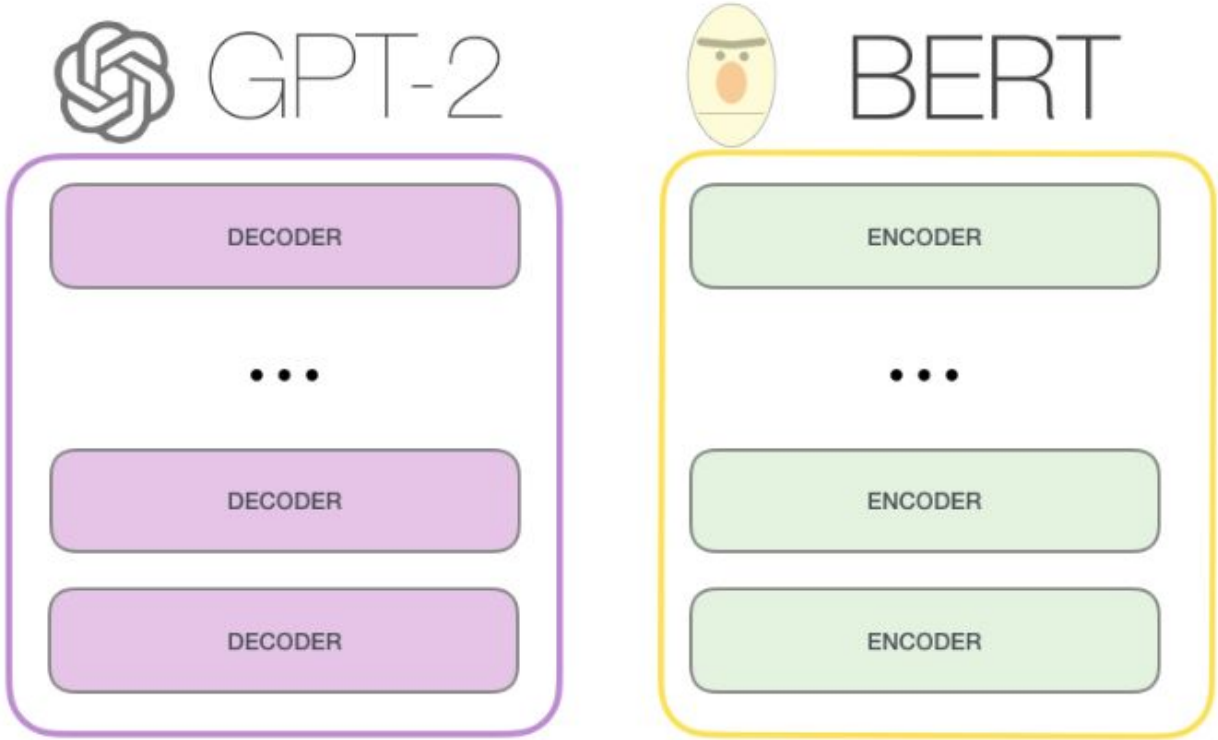
Provides Initial hidden state for Decoder

Decoder RNN is a **Conditional Language Model** that generates target sentence, conditioned on encoding.

Encoders vs Decoders

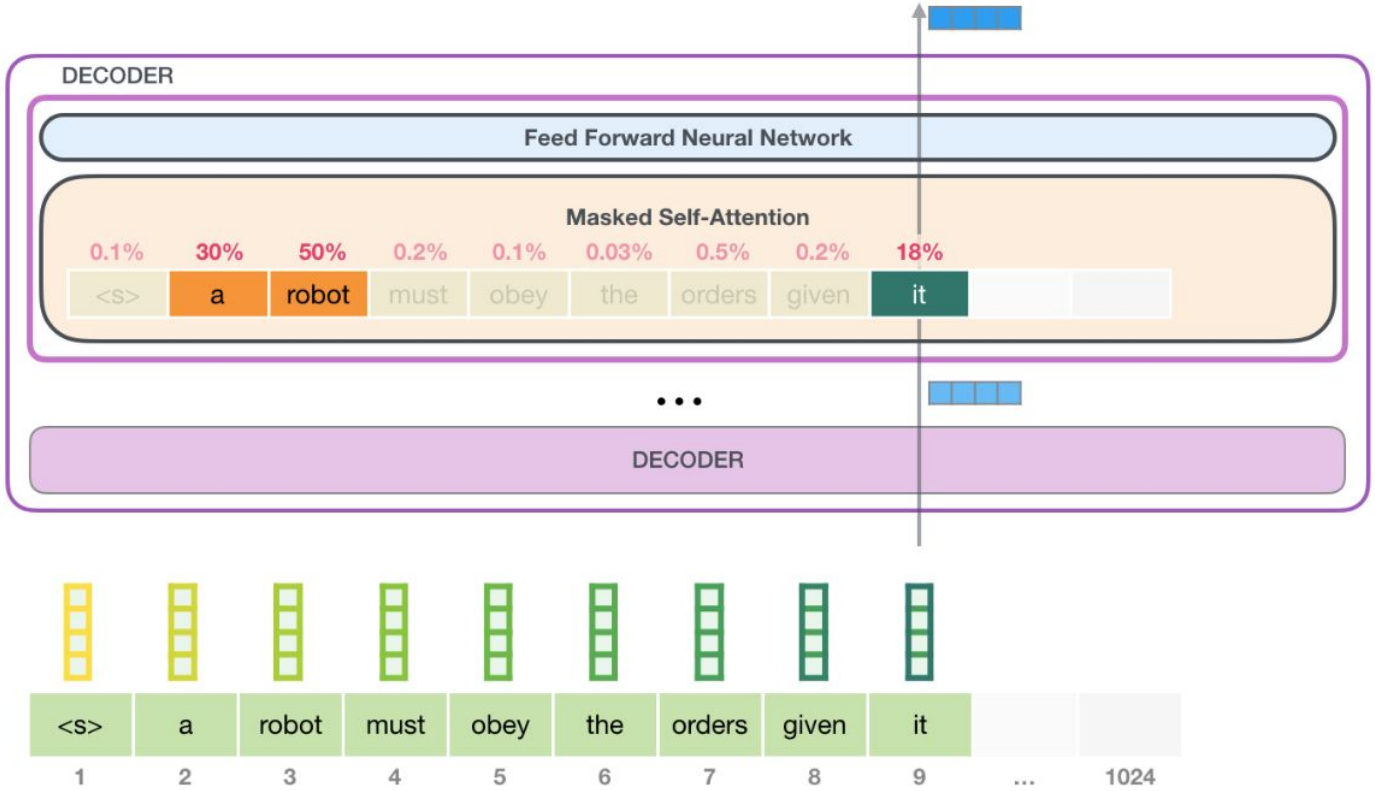


Encoders vs Decoders



Decoders

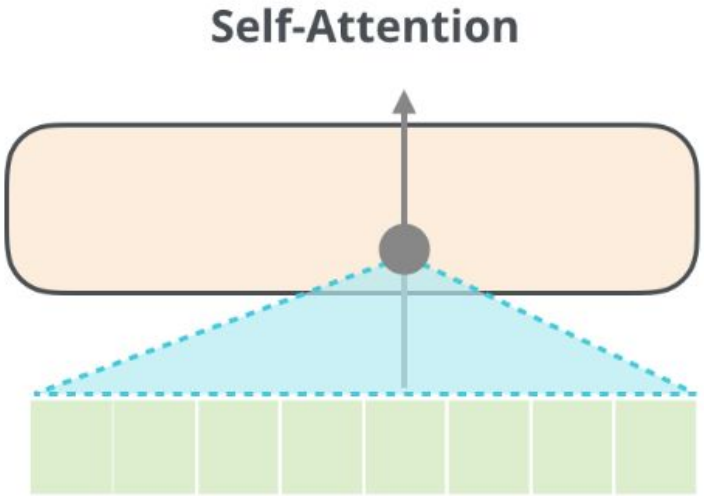
https://jalammar.github.io/illustrated-gpt2/



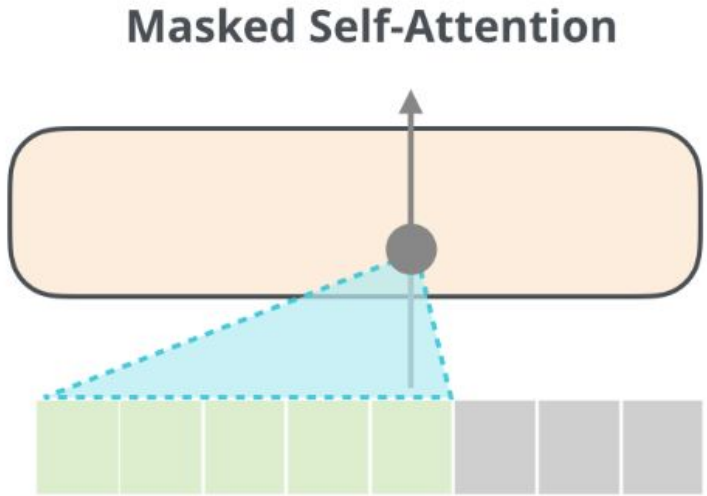
Decoders

https://jalammar.github.io/illustrated-gpt2/

this is the typical way self attention work in parallel



so now we mask the future, to train the model to work on past



Decoders

Source: stanford 224n

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

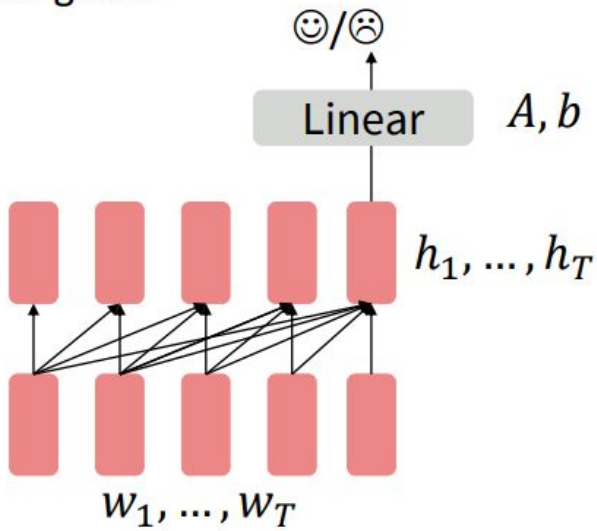
We can finetune them by training a classifier on the last word's hidden state.

with masked self attention

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

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.]

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
- 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"

GPT2 [Radford et al., 2018]

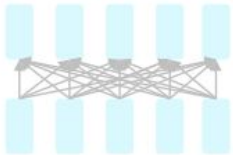
GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

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.

Encoder-Decoders

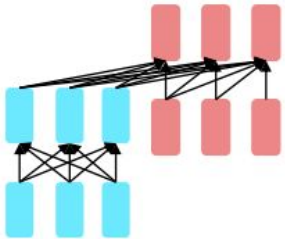
Source: stanford 224n

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



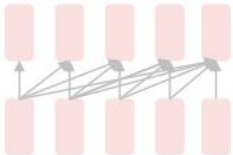
Encoders

- 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?

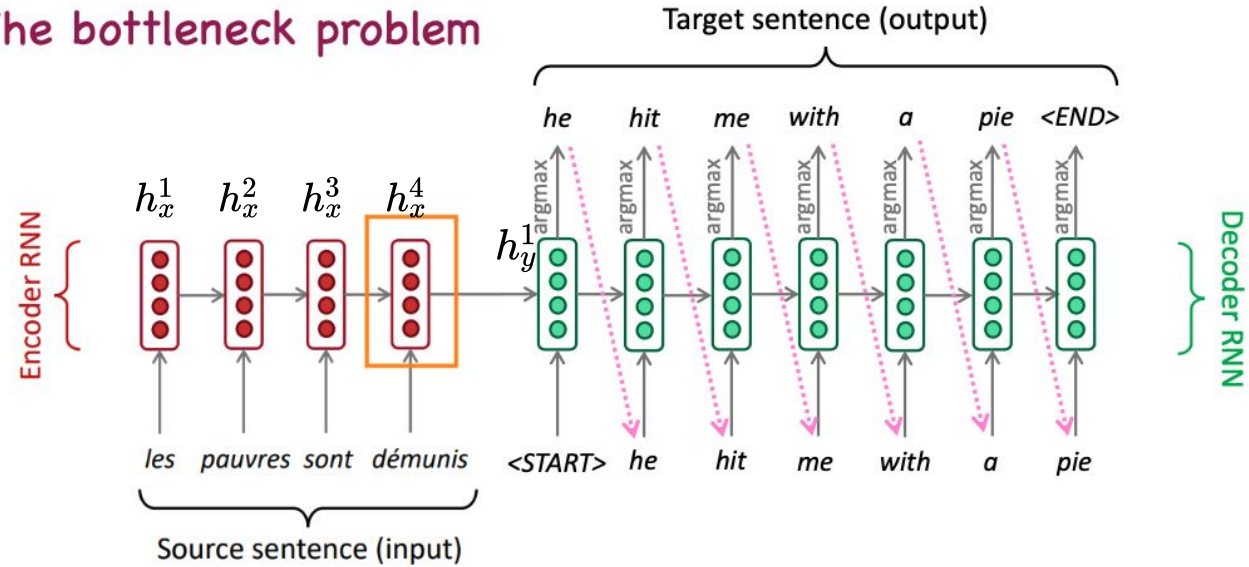


Decoders

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

Seq2Seq Modeling with RNNs (Revisit)

- The bottleneck problem



Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

$$h_y^1 = f(W_{ye} \cdot e_{<START>} + W_{yh} \cdot h_x^4 + b_y)$$

$$h_x^1 = f(W_{xe} \cdot e_{les} + W_{xh} \cdot h_x^0 + b_x)$$

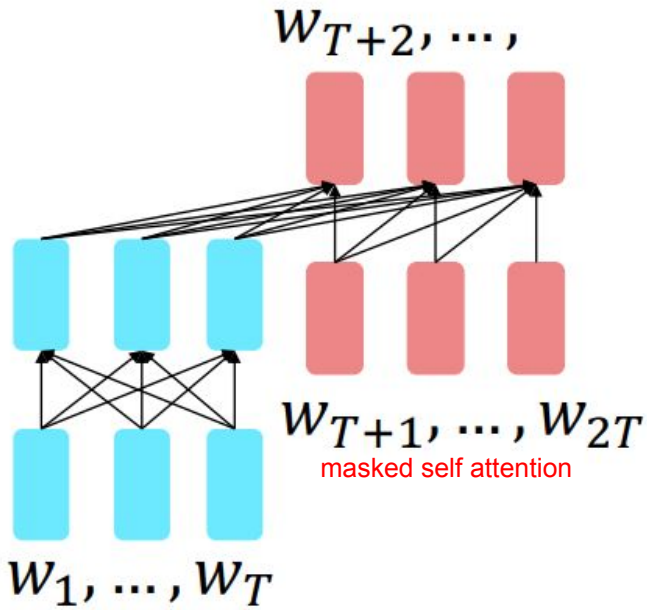
Encoder-Decoders [Raffel et al., 2018]

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.

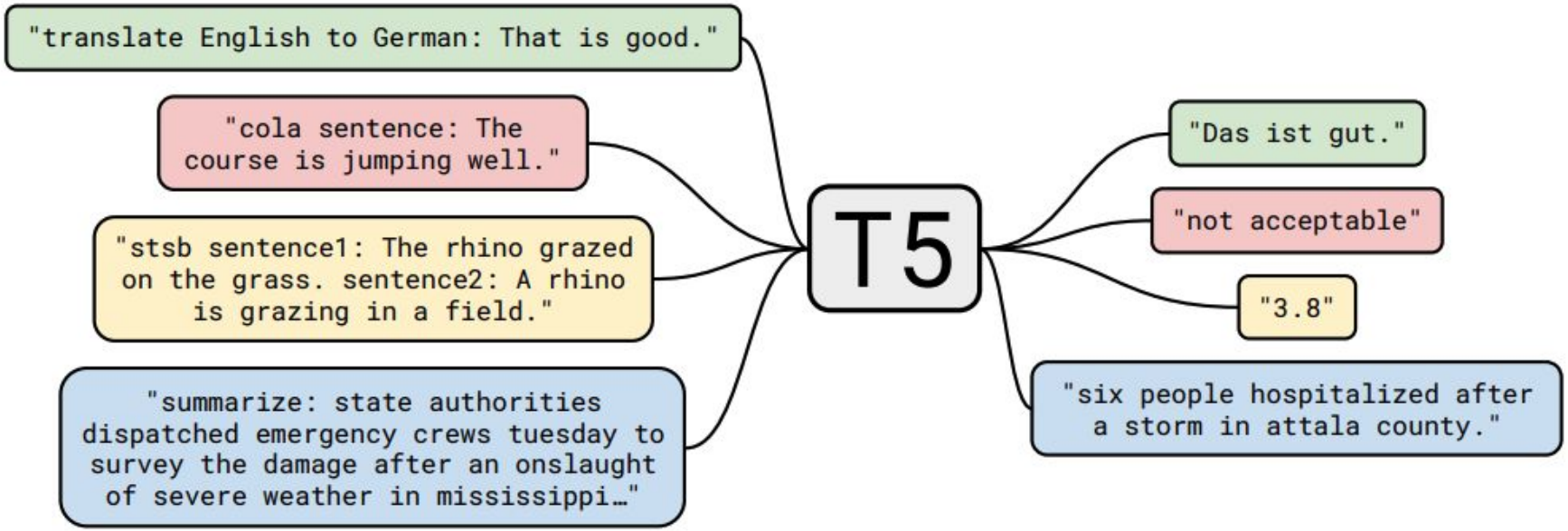
$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$h_{T+1}, \ldots, h_2 = Decoder(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$

Decoder applies all connection of

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



Encoder-Decoders [Raffel et al., 2018]



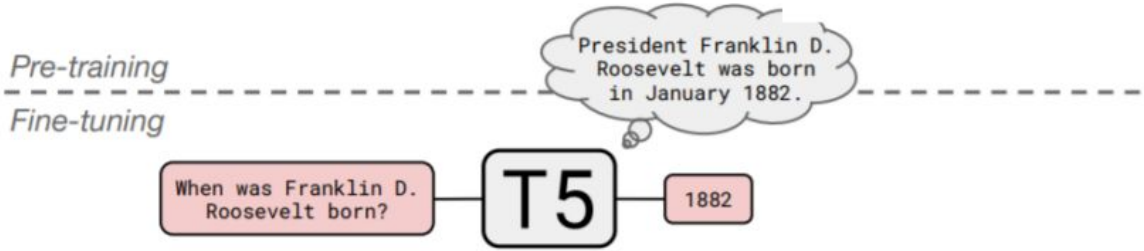
Encoder-Decoders [Raffel et al., 2018]

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

Encoder-Decoders [Raffel et al., 2018]

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

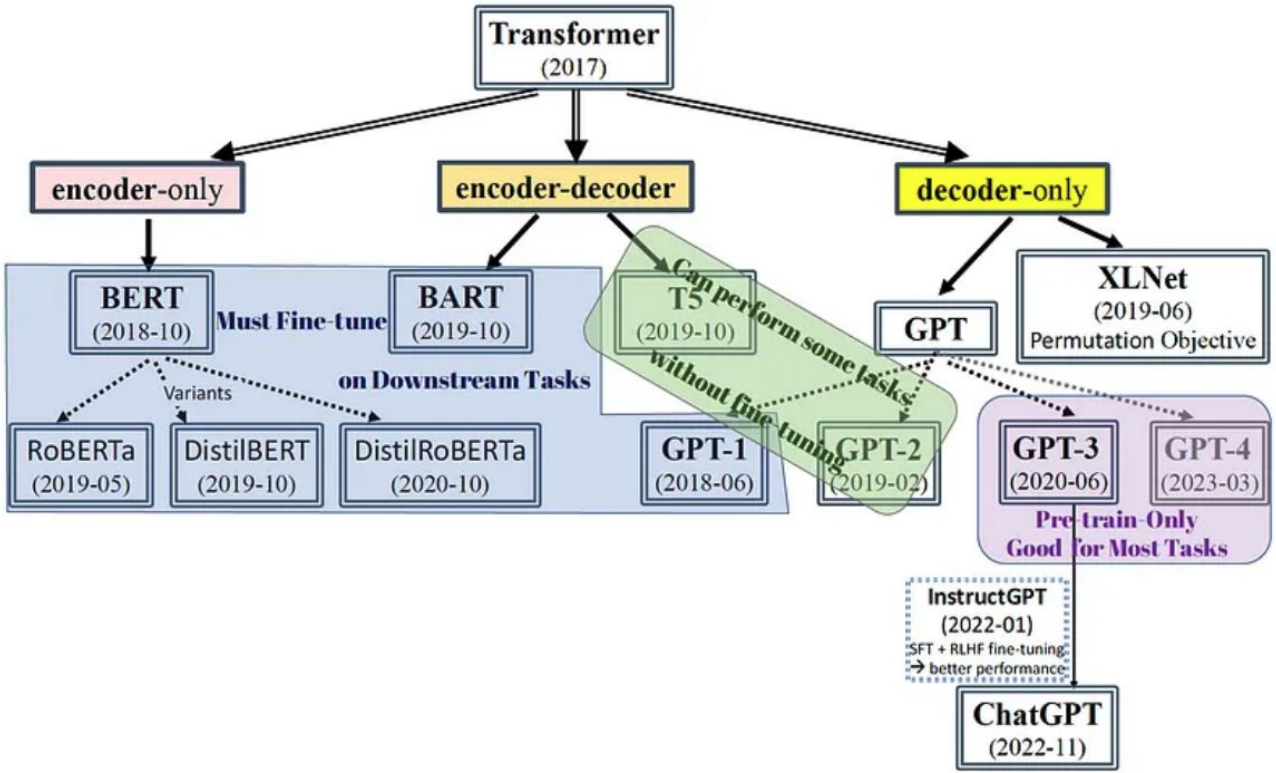
All "open-domain"

	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**	

What Does the Pretrained Model Learn

- *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 Fibonnaci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.

The Transformer Family



Outline for today

01 Subword Modeling

02 Knowledge Transfer

03 Pretraining vs. Finetuning

04 Natural Language Generation

Seq2Seq (Revisit)

- Not just MT!
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse tree)
  - Code generation (natural language → Python code)
  - Segmentation/tagging (input text → output tag sequence)

They all belong to the task of Natural Language Generation

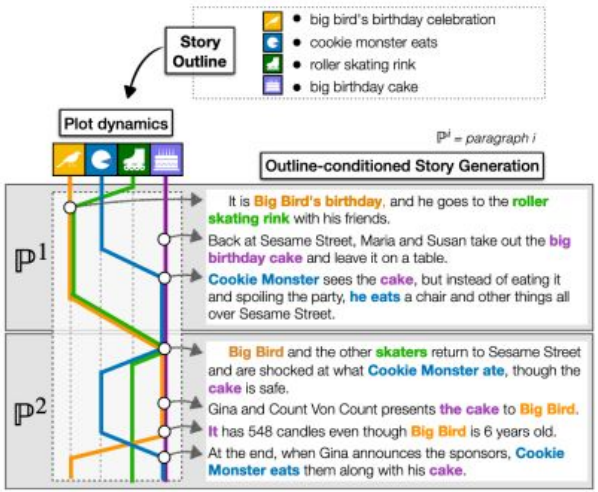
What is Natural Language Generation

- Natural Language Generation focuses on systems that produce fluent, coherent and useful language output for human consumption
- Natural language generation is one side of natural language processing.
- NLP = Natural Language Understanding (NLU) + Natural Language Generation (NLG)

More Interesting Examples

Source: stanford 224n

Creative stories



(Rashkin et al.., EMNLP 2020)

Data-to-text

Table Title: Robert Craig (American football)
Section Title: National Football League statistics
Table Description: None

			RUSHING					RECEIVING				
YEAR	TEAM	ATT	YDS	AVG	LNG	TD	NO.	YDS	AVG	LNG	TD
1983	SF	176	725	4.1	71	8	48	427	8.9	23	4
1984	SF	155	649	4.2	28	4	71	675	9.5	64	5
1985	SF	214	1050	4.9	62	9	92	1016	11	73	6
1986	SF	204	830	4.1	25	7	81	624	7.7	48	0
1987	SF	215	815	3.8	25	3	66	492	7.5	35	1
1988	SF	310	1502	4.8	46	9	76	534	7.0	22	1
1989	SF	271	1054	3.9	27	6	49	473	9.7	44	1
1990	SF	141	439	3.1	26	1	25	201	8.0	31	0
1991	RAI	162	590	3.6	15	1	17	136	8.0	20	0
1992	MIN	105	416	4.0	21	4	22	164	7.5	22	0
1993	MIN	38	119	3.1	11	1	19	169	8.9	31	1
Totals	-	1991	8189	4.1	71	56	566	4911	8.7	73	17

Craig finished his eleven NFL seasons with 8,189 rushing yards and 566 receptions for 4,911 receiving yards.

(Parikh et al.., EMNLP 2020)

Visual description

Two children are sitting at a table in a restaurant. The children are one little girl and one little boy. The little girl is eating a pink frosted donut with white icing lines on top of it. The girl has blonde hair and is wearing a green jacket with a black long sleeve shirt underneath. The little boy is wearing a black zip up jacket and is holding his finger to his lip but is not eating. A metal napkin dispenser is in between them at the table. The wall next to them is white brick. Two adults are on the other side of the short white brick wall. The room has white circular lights on the ceiling and a large window in the front of the restaurant. It is daylight outside.

(Krause et al. CVPR 2017)

Categorization of NLG Tasks

Source: stanford 224n

Spectrum of open-endedness for Generation Tasks

Source Sentence: 当局已经宣布今天是节假日。

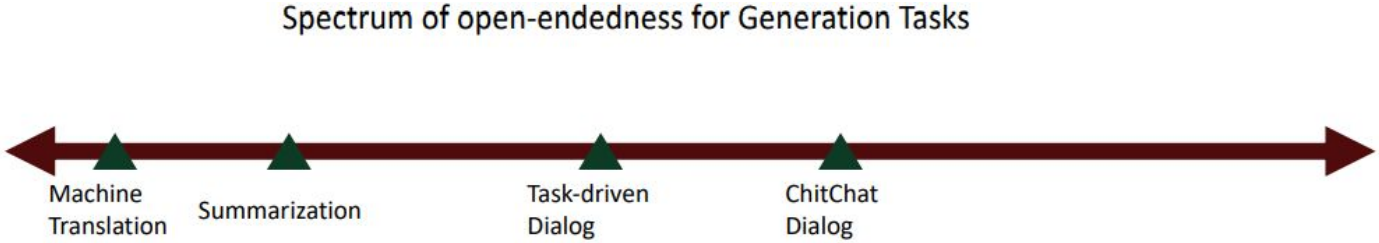
Reference Translation:

1. Authorities have announced a national holiday today.
2. Authorities have announced that today is a national holiday.
3. Today is a national holiday, announced by the authorities.

The output space is not very diverse.

Categorization of NLG Tasks

Source: stanford 224n



Input: Hey, how are you?

Outputs:

1. Good! You?
2. I just heard an exciting news, do you want to hear it?
3. Thx for asking! Barely surviving my hws.

The output space is getting more diverse…

Categorization of NLG Tasks

Source: stanford 224n

Spectrum of open-endedness for Generation Tasks

Input: Write a story about three little pigs?
Outputs:
… (so many options) …

The output space is extremely diverse…

Categorization of NLG Tasks

Source: stanford 224n

Spectrum of open-endedness for Generation Tasks

Input: Write a story about three little pigs?
Outputs:
… (so many options) …

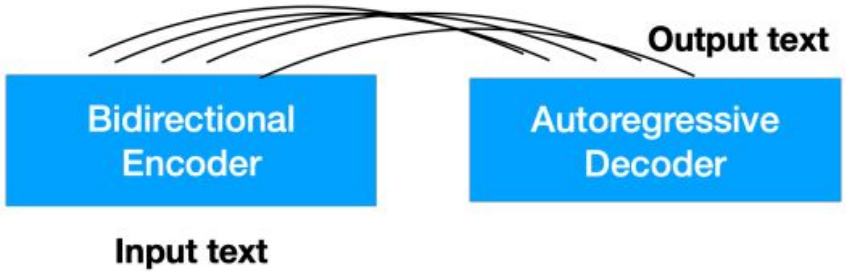
long story short, NLG is very diverse and its hard to

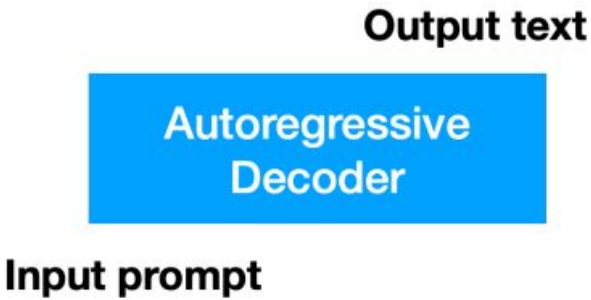
Less Open-Ended

More Open-Ended

NLG Model (Revisit)

- Encoder-Decoder model encodes the input via an encoder, and generates (decodes) tokens via a decoder.
- Decoder model directly uses the decoder for autoregressive generation (decoding).





Decoding Algorithm – Greedy Decoding

- Core idea: Always selects the next token with highest probability

$$\hat{y}_t = \underset{w \in V}{\operatorname{argmax}} P(y_t = w | y_{<t})$$

- Problem: Greedy decoding cannot correct previous mistakes

  - Input: il a m'entarté (he hit me with a pie)
  - → he _____
  - → he hit _____
  - → he hit a _____ no going back now!

Decoding Algorithm – Beam Search

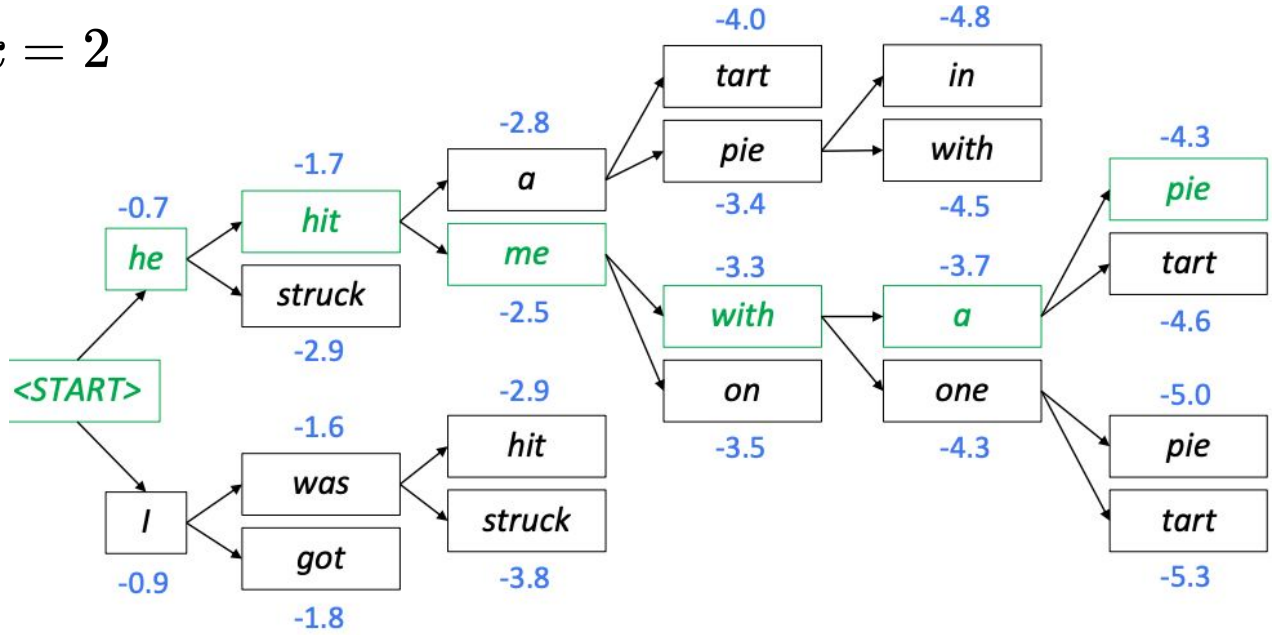
- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses). k (beam size) usually ranges from 5 to 10.

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Beam search is not guaranteed to find optimal solution, but is much more efficient than exhaustive search.

Decoding Algorithm – Beam Search

$k = 2$



Backtrack to obtain the full hypothesis

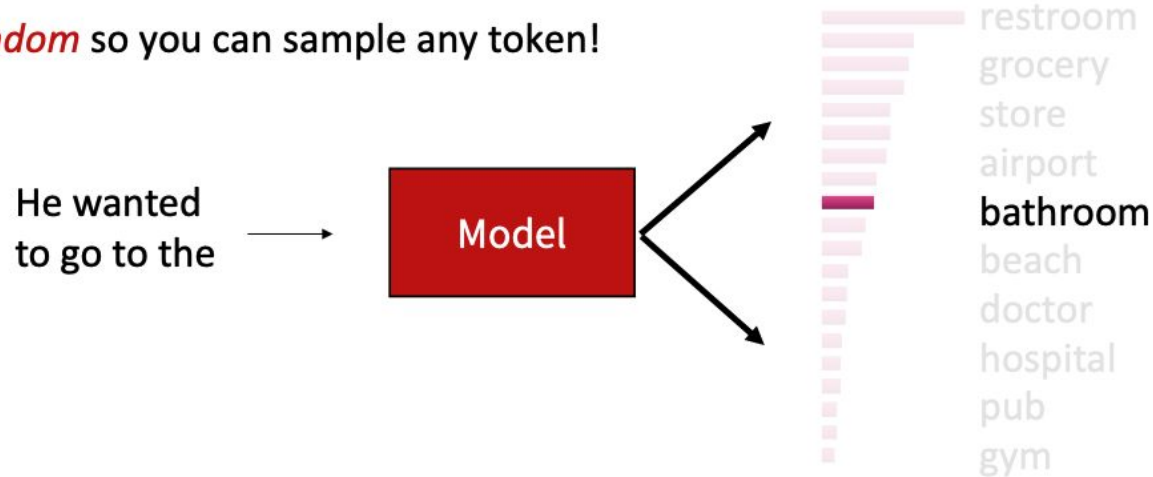
Decoding with Sampling

Source: stanford 224n

- Sample a token from the distribution of tokens

$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

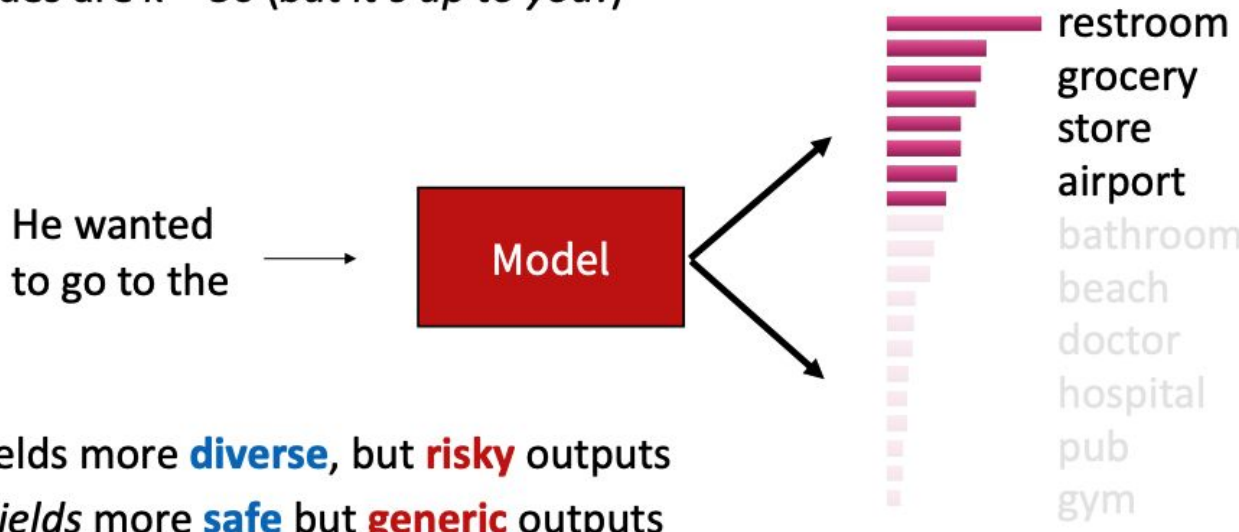
- It's *random* so you can sample any token!



Decoding with Top-k Sampling

Source: stanford 224n

- Solution: Top-*k* sampling
  - Only sample from the top *k* tokens in the probability distribution
  - Common values are *k* = 50 (*but it's up to you!*)



- Increase *k* yields more **diverse**, but **risky** outputs
- Decrease *k yields* more **safe** but **generic** outputs

Decoding with Temperature

- Recall: On timestep t, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t:

$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - **More** diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - **Less** diverse output (probability is concentrated on top words)

Temperature is a hyperparameter for decoding:
It can be tuned for both beam search and sampling.

Decoding Takeaways

Source: stanford 224n

- Decoding is still a challenging problem in NLG – there's a lot more work to be done!
- Different decoding algorithms can allow us to inject biases that encourage different properties of coherent natural language generation
- Some of the most impactful advances in NLG of the last few years have come from simple but effective modifications to decoding algorithms

Evaluations for NLG

Source: stanford 224n

Content Overlap Metrics

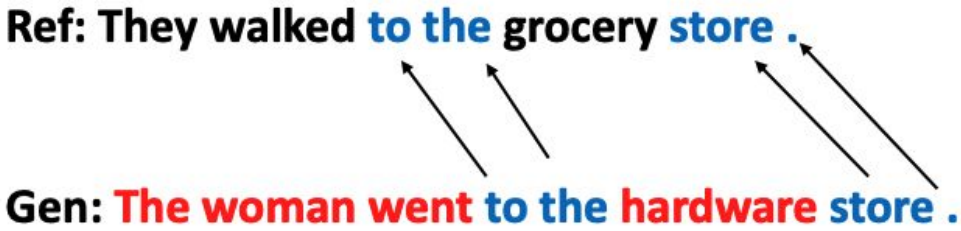
BLEU score

Model-based Metrics

Similarity metrics

Human Evaluations

Content Overlap Metrics (Optional)



- Compute a score that indicates the lexical similarity between *generated* and *gold-standard (human-written) text*
- Fast and efficient and widely used
- *N*-gram overlap metrics (e.g., **BLEU**, ROUGE, METEOR, CIDEr, etc.)

Source: stanford 224n

N-Gram Overlap Metrics (Optional)

Source: stanford 224n

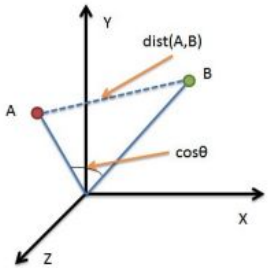
Word overlap–based metrics (BLEU, ROUGE, METEOR, CIDEr, etc.)

- They're not ideal for machine translation

- They get progressively much worse for tasks that are more open-ended than machine translation
  - **Worse** for summarization, as longer output texts are harder to measure
  - **Much worse** for dialogue, which is more open-ended that summarization
  - **Much, much worse** story generation, which is also open-ended, but whose sequence length can make it seem you're getting decent scores!

Model-based Metrics (Optional)

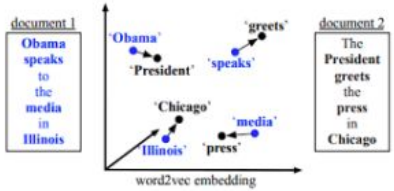
Source: stanford 224n



Vector Similarity

Embedding based similarity for semantic distance between text.

- **Embedding Average** (Liu et al., 2016)
- **Vector Extrema** (Liu et al., 2016)
- **MEANT** (Lo, 2017)
- **YISI** (Lo, 2019)

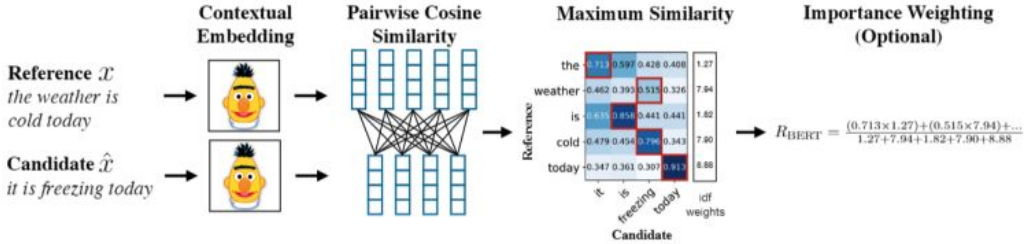


Word Mover's Distance

Measures the distance between two sequences (e.g., sentences, paragraphs, etc.), using word embedding similarity matching.
(Kusner et.al., 2015; Zhao et al., 2019)

BERTSCORE

Uses pre-trained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity.
(Zhang et.al. 2020)



Human Evaluations

Source: stanford 224n

- Automatic metrics fall short of matching human decisions
- Human evaluation is most important form of evaluation for text generation systems.
- Gold standard in developing new automatic metrics
  - New automated metrics must correlate well with human evaluations!

Human Evaluations

Source: stanford 224n

- Ask humans to evaluate the quality of generated text
- Overall or along some specific dimension:
  - fluency
  - coherence / consistency
  - factuality and correctness
  - commonsense
  - style / formality
  - grammaticality

Evaluation Takeaways

Source: stanford 224n

- *Content overlap metrics* provide a good starting point for evaluating the quality of generated text, but they're not good enough on their own.
- *Model-based metrics* can be more correlated with human judgment, but behavior is not interpretable
- *Human judgments* are critical
  - But humans are inconsistent!
- In many cases, the best judge of output quality is YOU!
  - **Look at your model generations. Don't just rely on numbers!**
  - **Publicly release large samples of the output of systems that you create!**