

# Natural Language Processing

AI51701/CSE71001

Lecture 15

11/9/2023

Instructor: Taehwan Kim

# Announcement

- ❑ Final project proposal grading was done
  - If you have any questions regarding grading, please let me know.

# Motivating model pretraining from word embeddings

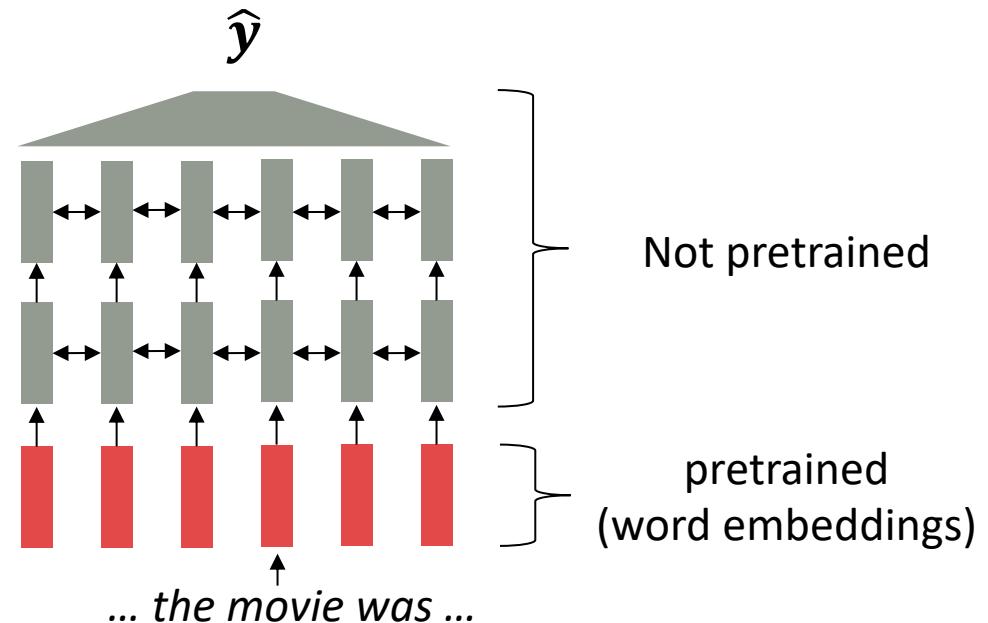
# Where we were: pretrained word embeddings

## ❑ Circa 2017:

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

## ❑ Some issues to think about:

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



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

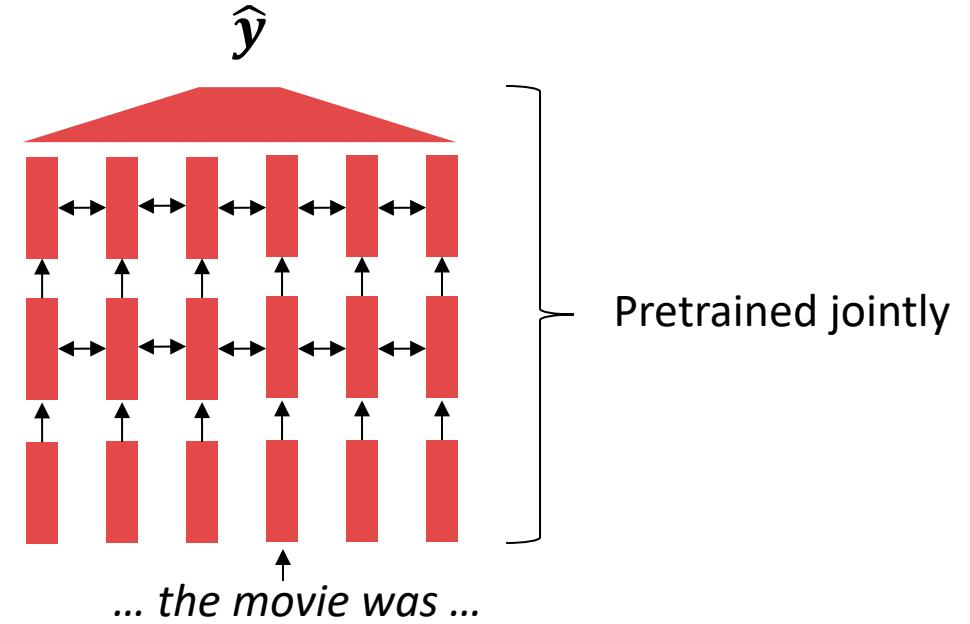
# Where we're going: pretraining whole models

## ❑ In modern NLP:

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

## ❑ This has been exceptionally effective at building strong:

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



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

## What can we learn from reconstructing the input?

- 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]
- I was thinking about the sequence that goes  
1, 1, 2, 3, 5, 8, 13, 21, \_\_\_\_\_ [some basic arithmetic]

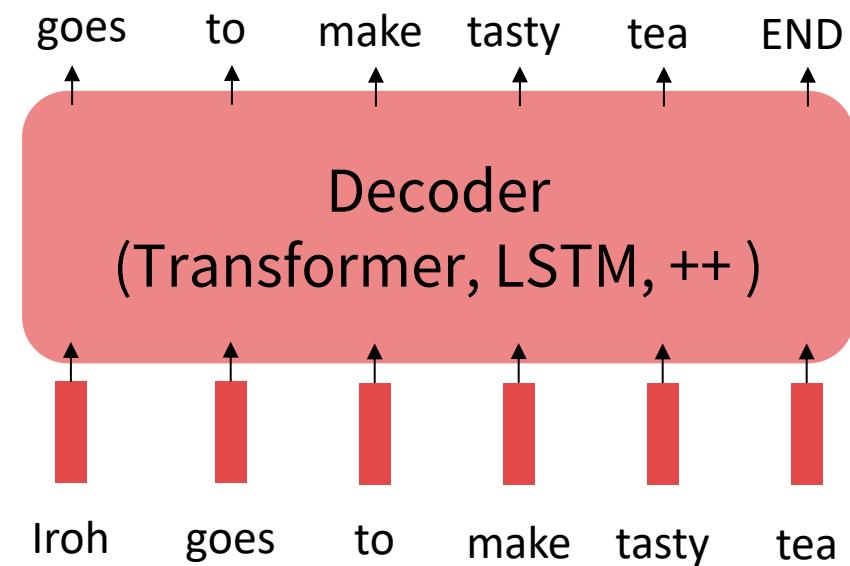
# Pretraining through language modeling [\(Dai and Le, 2015\)](#)

## ❑ 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.

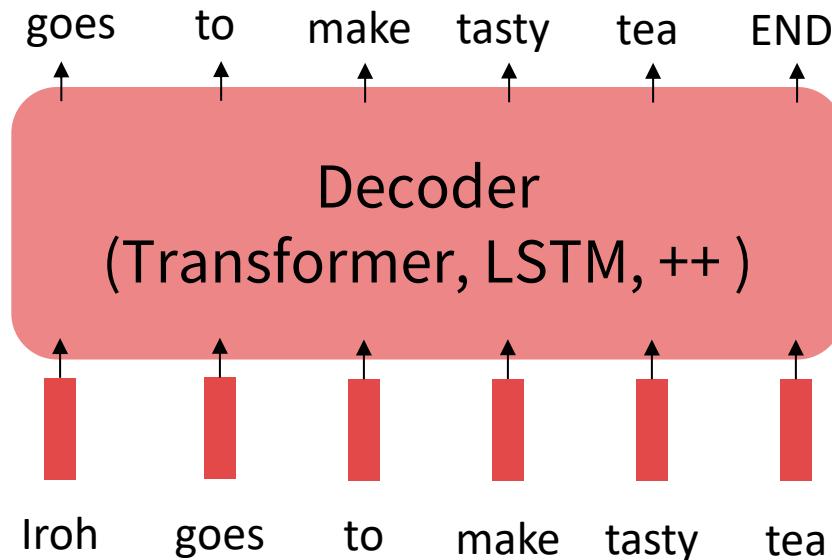


# The Pretraining / Finetuning Paradigm

- ❑ Pretraining can improve NLP applications by serving as parameter initialization.

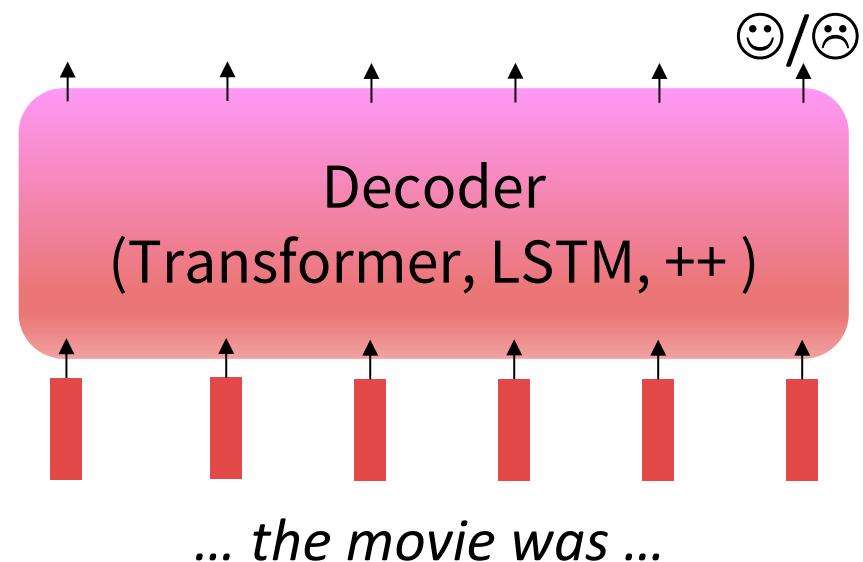
## Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



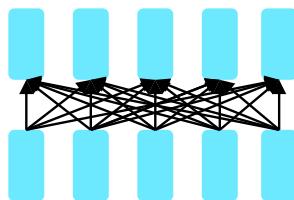
## Step 2: Finetune (on your task)

Not many labels; adapt to the task!



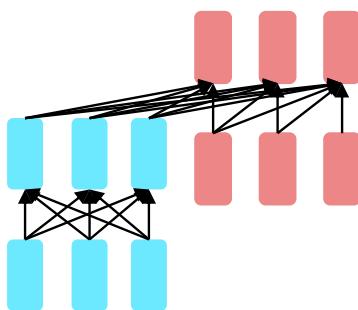
# Pretraining for three types of architectures

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



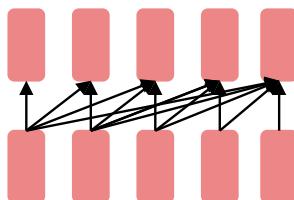
**Encoders**

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?



**Encoder-Decoders**

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

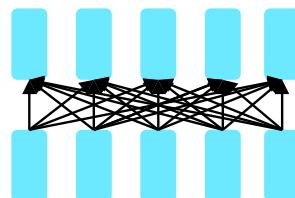


**Decoders**

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

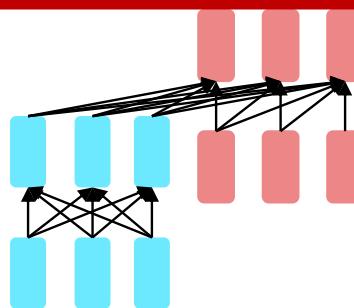
# Pretraining for three types of architectures

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



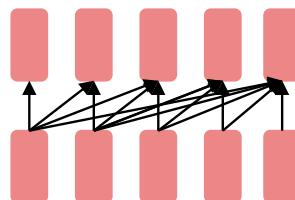
**Encoders**

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?



**Encoder-Decoders**

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

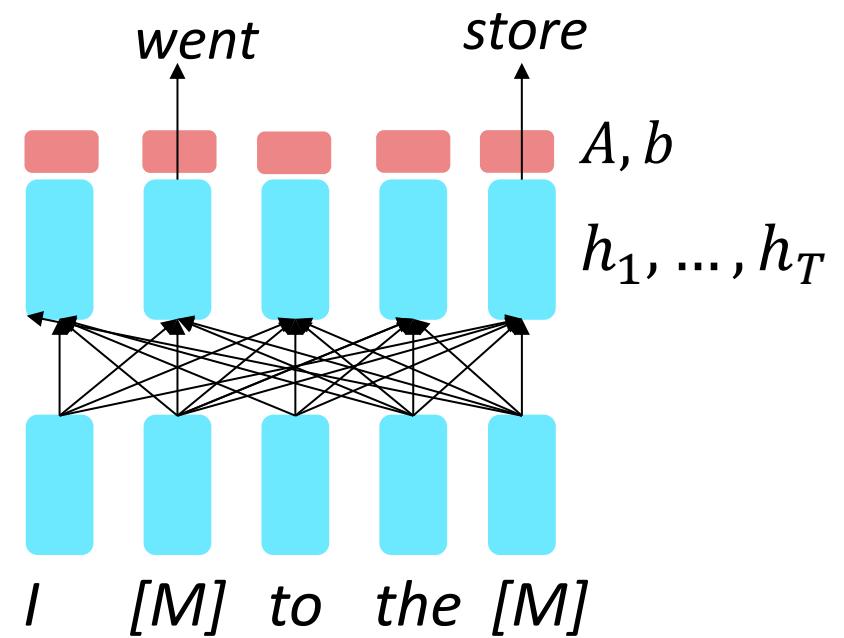


**Decoders**

- 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.
- ❑ 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**.

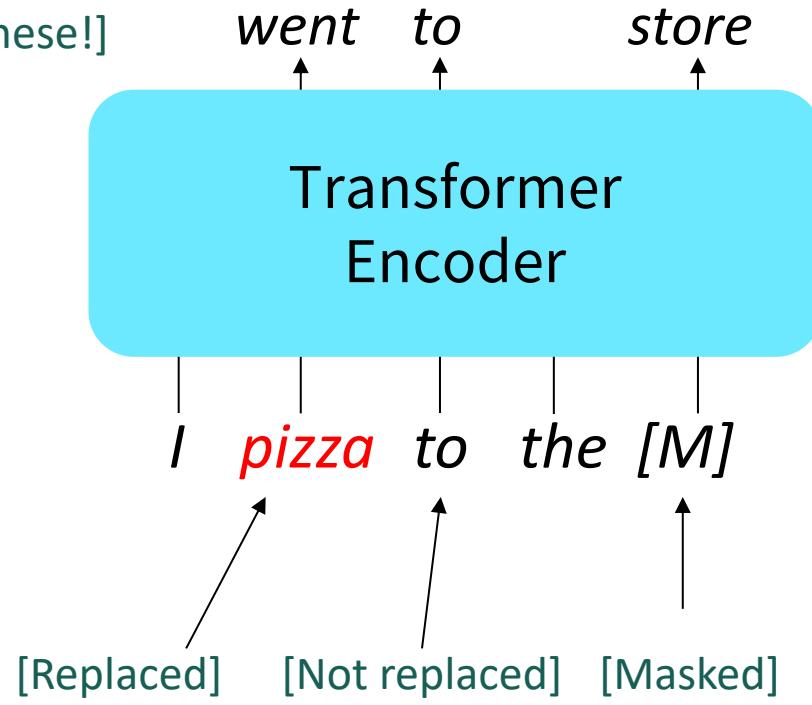


(Devlin et al., 2018)

# BERT: Bidirectional Encoder Representations from Transformers

- Devlin et al., 2018 proposed the “Masked LM” objective, and **released the weights of their pretrained Transformer (BERT)**.
- Some more details about Masked LM for BERT:
  - Predict a random 15% of (sub)word tokens.
    - Replace input word with [MASK] 80% of the time
    - Replace input word with a random token 10% of the time
    - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn’t let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)

[Predict these!]



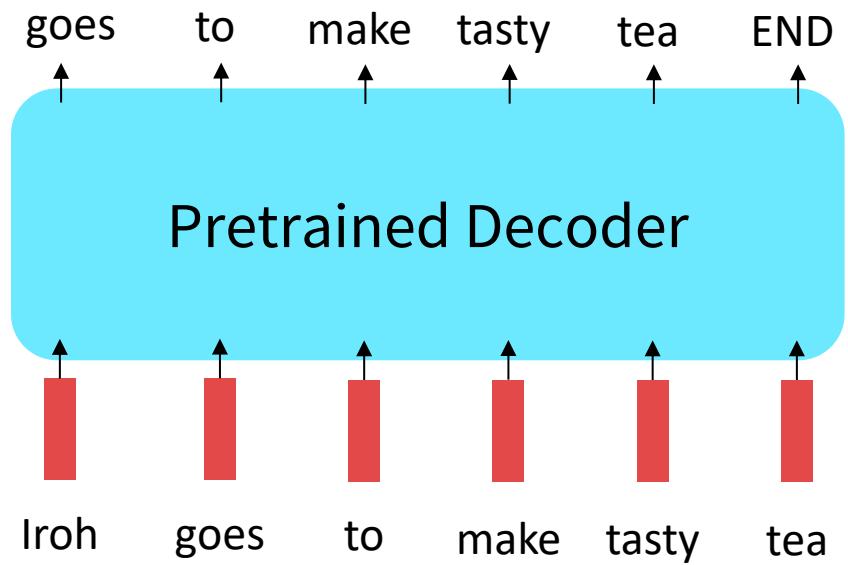
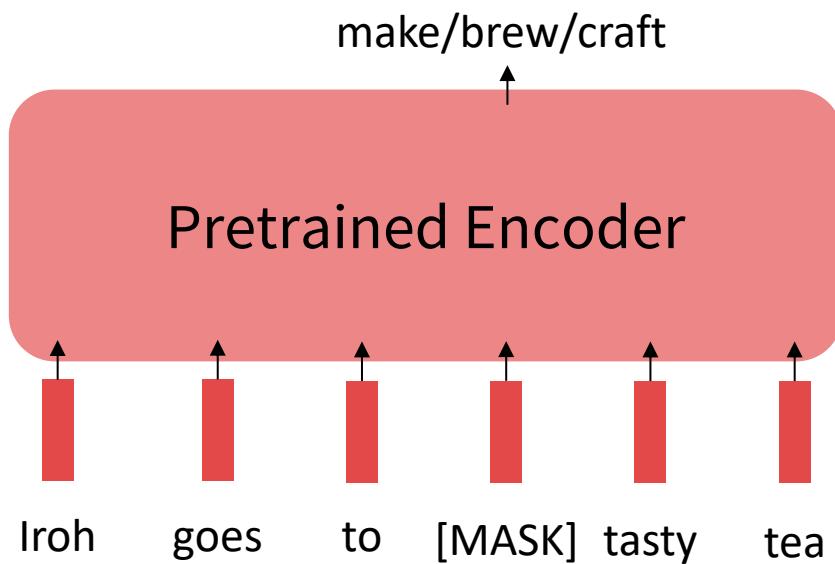
[\(Devlin et al., 2018\)](#)

# BERT: Bidirectional Encoder Representations from Transformers

- ❑ BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.
  - QQP: Quora Question Pairs (detect paraphrase questions)
  - QNLI: natural language inference over question answering data
  - SST-2: sentiment analysis
  - CoLA: corpus of linguistic acceptability (detect whether sentences are grammatical.)
  - STS-B: semantic textual similarity
  - MRPC: microsoft paraphrase corpus
  - RTE: a small natural language inference corpus

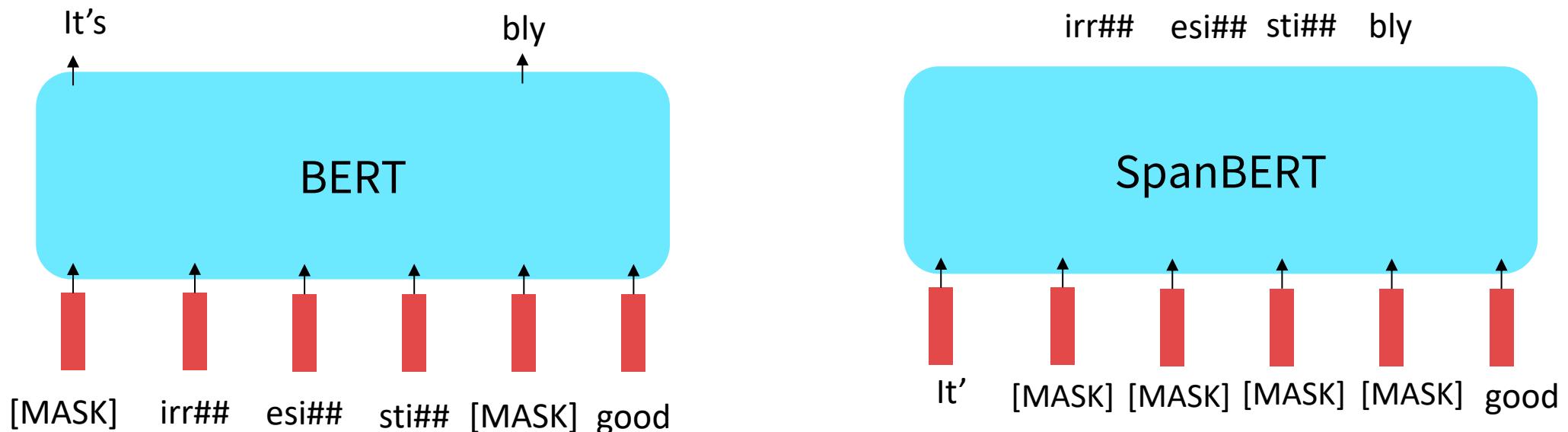
# Limitations of pretrained encoders

- ❑ Those results looked great! Why not used pretrained encoders for everything?
- ❑ If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.



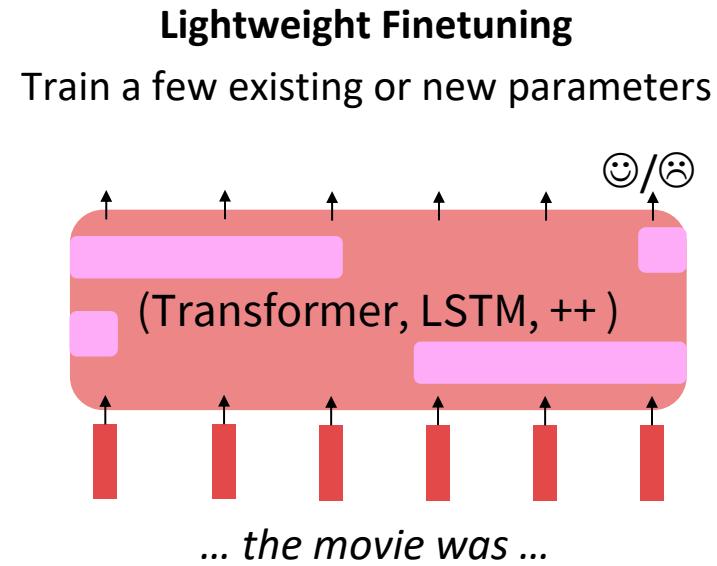
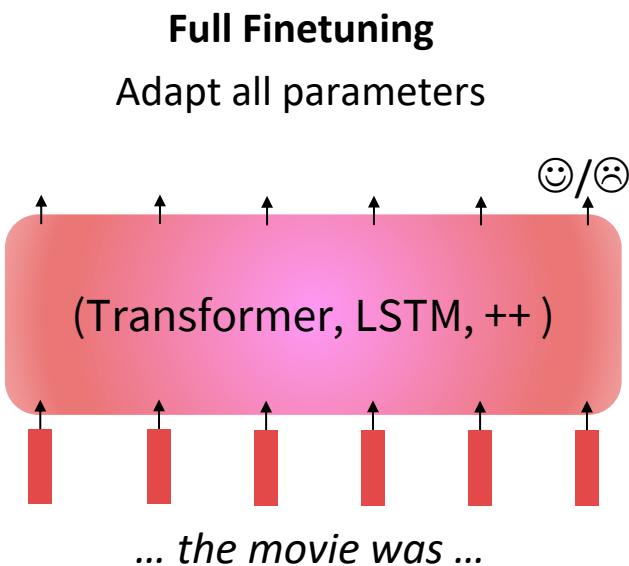
# Extensions of BERT

- ❑ You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++
- ❑ Some generally accepted improvements to the BERT pretraining formula:
  - RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
  - SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task



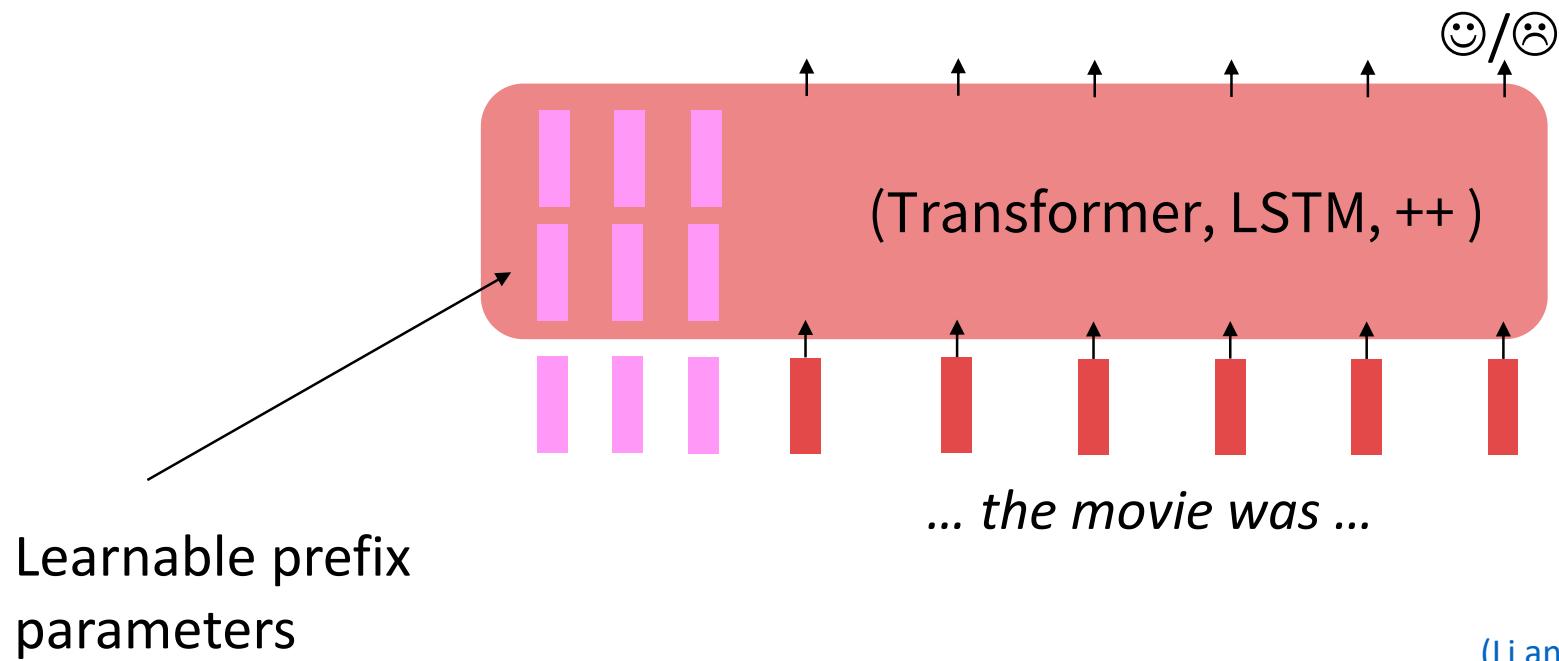
# Full Finetuning vs. Parameter-Efficient Finetuning

- ❑ 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.



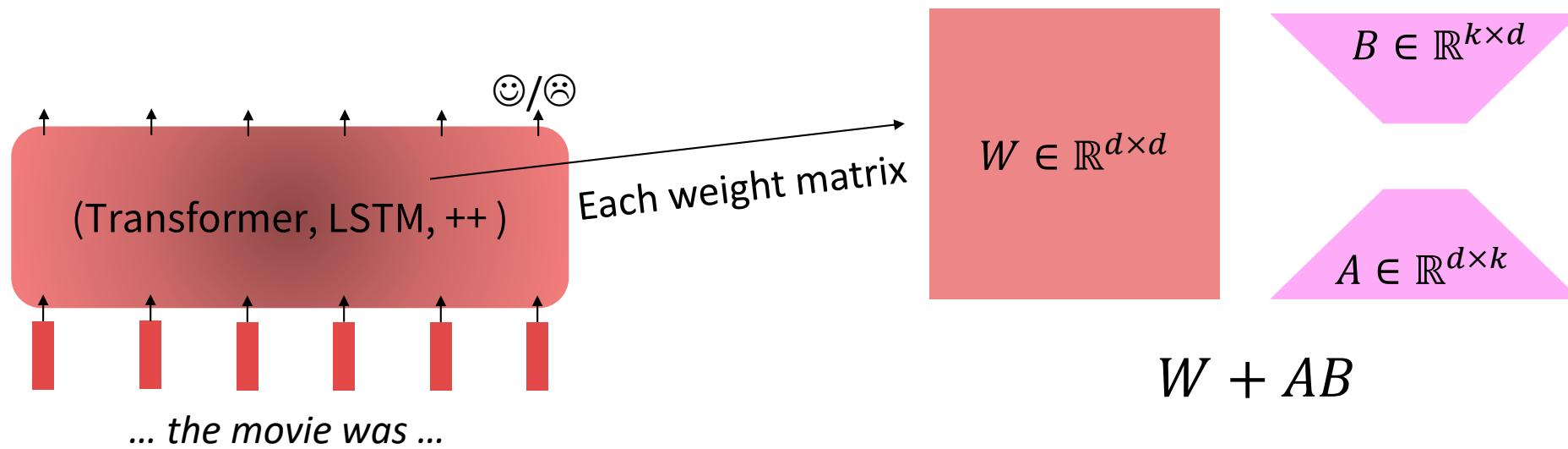
# Parameter-Efficient Finetuning: Prefix-Tuning, Prompt tuning

- ❑ 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.



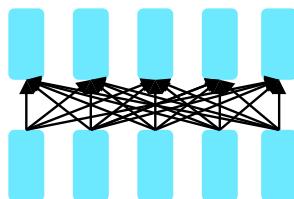
# Parameter-Efficient Finetuning: Low-Rank Adaptation

- ❑ Low-Rank Adaptation Learns a low-rank “diff” between the pretrained and finetuned weight matrices.
- ❑ Easier to learn than prefix-tuning.



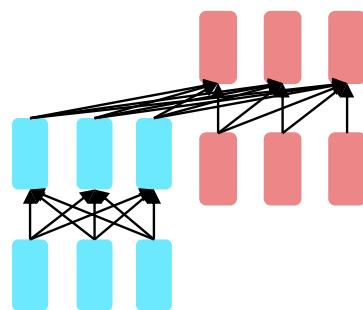
# Pretraining for three types of architectures

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



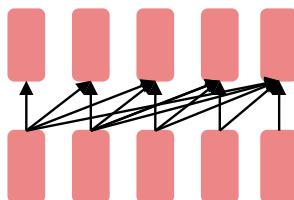
**Encoders**

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?



**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?

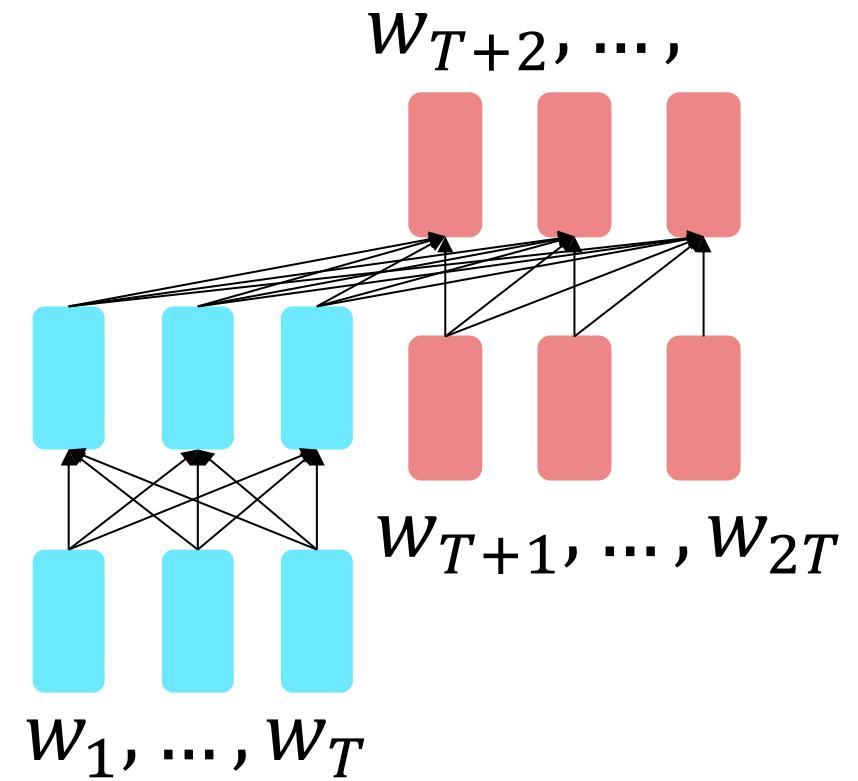
- For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

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

$$h_{T+1}, \dots, h_2 = \text{Decoder}(w_1, \dots, w_T, h_1, \dots, h_T)$$

$$y_i \sim Aw_i + b, i > T$$

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

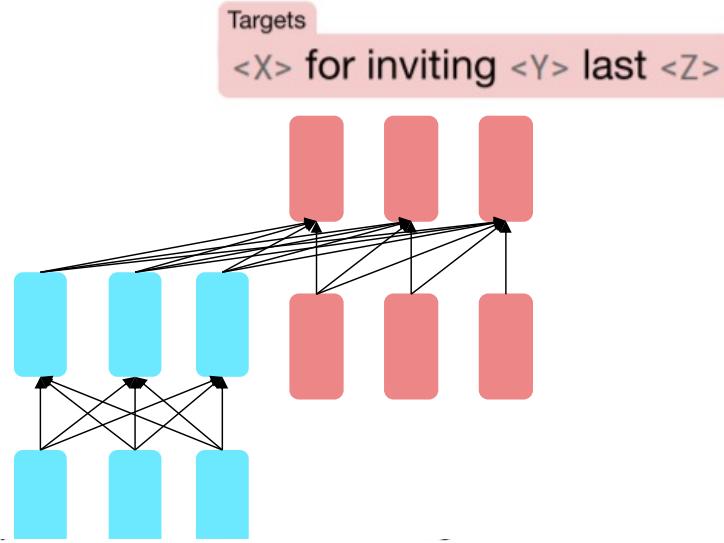


(Raffel et al., 2018)

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

- ❑ What Raffel et al., 2018 found to work best was **span corruption**. Their model: **T5**.
- ❑ Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

Original text  
Thank you for inviting me to your party last week.



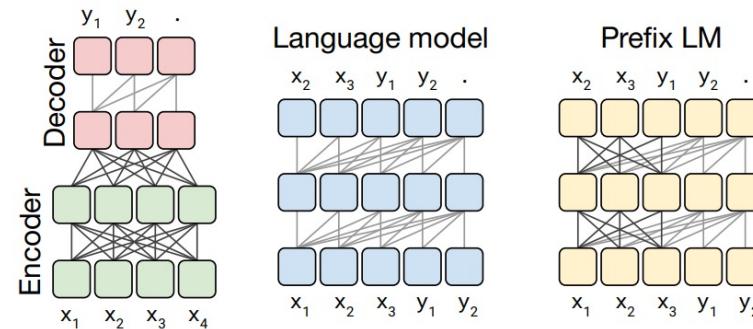
- ❑ This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.

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

(Raffel et al., 2018)

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

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

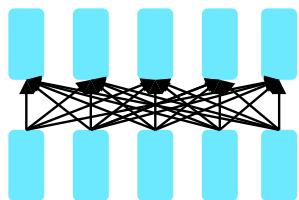


Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec,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

(Raffel et al., 2018)

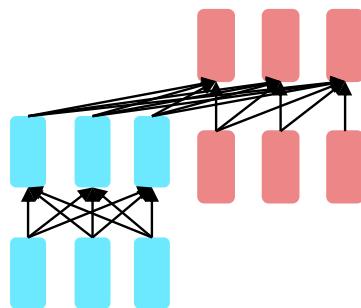
# Pretraining for three types of architectures

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



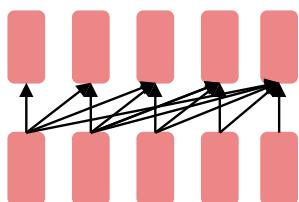
**Encoders**

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?



**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

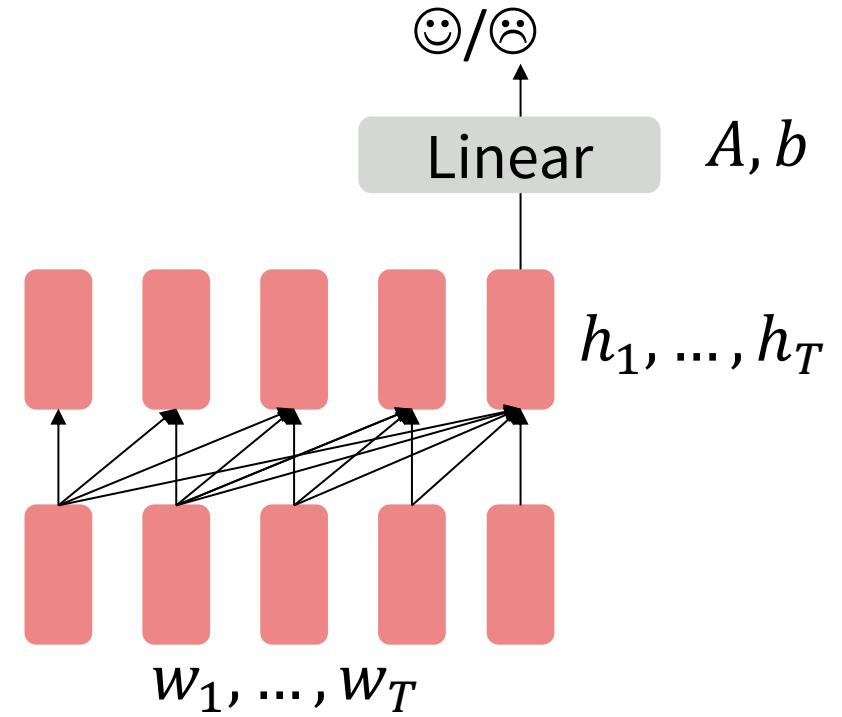
# Pretraining decoders

- ❑ When using language model pretrained decoders, we can ignore that they were trained to model  $p_\theta(w_t|w_{1:t-1})$
- ❑ We can finetune them by training a classifier on the last word's hidden state.

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

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

- ❑ Gradients backpropagate through the whole network.



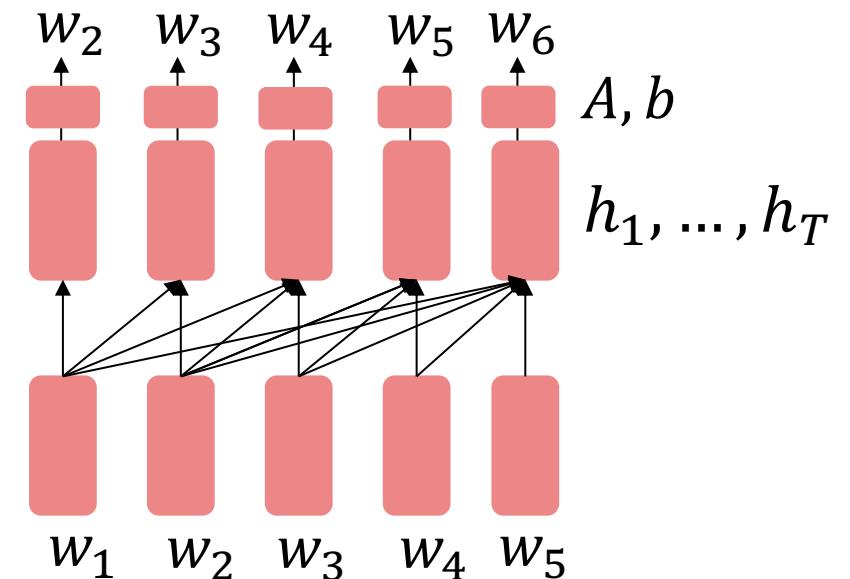
[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

# Pretraining decoders

- ❑ It's natural to pretrain decoders as language models and then use them as generators, finetuning their  $p_\theta(w_t|w_{1:t-1})$
- ❑ This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!
  - Dialogue (context=dialogue history)
  - Summarization (context=document)

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

where  $A, b$  were pretrained in the language model!



[Note how the linear layer has been pretrained.]

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

- 2018's GPT was a big success in pretraining a decoder!
  - Transformer decoder with 12 layers.
  - 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
  - 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.

## 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

# Increasingly convincing generations (GPT2) [\(Radford et al., 2018\)](#)

- We mentioned how pretrained decoders can be used in their capacities as language models.
- GPT-2, a larger version of GPT (48 layers) trained on more data (~ x10), 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 then take their predictions.
- ❑ Emergent behavior: Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.
- ❑ GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters. **GPT-3 has 175 billion parameters.**

# Natural Language Generation

# What is natural language generation?

- ❑ Natural language generation is one side of natural language processing.
  - NLP = Natural Language Understanding (NLU) + Natural Language Generation (NLG)
- ❑ Any task involving language production for human consumption requires natural language generation
- ❑ NLG focuses on systems that produce **coherent** and **useful** language output for human consumption
- ❑ Deep Learning is powering (some) next-gen NLG systems!

# Uses of natural language generation

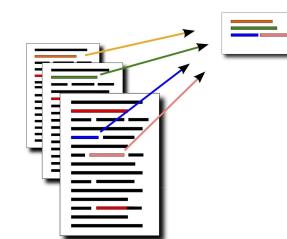
- ❑ Machine Translation systems use NLG for output



- ❑ Digital assistant (dialogue) systems use NLG



- ❑ Summarization systems (for research articles, email, meetings, documents) use NLG

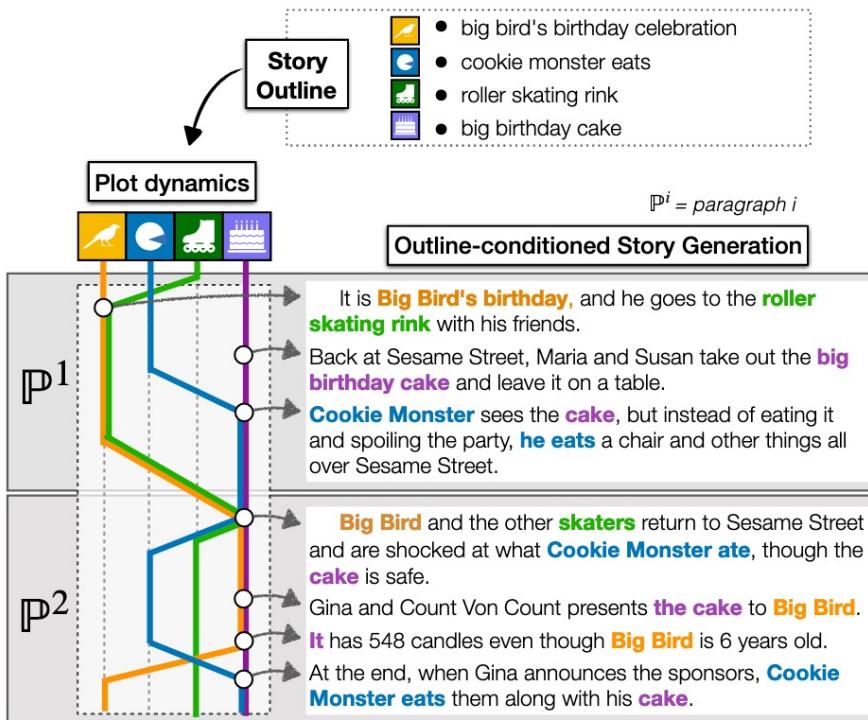


C: Looking at what we've got, we we want an LCD display with a spinning wheel.  
B: You have to have some push-buttons, don't you?  
C: Just spinning and not scrolling, I would say.  
B: I think the spinning wheel is definitely very now.  
A: but since LCDs seems to be uh a definite yes,  
C: We're having push-buttons on the outside  
C: and then on the inside an LCD with spinning wheel,

**Decision Abstract (Summary):**  
The remote will have push buttons outside, and an LCD and spinning wheel inside.

# More interesting NLG uses

## 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

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

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)

# SOTA NLG system

- ❑ ChatGPT is an NLG system!
- ❑ It's general purpose and can do many NLG tasks!
  - e.g., Chatbot

XL Got any creative ideas for a 10 year old's birthday?

 Here are some creative birthday ideas for a 10 year old:

1. Scavenger Hunt: A birthday scavenger hunt can be a fun and interactive activity for kids. You can create a series of clues that lead the birthday child to the next one, until they reach the final prize.

2. DIY Crafts: Crafts are always a hit with kids. You can arrange a crafting session where the birthday child and their friends can make their own personalized gifts or decorations.

3. Movie Night: Have a movie night with friends and family at home or at a local theater. You can also make it a sleepover by setting up sleeping bags, pillows, and blankets.

thumb up thumb down

# SOTA NLG system

- ❑ ChatGPT is an NLG system!
- ❑ It's general purpose and can do many NLG tasks!
  - e.g., Poetry Generation:



Sorting algorithms, oh so wise,  
Bringing order to chaotic ties.  
Bubble sort, quick sort, merge sort too,  
Each one unique, with their own special view.

Bubble sort, gentle as can be,  
Compares and swaps, so efficiently.  
It moves like waves, across the sea,  
Bringing order, so beautifully.

Quick sort, so fast and strong,  
Divides and conquers all day long.  
With each partition, it grows so wide,  
Order is found, on every side.

Merge sort, so neat and trim,  
Divides and merges, one by one.  
It builds a bridge, to conquer all,  
Order is its overall call.

# Categorization of NLG tasks

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

Spectrum of open-endedness for Generation Tasks



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

Spectrum of open-endedness for Generation Tasks



Input: Write a story about three little pigs?

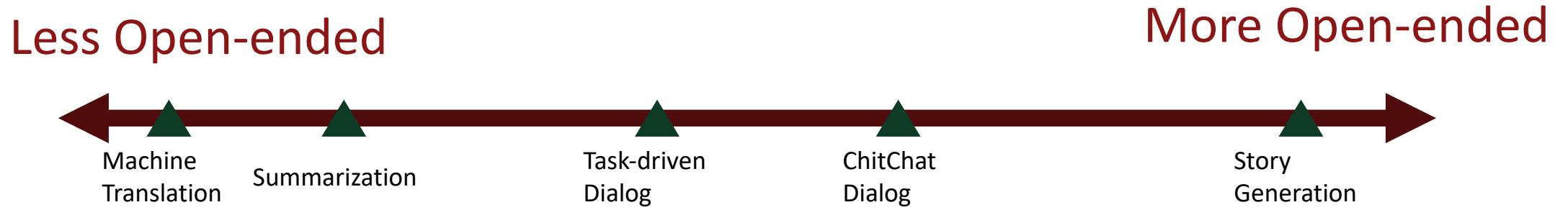
Outputs:

... (so many options) ...

The output space is extremely diverse...

# Categorization of NLG tasks

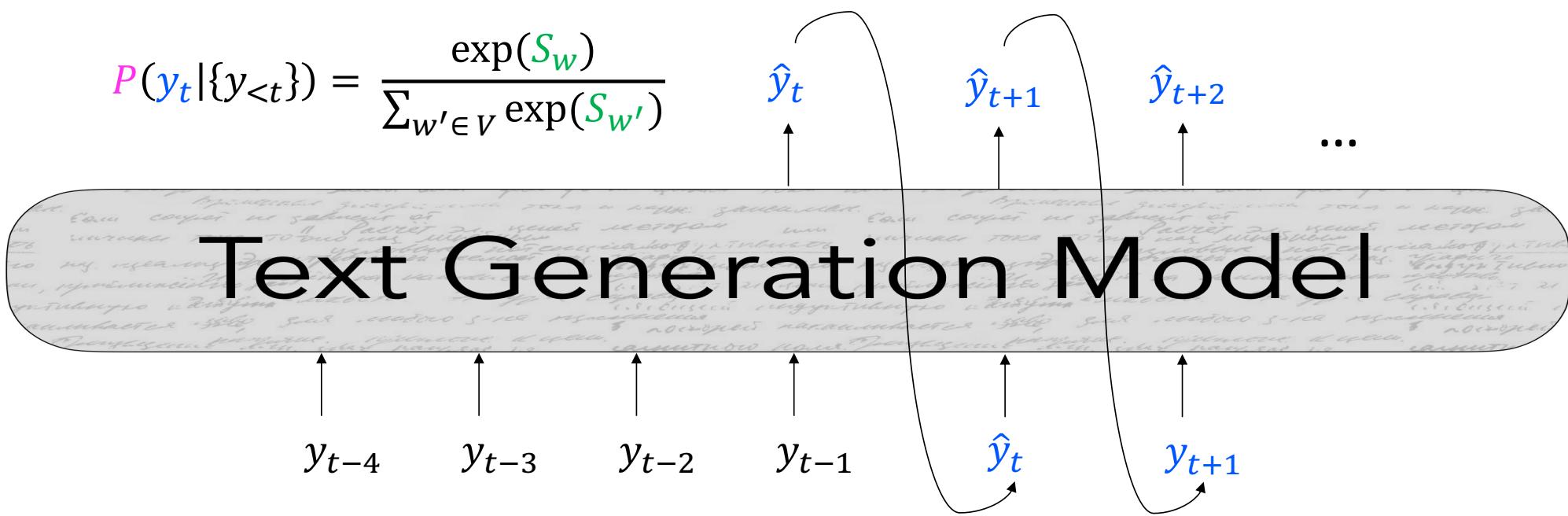
- ❑ Open-ended generation: the output distribution still has high freedom
  - ❑ Non-open-ended generation: the input mostly determines the output generation.



- Remark: One way of formalizing categorization this is by **entropy**.
    - These two classes of NLG tasks require different decoding and/or training approaches!

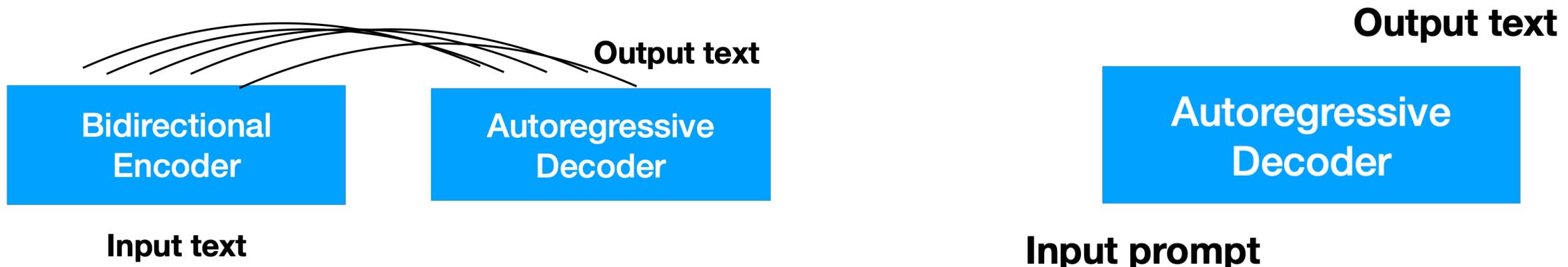
# Basics of natural language generation (review)

- In autoregressive text generation models, at each time step  $t$ , our model takes in a sequence of tokens of text as input  $\{y\}_{<t}$  and outputs a new token,  $\hat{y}_t$
- For model  $f(\cdot)$  and vocab  $V$ , we get scores  $S = f(\{y_{<t}\}, \theta) \in \mathbb{R}^V$



# Basics of natural language generation (review)

- ❑ For non-open-ended tasks (e.g., MT), we typically use a encoder-decoder system, where this autoregressive model serves as the decoder, and we'd have another bidirectional encoder for encoding the inputs.
- ❑ For open-ended tasks (e.g., story generation), this autoregressive generation model is often the only component.

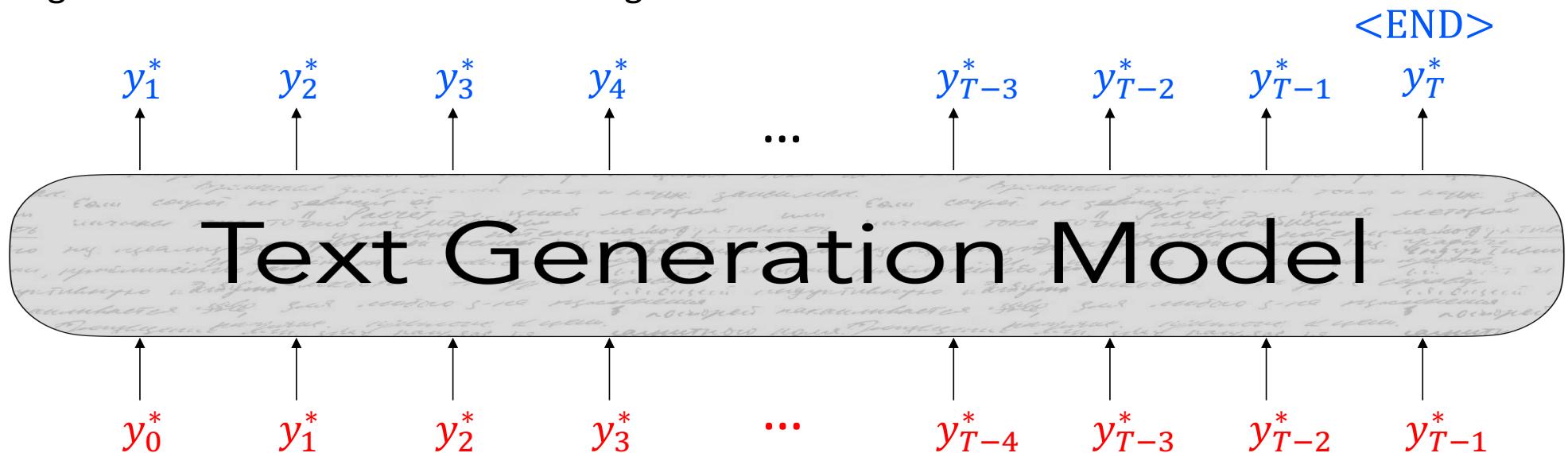


Trained one token at a time by maximum likelihood *teacher forcing*

- Trained to maximize the probability of the next token  $y_t^*$  given preceding words  $\{y^*\}_{<t}$

$$\mathcal{L} = - \sum_{t=1}^T \log P(y_t^* | \{y^*\}_{<t})$$

- This is a classification task at each time step trying to predict the actual word  $y_t^*$  in the training data
- Doing this is often called “teacher forcing”



# Basics of natural language generation (review)

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

$$\hat{y}_t = g(P(y_t | \{y_{<t}\}))$$

*g(.)* is your decoding algorithm

- The “obvious” decoding algorithm is to greedily choose the highest probability next token according to the model at each time step
- While this basic algorithm sort of works, to do better, the two main avenues are:
  - 1. Improve the decoder
  - 2. Improve the training
  - Of course, there’s also improving your training data or model architecture

# Decoding from NLG models

# Decoding: what is it all about?

- At each time step  $t$ , our model computes a vector of scores for each token in our vocabulary,  $S \in \mathbb{R}^V$ :

$$S = f(\{y_{<t}\})$$

$f(\cdot)$  is your model

- Then, we compute a probability distribution  $P$  over these scores (usually with a softmax function):

$$P(y_t = w | \{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- Our decoding algorithm defines a function to select a token from this distribution:

$$\hat{y}_t = g(P(y_t | \{y_{<t}\}))$$

$g(\cdot)$  is your decoding algorithm

# How to find the most likely string? (review)

## □ Greedy Decoding

- Selects the highest probability token in  $P(y_t | y_{<t})$

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

## □ Beam Search

- Discussed in the lecture on Machine Translation
- Also aims to find strings that maximize the log-prob, but with wider exploration of candidates

## □ Overall, maximum probability decoding is good for low-entropy tasks like MT and summarization!

# The most likely string is repetitive for Open-ended Generation

## Context:

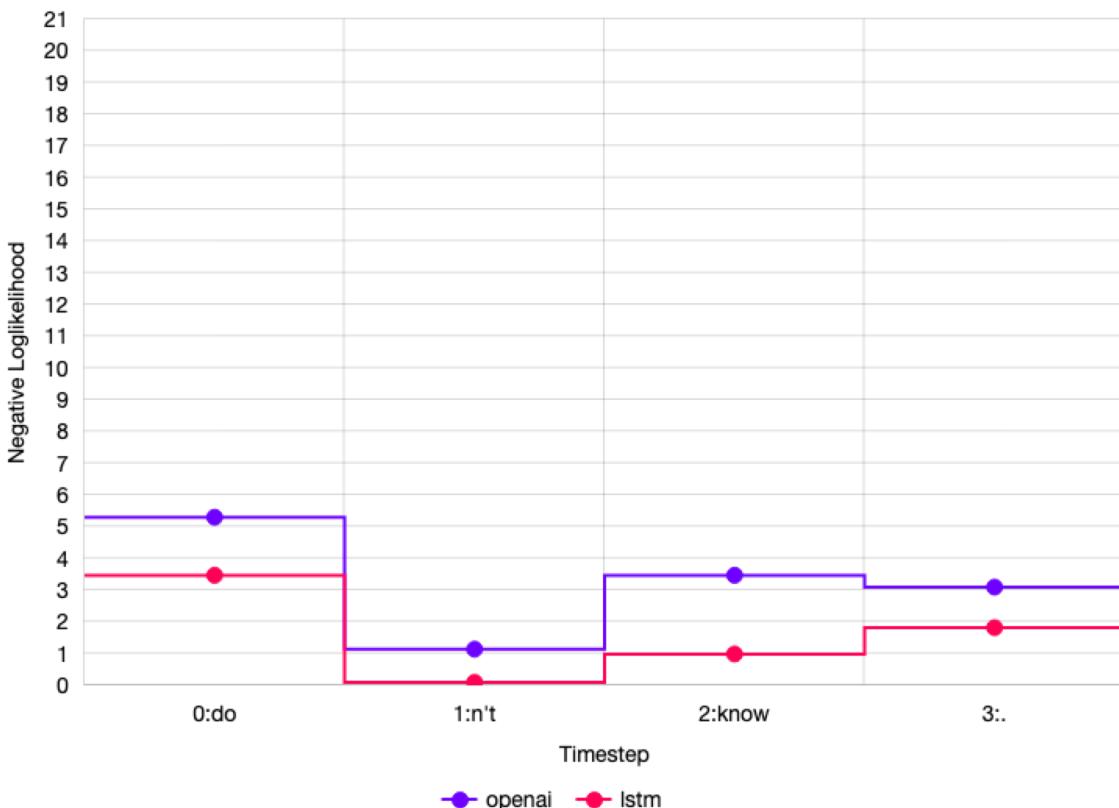
In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

## Continuation:

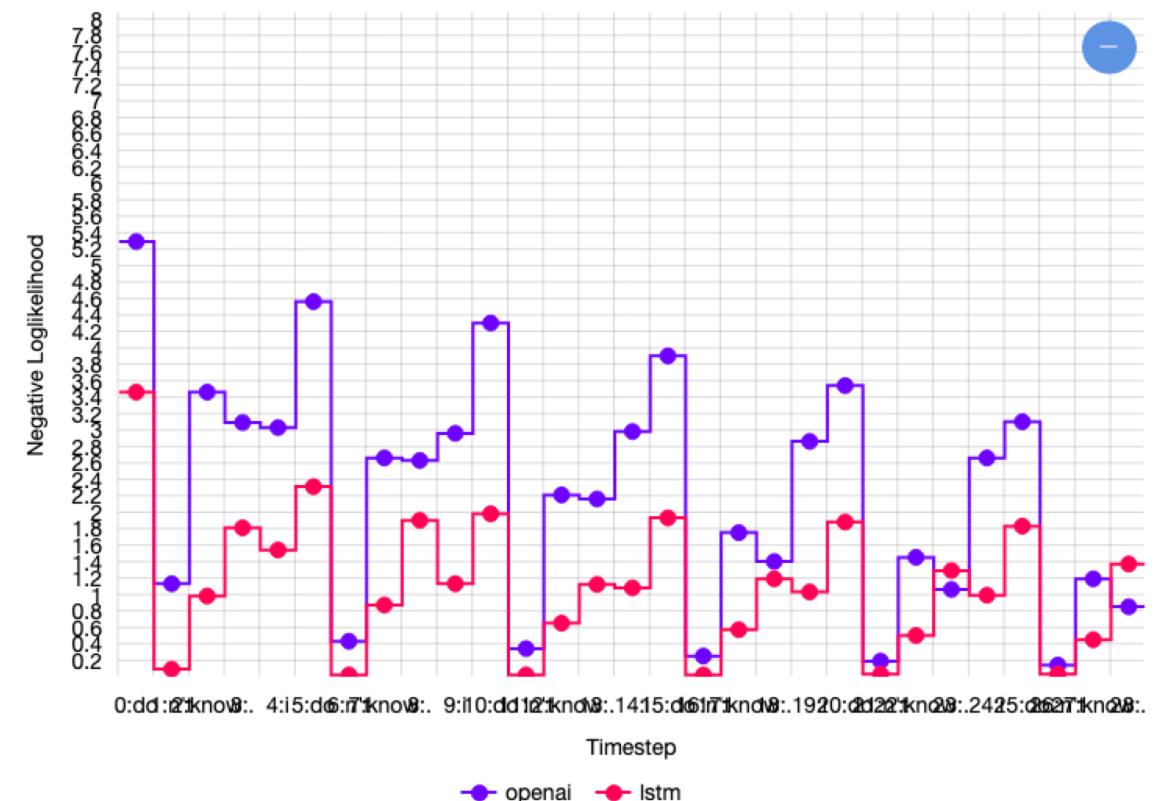
The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México...)**

# Why does repetition happen?

I don't know.



I don't know. I don't know.

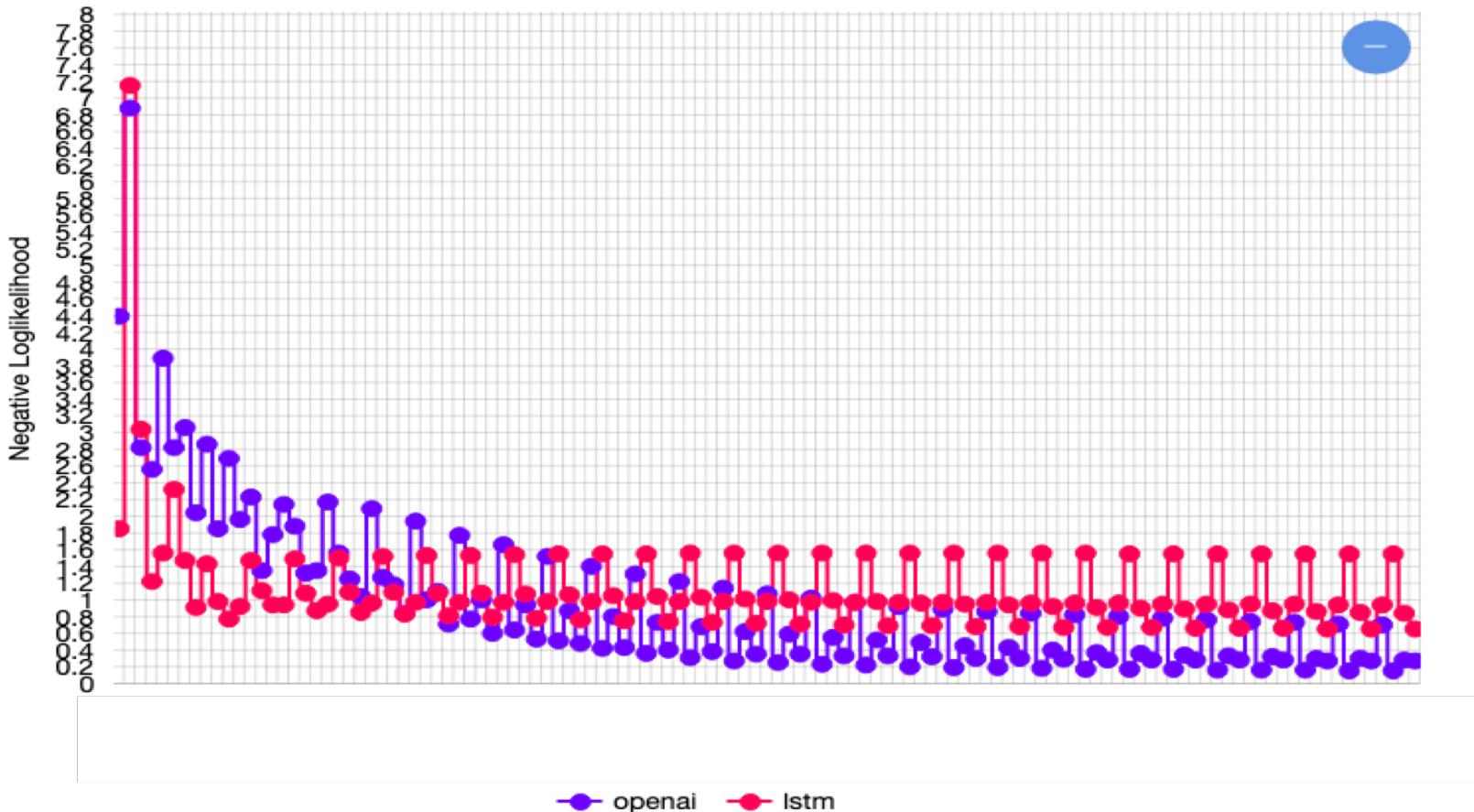


A self-amplification effect!

(Holtzman et. al., ICLR 2020)

# And it keeps going...

I'm tired. I'm tired.



Scale doesn't solve this problem: even a 175 billion parameter LM still repeats when we decode for the most likely string.  
(Holtzman et. al., ICLR 2020)

# How can we reduce repetition?

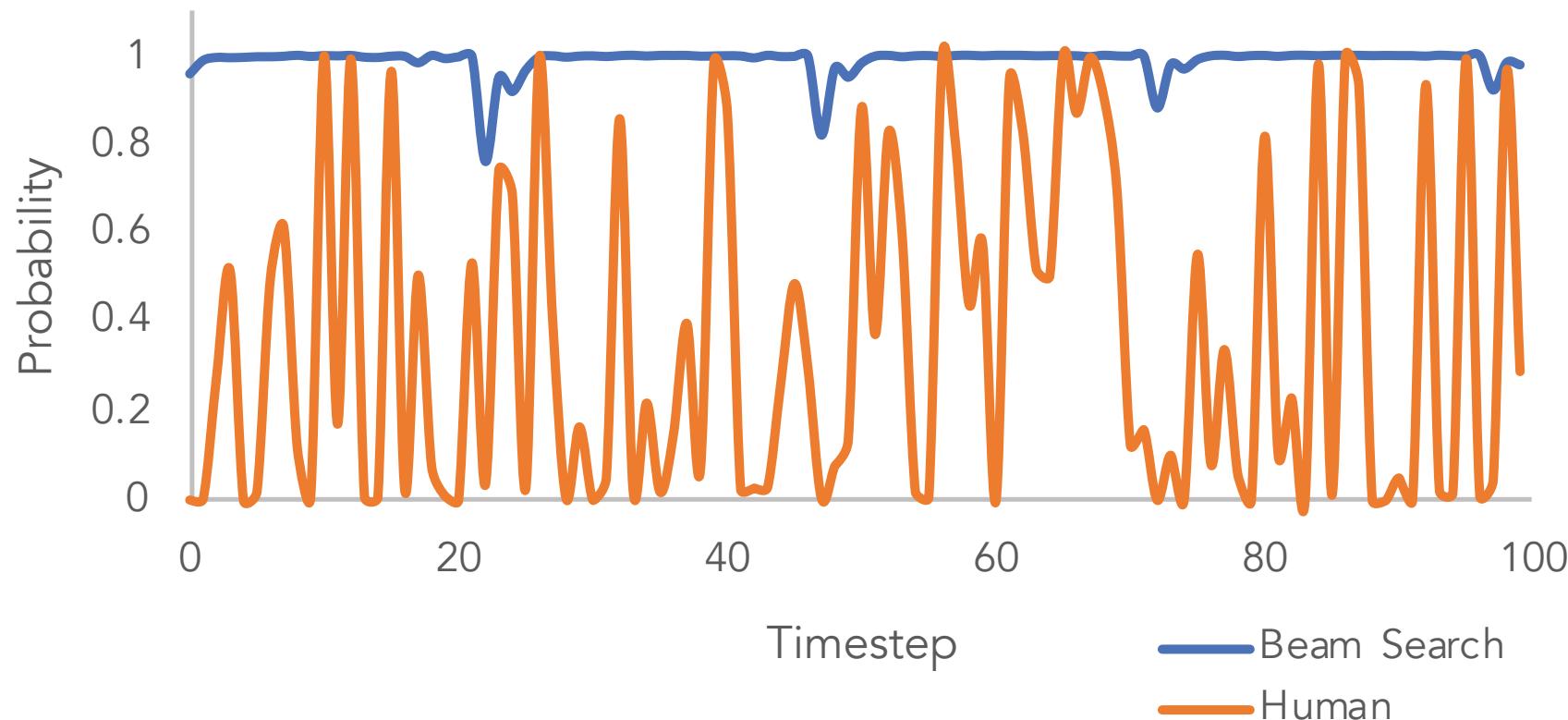
## □ Simple option:

- Heuristic: Don't repeat n-grams

## □ More complex:

- Use a different training objective:
  - Coverage loss (See et al., 2017)
    - Prevents attention mechanism from attending to the same words
  - Unlikelihood objective (Welleck et al., 2020)
    - Penalize generation of already-seen tokens
- Use a different decoding objective:
  - Contrastive decoding (Li et al, 2022)
    - Searches for strings  $x$  that maximize  $\text{logprob\_largeLM}(x) - \text{logprob\_smallLM}(x)$

# Is finding the most likely string reasonable for open-ended generation?



It fails to match the uncertainty distribution for human generated text.

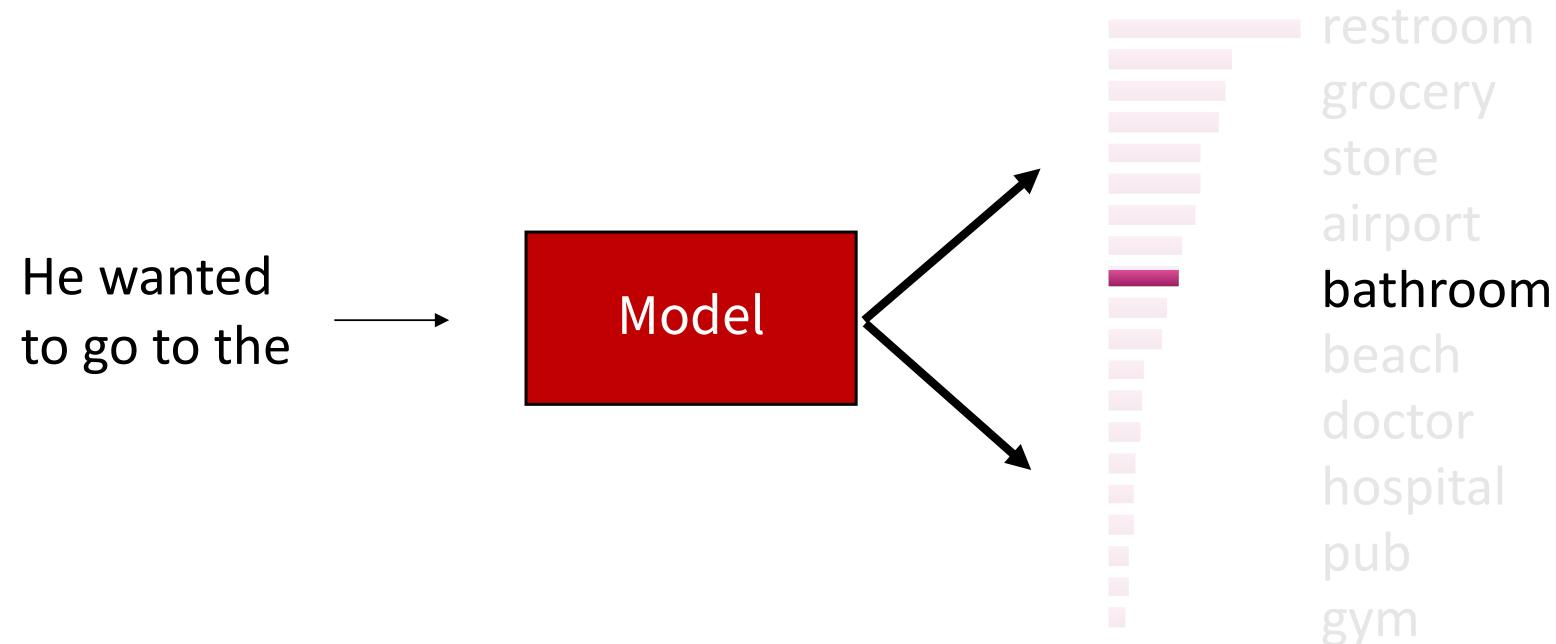
(Holtzman et. al., ICLR 2020)

# Time to get random : Sampling!

- Sample a token from the distribution of tokens

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

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



# Decoding: Top- $k$ sampling

## ❑ Problem: Vanilla sampling makes every token in the vocabulary an option

- Even if most of the **probability mass** in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass (statistics speak: we have “**heavy tailed**” distributions)
- Many tokens are probably *really wrong* in the current context
- For these wrong tokens, we give them *individually* a tiny chance to be selected.
- But because there are many of them, we still give them *as a group* a high chance to be selected.

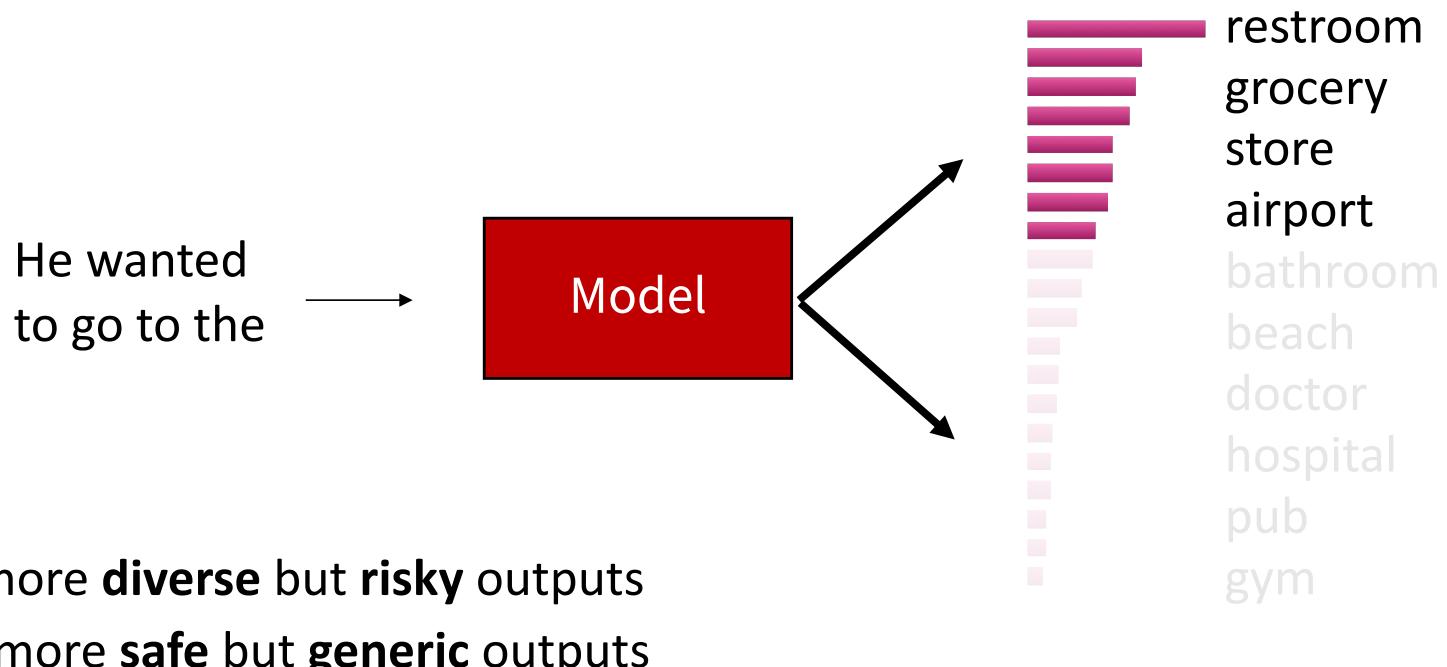
## ❑ Solution: Top- $k$ sampling

- Only sample from the top  $k$  tokens in the probability distribution

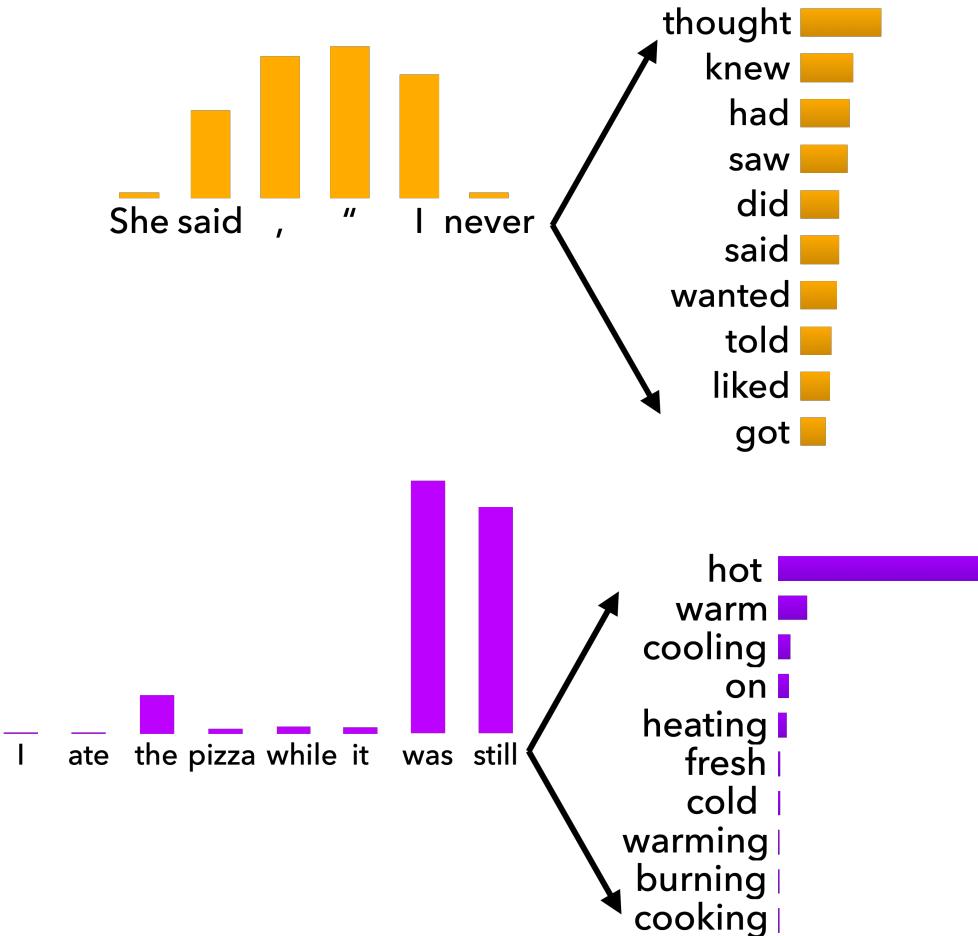
# Decoding: Top- $k$ sampling

## □ Solution: Top- $k$ sampling

- Only sample from the top  $k$  tokens in the probability distribution
- Common values are  $k = 5, 10, 20$  (but it's up to you!)



# Issues with Top- $k$ sampling



Top- $k$  sampling can cut off too *quickly*!

Top- $k$  sampling can also cut off too *slowly*!

# Decoding: Top- $p$ (nucleus) sampling

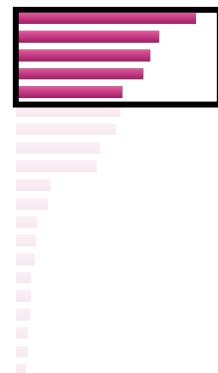
- **Problem:** The probability distributions we sample from are dynamic
  - When the distribution  $P_t$  is flatter, a limited  $k$  removes many viable options
  - When the distribution  $P_t$  is peakier, a high  $k$  allows for too many options to have a chance of being selected
  
- **Solution:** Top- $p$  sampling
  - Sample from all tokens in the top  $p$  cumulative probability mass (i.e., where mass is concentrated)
  - Varies  $k$  depending on the uniformity of  $P_t$

# Decoding: Top- $p$ (nucleus) sampling

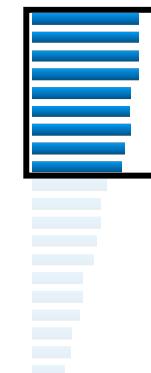
## □ Solution: Top- $p$ sampling

- Sample from all tokens in the top  $p$  cumulative probability mass (i.e., where mass is concentrated)
- Varies  $k$  depending on the uniformity of  $P_t$

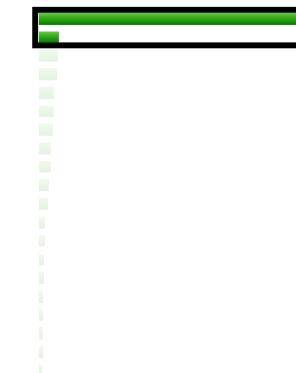
$$P_t^1(y_t = w | \{y\}_{<t})$$



$$P_t^2(y_t = w | \{y\}_{<t})$$



$$P_t^3(y_t = w | \{y\}_{<t})$$



# Scaling randomness: Softmax 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  $\tau$  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)

- ❑ Note: softmax temperature is not a decoding algorithm!
  - Temperature is a hyperparameter for decoding:  
It can be tuned for both beam search and sampling.

# Improving decoding: Re-ranking

- **Problem:** What if I decode a bad sequence from my model?
- Decode a bunch of sequences
  - 10 candidates is a common number, but it's up to you
- Define a score to approximate quality of sequences and re-rank by this score
  - Simplest is to use (low) perplexity!
    - Careful! Remember that repetitive utterances generally get low perplexity.
  - Re-rankers can score a variety of properties:
    - style(Holtzman et al., 2018), discourse(Gabriele et al., 2021), entailment/factualty(Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more ...
    - Beware poorly-calibrated re-rankers
  - Can compose multiple re-rankers together.

## Decoding: Takeaways

- ❑ 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

# Training NLG models

# Is repetition due to how LMs are trained?

## Context:

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.

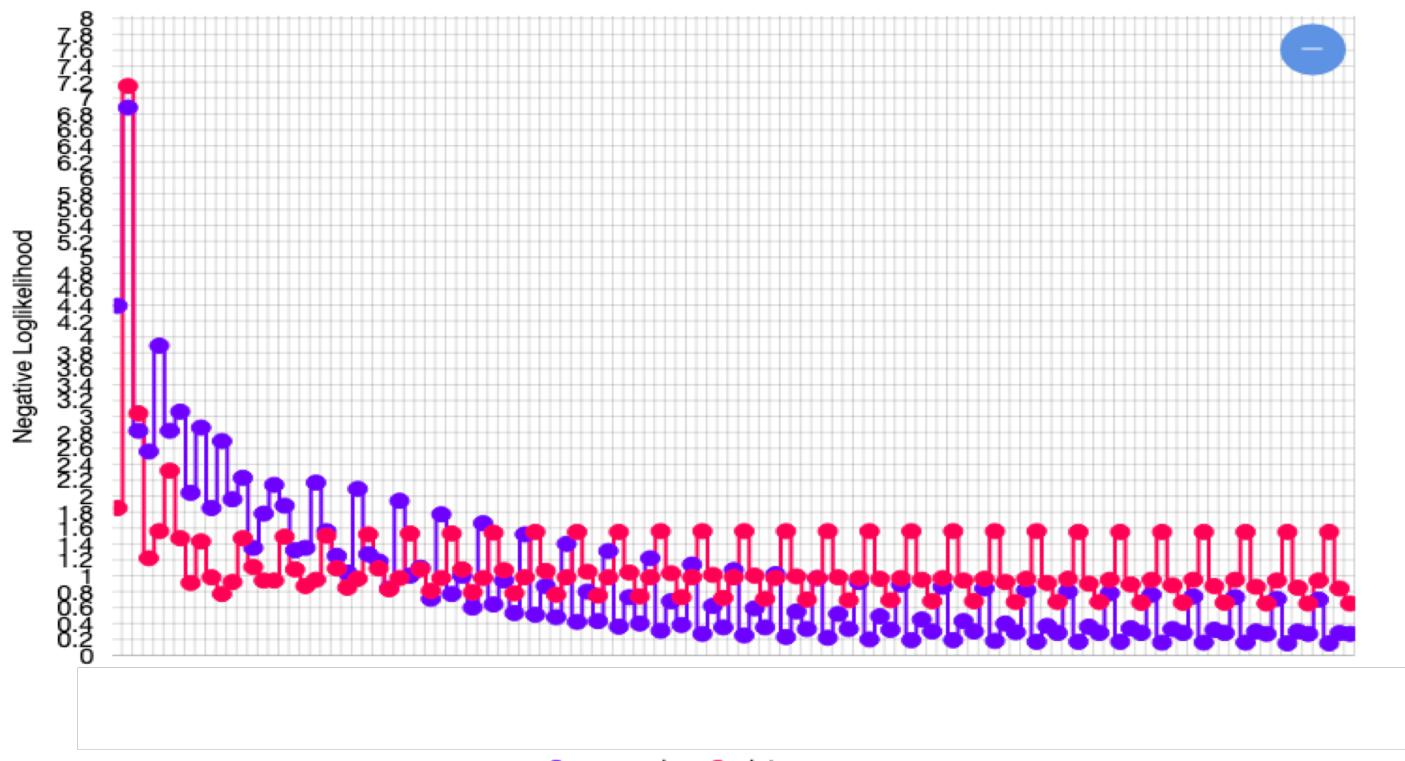
## Continuation:

The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México...)**

# Diversity Issues

- Maximum Likelihood Estimation model learns bad mode of the text distribution.

I'm tired. I'm tired.



(Holtzman et. al., ICLR 2020)

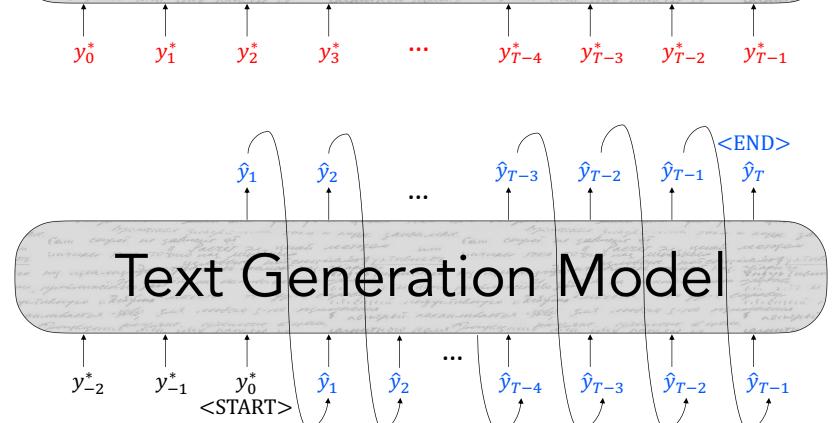
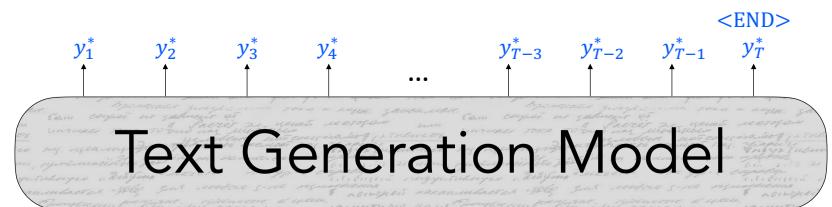
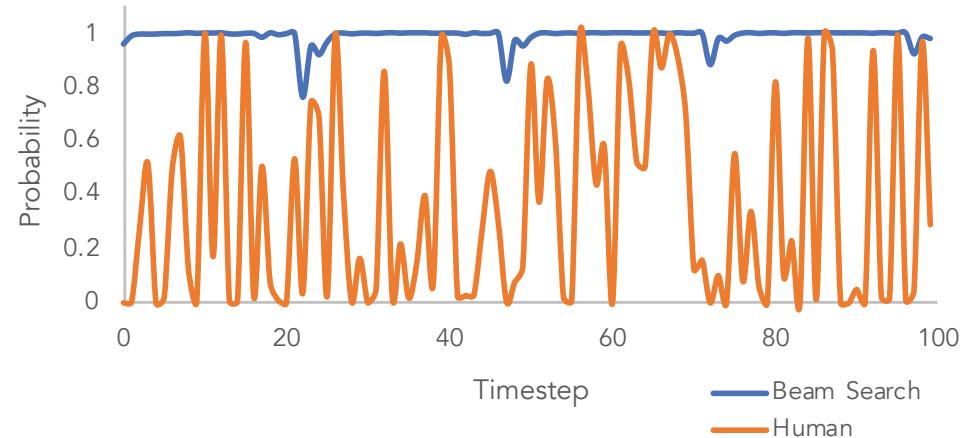
# Exposure Bias

- Training with teacher forcing leads to **exposure bias** at generation time
  - During training, our model's inputs are gold context tokens from real, human-generated texts

$$\mathcal{L}_{MLE} = -\log P(y_t^* | \{y^*\}_{<t})$$

- At generation time, our model's inputs are previously-decoded tokens

$$\mathcal{L}_{dec} = -\log P(\hat{y}_t | \{\hat{y}\}_{<t})$$



# Exposure Bias Solutions

## ❑ Scheduled sampling (Bengio et al., 2015)

- With some probability  $p$ , **decode a token** and feed that as the next input, rather than the **gold token**.
- Increase  $p$  over the course of training
- Leads to improvements in practice, but can lead to **strange training objectives**

## ❑ Dataset Aggregation (DAgger; Ross et al., 2011)

- At various intervals during training, generate sequences from your current model
- **Add these sequences** to your training set as additional examples

# Exposure Bias Solutions

## ❑ Retrieval Augmentation (Guu\*, Hashimoto\*, et al., 2018)

- Learn to retrieve a sequence from an existing corpus of human-written prototypes (e.g., dialogue responses)
- Learn to edit the retrieved sequence by adding, removing, and modifying tokens in the prototype – this will still result in a more “human-like” generation

## ❑ Reinforcement Learning: cast your text generation model as a *Markov decision process*

- **State  $s$**  is the model’s representation of the preceding context
- **Actions  $a$**  are the words that can be generated
- **Policy  $\pi$**  is the decoder
- **Rewards  $r$**  are provided by an external score
- Learn behaviors by rewarding the model when it exhibits them

# Reward Estimation

- ❑ How should we define a reward function? Just use your evaluation metric!
  - **BLEU** (machine translation; Ranzato et al., ICLR 2016; Wu et al., 2016)
  - **ROUGE** (summarization; Paulus et al., ICLR 2018; Celikyilmaz et al., NAACL 2018)
  - CIDEr (image captioning; Rennie et al., CVPR 2017)
  - SPIDER (image captioning; Liu et al., ICCV 2017)
  
- ❑ Be careful about **optimizing for the task** as opposed to “**gaming**” the reward!
  - Evaluation metrics are merely proxies for generation quality!
  - “even though RL refinement can achieve better BLEU scores, it barely improves the human impression of the translation quality” – Wu et al., 2016

# Reward Estimation

- What behaviors can we tie to rewards?
  - Cross-modality consistency in image captioning (Ren et al., CVPR 2017)
  - Sentence simplicity (Zhang and Lapata, EMNLP 2017)
  - Temporal Consistency (Bosselut et al., NAACL 2018)
  - Utterance Politeness (Tan et al., TACL 2018)
  - Paraphrasing (Li et al., EMNLP 2018)
  - Sentiment (Gong et al., NAACL 2019)
  - Formality (Gong et al., NAACL 2019)

# Reward Estimation

- ❑ Human Preference (RLHF): this is the technique behind ChatGPT!
  - (Ziegler et al. 2019, Stiennon et al., 2020)
  - Human ranking the generated text based on their preference.
  - Learn a reward function of the human preference.
  - We will talk about it more later
- ❑ If you can formalize a behavior as a Python function (or train a neural network to approximate it!), you can train a text generation model to exhibit that behavior!

## Training: Takeaways

- ❑ *Teacher forcing* is still the main algorithm for training text generation models
- ❑ **Diversity** is an issue with sequences generated from teacher forced models
  - New approaches focus on mitigating the effects of common words
- ❑ **Exposure bias** causes text generation models to **lose coherence** easily
  - Models must learn to recover from their own bad samples
    - e.g., scheduled sampling, DAgger
  - Or not be allowed to generate bad text to begin with (e.g., retrieval + generation)
- ❑ Training with RL can allow models to learn behaviors that are preferred by human preference / metrics

# Evaluating NLG Systems

# Types of evaluation methods for text generation

Ref: They walked **to the grocery store** .  
Gen: **The woman went to the hardware store** .

Content Overlap Metrics



Model-based Metrics

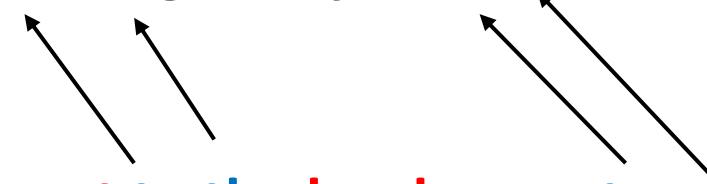


Human Evaluations

# Content overlap metrics

Ref: They walked **to the grocery store** .

Gen: **The woman went to the hardware store** .



- 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.)

# N-gram overlap metrics

## 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 than summarization
  - Much, much worse story generation, which is also open-ended, but whose sequence length can make it seem you're getting decent scores!

# A simple failure case

- ❑ n-gram overlap metrics have no concept of semantic relatedness!

Are you enjoying the NLP lectures?

Heck yes !



Score:

0.61

Yes !

0.25

You know it !

False negative

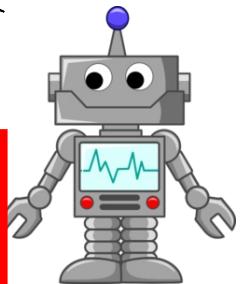
0

Yup .

False positive

0.67

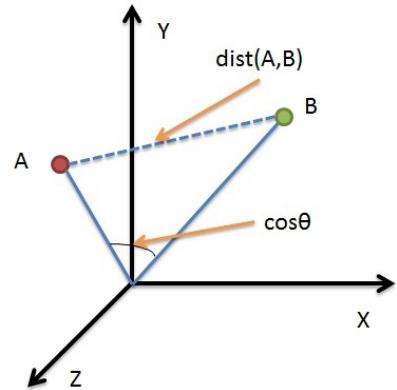
Heck no !



## Model-based metrics to capture more semantics

- ❑ Use **learned representations** of words and sentences to compute semantic similarity between generated and reference texts
- ❑ No more **n-gram bottleneck** because text units are represented as **embeddings!**
- ❑ The embeddings are **pretrained**, distance metrics used to measure the similarity can be **fixed**

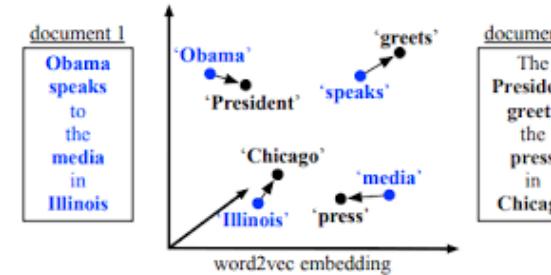
# Model-based metrics: Word distance functions



## 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)



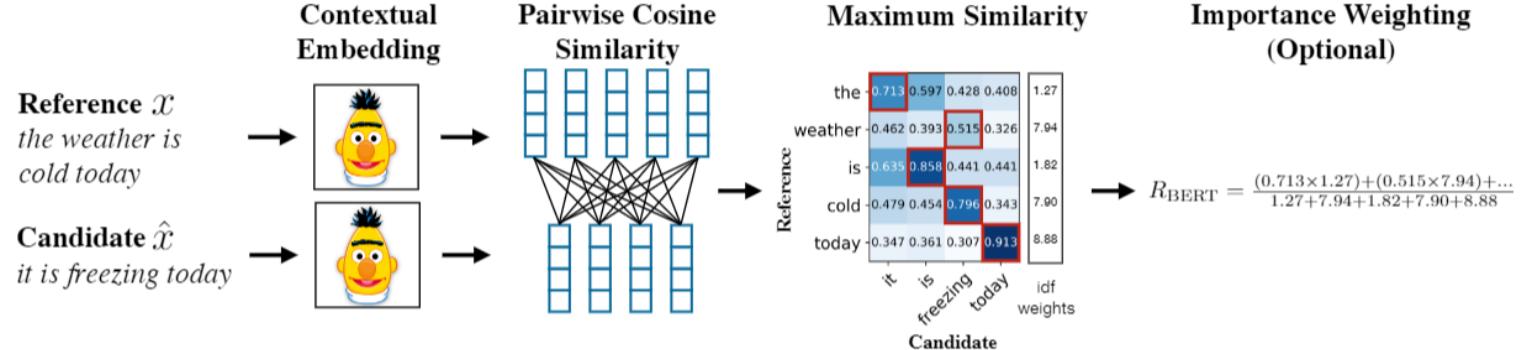
## Word Mover's Distance

Measures the distance between two sequences (e.g., sentences, paragraphs, etc.), using word embedding similarity matching.

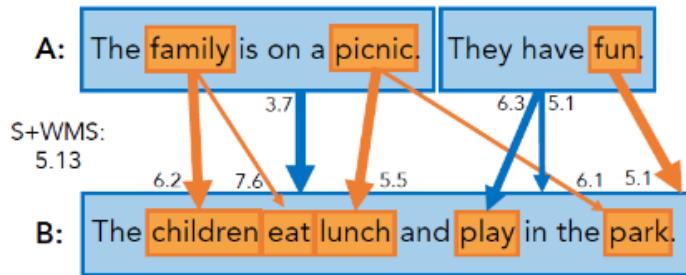
(Kusner et.al., 2015; Zhao et al., 2019)

## BERTSCORE

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



# Model-based metrics: Beyond word matching



## Sentence Movers Similarity :

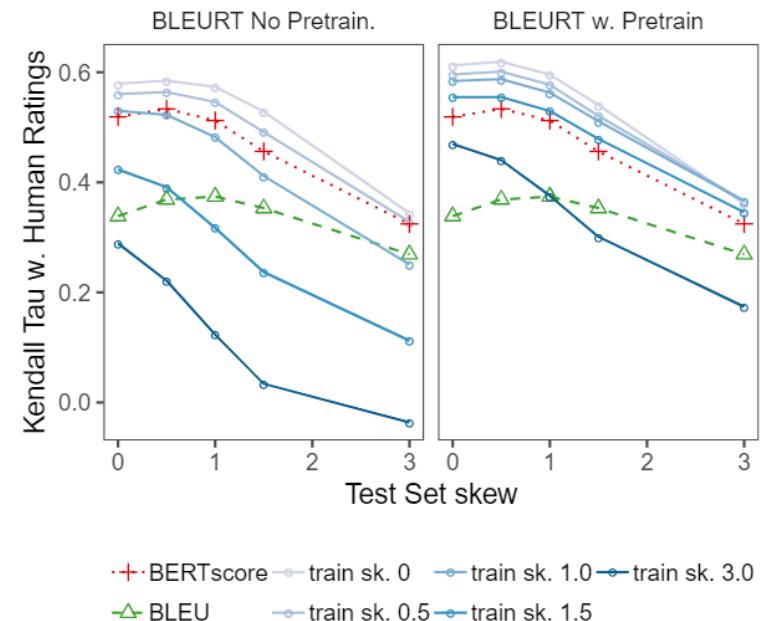
Based on Word Movers Distance to evaluate text in a continuous space using sentence embeddings from recurrent neural network representations.

(Clark et.al., 2019)

## BLEURT:

A regression model based on BERT returns a score that indicates to what extent the candidate text is grammatical and conveys the meaning of the reference text.

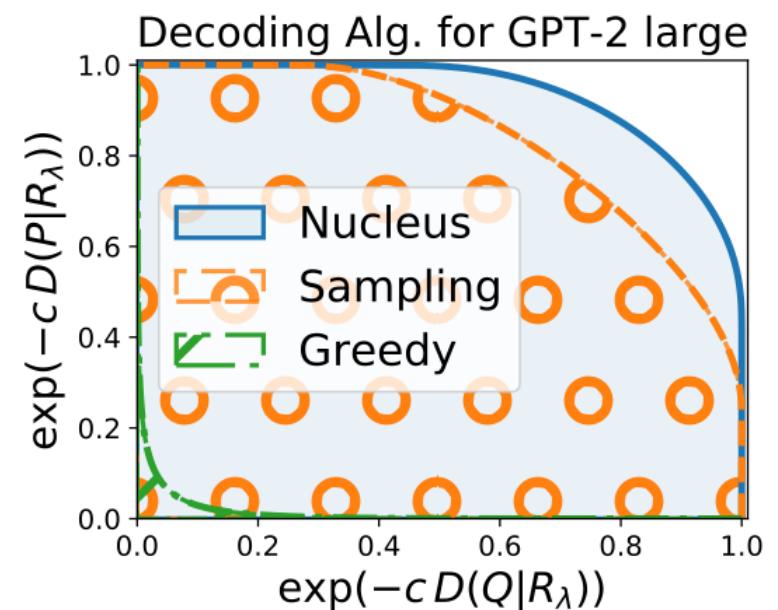
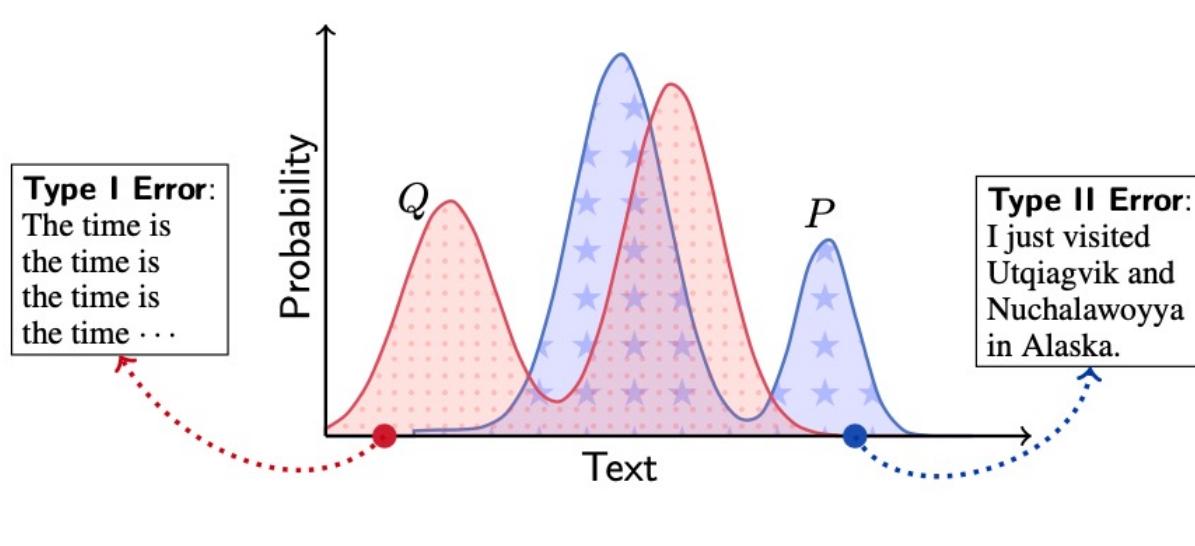
(Sellam et.al. 2020)



# Evaluating Open-ended Text Generation

## MAUVE

MAUVE computes information divergence in a quantized embedding space, between the generated text and the gold reference text (Pillutla et.al., 2022).



# MAUVE (details)

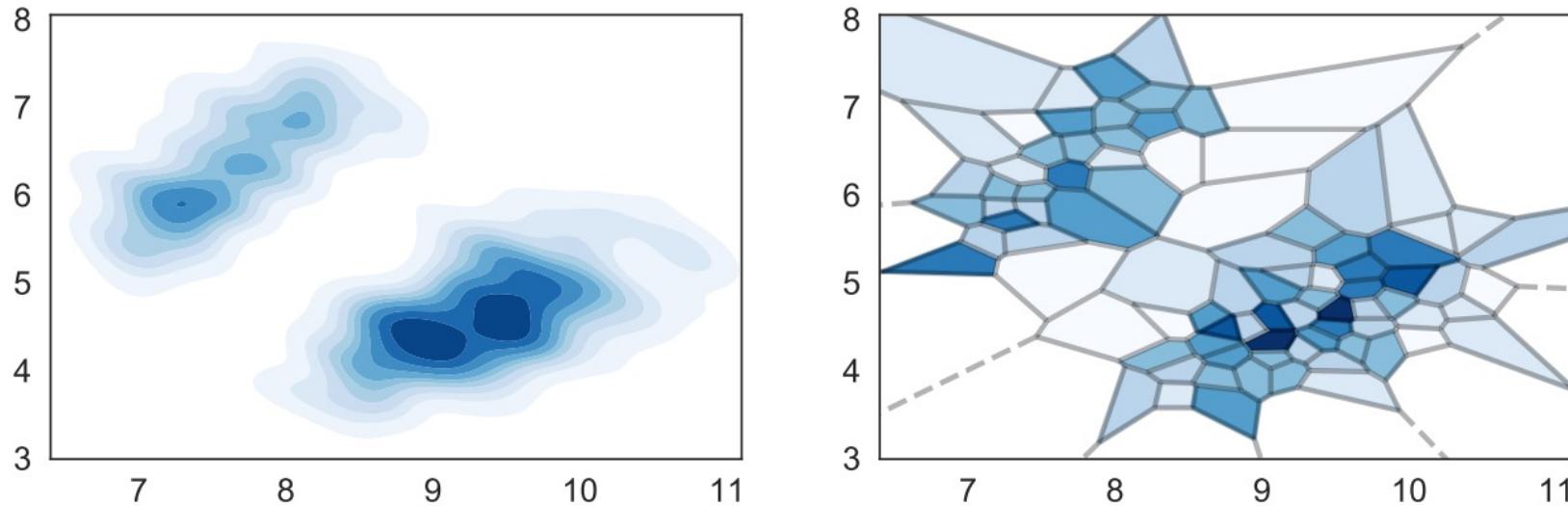


Figure 3: Illustration of the quantization. **Left:** A continuous two-dimensional distribution  $P$ . **Right:** A partitioning of the Euclidean plane  $\mathbb{R}^2$  and the corresponding quantized distribution  $\tilde{P}$ .

# How to evaluate an evaluation metric?

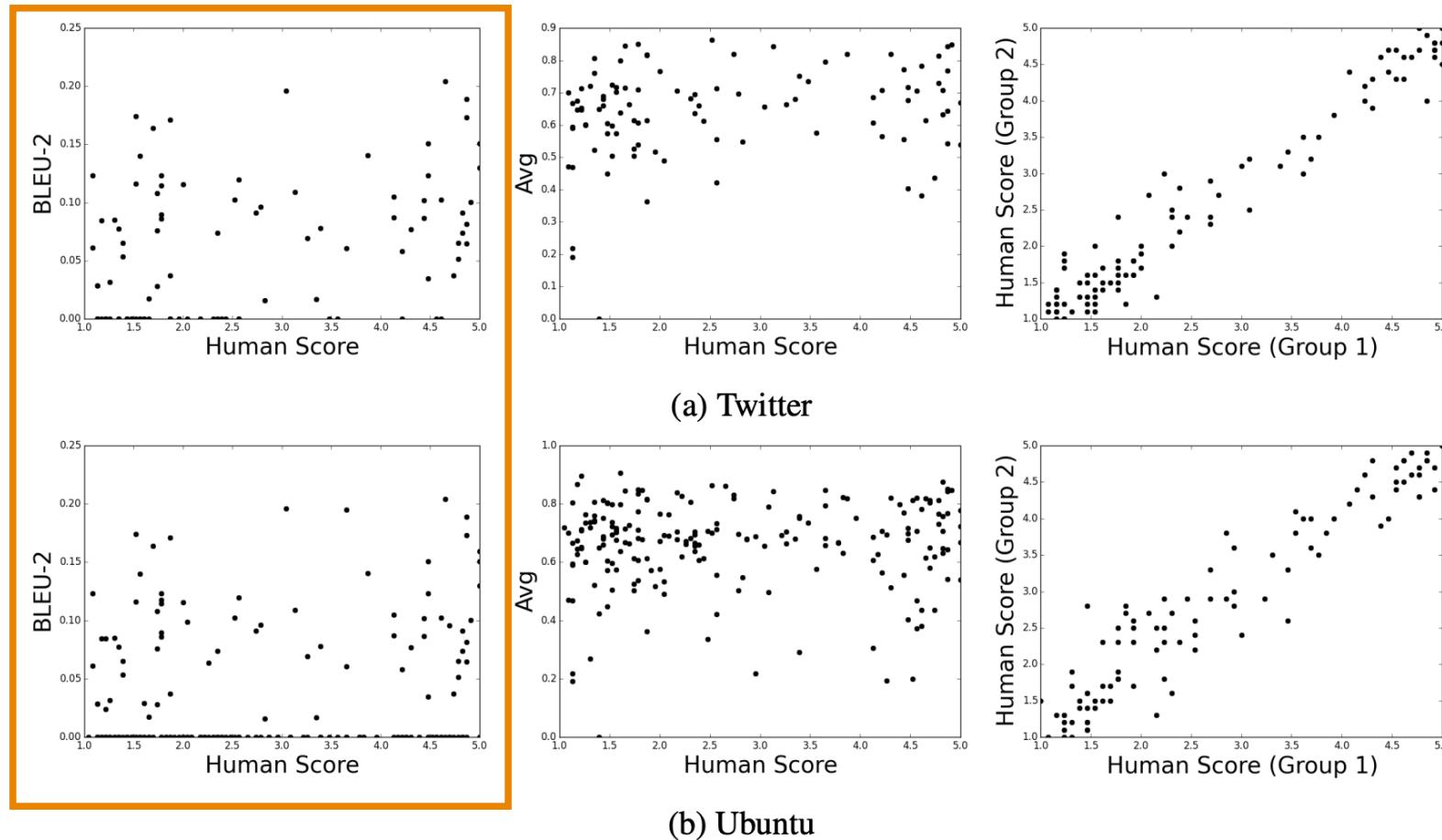
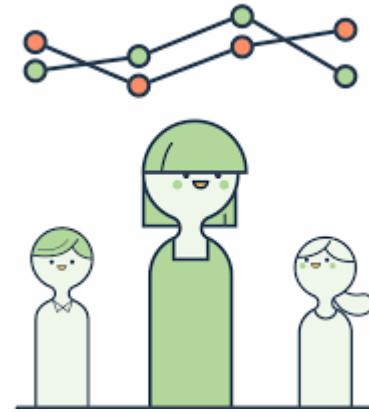


Figure 1: Scatter plots showing the correlation between metrics and human judgements on the Twitter corpus (a) and Ubuntu Dialogue Corpus (b). The plots represent BLEU-2 (left), embedding average (center), and correlation between two randomly selected halves of human respondents (right).

# Human evaluations



- ❑ 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!

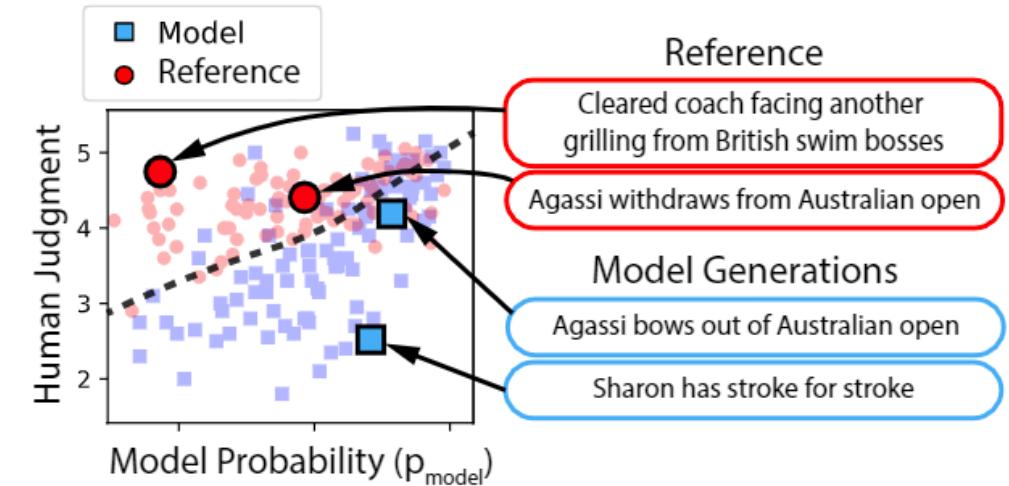
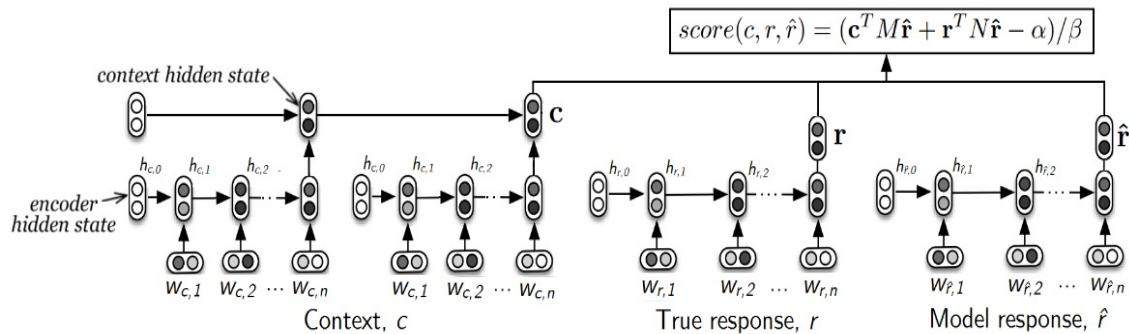
# Human evaluations

- 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
  - typicality
  - redundancy

## Human evaluation: Issues

- Human judgments are regarded as the **gold standard**
- Of course, we know that human eval is **slow** and **expensive**
- Beyond the cost of human eval, it's still far from perfect:
- Humans Evaluation is hard:
  - Results are inconsistent / not reproducible
  - can be illogical
  - misinterpret your question
  - Precision not recall.

# Learning from human feedback



## ADEM:

A learned metric from human judgments for dialog system evaluation in a chatbot setting.

(Lowe et.al., 2017)

## HUSE:

Human Unified with Statistical Evaluation (HUSE), determines the similarity of the output distribution and a human reference distribution.

(Hashimoto et.al. 2019)

## Evaluation: Takeaways

- ❑ *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!**