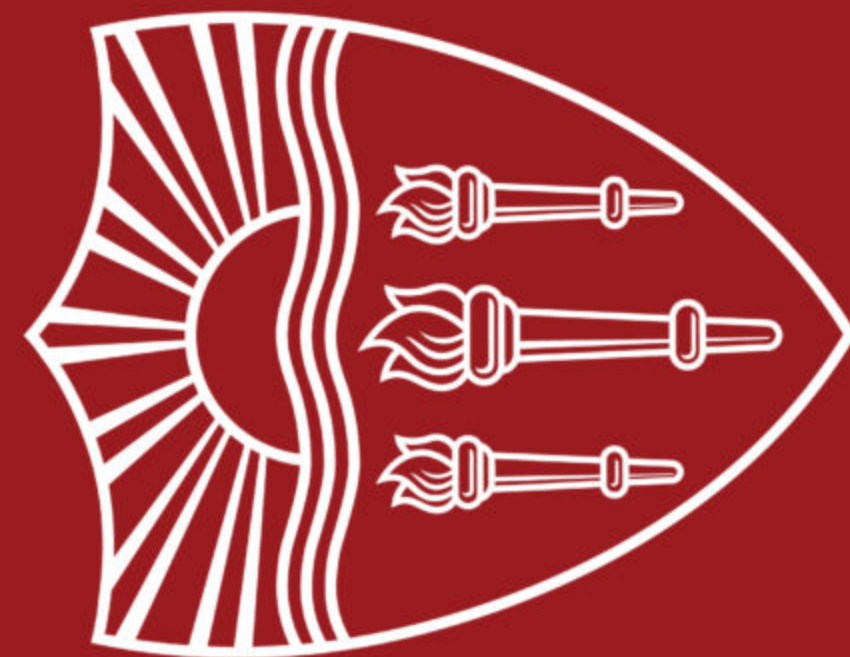


CSCS



Lecture 16: Tokenization and Language Generation

Instructor: Swabha Swayamdipta
USC CSCI 544 Applied NLP
Oct 22, Fall 2024



Announcements

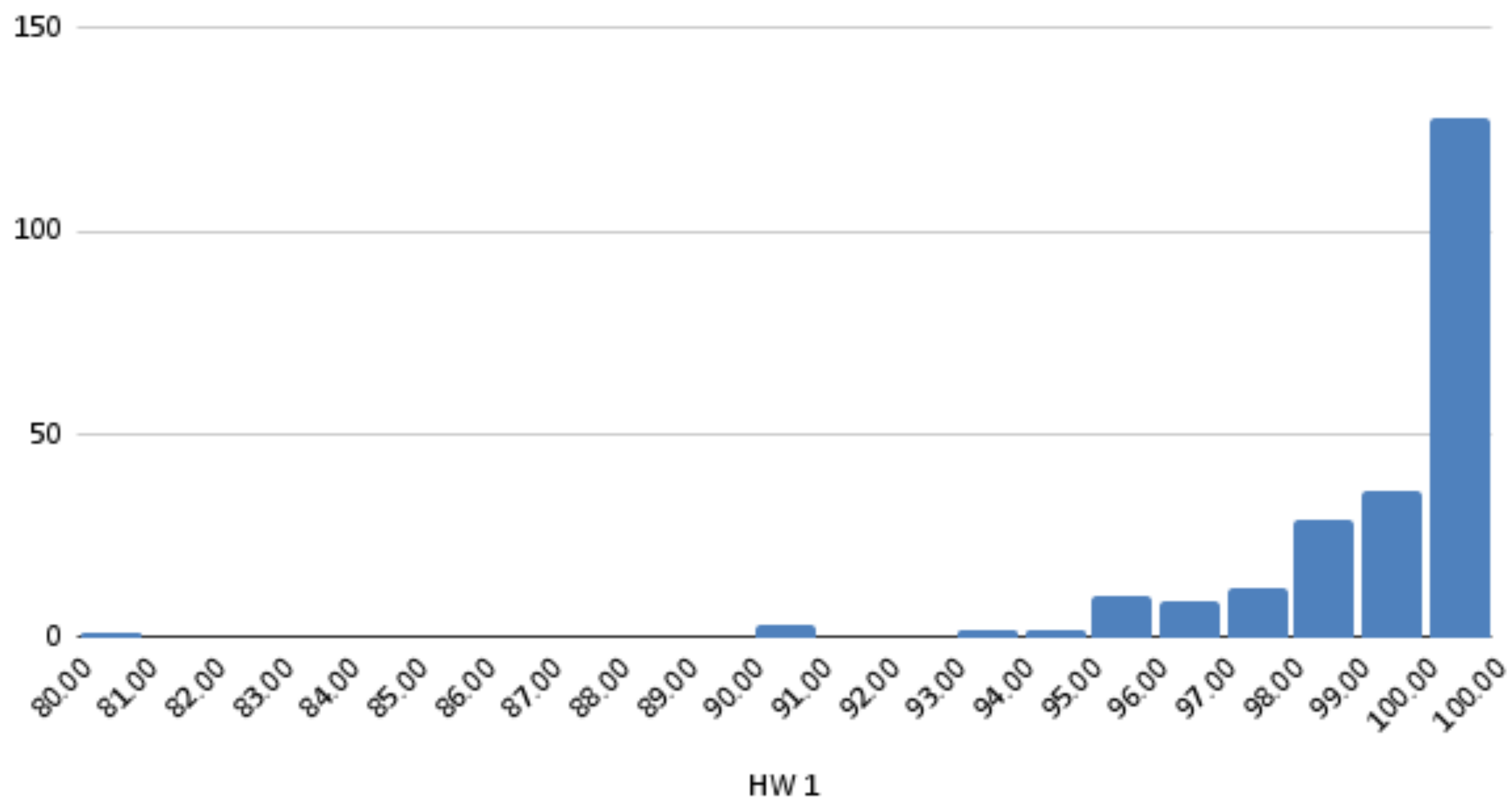
- Today, 10/22 - HW3
- Fri, 10/25 - Project Progress Report
 - What are we expecting? See class website: <https://swabhs.com/f24-csci544-appliednlp/details/project/>
 - once again describe the project's goals
 - contain all details on the dataset (your dataset should mostly be collected by this time),
 - contain some initial results (think of this as a motivating results), and
 - must outline a concrete plan of what will be done before the final report.
- Tue, 10/29 - Quiz 4
- Thu, 10/31 onwards: Paper presentations and project presentations
 - Also two remaining lectures on 10/31 and 11/5
 - Guest lecture by Prof. Willie Neiswanger on 11/14

Announcements: Paper Presentations

- Paper Presentations Format:
 - Every team has been assigned a paper by their TA
 - See your presentation schedule: https://docs.google.com/spreadsheets/d/13cgBXINq3679VCju9wmLcE5_pySP2HxpczNNje6xqLI/edit?gid=1378308384#gid=1378308384
 - All teammates prepare to present—on their own—the assigned paper
 - We will announce the actual presenter a couple of hours before class
 - All teammates should be prepared to answer questions from the audience about the paper
 - Total time for presentation + questions: 5 mins / team
 - Remaining time: lecture
 - Slides encouraged - we will ask you to send us your Google slides in advance

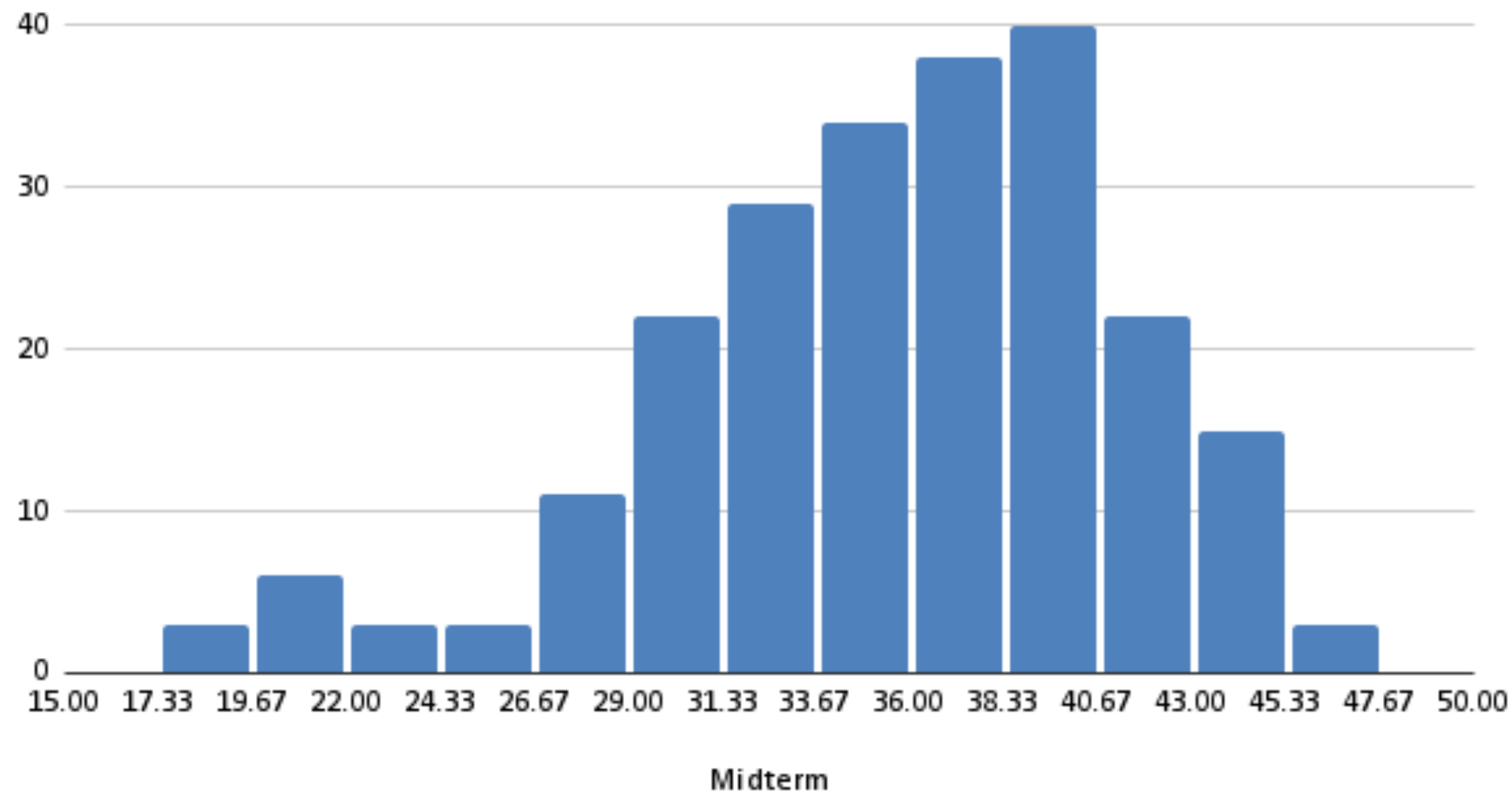
HW1 Grades

Histogram of HW 1

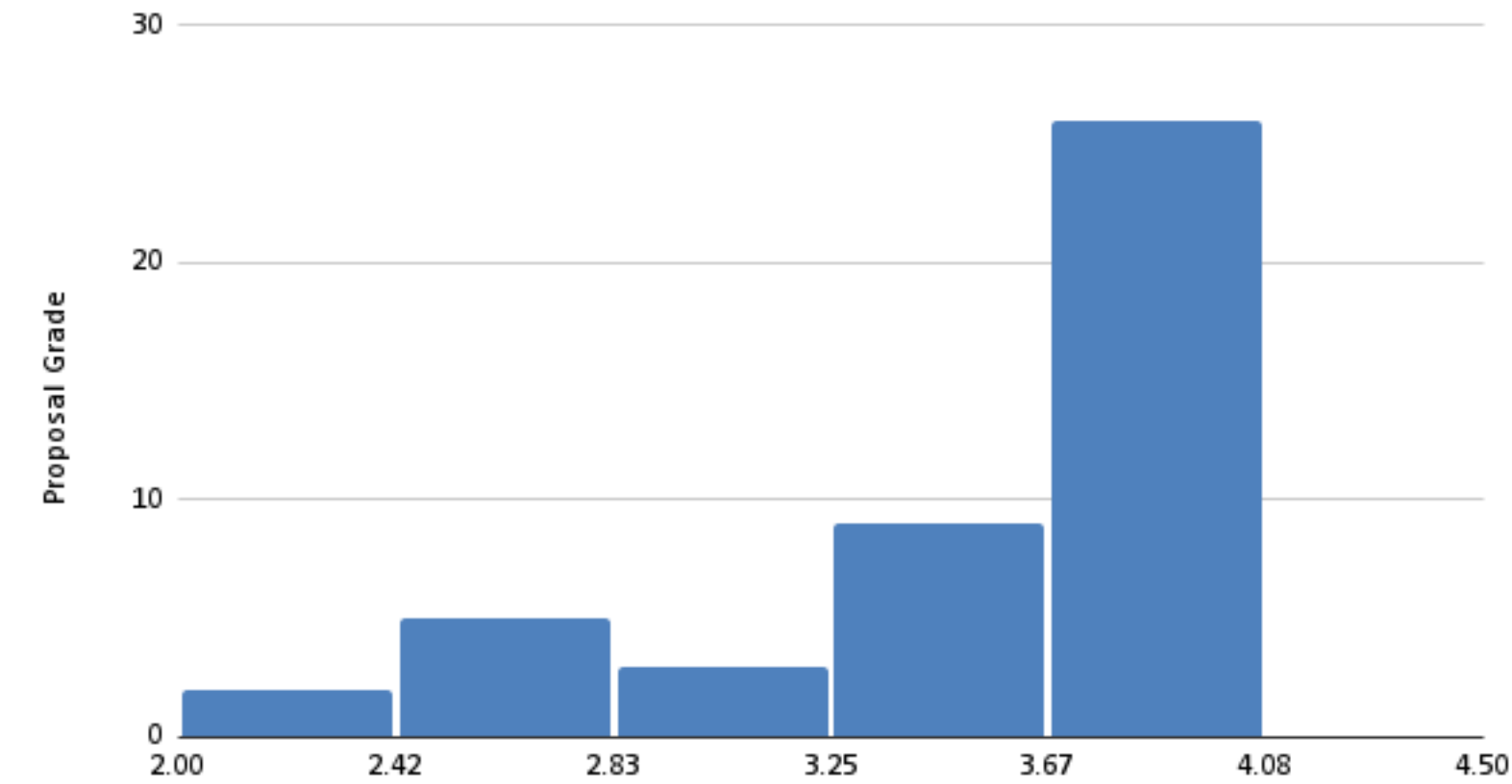


Class Grades

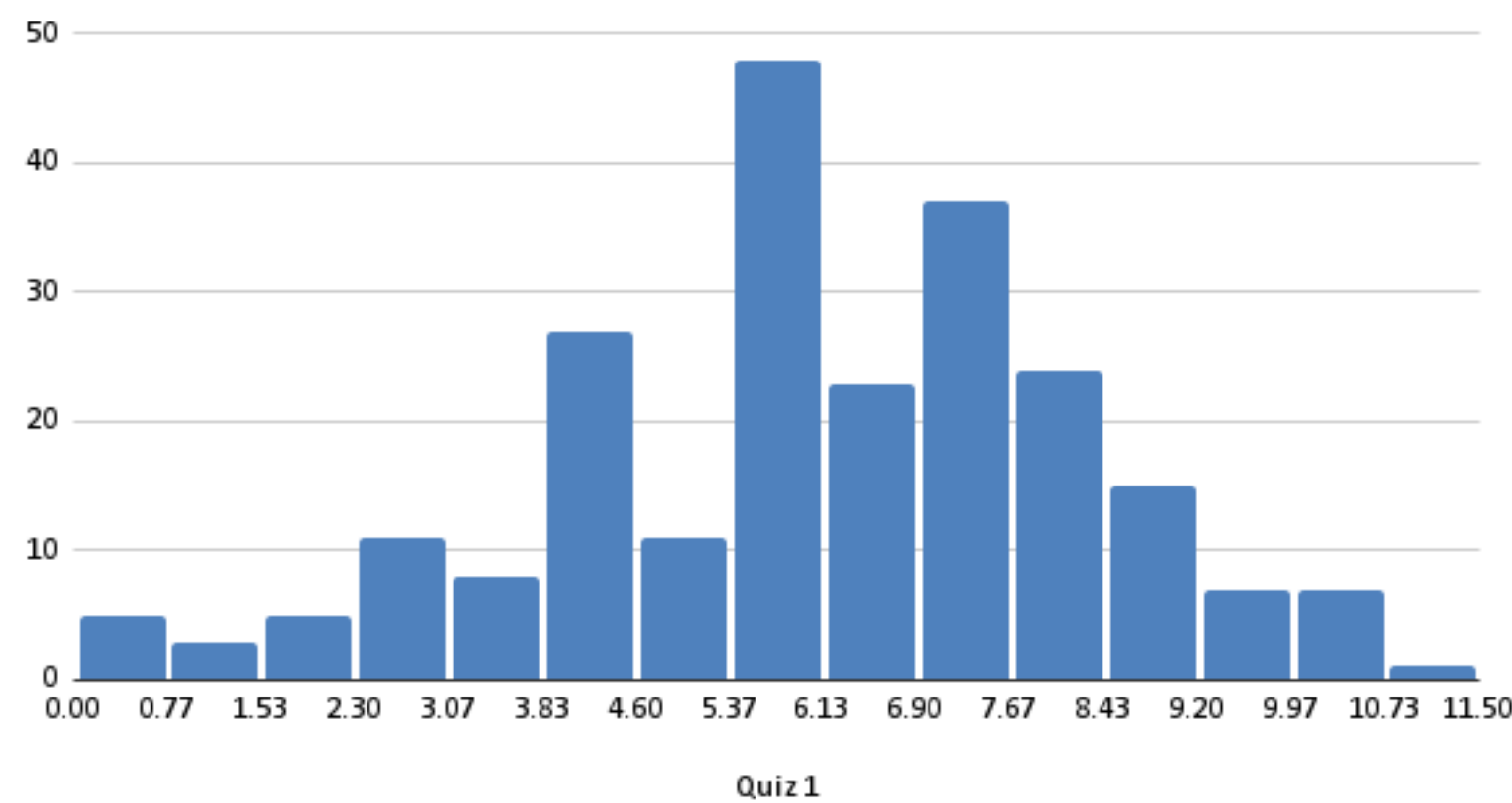
Histogram of Midterm



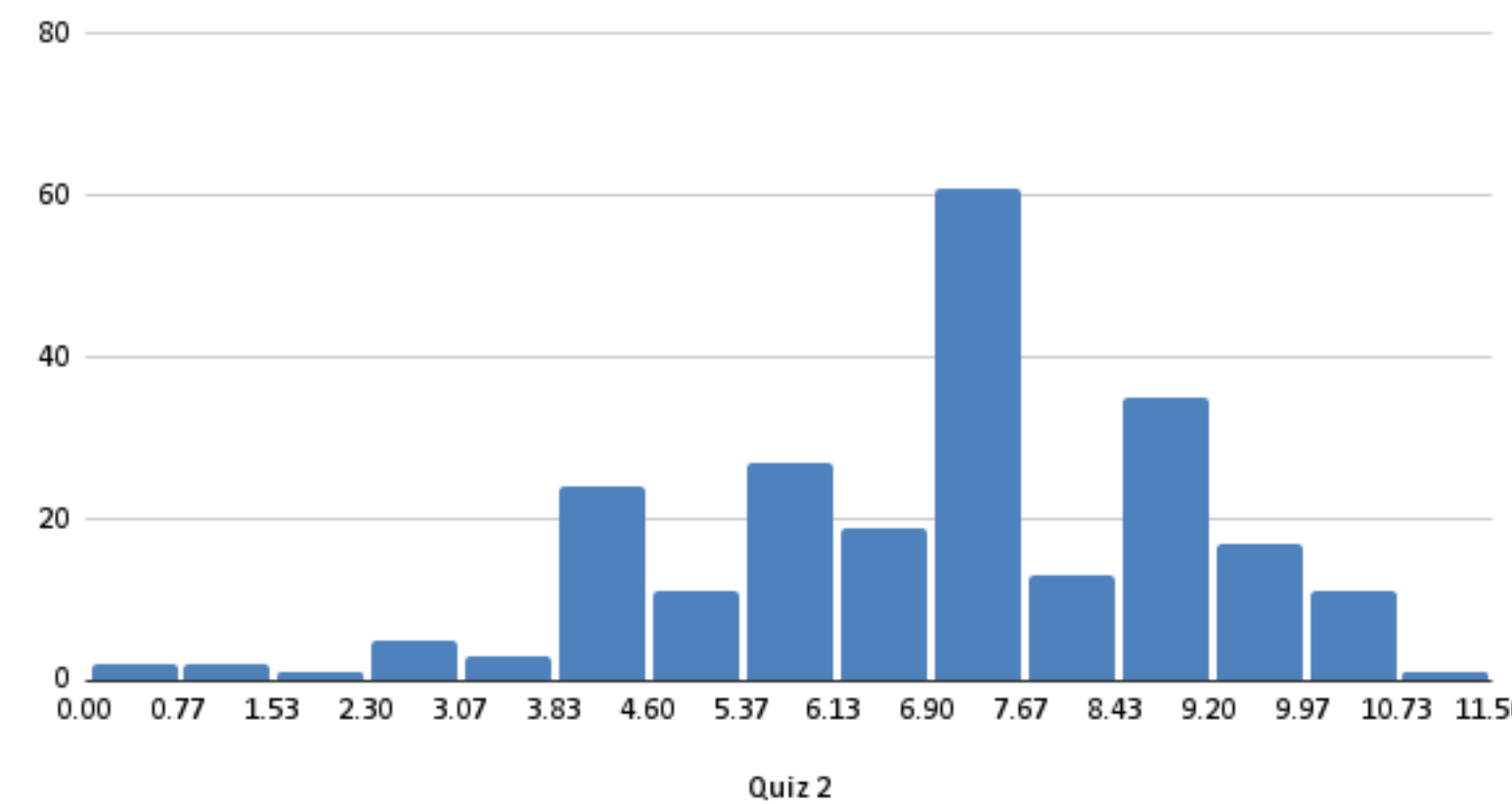
Project Proposal Grades



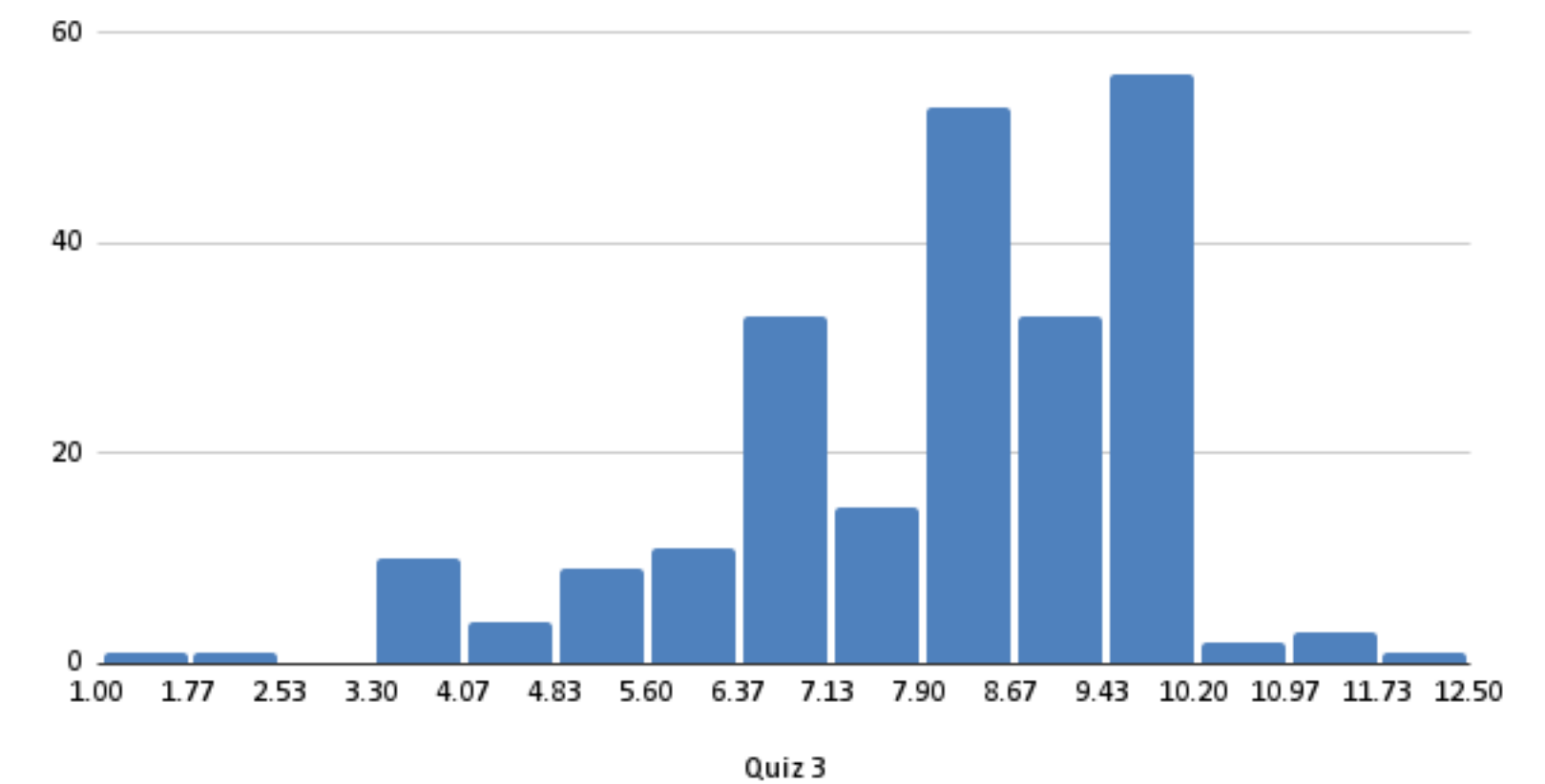
Histogram of Quiz 1



Histogram of Quiz 2



Histogram of Quiz 3



Lecture Outline

- Announcements
- Recap: The pre-training and fine-tuning paradigm
 - Pre-training Decoder-Only Models
 - Pre-training Encoder-Only Models
- Pre-training Encoder-Decoder Models
- Tokenization
- Natural Language Generation

Recap: Pre-training and Fine-tuning

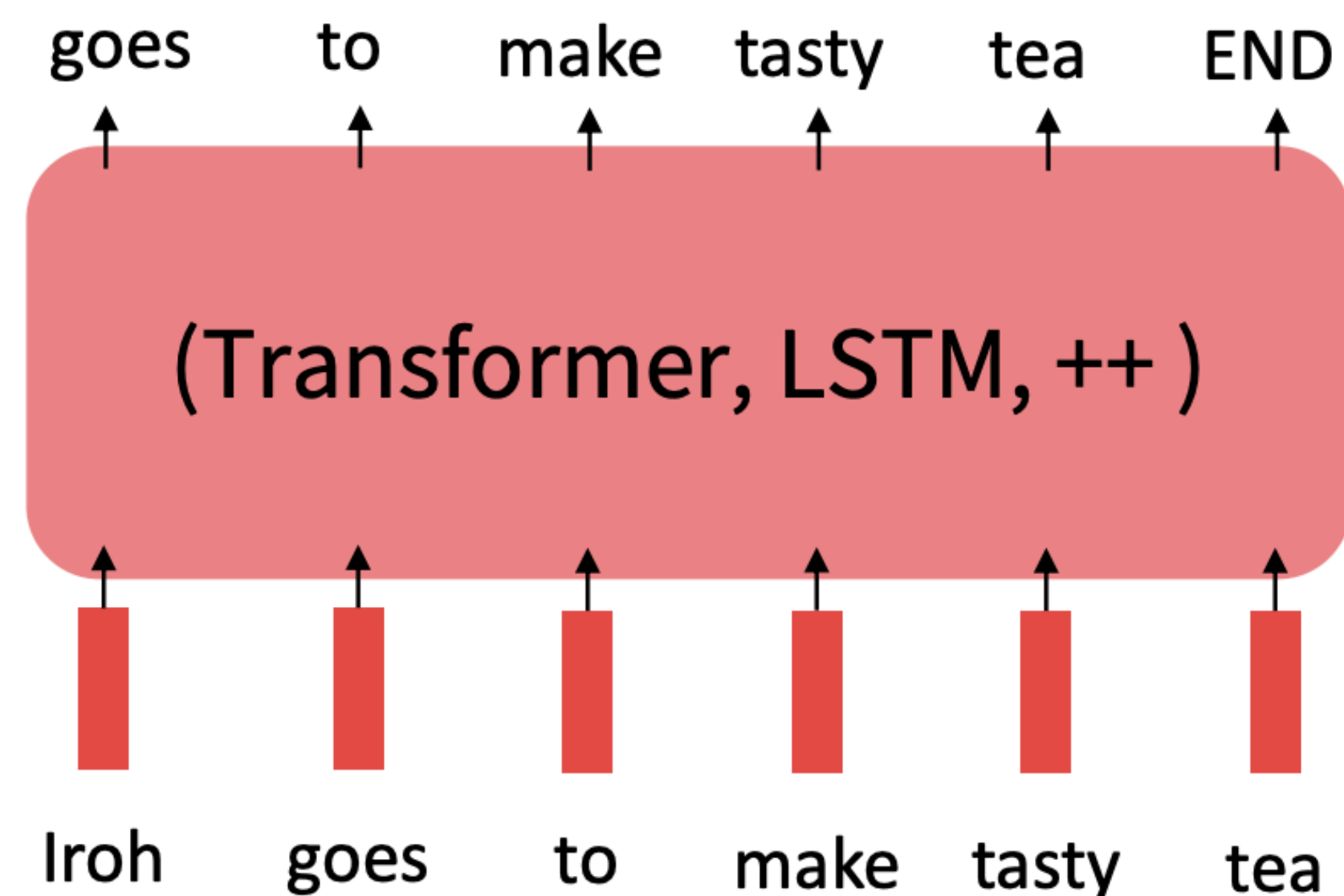
The Pretraining / Finetuning Paradigm

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

Key idea: "Pretrain once, finetune many times."

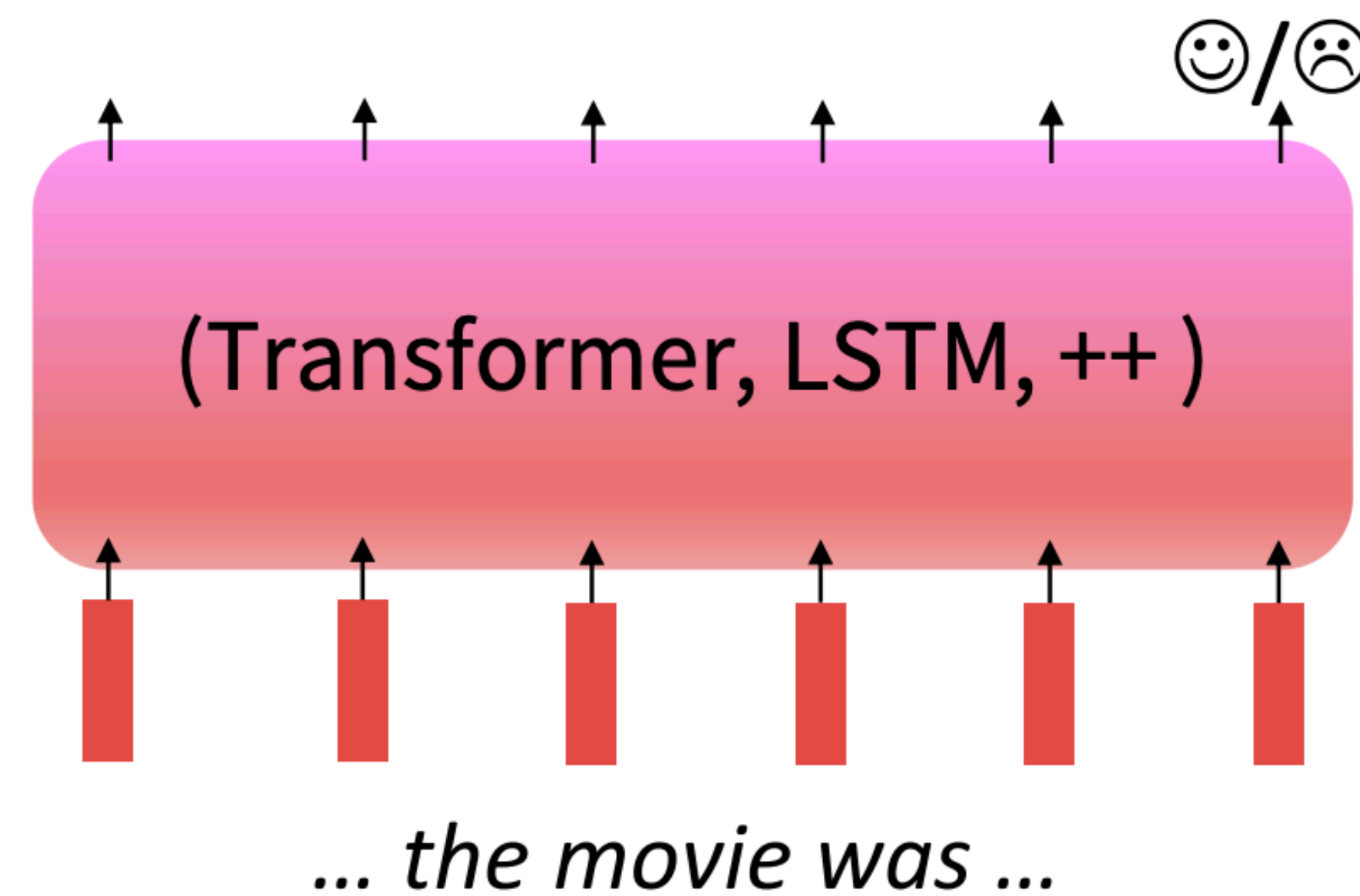
Step 1: Pretrain (on language corpora)

Lots of text; learn general things!



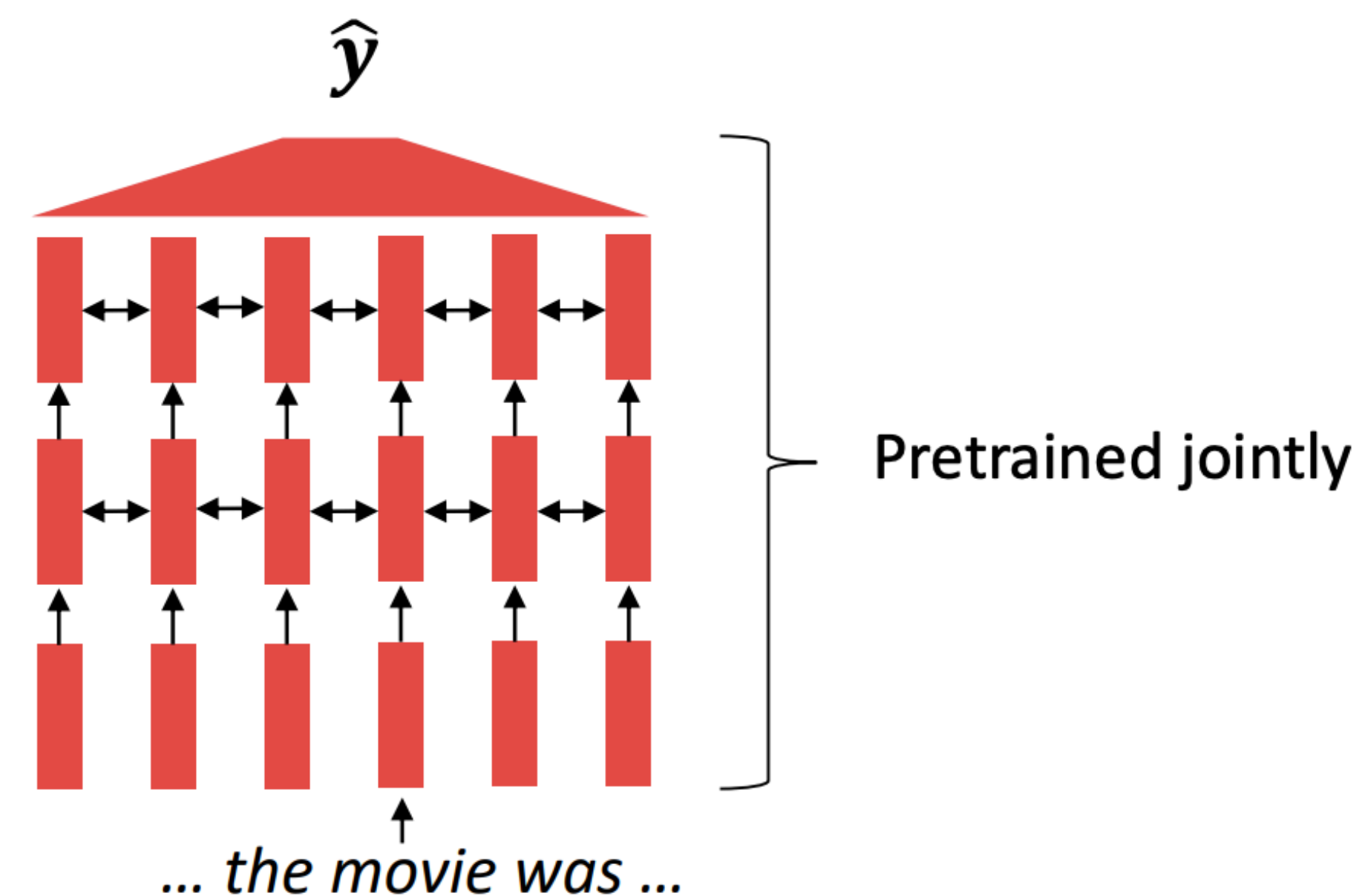
Step 2: Finetune (on your task data)

Not many labels; adapt to the task!



Pretraining Entire Models

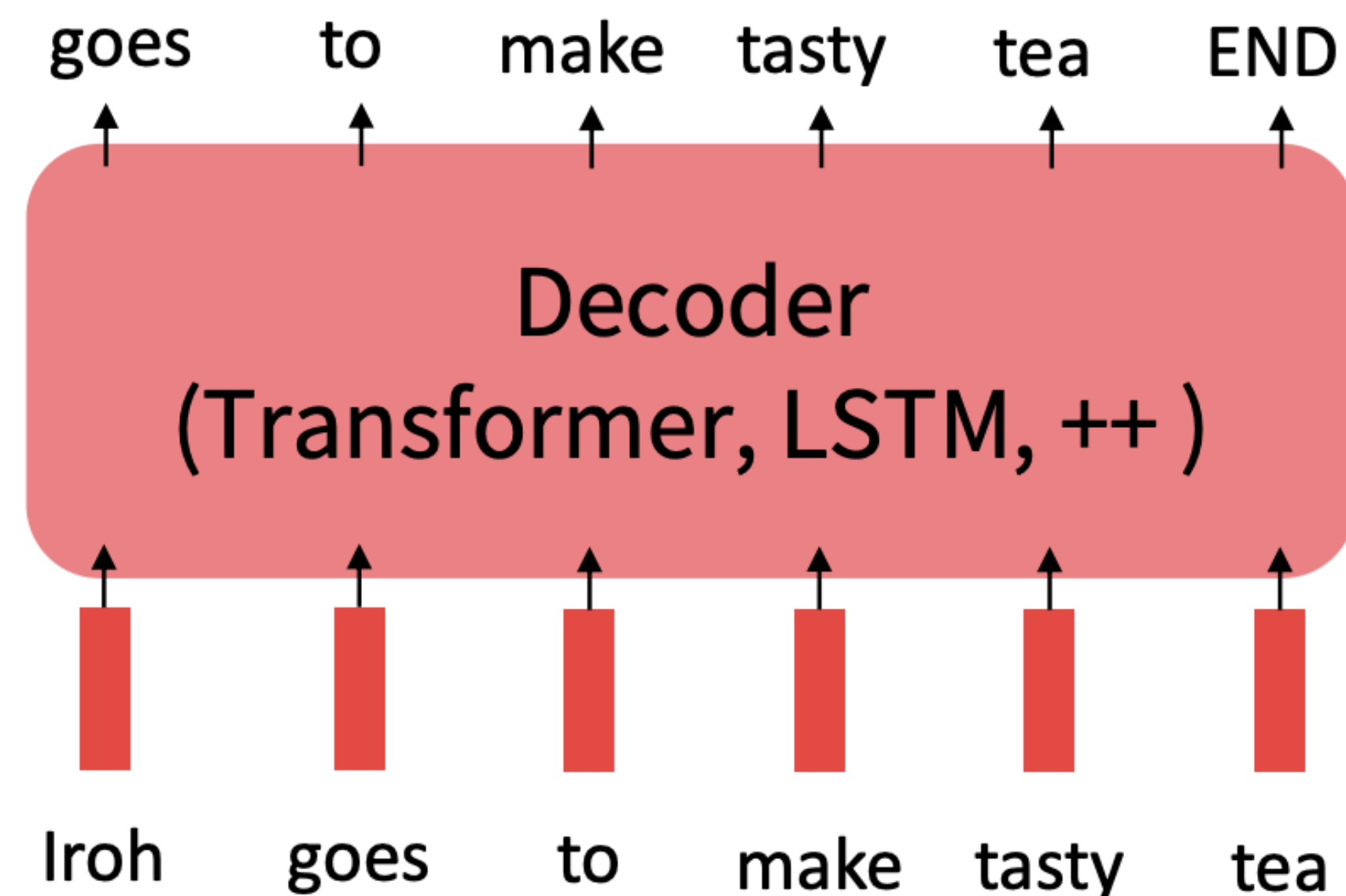
- In modern NLP:
 - All (or almost all) parameters in NLP networks are initialized via pretraining.
 - This has been exceptionally effective at building strong:
 - representations of language
 - parameter initializations for strong NLP models.
 - probability distributions over language that we can sample from



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

Pretraining: Language Models

- Recall the language modeling task:
 - Model $p_{\theta}(w_t | w_{1:t-1})$, the probability distribution over words given their past contexts.
 - There's lots of data for this! (In English.)
- Pretraining through language modeling:
 - Train a neural network to perform language modeling on a large amount of text.
 - Save the network parameters.



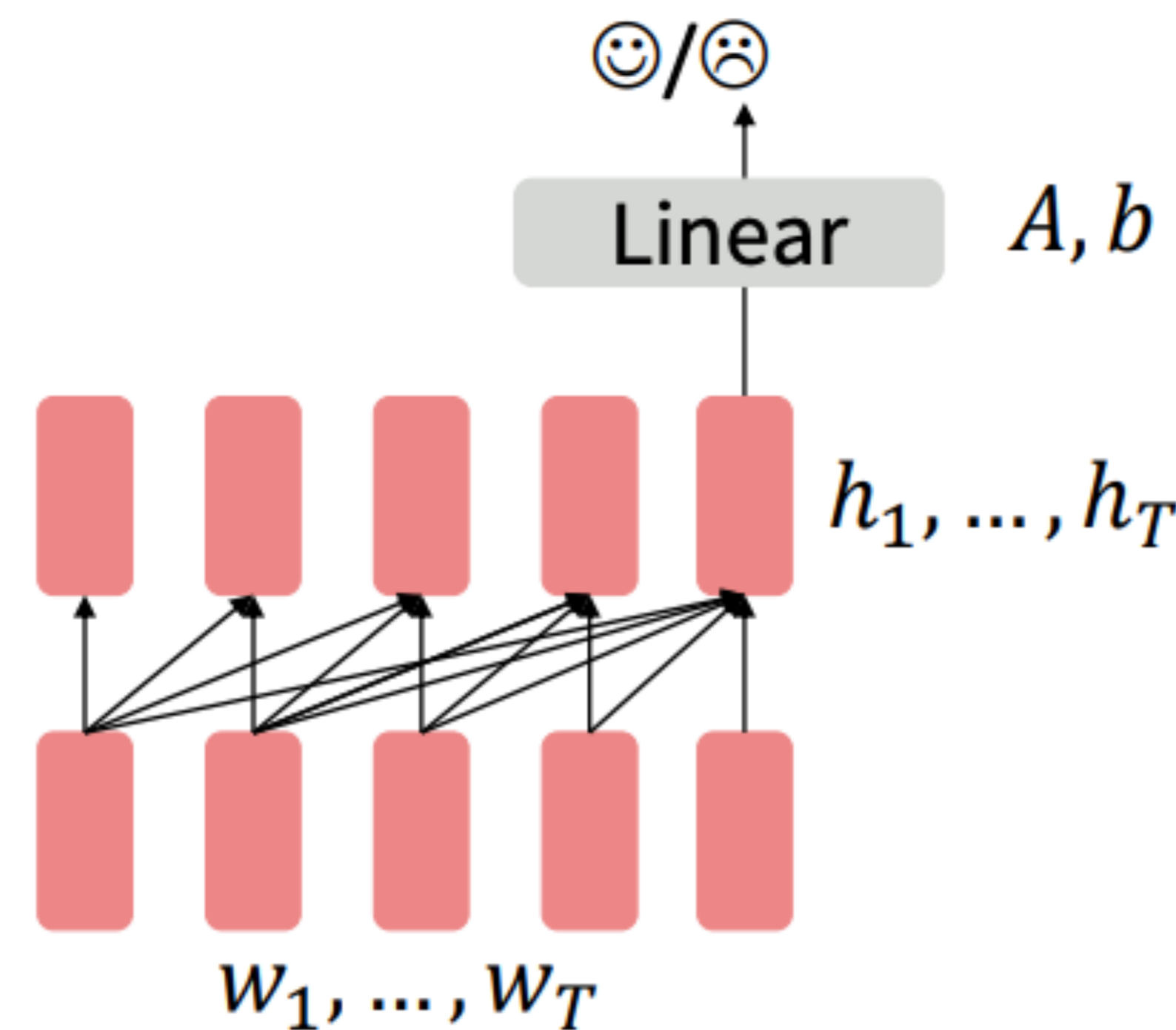
Semi-supervised Sequence Learning

Andrew M. Dai
Google Inc.
adai@google.com

Quoc V. Le
Google Inc.
qvl@google.com

Pretraining Decoders: Classifiers

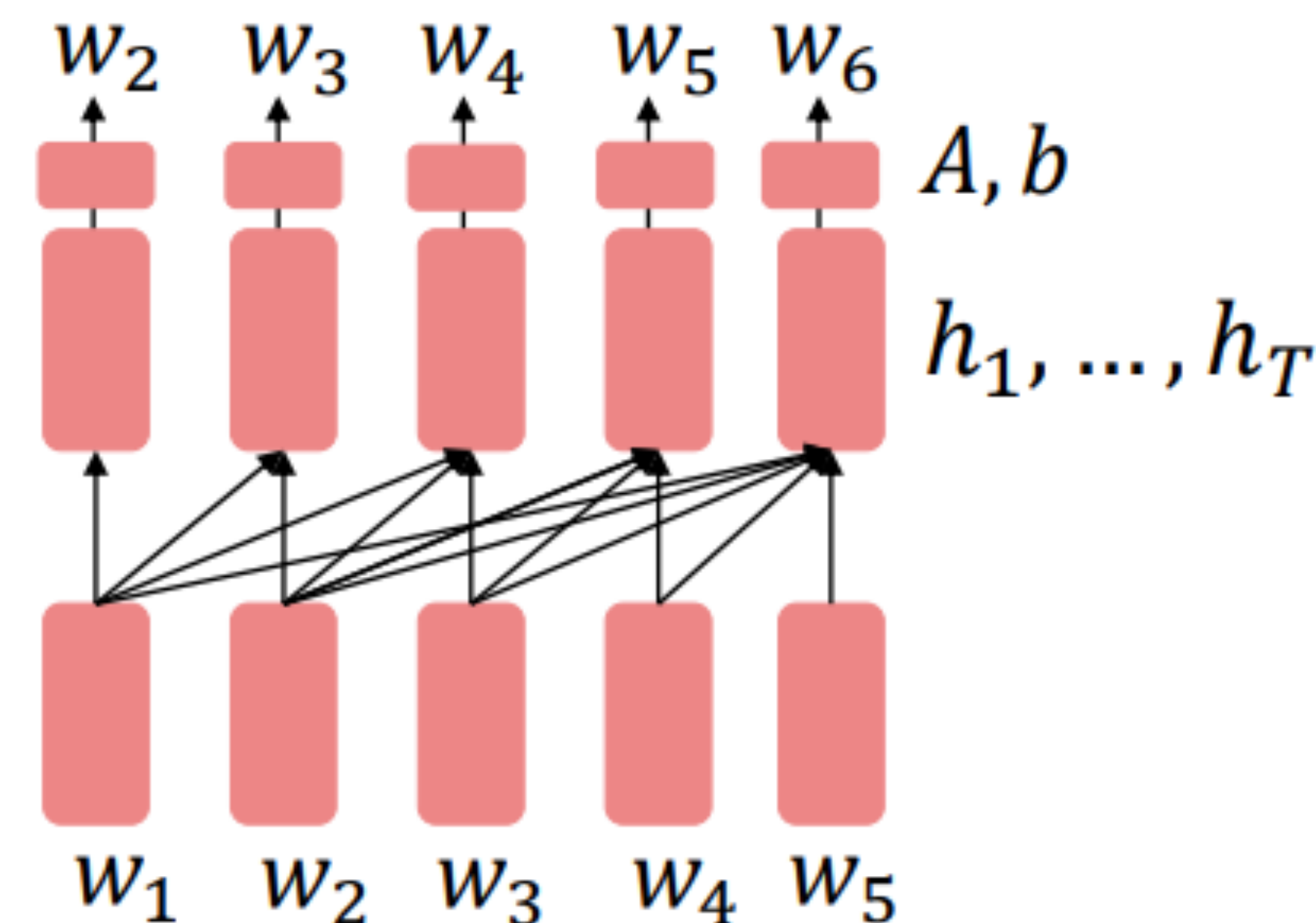
- When using language model pretrained decoders, we can ignore that they were trained to model $p_{\theta}(w_t | w_{1:t-1})$
- We can finetune them by training a classifier on the last word's hidden state
 - $h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$
 - $y \approx Ah_T + b$
 - Where A and b are randomly initialized and specified by the downstream task.
- Gradients backpropagate through the whole network.



The linear layer hasn't been pretrained and must be learned from scratch.

Pretraining Decoders: Generators

- More natural: pretrain decoders as language models and then use them as generators, finetuning their $p_{\theta}(w_t | w_{1:t-1})$
 - $h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$
- $w_t \approx Ah_{t-1} + b$
- Where A, b were pretrained in the language model!
- 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)



The linear layer has been pretrained

Pretraining Encoders: Bidirectional Context

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

Universal Studios Theme Park is located in _____, California

Problem: Input
Reconstruction

'Cause darling i'm a _____ dressed like a daydream

Bidirectional context is important to reconstruct the input!

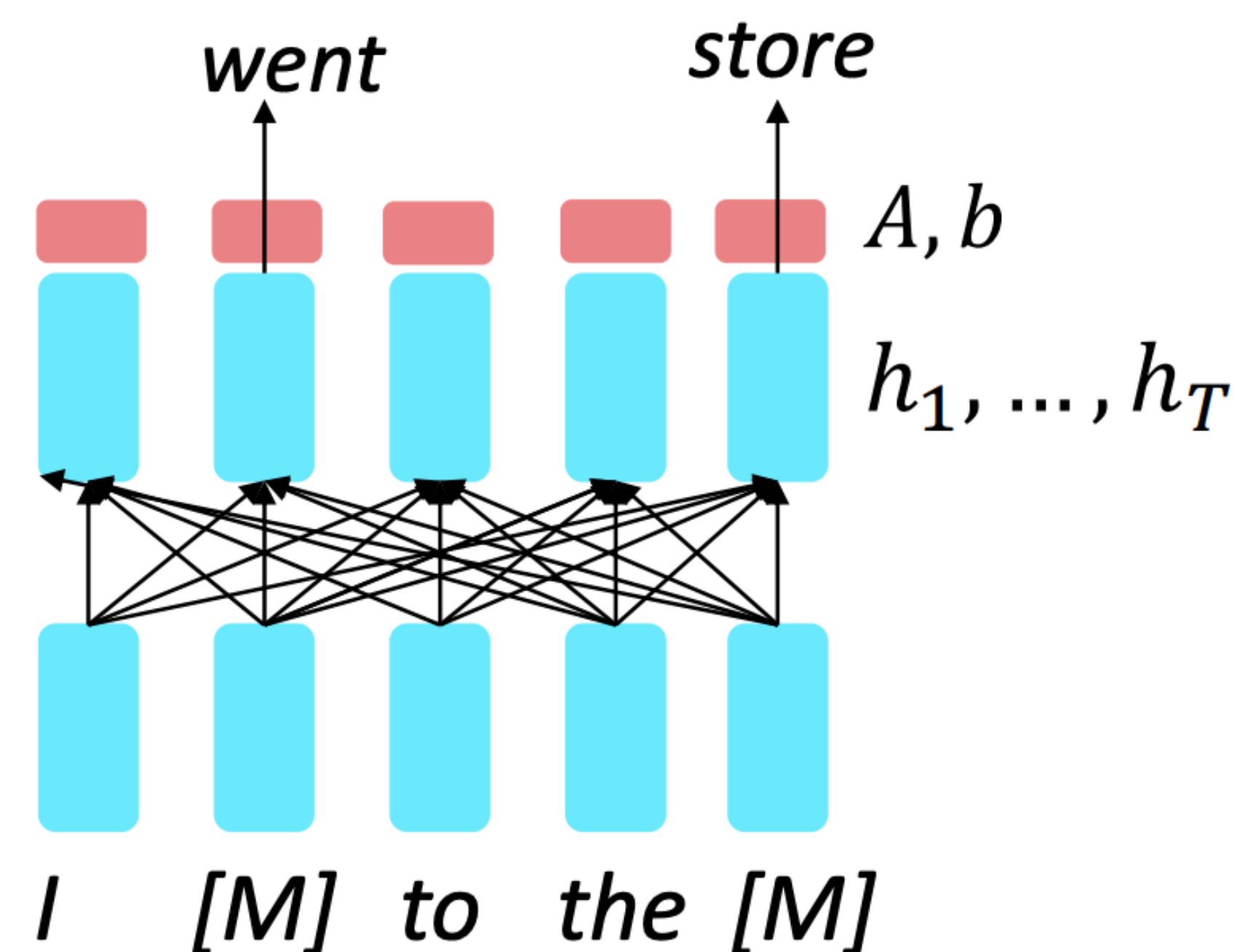
Pretraining Encoders: Objective

- 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.

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

- $y_i \approx Ah_i + b$

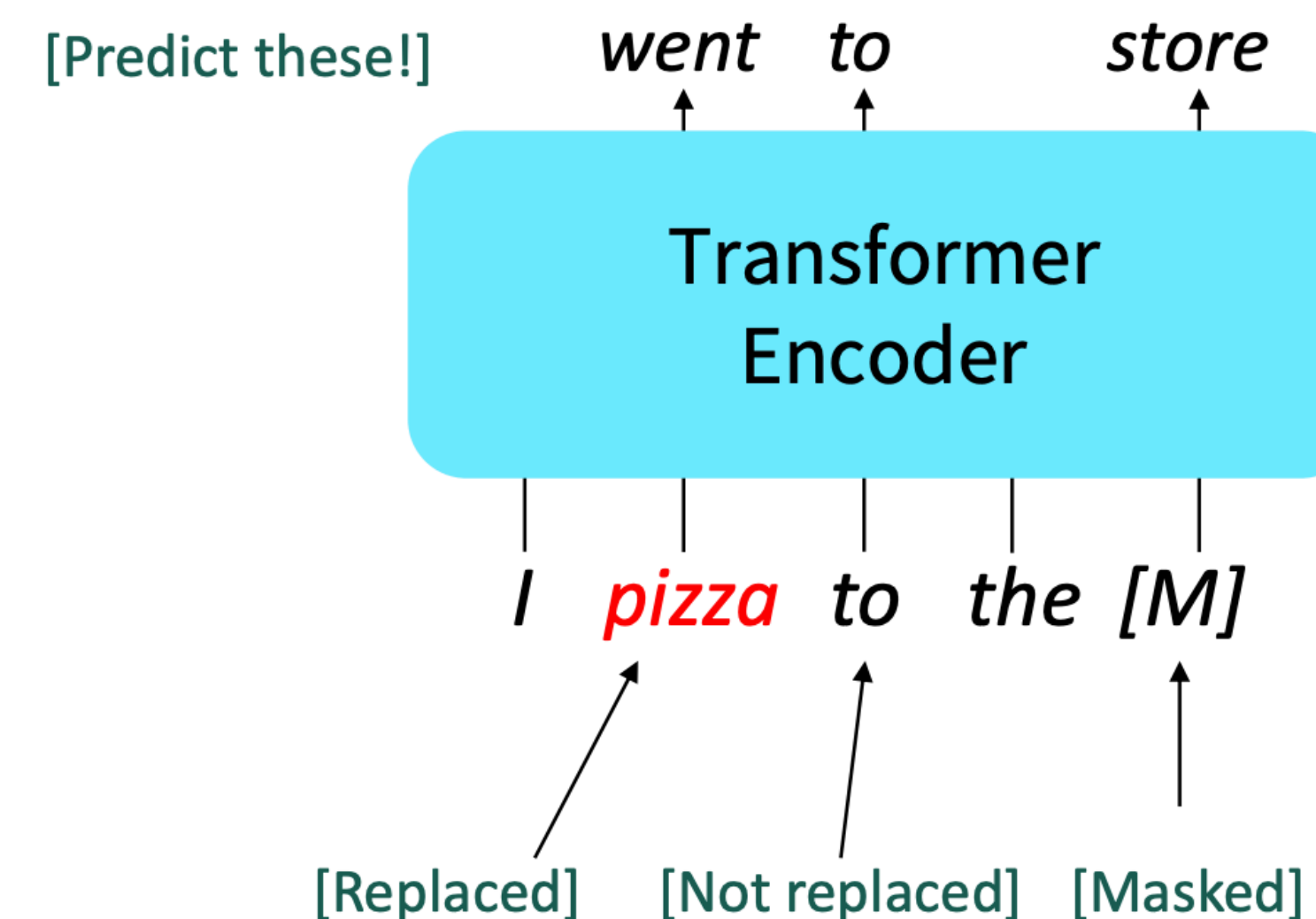
- Only add loss terms from words that are "masked out."
- If \tilde{x} is the masked version of x , we're learning $p_\theta(\tilde{x} | x)$.
- Called Masked LM
- Special type of language modeling



BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the “Masked LM” objective and released BERT, a Transformer, pretrained to:

- 15% of the input tokens in a training sequence are sampled for learning, these are to be predicted by the model
- Of these
 - 80% are replaced with [MASK]
 - 10% are replaced with randomly selected tokens,
 - Remaining 10% are left unchanged

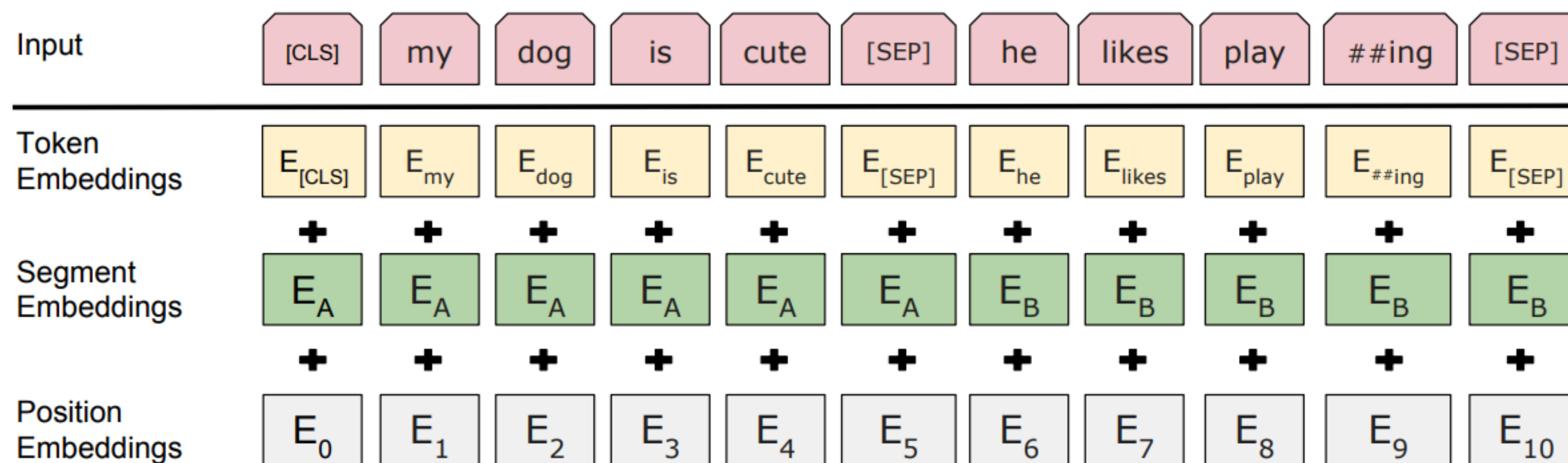


Why?

Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)

BERT: Bidirectional Encoder Representations from Transformers

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



- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
 - [CLS] and [SEP] tokens
 - [SEP] is used for next sentence prediction - do these sentences follow each other?
 - [CLS] for text classification / connection to fine-tuning

BERT: Extensions

- 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
- A lot of BERT variants that used the BERT formula
 - ALBERT: BERT with parameter-reduction techniques
 - DistilBERT:
 - DeBERTa: Decoding-enhanced BERT with disentangled attention
 - FlauBERT: BERT for French
 - XLNet: Multilingual BERT
 - Etc.
- **BERTology**: How and why BERT worked so well



BERT: Overview

- [SEP]: Later work has argued this “next sentence prediction” is not necessary
- In general, more compute, more data can improve pretraining even when not changing the underlying Transformer encoder
- Results in contextual embeddings
- Key Limitation:
 - Cannot be used for generation
 - No pretraining encoders can be used for autoregressive generation very naturally
 - There are clunky ways in which you could try...but not a natural fit
 - For this, we need to have a decoder!

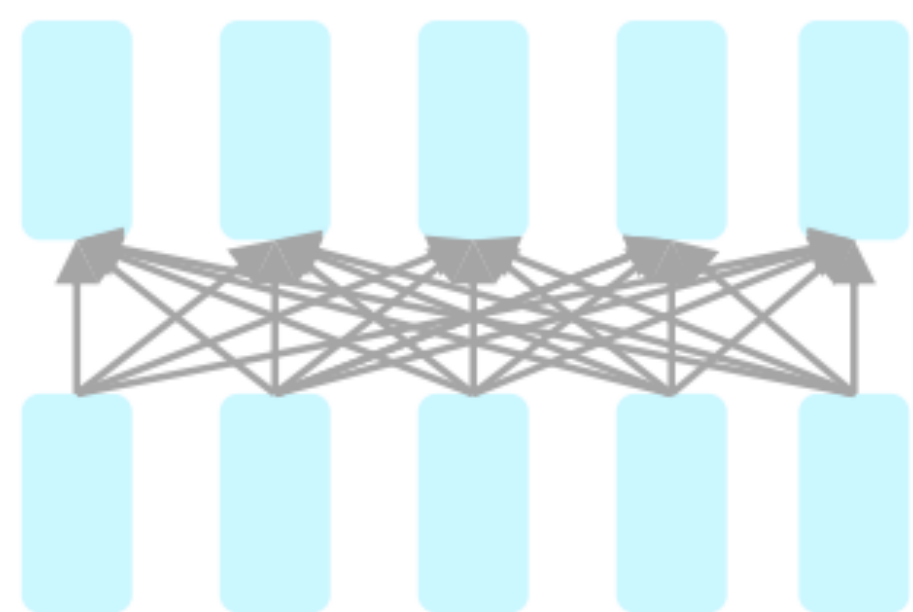


Lecture Outline

- Announcements
- Recap: The pre-training and fine-tuning paradigm
 - Pre-training Decoder-Only Models
 - Pre-training Encoder-Only Models
- Pre-training Encoder-Decoder Models
- Tokenization
- Natural Language Generation

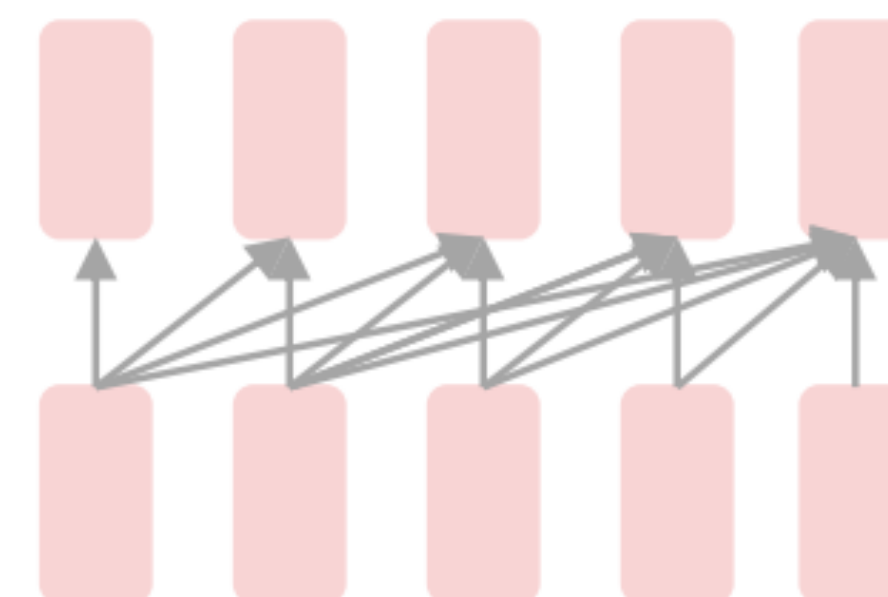
Pre-training Encoder-Decoder Models

Pretraining for three types of architectures



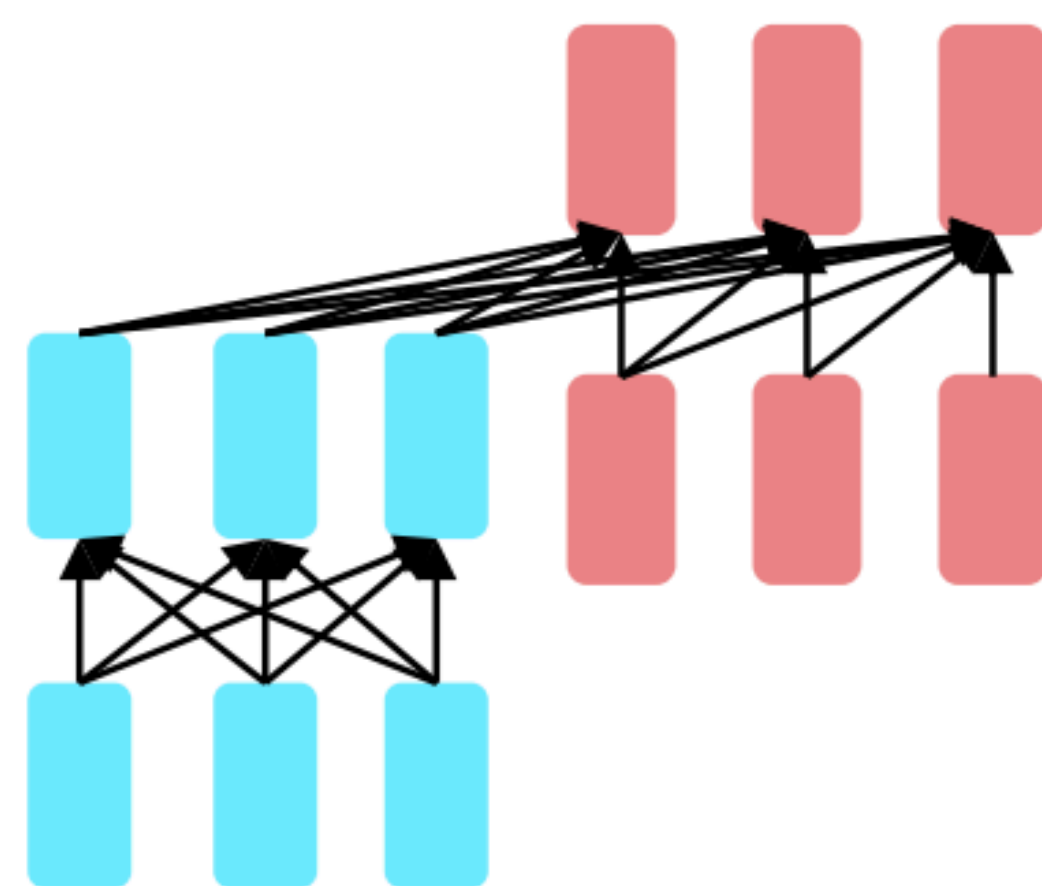
Encoders

Bidirectional Context



Decoders

Language Models



**Encoder-
Decoders**

Sequence-to-sequence

Pretraining Encoder-Decoder Models

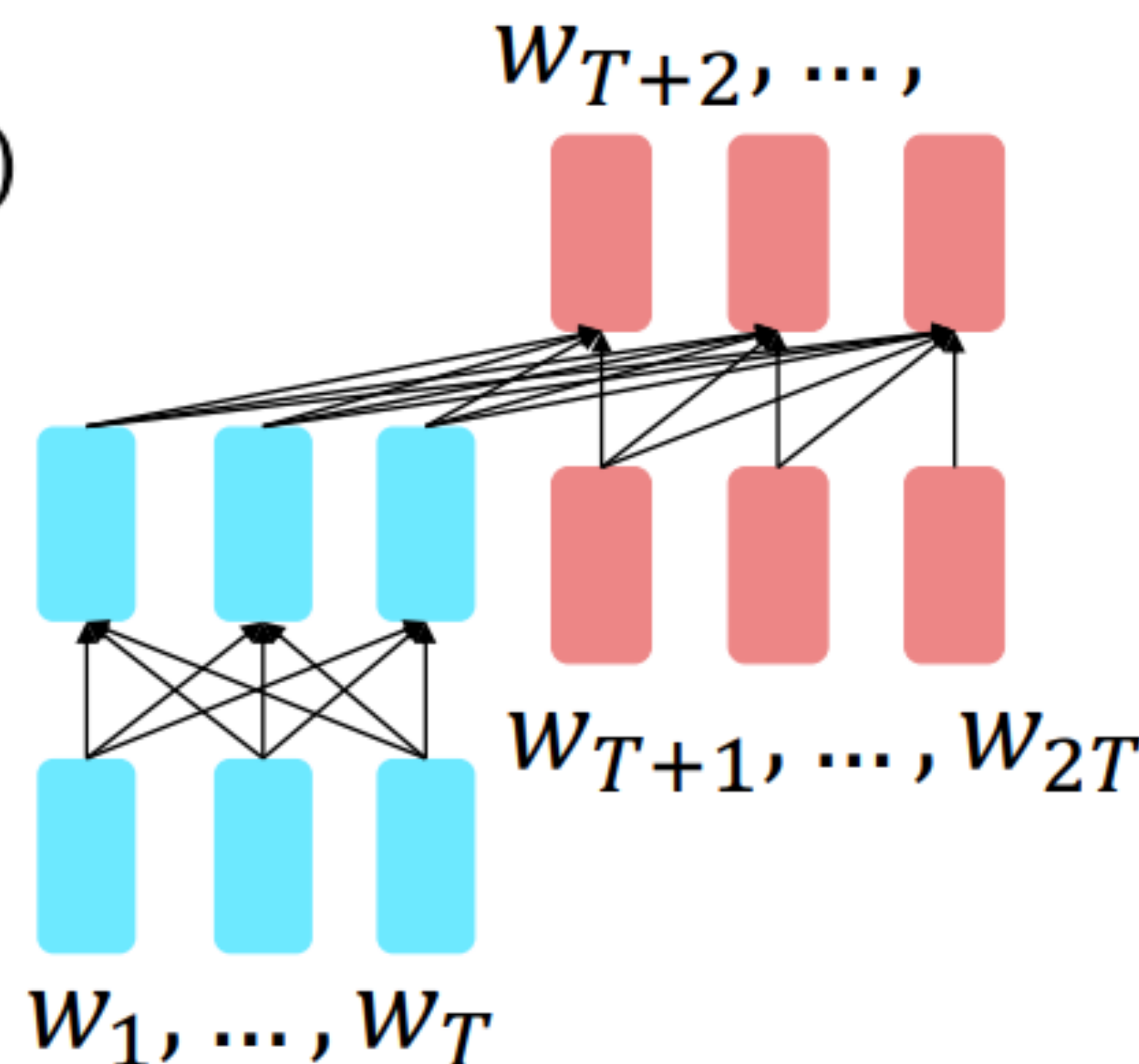
- For encoder-decoders, we could do something like language modeling, but where a prefix of every input is provided to the encoder and is not predicted.

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

$$h_{T+1}, \dots, h_{2T} = \text{Decoder}(w_1, \dots, w_T, h_1, \dots, h_T)$$

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

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



T5: A Pretrained Encoder-Decoder Model

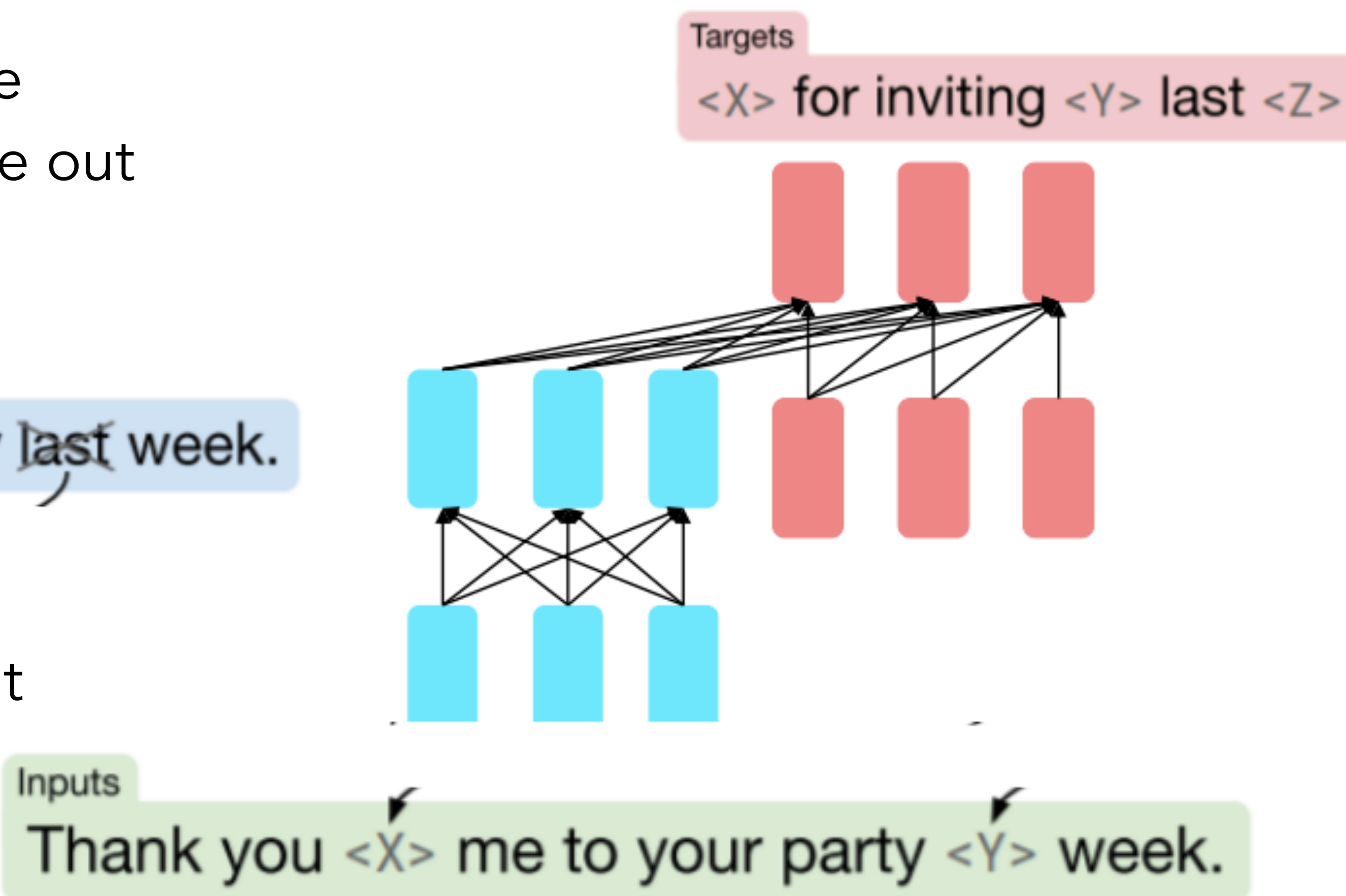
- Raffel et al., 2018 built T5, which uses as a span corruption pretraining objective

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

Original text

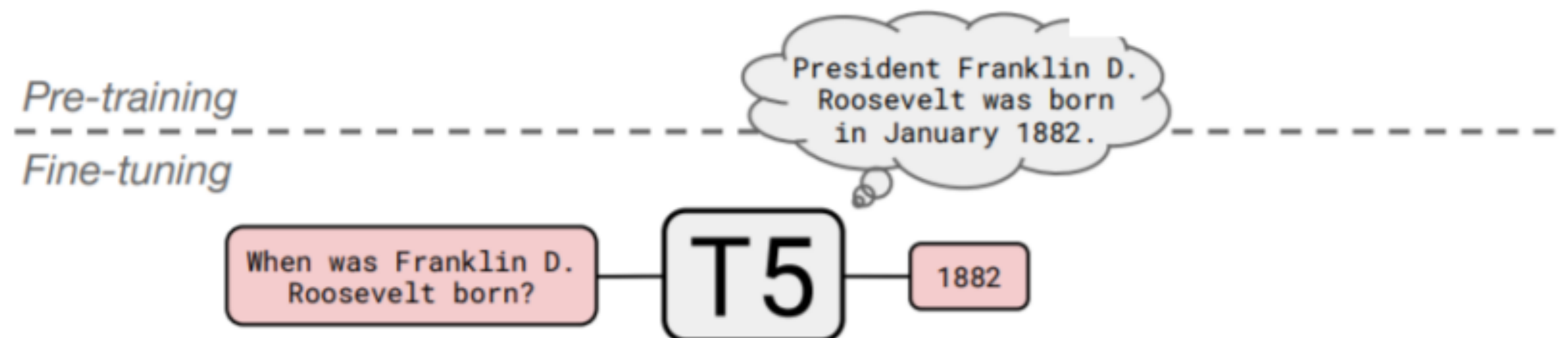
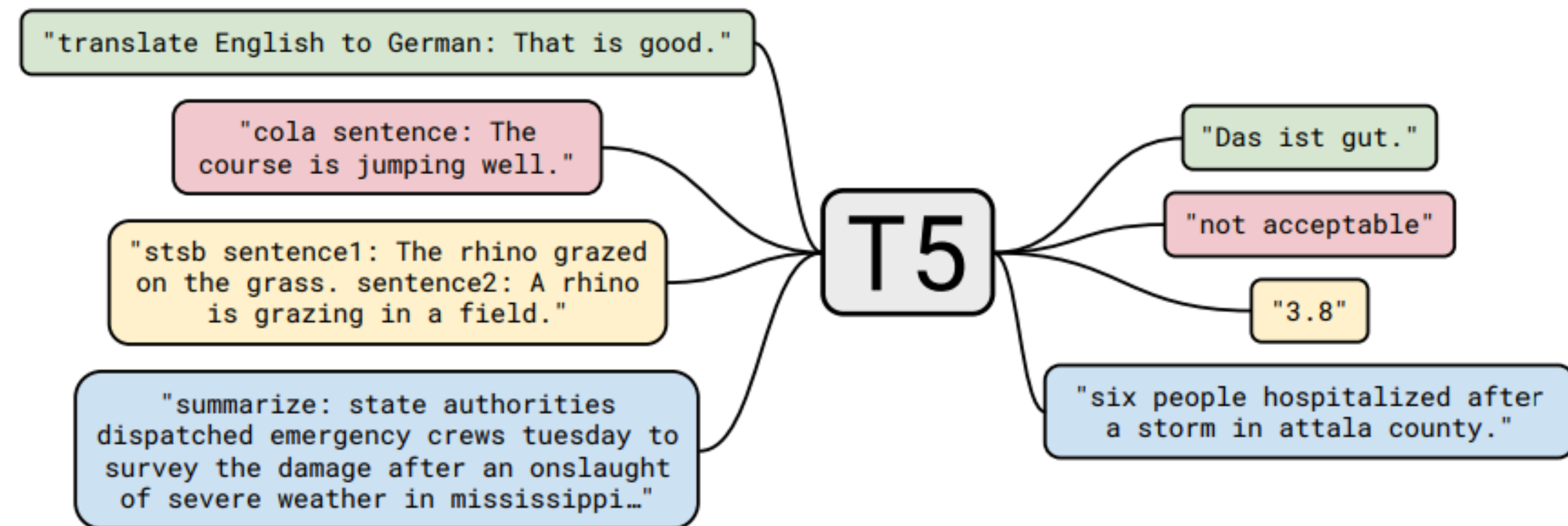
Thank you ~~for inviting~~ me to your party ~~last~~ week.

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



T5: Task Preparation

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



T5 Results

- 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

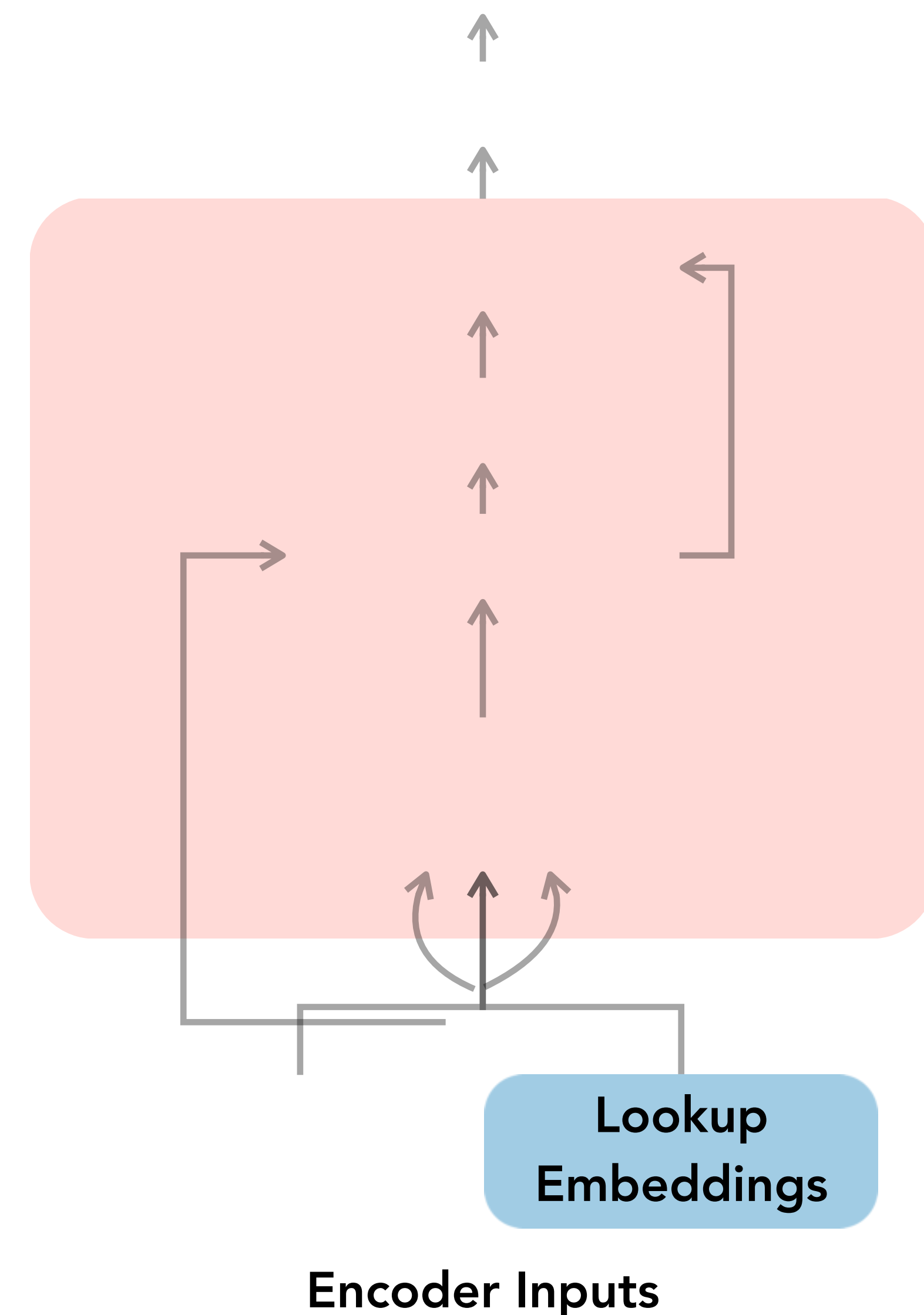
Lecture Outline

- Announcements
- Recap: The pre-training and fine-tuning paradigm
 - Pre-training Decoder-Only Models
 - Pre-training Encoder-Only Models
- Pre-training Encoder-Decoder Models
- Tokenization
- Natural Language Generation

Tokenization in Transformers

The Input Layer

- So far, we have made some assumptions about a language's vocabulary
- Our approach so far: use a known, fixed vocabulary
 - Built from training data, with tens of thousands of components
 - However, even with the largest vocabulary, we may encounter out-of-vocabulary words at test time
 - Our approach so far: map novel words seen at test time (OOV) to a single UNK



How to get the words?

Or, more accurately, the tokens?

- Problem: break the text into a sequence of discrete tokens
- For alphabetic languages such as English, deterministic scripts usually suffice to achieve accurate tokenization
- However, in languages such as Chinese and Swahili, words are typically composed of a small number of characters, without intervening whitespace

Word Structure in Language

- Finite vocabulary assumptions make even less sense in many languages.
 - Many languages exhibit complex morphology, or word structure.
 - The effect is more word types, each occurring fewer times.

-ambia = to tell

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

Conjugation of -ambia																		
Form		Non-finite forms																
		Positive								Negative								
Infinitive		kuambia								kutoambia								
Positive form		Simple finite forms																
		Singular								Plural								
Imperative		ambia								ambia								
Habitual										huambia								
Complex finite forms																		
Polarity	Persons				Persons / Classes		Classes											
	Sg.	1st Pl.	2nd Sg.	2nd Pl.	3rd Sg. / 1st Pl.	3rd Pl. / 2nd Pl.	3	4	5	6	7	8	9	10	11 / 14	15 / 17	16	18
Past																		
Positive	niliambia	tuliambia	uliambia	mliambia	aliambia	waliambia	uliambia	iliambia	liliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	muliambia
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambia	haukuambia	haikuambia	halikuambia	hayakuambia	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambia	hapakuambia	hamukuambia
Present																		
Positive	ninaambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	haliambii	hayaambii	hakiambii	haviambii	haiambii	haziambii	hauambii	hakuambii	hapaambii	hamuambii
Future																		
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	litaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambia	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambia
Subjunctive																		
Positive	niambie	tuambie	uambie	mambie	aambie	waambie	uambie	iambie	liambie	yaambie	kiambie	viambie	iambie	ziambie	uambie	kuambie	paambie	muambie
Negative	nisiambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	lisiambie	yasiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie
Present Conditional																		
Positive	ningeambia	tungeambia	ungeambia	mngeambia	angeambia	wangeambia	ungeambia	ingeambia	lingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia
Negative	nisingeambia	tusingeambia	usingeambia	msingeambia	asingeambia	wasingeambia	usingeambia	isingeambia	lisingeambia	yasingeambia	kisingeambia	visingeambia	isingeambia	zisingeambia	usingeambia	kusingeambia	pasingeambia	musingeambia
Past Conditional																		
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia	ungaliambia	ingaliambia	lingaliambia	yangaliambia	kingaliambia	vingaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia	pangaliambia	mungaliambia
Negative	nisingaliambia	tusingaliambia	usingaliambia	msingaliambia	asingaliambia	wasingaliambia	usingaliambia	isingaliambia	lisingaliambia	yasingaliambia	kisingaliambia	visingaliambia	isingaliambia	zisingaliambia	usingaliambia	kusingaliambia	pasingaliambia	musingaliambia
Conditional Contrary to Fact																		
Positive	ningeliambia	tungeliambia	ungeliambia	mngeliambia	angeliambia	wangeliambia	ungeliambia	ingeliambia	lingeliambia	yangeliambia	kingeliambia	vingeliambia	ingeliambia	zingeliambia	ungeliambia	kungeliambia	pangeliambia	mungeliambia
Gnomic																		
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia
Perfect																		

Source: Wiktionary

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
 - Subwords may be words, 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
- Different algorithms:
 - Byte-Pair Encoding
 - WordPiece Modeling
 - Follow different strategies. Often contain prepending / appending special tokens (##, </w>)

Hello how are U tday?



hello how are u tday?



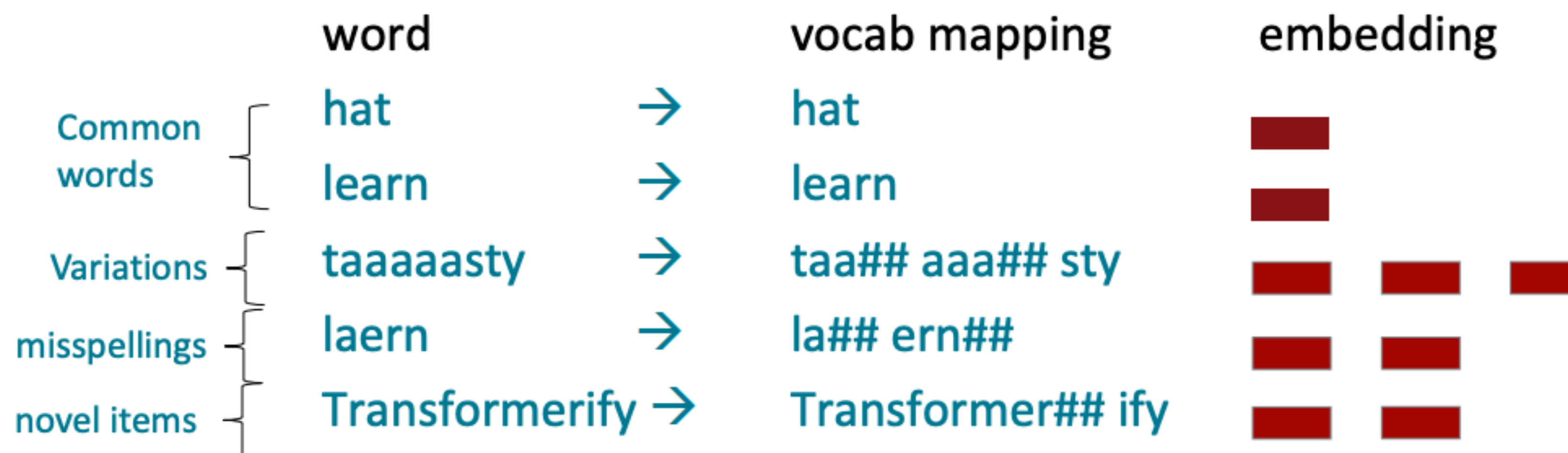
[hello, how, are, u, tday, ?]



[hello, how, are, u, td, ##ay, ?]

Word structure and subword models

- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.
- In the worst case, words are split into as many subwords as they have characters.
 - Llama 3 uses a tokenizer with a vocabulary of 128K tokens



Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
 - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
 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
 - This is a learned operation! However, not a parametric function
 - Only combine pairs (hence the name!)
 3. Replace instances of the character pair with the new subword; repeat until desired vocabulary size.
- At test time, first split words into sequences of characters, then apply the learned operations to merge the characters into larger, known symbols
- Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Vocabulary

d	e	i	l	n	o	s	t	w
es								

Frequency

d-e (3)	l-o (7)	t-</w> (8)
e-r (2)	n-e (5)	w-</w> (5)
e-s (8)	o-w (7)	w-e (7)
e-w (5)	r-</w> (2)	w-i (3)
i-d (3)	s-t (8)	

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Vocabulary

d	e	i	l	n	o	s	t	w
es	est							

Frequency

d-es (3)	l-o (7)	w-</w> (5)
e-r (2)	n-e (5)	w-es (5)
e-w (5)	o-w (7)	w-e (2)
es-t (8)	r-</w> (2)	w-i (3)
i-d (3)	t-</w> (8)	

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Vocabulary

d	e	i	l	n	o	s	t	w
es	est	est</w>	lo	low	low</w>	ne	new	newest</w>

After 10 merges










WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
 - Each word is initially split by adding ## to all the characters inside a word
 - So, for instance, "word" gets split like this: w ##o ##r ##d
 - For this vocabulary:
 - ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
 - Splits may look like:
 - ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)
- On merging, ## **between** the two tokens is removed
 - This explains the presence of the token "##ing"

WordPiece Modeling Outcome

- Different stopping criteria: number of merges or size of resulting vocabulary
- In the worst case, at test time, words are split into as many subwords as they have characters
- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components

WordPiece Outcome

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

Tokenization: Frequently Asked Questions

- Where does the token “##ing” come from?
 - In WordPiece tokenization, all non-starting characters are initialized as ##x.
 - Like: h, ##e, ##l, ##l, ##o.
 - Upon merging, only the first segment keeps its ##.
- How is tokenization done in Chinese?
 - Follows the same broad overall algorithm, but the initial split into characters involve language-specific rules
 - e.g. stroke-level tokenization [Si et al., 2023]

Source: <https://huggingface.co/learn/nlp-course/chapter6/6?fw=pt>

Lecture Outline

- Announcements
- Recap: The pre-training and fine-tuning paradigm
 - Pre-training Decoder-Only Models
 - Pre-training Encoder-Only Models
- Pre-training Encoder-Decoder Models
- Tokenization
- Natural Language Generation
 - Classic Inference Algorithms: Greedy and Beam Search

Natural Language Generation

Natural Language Generation

- Natural language understanding and natural language generation are two sides of the same coin
 - Natural language understanding: Learning representations that perform well on downstream tasks
 - To generate good language, one needs to understand language
 - If you understand language, you should be able to generate it (with some effort)
- NLG is the workhorse of many classic and novel applications
 - AI Assistants
 - Translators
 - Search summarizers



NLG Use Cases

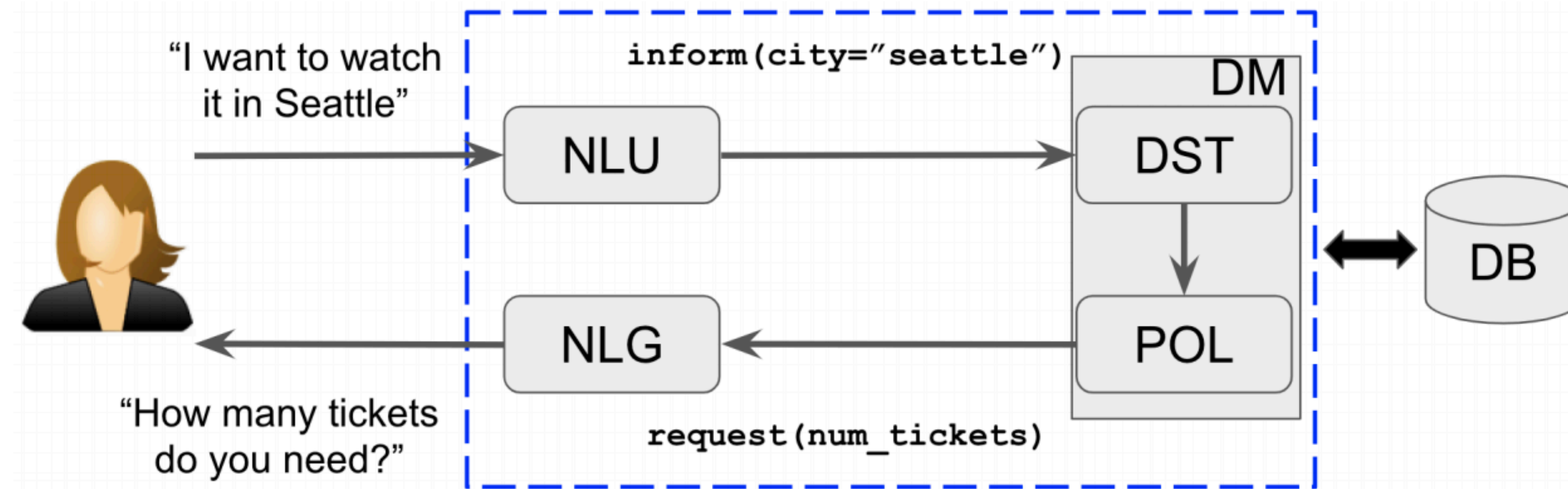
Simple and Effective Multi-Paragraph Reading Comprehension

Christopher Clark, Matt Gardner · Computer Science · ACL · 29 October 2017

TLDR We propose a state-of-the-art pipelined method for training neural paragraph-level question answering models on document QA data. [Expand](#)

236 PDF · View PDF on arXiv · Save · Alert · Cite · Research Feed

Summarization



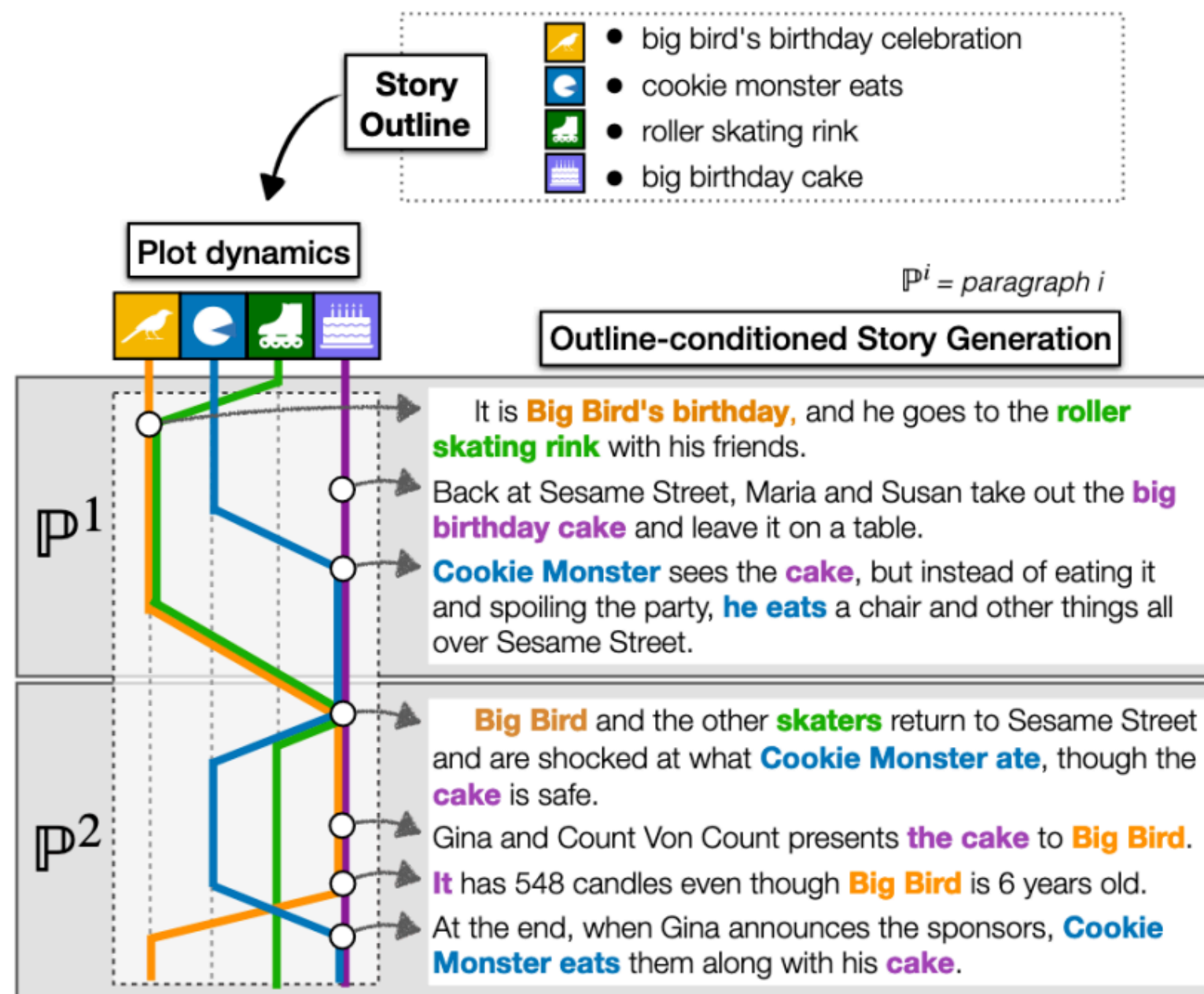
Task-driven Dialog



Chit-chat Dialog

More Interesting NLG Uses

Creative stories



Rashkin et al., 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	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	3
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., 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., 2017

Broad Spectrum of NLG Tasks



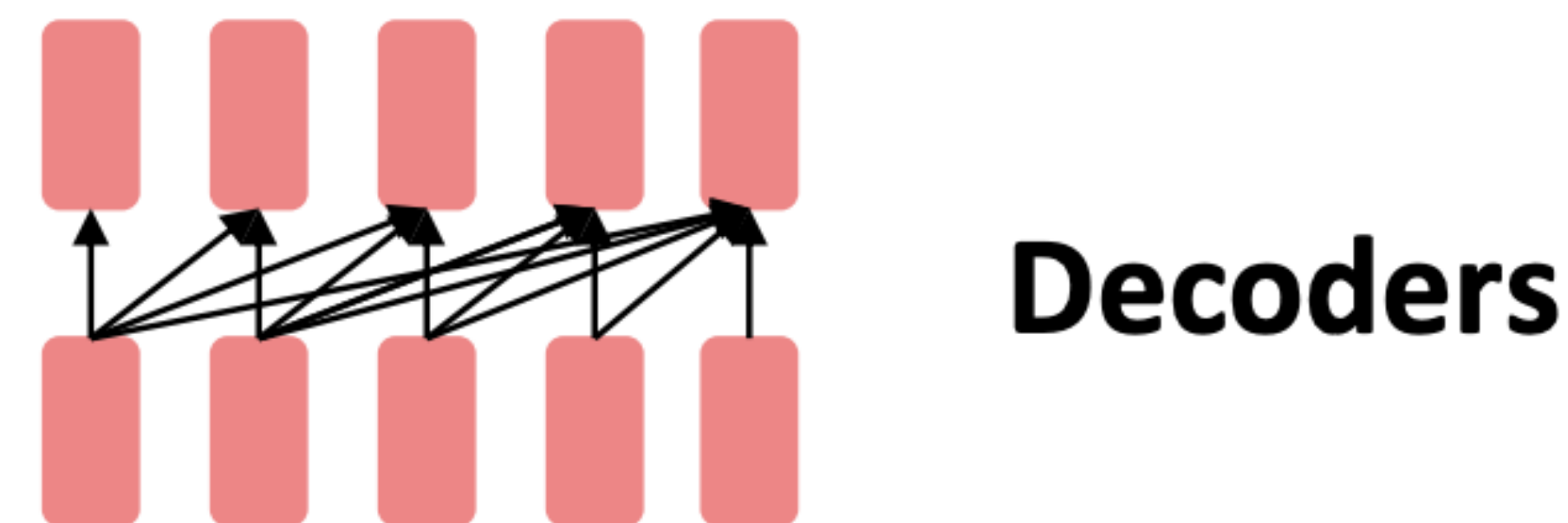
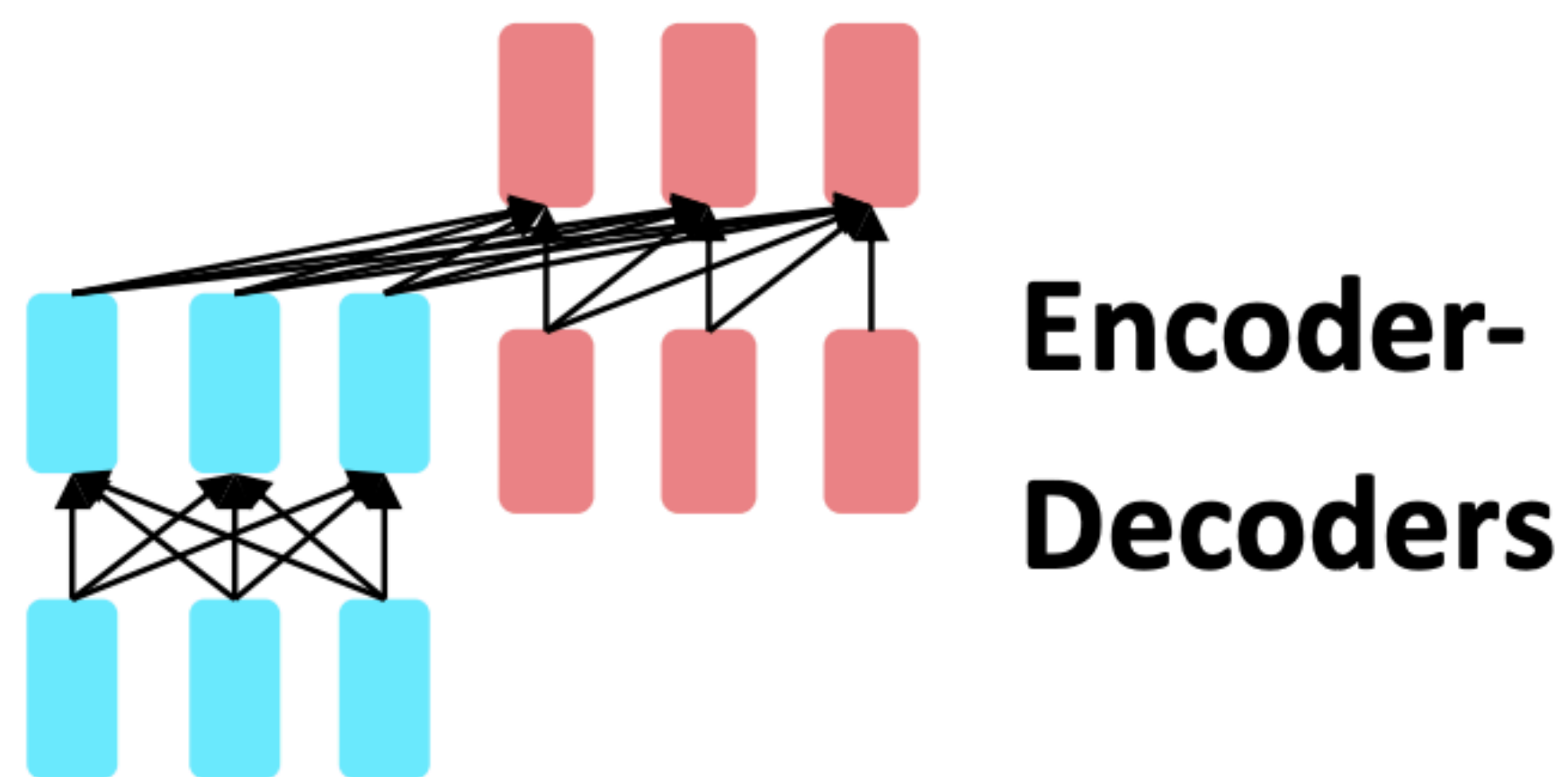
Open-ended generation: the output distribution still has high freedom.

Non-open-ended generation: the input mostly determines the output generation.

Broad Spectrum of NLG Tasks

Less Open-Ended

More Open-Ended

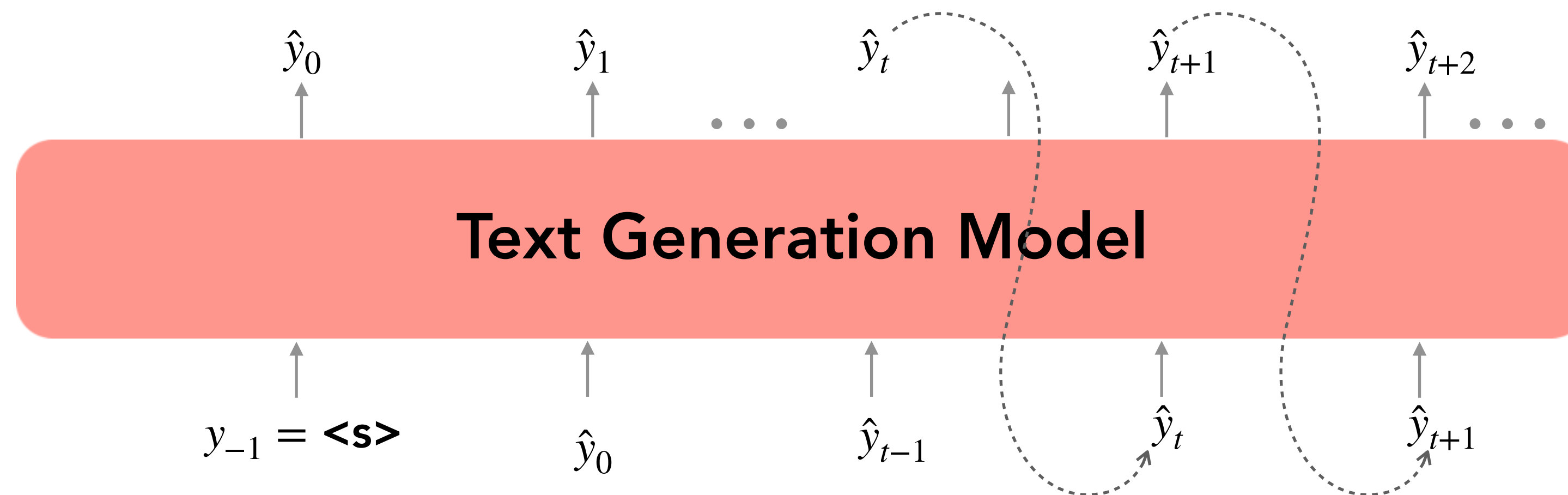


Language Generation: Fundamentals

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

For model $f_{\theta}(\cdot)$ and vocabulary V , we get scores $S = f_{\theta}(y_{<t}) \in \mathbb{R}^V$

$$P(w | y_{<t}) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$$

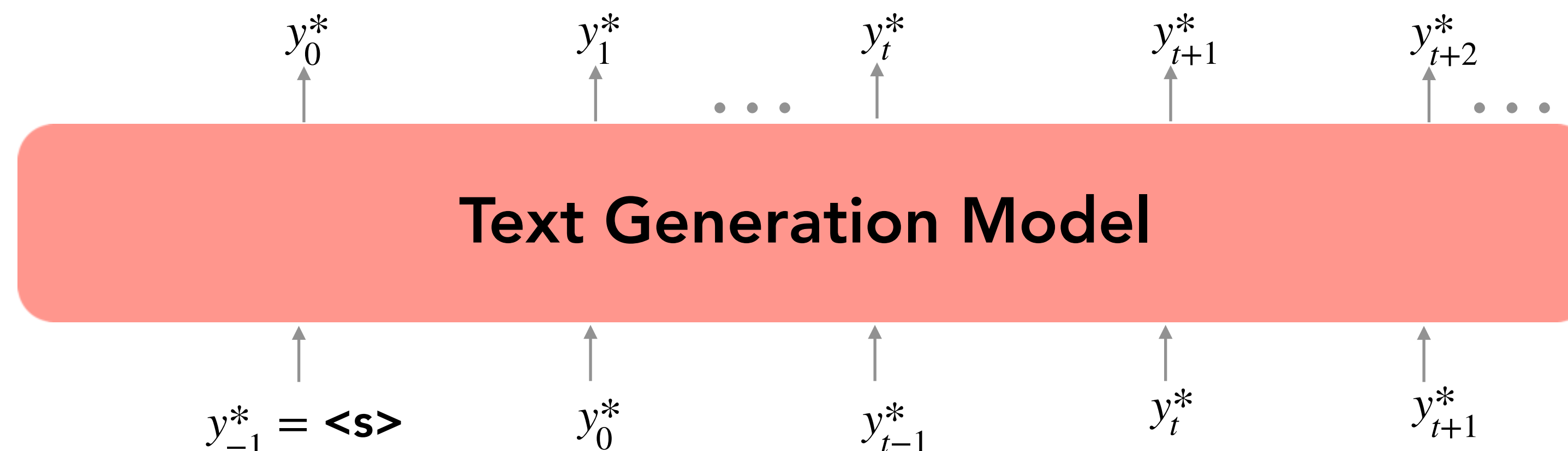


Language Generation: Training

- Trained one token at a time to maximize the probability of the next token y_t^* given preceding words $y_{<t}^*$

$$\mathcal{L} = - \sum_{t=1}^T \log P(y_t | y_{<t}) = - \sum_{t=1}^T \log \frac{\exp(S_{y_t | y_{<t}})}{\sum_{v \in V} \exp(S_{v | y_{<t}})}$$

- Classification task at each time step trying to predict the actual word y_t^* in the training data
- “Teacher forcing” (reset at each time step to the ground truth)



Teacher Forcing

- Strategy for **training** decoders / language models
- At each time step t in decoding we force the system to use the gold target token from training as the next input x_{t+1} , rather than allowing it to rely on the (possibly erroneous) decoder output \hat{y}_t
- Runs the risk of **exposure bias!**
 - During training, our model's inputs are gold context tokens from real, human-generated texts
 - At generation time, our model's inputs are previously-decoded tokens

Language Generation: Inference

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

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

Inference / Decoding Algorithm

- The “obvious” decoding algorithm is to greedily choose the highest probability next token according to the model at each time step

$$g = \arg \max$$

$$\hat{y}_t = \arg \max_{w \in V} (P(y_t = w | y_{<t}))$$

Classic Inference Algorithms: Greedy and Beam Search

Decoding

- Generation from a language model is also called decoding
 - Think encoder-decoder
 - Also called inference
- Strategy so far: Take $\arg \max$ on each step of the decoder to produce the most probable word on each step
- This is called greedy decoding
 - Greedy Strategy: we are not looking ahead, we are making the hastiest decision given all the information we have

Greedy Decoding: Issues

- Greedy decoding has no wiggle room for errors!
 - Input: the green witch arrived
 - Output: Ilego
 - Output: Ilego la
 - Output: Ilego la **verde**
- How to fix this?
 - Need a lookahead strategy / longer-term planning

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$\begin{aligned} P(y|x) &= P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots, P(y_T|y_1, \dots, y_{T-1}, x) \\ &= \prod_{t=1}^T P(y_t|y_1, \dots, y_{t-1}, x) \end{aligned}$$

- We could try computing all possible sequences y
 - This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size
 - This $O(V^T)$ complexity is far too expensive!

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)

- k is the beam size (in practice around 5 to 10, in NMT)

- A hypothesis has a score which is its log probability:

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

- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top k on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!