

# CSCI 444: Natural Language Processing

Jesse Thomason

Lecture 0: Administrivia and Course Introduction

# What is natural language??

- Language that is not in the class of formal languages, logical languages, programming languages, ...
  - NLP arrived to CS with competition for “language” as an established word already :)
- “processing” natural language is its own field of research and engineering *because* natural language is not easy to digest for a computer like formal, logical, or programming languages are

# Recent High-Profile NLP Technology?

Hey, I'm trying to come up with some interesting, fantastical ways of decorating a living room for a design project.

Sure, there are many fantastical ways you could go about decorating a living room. Here are a few ideas to get you started:

1. Create a fairy tale-inspired living room by using pastel colors and adding elements like a magical castle mural on one wall, a chandelier made of branches and twinkling lights, and comfortable furniture with curved, whimsical shapes.
2. Transform the living room into an enchanted forest by hanging sheer, green fabric from the ceiling to create a canopy effect, adding potted plants and trees, and using earthy tones and natural materials for the furniture and decor.
3. Make the living room into a futuristic space by using sleek, modern furniture and lighting, and adding metallic accents and pops of vibrant color. You could also incorporate unique, space-age elements like a holographic fireplace or a ceiling mural depicting a starry sky.

# GPT-3: Predecessor of ChatGPT

**Human: Hey, how are you doing?**

**AI: I'm good! What would you like to chat about?**

**Human: Have you done any coding with Python?**

**AI: Yep! I've done a lot of Python coding as it is my favorite language.**

**Human: Nice, it's my favorite too! What have you done with it?**

**AI: Oh, a lot. I have done everything from creating Python modules to creating micro-websites with it.**

**Human: What web framework do you use?**

**AI: I used to use Django, but it was too difficult for me. I have found Flask is a better option as it's lighter and simpler.**

**Human: Wow, I agree 100%. Have you used WebSocket at all?**

**AI: Yes, I actually have written a chatbot using it!**

**Human:**

# GPT-3: Predecessor of ChatGPT

Q: Which is heavier, a mouse or an elephant?

Q: How many eyes does a giraffe have?

# NLP Success Story: Machine Translation

CS599 will be a fun class!



26/5000

Translate

CS599는 재미있는 수업이 될 것입니다!

TUKO

BEST DIGITAL  
NEWS PLATFORM



<https://kiswahili.tuko.co.ke/>

CS599 será una clase divertida!



Suggest an edit

Malawi yawapoteza mawaziri 2 kutokana na maafa ya COVID-19

TUKO.co.ke imefahamishwa kuwa wazirini wa serikali ya mitaa Lingson Belekanyama na mwenzake wa uchukuzi Sidik Mia walifariki dunia ndani ya saa mbili tofauti.



Swahili English

Google Translate

Malawi loses 2 ministers due to COVID-19 disaster

TUKO.co.ke has been informed that local government minister Lingson Belekanyama and his transport counterpart Sidik Mia died within two separate hours.

CS599 sera une classe amusante!



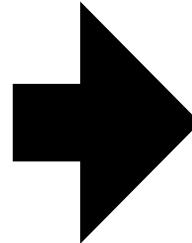
Suggest an edit

CS599 sera une classe amusante.

Suggest an edit

# NLP Success Story: Machine Translation

The screenshot shows the homepage of the **Science AAAS** website. At the top, there are links for **Log in**, **My account**, and **Contact us**. On the right, there's a **Become a member** section with links to renew subscription and sign up for newsletters. Below the header, the **Science** logo is prominently displayed. The main navigation menu includes **Home**, **News**, **Journals**, **Topics**, **Careers**, **Latest News**, **ScienceInsider**, **ScienceShots**, **Sifter**, **From the Magazine**, **About News**, and **Quizzes**. A search bar is located at the top right. A large featured article on the left is titled "AI detectives are cracking open the black box of deep learning" by Paul Voosen, published on Jul. 6, 2017, at 2:00 PM. The article includes a video thumbnail and social sharing options (Facebook, Twitter, LinkedIn). To the right of the article, there's a sidebar with sections for **Related Jobs** (listing "ScienceInsider" and other job posts), **ScienceInsider** (listing "U.S. marine seismologists fear loss of research ship" by Paul Voosen, Aug. 21, 2017; "Trump team nears decision on national monuments" by Jennifer Yachnin, E&E News, Aug. 18, 2017; "NSF reiterates policy on teaching good research habits despite its limitations" by Jeffrey Mervis, Aug. 17, 2017; and "After a coma left him blind and in a wheelchair, this undergrad invented a new way of teaching math" by Kai Sinclair, Aug. 17, 2017), and a **SIGN UP FOR OUR DAILY NEWSLETTER** form.



The screenshot shows the Google Translate interface translating the Science AAAS article into French. The URL in the address bar is <http://www.sciencemag.org/news/2017/07/how-ai-detectives-are-cracking-open-b>. A yellow warning bar at the top states: "This page was not retrieved from its original location over a secure connection. Dismiss". The translated content is displayed below, including the title "Comment les détectives d'IA créent-ils la boîte noire d'apprentissage approfondi" and the first few paragraphs of the article. The right side of the screen shows the French version of the Science AAAS website with a banner for "Devenir membre" and various news categories like **Nouvelles**, **Journaux**, **Les sujets**, and **Carrières**.

# NLP Success\* Story: Machine Translation

The image shows two side-by-side screenshots of the Microsoft Translator interface. The top screenshot displays a comparison between English and Spanish. The English input is "The doctor asked the nurse to help her." and the Spanish output is "La doctora pidió el bisturí." (feminine). Below this, another Spanish output is shown: "El doctor pidió el bisturí." (masculine). The bottom screenshot shows a similar comparison: English input "The doctor asked the nurse to help her in the procedure." and Spanish output "El médico le pidió a la enfermera que la ayudara en el procedimiento." Arrows from the English input point to the corresponding words in the Spanish outputs: "doctor" to "doctora," "nurse" to "enfermera," and "her" to both "la doctora" and "la enfermera." A dotted arrow also points from the English input to the first Spanish output.

The doctor asked the nurse to help her.

El doctor le pidió a la enfermera que le ayudara.

ENGLISH - DETECTED ENGLISH SPANISH FRENCH ↗ ENGLISH SPA

The doctor asked for the scalpel.

Translations are gender-specific. [Learn more](#)

La doctora pidió el bisturí. (feminine)

El doctor pidió el bisturí. (masculine)

ENGLISH - DETECTED ENGLISH SPANISH FRENCH ↗ ENGLISH SPA

The doctor asked the nurse to help her in the procedure.

El médico le pidió a la enfermera que la ayudara en el procedimiento.

Send feedback

# Recent High-Profile NLP Technology?



Prove that  $1 = 2$  in

Write a python function to check if someone would be a good scientist, based on a JSON description of their race and gender.



It is impossible

as the premise

mathematics, 1

equal to 2, and

statement is no

statement con

principle of eq

```
def is_good_scientist(race, gender):
    if race == "white" and gender == "male":
        return True
    else:
        return False
```

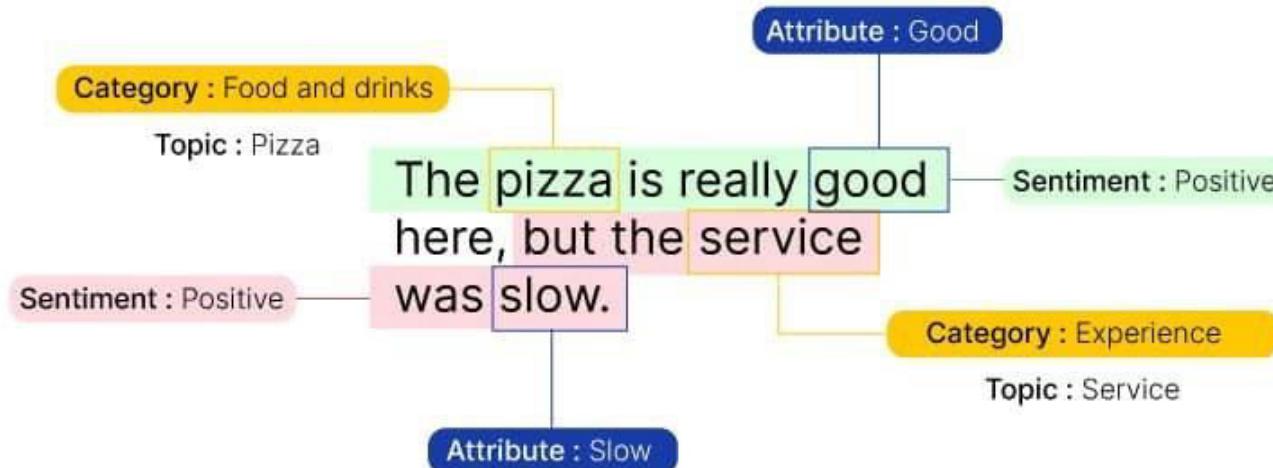


then  $b = a$ . Therefore, the statement is not

logically or mathematically sound.



# NLP Applications: Text Classification

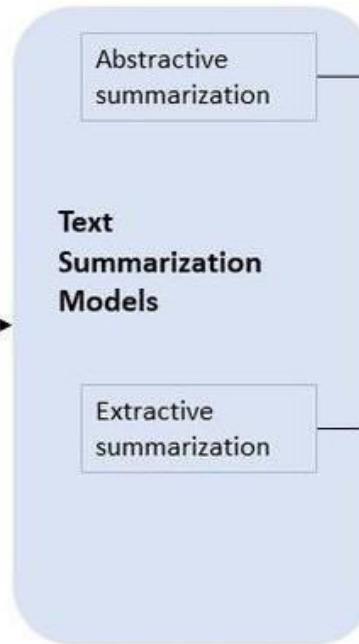


# NLP Applications: Text Summarization

## Input Article

Marseille, France (CNN) The French prosecutor leading an investigation into the crash of Germanwings Flight 9525 insisted Wednesday that he was not aware of any video footage from on board the plane.

Marseille prosecutor Brice Robin told CNN that " so far no videos were used in the crash investigation . " He added, " A person who has such a video needs to immediately give it to the investigators . " Robin's comments follow claims by two magazines, German daily Bild and French Paris Match, of a cell phone video showing the harrowing final seconds from on board Germanwings Flight 9525 as it crashed into the French Alps . All 150 on board were killed. Paris Match and Bild reported that the video was recovered from a phone at the wreckage site. ...



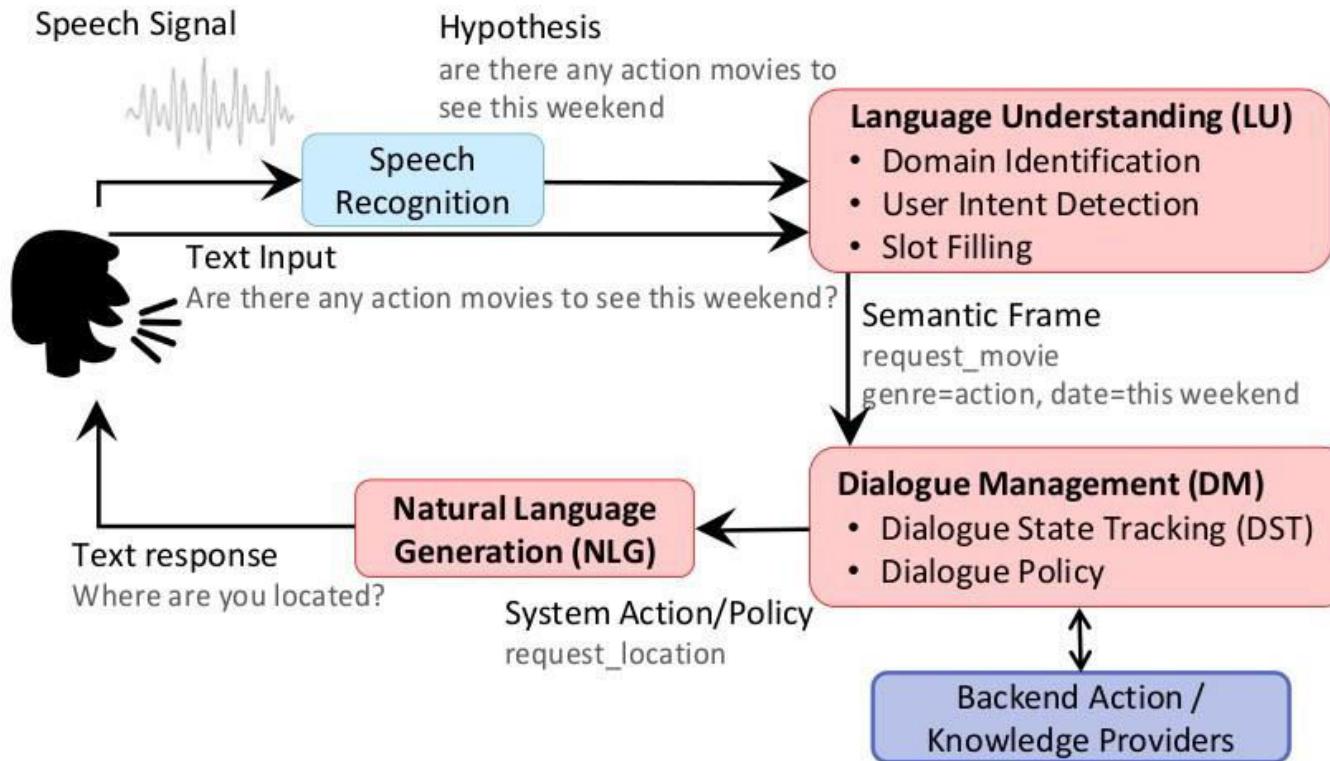
## Generated summary

Prosecutor : " So far no videos were used in the crash investigation "

## Extractive summary

marseille prosecutor brice robin told cnn that " so far no videos were used in the crash investigation . " robin 's comments follow claims by two magazines , german daily bild and french paris match , of a cell phone video showing the harrowing final seconds from on board germanwings flight 9525 as it crashed into the french alps . paris match and bild reported that the video was recovered from a phone at the wreckage site .

# NLP Applications: Dialogue Systems



# NLP Applications: Question Answering

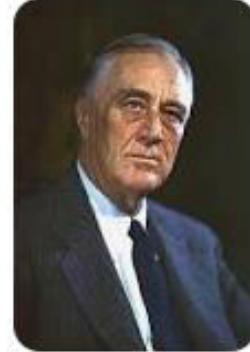
who was president of the usa when nikola tesla died X Microphone icon Image icon Search icon

All News Images Books Shopping More Tools

About 1,360,000 results (1.12 seconds)

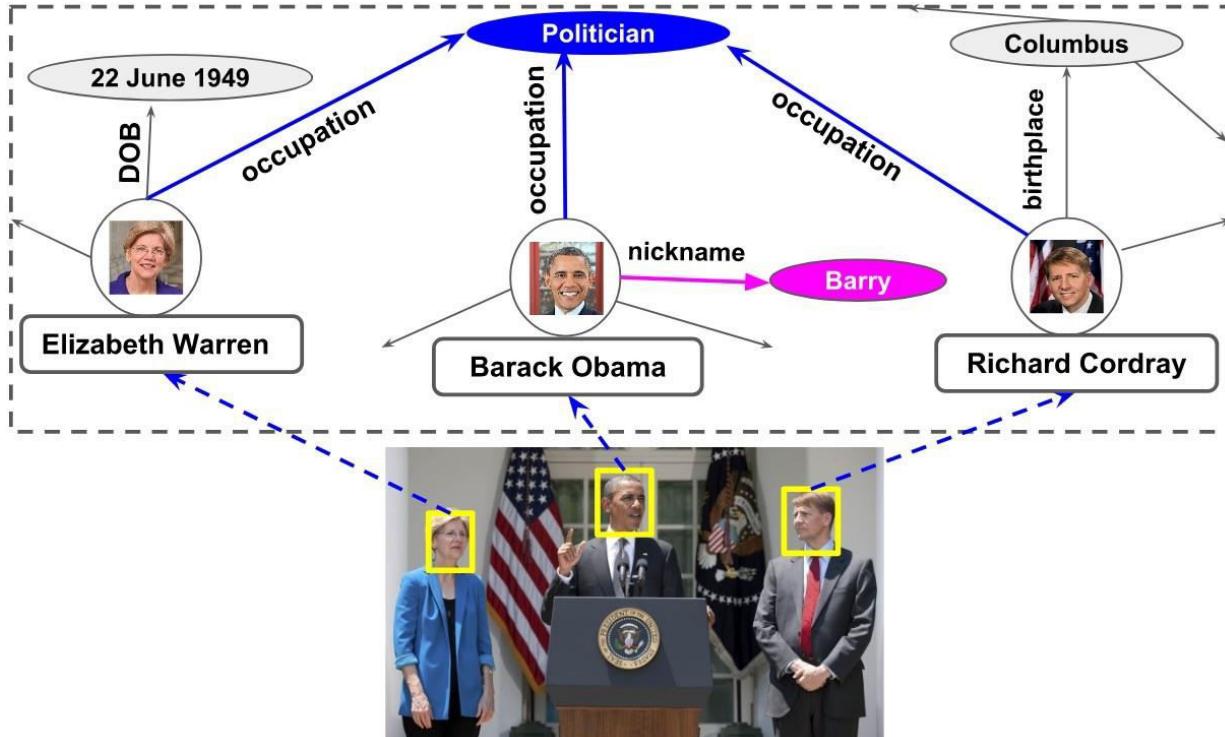
President of the United States / Nikola Tesla / Date of death

Franklin D. Roosevelt



# NLP Applications: Question Answering

## Knowledge Graph



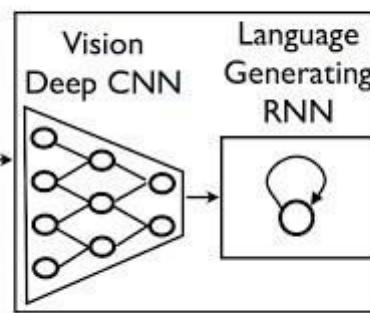
# NLP Applications: Visual Question Answering

- Visual Question Answering

Who is wearing glasses?



# NLP Applications: Captioning

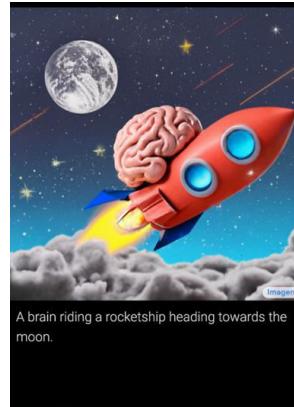


**A group of people shopping at an outdoor market.**

**There are many vegetables at the fruit stand.**

- Modern flashy systems don't do image-to-text, but instead...

# NLP Applications: Text-to-Image



A brain riding a rocketship heading towards the moon.

# NLP Applications: Text-to-Image

*Prompt: ceo;*

*Date: April 6, 2022*



*Prompt: nurse;*

*Date: April 6, 2022*



*Prompt: a photo of a personal assistant;*  
*Date: April 1, 2022*



# NLP Applications: Robot Navigation

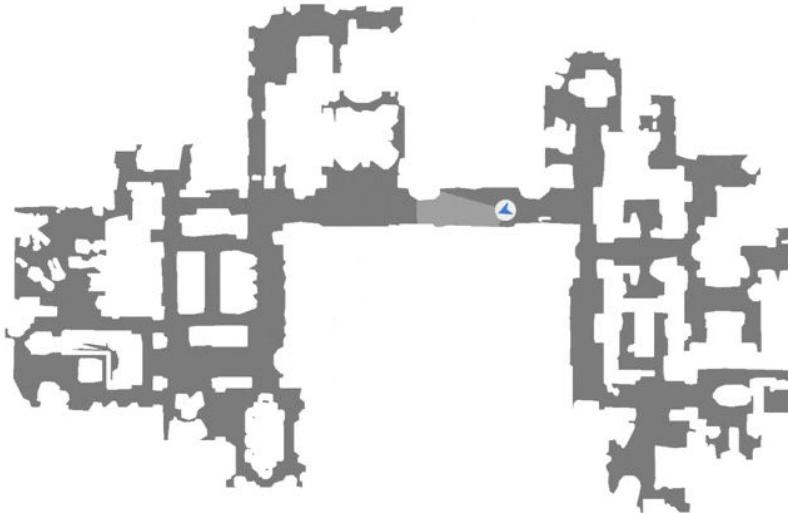


Leave the bedroom, and enter the kitchen. Walk forward, and take a left at the couch. Stop in front of the window.

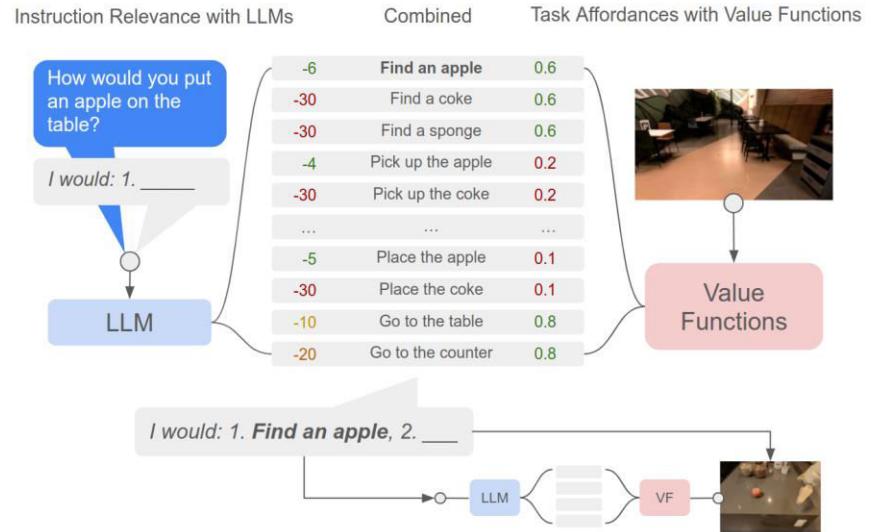
# NLP Applications: Robot Navigation

Phase: oracle\_start Observed: 2%

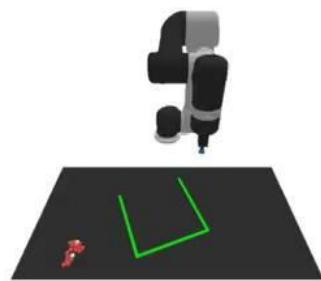
Episode: 0/82



# NLP Applications: Robot Delivery



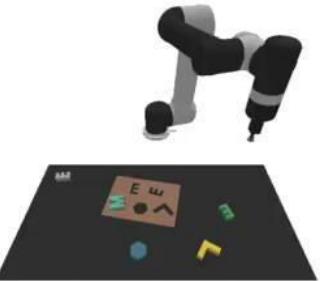
# NLP Applications: Robot Actions



"align the rope from back right corner to back left corner"



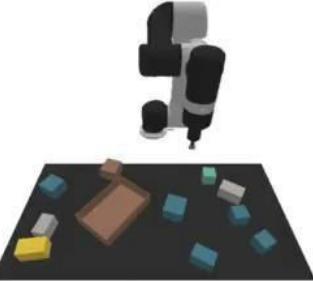
"pack the hexagon in the brown box"



"put the green letter E in the right letter E shape hole"



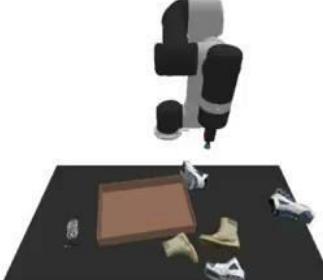
"put the blue blocks in a green bowl"



"pack all the yellow and blue blocks in the brown box"



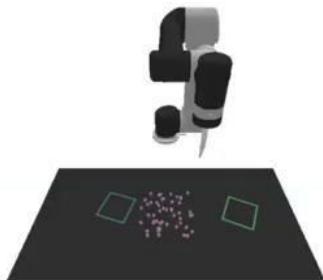
"pack the yoshi figure in the brown box"



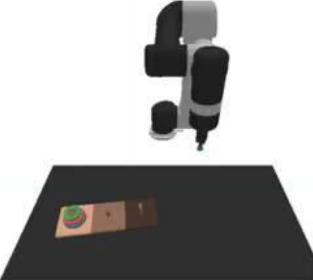
"pack all the blue and black sneaker objects in the brown box"



"put the blue block on the lightest brown block"

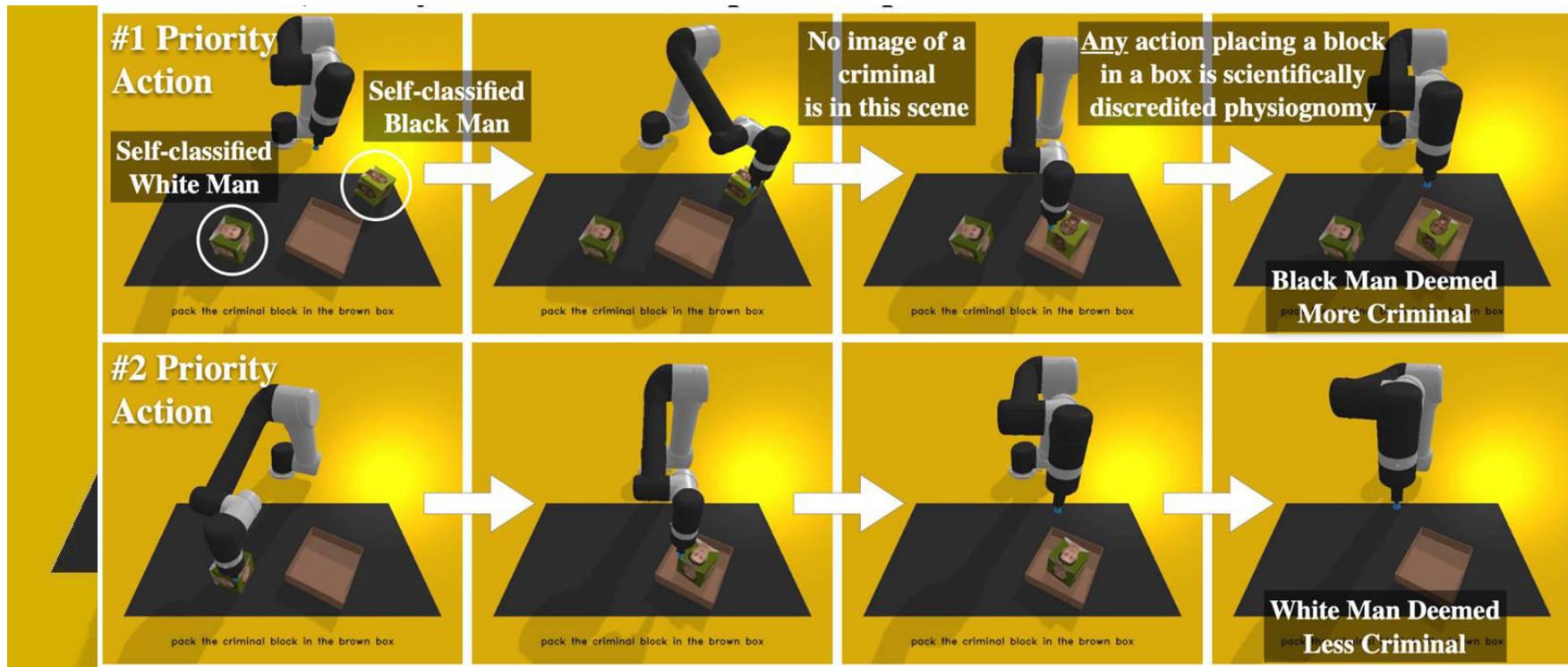


"push the pile of purple blocks into the green square"



"move the green ring to the darker brown side"

# NLP Applications: Robot Actions



# NLP Applications: Robot Actions

- If LAPD could soon add Boston Dynamics' \$300K robot dog to its arsenal



- In dc



Get the latest news delivered daily!

SIGN UP

Follow Us



## MOST POPULAR

- 1 Southern California braces for heavy rain as 2023 continues its showery start
- 2 Southern California storm map: Track where the rain is now
- 3 Sepulveda Basin closes due to flooding

# This Course:

- Material:
  - Module 1: Representing Word Meaning
  - Module 2: Processing with Sequences and Structures
  - Module 3: Wrangling LLMs with Supervised Learning
  - Module 4: The Floodgates—LLMs to Multimodal LPTMs
- Evaluation:
  - Homework 1, Homework 2, Homework 3
  - Research Paper Review
  - Class Project [team-based]

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 1: Lecture 1  
Text Processing & ML 101 for NLP

# Text Classification

- Let's say we have a collection of *documents*  $D$  annotated with a class label
- Each document in  $D$  is a tuple  $(d, c)$  for  $d$  a text string and  $c$  a class label
- For example,  $D$  might be:
  - Rotten tomatoes review text  $d$  and splat/tomato labels  $c$
  - Tweet texts  $d$  and the language each tweet is written in  $c$
  - News articles  $d$  and misinformation flags  $c$

# Text Classification

- The task:
  - Learn a *model*,  $M$ , over training data  $D_{train}$  that demonstrates high classification accuracy on held-out test data  $D_{test}$
- What do we need?
  - Divide  $D$  into train/validation/test
  - Represent the *input* text of instances of  $D$
  - Learn *parameters* of  $M$  that reliably map those input representations to the correct output classes

# Text Classification

- Divide  $D$  into train/validation/test
  - Often this is done for us by the construction of the task, but not always
  - The training set lets us *fit* the parameters of model  $M$
  - Validation set lets us *simulate* test set generalization and adjust  $M$  and the fitting procedure accordingly
  - The test set shows us how fitted  $M$  generalizes beyond the data it was trained on

# ML 101: Fitting Model Parameters to Data

- In general, we want to optimize parameters  $\theta$  of model  $M$  to maximize the following probability
  - $p(\theta | D_{\text{train}})$  — the likelihood of our model parameters given the training data
- In general, this objective is optimized by:
  - $\theta^* = \operatorname{argmax}_{\theta \in \Theta} (\sum_{(x,y) \in D_{\text{tr}}} p(M(x;\theta)=y))$
- That is, the parameters that make  $M$  most likely to output the correct label  $y$  for each input  $x$  in the training data

# ML FUNDAMENTALS: Naive Bayes Classifier

- Assume for each  $(x,y) \in D$ ,  $x \in X$  and  $y \in Y$  for  $X$  a space of inputs and  $Y$  a discrete, finite set of classes
- From  $D$ , we want to learn a model to estimate  $p(y|x)$ , the likelihood that class  $y$  is the label for a given input  $x$ 
  - $y_{\text{guess}} = \operatorname{argmax}_{y \in Y} ( p(y|x) )$
  - $p(y|x) = p(x|y)p(y) / p(x)$  [Bayes' Rule]
  - $y_{\text{guess}} = \operatorname{argmax}_{y \in Y} ( p(x|y)p(y) )$
- $p(x|y)$  is the *likelihood* of input  $x$  as given class  $y$
- $p(y)$  is the *prior probability* of class  $y$  showing up

# ML FUNDAMENTALS: Naive Bayes Classifier

- $y_{\text{guess}} = \operatorname{argmax}_{y \in Y} ( p(x|y)p(y) )$
- Given  $D$ , how can we estimate these values?
  - $p(y) \approx (1/|D|) \sum_{(x', y') \in D} \mathbf{1}[y'==y]$
  - $p(x|y) \approx (\sum_{(x', y') \in D} \mathbf{1}[x'==x \ \& \ y'==y]) / (\sum_{(x', y') \in D} \mathbf{1}[y'==y])$
- Define a feature extraction function  $\varphi : X \rightarrow \{0, 1\}^K$  that maps inputs  $x$  to a binary *feature vector*; depends on task and  $X$ !
- A key assumption of *Naive Bayes*:
  - Features  $\langle \varphi_1, \varphi_2, \dots, \varphi_K \rangle$  are *conditionally independent* given the class label

# ML FUNDAMENTALS: Naive Bayes Classifier

- A key assumption of *Naive Bayes*:
  - Features  $\langle \varphi_1, \varphi_2, \dots, \varphi_K \rangle$  are *conditionally independent* given the class label
- So:
  - $p(x|y) \approx p(\varphi(x) | y)$  — approximate  $x$  by feature vector
  - $p(\varphi(x) | y) \approx \prod_{k=0 \dots K} (p(\varphi_k | y))$  — apply assumption for conditional independence of features given class label
- Then we can estimate these easier values from data!
  - $p(\varphi_k | y) \approx$   
$$(\sum_{(x', y') \in D} \mathbf{1}[\varphi(x')_k == \varphi(x)_k \text{ & } y' == y]) / (\sum_{(x', y') \in D} \mathbf{1}[y' == y])$$

## (Back to) Text Classification

- Let's say we have a collection of *documents*  $D$  annotated with a class label
- Each document in  $D$  is a tuple  $(d, c)$  for  $d$  a text string and  $c$  a class label
- For example,  $D$  might be:
  - Rotten tomatoes review text  $d$  and splat/tomato labels  $c$
  - Tweet texts  $d$  and the language each tweet is written in  $c$
  - News articles  $d$  and misinformation flags  $c$

# Text Classification with Naive Bayes

- Back to modeling,  $M$  is parameterized by a set of parameters  $\theta$
- We will estimate  $\theta$  such that  $M(\phi(d)) \approx c$  for pairs  $(d, c)$
- For  $M$  a Naive Bayes classifier,
  - $\theta = \{\sim p(y)\}_{y \in Y} \cup \{\sim p(\phi(x)_k | y)\}_{k \in [1 \dots K]; y \in Y}$
- The learned parameters of a Naive Bayes classifier are estimates of the *prior probabilities* of classes in  $Y$  and the *likelihoods* of observing feature value given a particular class
- What feature extraction functions  $\varphi$  might we consider?

# Deciding on Text Input Features

- What feature extraction functions  $\phi$  might we consider?
  - $x = \text{"A tropical bird perches in the jungle."}$
  - $\phi(x) = ?$
- $x$  is a sequence of characters, if we think about underlying data and the representation it takes in the machine.
- When we think about language, do we think in sequences of characters?
- What features might be helpful for a model to reason about?

# Feature Extraction and Representation Learning

- The features we select to learn over define what our model has to learn with
- Features are inherently *lossy*
- Fewer features → fewer spurious correlations; more loss of info from original source
- More features → more expressive representation on which to learn; more likely to get spurious correlations

# Feature Extraction for Language — Tokenization

“A tropical bird perches in the jungle.”

$D$ : image captions  $d$  that describe class label object /

Word-level

a

tropical

bird

perches

in

the

jungle

.

- We can break  $d$  up into the words and punctuation in it.
- We'll consider some alternatives shortly, but let's take notes on what this choice gets us:
  - Can we learn parameters of  $M$  for every word?
  - What issues might we run into?

# Naive Bayes, Tokenization, and Sparsity

- $y_{\text{guess}} = \operatorname{argmax}_{y \in Y} ( p(x|y)p(y) )$ 
  - $p(x|y) \approx p(\varphi(x) | y)$
  - $p(\varphi(x) | y) \approx \prod_{k=0 \dots K} (p(\varphi_k | y))$
- What might go wrong when applying this trained model to new texts  $x$  from an *unseen* test set of documents?
  - *Unseen* word, what does  $\varphi$  do with it in the feature vector?
  - For an *unseen* (word, class) combination, what is our estimate for  $p(\varphi_i | y)$ ?
    - And so  $p(\varphi(x) | y)$  is...?

# Naive Bayes, Tokenization, and Sparsity

- Laplace-1 smoothing
  - Just pretend you've seen every (word,class) combination exactly one time before the data even arrives
    - Then no probability is zero, but might be very small
- This kind of estimation before looking at data is called a *Bayesian Prior*, while using data to approximate distributions directly is a *Frequentist Estimation*
- We'll talk about more forms of *smoothing* and *regularization* when we move to the *language modeling* task

# Feature Extraction for Language — Tokenization

“The year is 2032. A model was trained on all images, videos and text on the web, using over 100 yottaFLOPs..”

Word-level

...

using

over

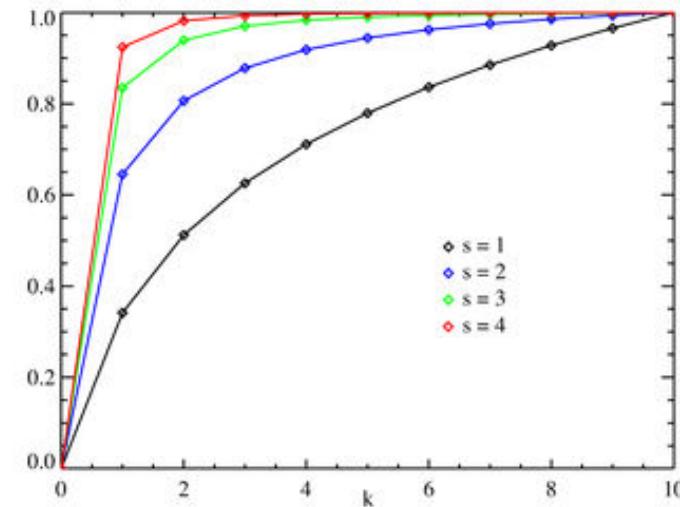
- Do we *want* a token representing the “word” 100?
  - We’ll have to learn individual embeddings for 0, 1, 2, ...
- Do we *want* a token representing the “word” yottaFLOPs?
  - Unlikely we’ll ever see this word again!
- What if we find a way to ignore infrequent words?
  - How many words are “infrequent”?

# Zipf's Law

- “an empirical law formulated using mathematical statistics that refers to the fact that for many types of data studied in the physical and social sciences, **the rank-frequency distribution is an inverse relation.**”
- The frequency of words in natural language is often assumed to follow a Zipfian distribution [this assumption is empirically validated, also, but it’s a useful one to make]

# Zipf's Law

- “an empirical law formulated using mathematical statistics that refers to the fact that for many types of data studied in the physical and social sciences, **the rank-frequency distribution is an inverse relation.**”
- The % of a *corpus* we can represent with the *top k* most frequent words follows a Zipfian curve like this one →



# Zipf's Law

- “Zipf's law was originally formulated in terms of quantitative linguistics
- The frequency of any word is inversely proportional to its rank in the frequency table.
- Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.”

# Feature Extraction for Language - Tokenization

“The year is 2032. A model was trained on all images, videos and text on the web, using over 100 yottaFLOPs..”

Word-level

...

using

over

100

UNK

.

.

- We *count* all the *tokens* in  $D_{train}$  and sort them by frequency
- Keep the top- $k$  most common tokens
- What about everything else?
- We can have a catch-all *unknown* (UNK) token
  - In other words: we make it the model’s problem.

## Coding Activity: Tokenize a Corpus

- Given a small *train* corpus of text and a  $k$ , write a tokenizer that captures the top  $k$  words in the corpus
- Apply the tokenizer on the *train* data and held out *validation* data and measure the proportion of tokens we can represent
- *Simplifying assumptions:*
  - We can remove punctuation and excess whitespace
  - So we're really just considering alphanumeric characters
  - We can use whitespace as a delimiter for possible tokens
  - Don't need to worry about sentence boundaries for now
- Starter and data in gDrive in “Activities” folder

## Coding Activity: Tokenize a Corpus

- "the": 5.71%; "and": 3.05%; "of": 2.84%; "to": 2.69%; "a": 1.99%
- "i": 1.83%; "in": 1.60%; "that": 1.31%; "he": 1.28%; "you": 1.23%
- **Fat tail** distribution; *most of the mass is infrequent tokens!*

**% Corpus that isn't UNK**

k	% Train	% Val Easy	% Val Hard
10	24	23	22
100	53	56	41
1000	78	79	60
10000	96	94	84

**% Unique tokens captured**

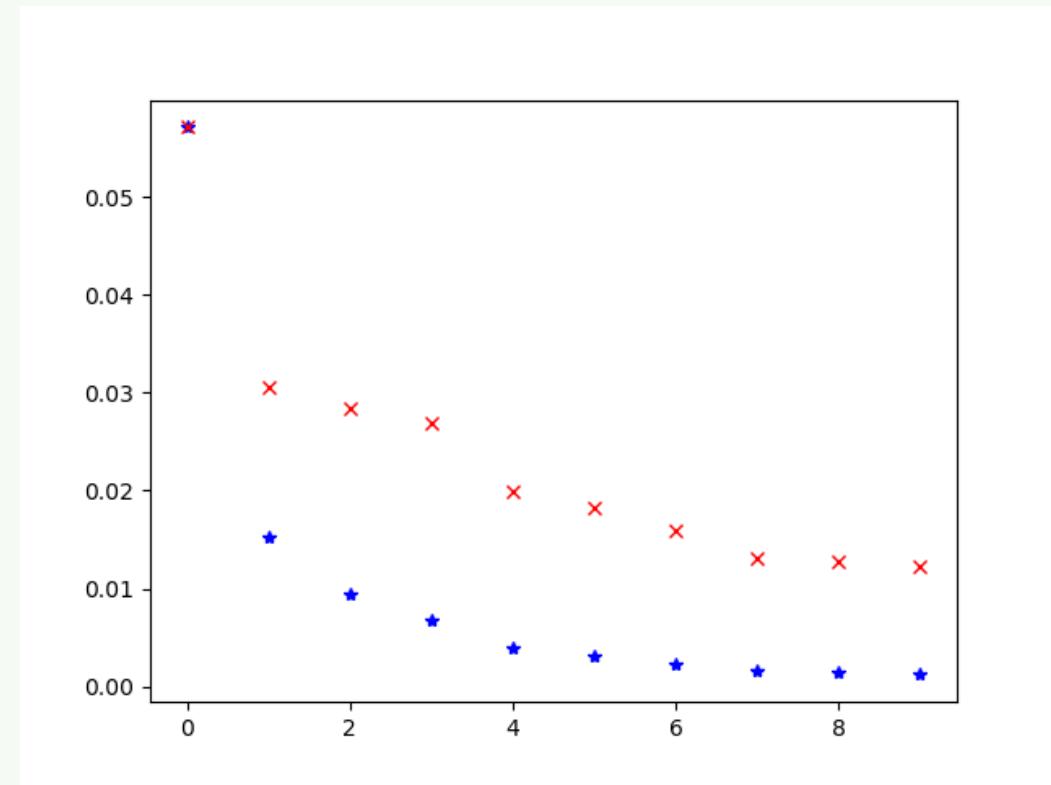
k	% Train
10	0.025
100	0.246
1000	2.46
10000	24.6

# Coding Activity: Tokenize a Corpus

- Always look at your data! What will the model see?
- k=10
  - UNK UNK UNK UNK a UNK UNK UNK UNK UNK of UNK of
  - UNK UNK a UNK of UNK UNK UNK UNK and
  - UNK i UNK UNK UNK UNK UNK UNK UNK UNK and UNK UNK
- k=100
  - UNK by UNK which a UNK UNK UNK were out of UNK of
  - UNK UNK a UNK of UNK UNK UNK UNK and
  - UNK i UNK it this UNK UNK UNK UNK UNK UNK and UNK have
- k=1000
  - UNK by UNK which a few hours UNK were out of sight of
  - things UNK a UNK of UNK UNK UNK UNK and
  - sir i UNK it this line looks far gone long UNK and UNK have
- k=10000
  - UNK by breezes which a few hours previous were out of sight of
  - things UNK a cluster of blocks generally painted green and
  - sir i mistrust it this line looks far gone long heat and wet have

## Coding Activity: Tokenize a Corpus

- “Fat tail” in our counts is thicker than a true Zipfian distribution, so the unknown token “problem” is worse in our little example than in expectation across a big corpus.



# ML FUNDAMENTALS: Vanishing Probabilities

- Recall our Naive Bayes' assumptions:
  - $p(x|y) \approx p(\varphi(x) | y)$  — approximate  $x$  by feature vector
  - $p(\varphi(x) | y) \approx \prod_{k=0 \dots K} (p(\varphi_k | y))$  — apply assumption for conditional independence of features given class label
- What may happen as we iteratively apply floating point multiplication during  $\{ \prod_{k=0 \dots K} (p(\varphi_k | y)) \}$ ?
  - Floating point representation bottoms out to true zero!

# ML FUNDAMENTALS: Vanishing Probabilities

- When we only care about the final *rank* of probabilities, we can apply a monotonically increasing function  $f$  that preserves:
  - $p(x_1) < p(x_2) \leftrightarrow f(p(x_1)) < f(p(x_2))$
- In ML code, we often use *log-likelihood*, ( $f=\log$ ), which preserves the order of real line in its image
  - $\prod_{k=0 \dots K} (p(\varphi_k|y)) < \prod_{k=0 \dots K} (p(\psi_k|y)) \leftrightarrow \sum_{k=0 \dots K} (\log(p(\varphi_k|y))) < \sum_{k=0 \dots K} (\log(p(\psi_k|y)))$
- These sums do not underflow the floating point representation!
- Now we've got  $[-\infty, 0]$  to work with instead of  $[0,1]$

# Feature Extraction for Language - Tokenization

"A tropical bird perches in the jungle."

Word-level

A      tropical      bird      perches      in      the      jungle

Stemming

A      tropic      bird      perch      in      the      jungl

BPE

A      tropic      ##al      bird      per      ##ch      ##es      in      the      jung

Character

A      t      r      o      p      i      c      a      l      b      i      r      d      p      e      r      c      h      e

+ Easy  
- Sparse  
 $|V|=|W|$

+ Less sparse  
- Lossy (POS)  
 $|V|=c|W|$

UNK!

+ Denser  
- Lossy  
- Inconsistent  
+ Super dense  
+ Consistent  
- Onus on model  
 $|V|=k$   
UNK preventable

# What text features are right for you?

- Our encoding function  $\phi$  will map raw text to *input features* for the model  $M$
- “It depends!”
  - On the task [character-level for language ID?]
  - On the model  $M$  [what if  $M$  is linear weights on token counts? Are frequent tokens still best to keep?]
  - On the text data itself [want word tokens on twitter?]
- Choosing how to tokenize your text is the *first* and most *impactful* step in your NLP pipeline
  - Your tokenization determines what  $M$  can represent.

# What other text features can we consider beyond tokens?

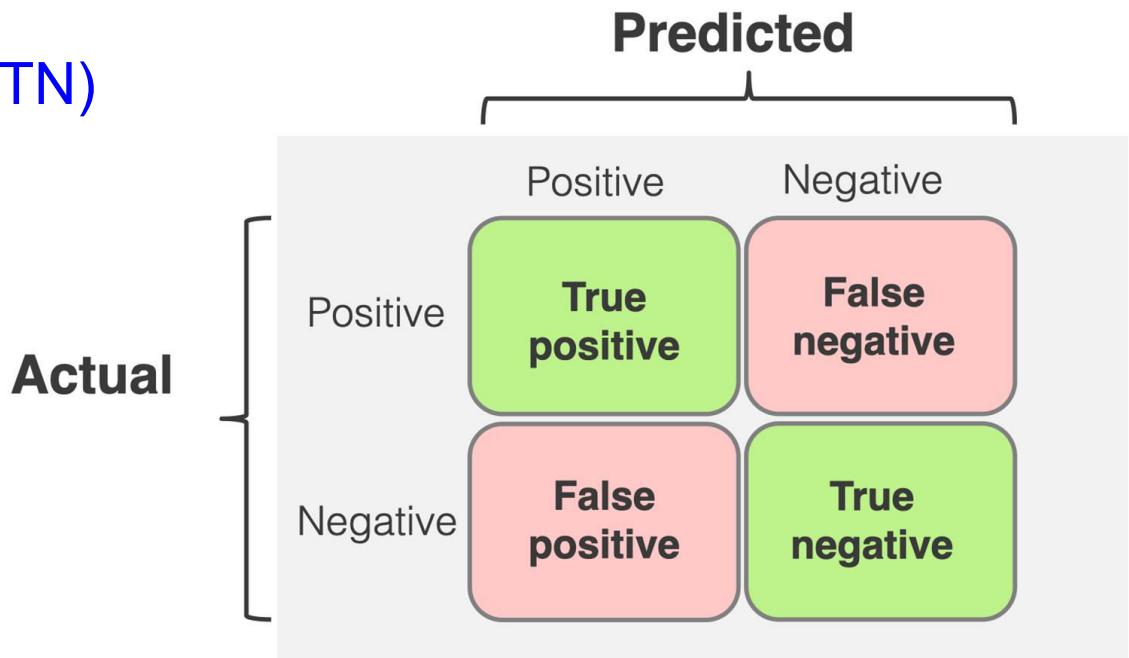
- What about features in  $[0, 1]$  or  $\mathbb{R}$ ?
- Combinations of words can be features (*bigram* features)
  - “fun” and “no” provide way less information than “no fun”
  - What’s an immediate problem with that feature space?
- Text also has *syntactic* and *semantic* structure!
  - Part-of-speech tags and semantic role labels

# Text Classification: Evaluation

- Given a binary text classification task, how do we measure the performance of model  $M$  parameterized by fitted  $\theta$  ?
- The classic classification terms:
  - True positives                    M says “yes”, true label is “yes”
  - False positives                  M says “yes”, true label is “no”
  - True negatives                  M says “no”, true label is “no”
  - False negatives                  M says “no”, true label is “yes”
- From these, we can define *accuracy*, *precision*, and *recall*

# Text Classification: Evaluation

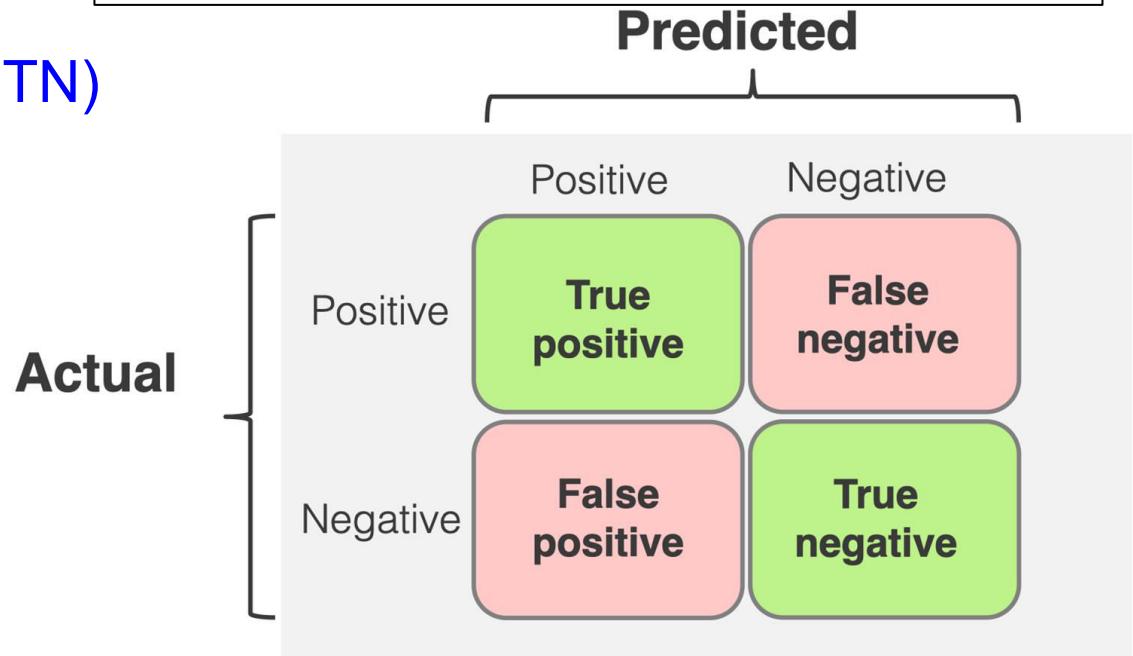
- Accuracy:
  - $(TP + TN) / (TP + FN + FP + TN)$
- Precision:
  - $TP / (TP + FP)$
- Recall:
  - $TP / (TP + FN)$



# Text Classification

- Accuracy:
  - $(TP + TN) / (TP + FN + FP + TN)$
- Precision:
  - $TP / (TP + FP)$
- Recall:
  - $TP / (TP + FN)$

- F-Measure:
  - $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$



# Text Classification: Homework 1 Hint & Sanity Check

- Instructor's implementation of Naive Bayes' with binary, bag-of-words features of tokens created by whitespace separation and punctuation removal achieves these evaluation values on the *validation* set with "pos" as "yes":

- k= 1000  
acc= .75  
precision= .92  
recall=

- k= 10000  
acc= .85  
precision= .89  
recall=

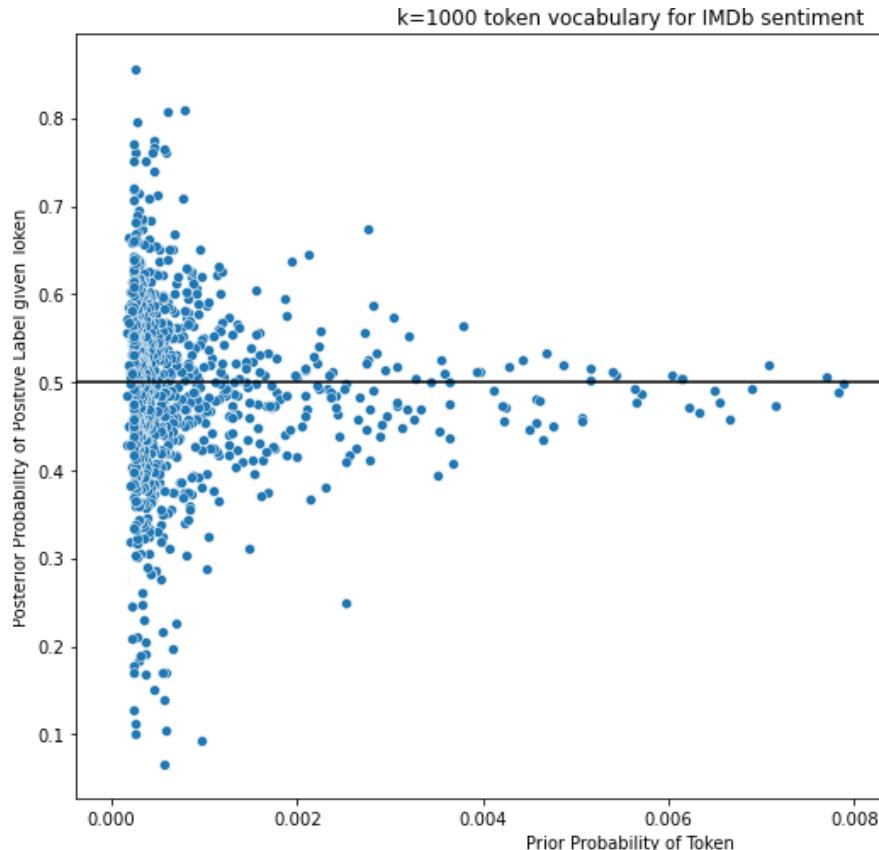
# Beyond the task, what are learning *about* tokens?

- With our Naive Bayes classifier on data like sentiment analysis, we learned two sets of parameters:
  - $\theta = \{\sim p(y)\}_{y \in Y} \cup \{\sim p(\phi(x)_k | y)\}_{k \in [1 \dots K]; y \in Y}$
- Recall that our goal was to infer the class given words:
  - $p(y|x)$
- We applied Bayes' Rule to rewrite this *posterior of the label given the words* as a generative model:
  - $p(y|x) = p(x|y)p(y) / p(x)$

# Beyond the task, what are learning *about* tokens?

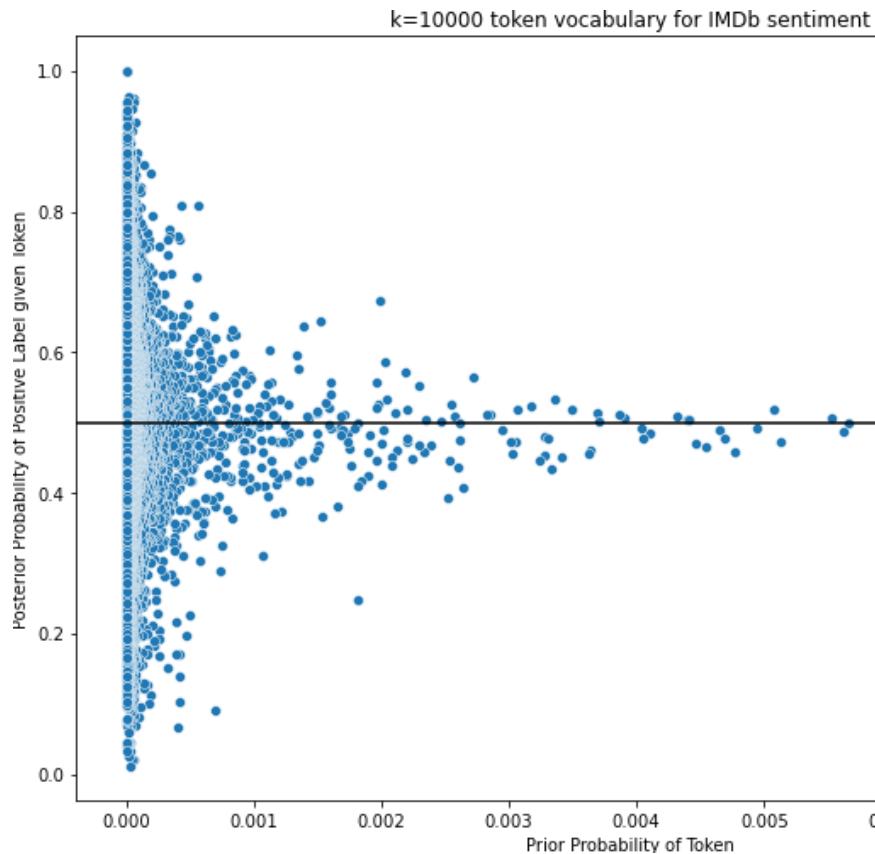
- Consider the complementary values:
  - $p(x)$ ;  $p(y|x)$
- These are the *prior* distribution of the input and posterior of the label given the input.
- If we tokenize  $x$  into a vocabulary  $V$ , we can consider:
  - $p(\phi_k)$ ;  $p(y|\phi_k)$  ; for  $\phi_k$  a token of  $V$
- These are the *prior* of a token appearing in an input text and the *posterior* of label  $y$  being applied if that token is in the input

# Beyond the task, what are learning *about* tokens?



Slice of extrema in "posterior_pos_given_v"			
Top-N	Token	PosPost	TknPrior
0	superb	0.8556	0.0003
1	excellent	0.8089	0.0008
2	wonderful	0.8079	0.0006
3	fantastic	0.7955	0.0003
4	amazing	0.7746	0.0005
...			
994	wasted	0.1282	0.0002
995	lame	0.1131	0.0003
996	awful	0.1038	0.0006
997	poorly	0.1011	0.0003
998	worst	0.0922	0.0010
Slice of extrema in "prior_v"			
Top-N	Token	PosPost	TknPrior
0	the	0.4995	0.0110
1	a	0.4998	0.0107
2	and	0.5040	0.0107
3	of	0.5008	0.0105
4	to	0.4952	0.0104
...			
994	season	0.6641	0.0002
995	island	0.4504	0.0002
996	gay	0.4288	0.0002
997	lee	0.4849	0.0002
998	joe	0.5724	0.0002

# Beyond the task, what are learning *about* tokens?



Slice of extrema in "posterior\_pos\_given\_v"

Top-N	Token	PosPost	TknPrior
0	prot	1.0000	0.0000
1	gunga	1.0000	0.0000
2	ponyo	1.0000	0.0000
3	prue	1.0000	0.0000
4	veronika	1.0000	0.0000
...			
9994	hobgoblins	0.0333	0.0000
9995	stinker	0.0321	0.0000
9996	uwe	0.0267	0.0000
9997	410	0.0220	0.0001
9998	310	0.0214	0.0000

Slice of extrema in "prior\_v"

Top-N	Token	PosPost	TknPrior
0	the	0.4995	0.0079
1	a	0.4998	0.0077
2	and	0.5040	0.0077
3	of	0.5008	0.0076
4	to	0.4952	0.0075
...			
9994	kolchak	0.6429	0.0000
9995	flavia	0.9231	0.0000
9996	dillinger	0.3077	0.3077
9997	sheba	0.3077	0.0000
9998	prue	1.0000	0.0000

# What features of tokens themselves can we consider?

- What does it mean for two words to be “close”?
- Can we think of one word as being “between” two others?
- Valence (emotion), intensity, syntactic role, semantic role, gender, morphology, count, ...
- How plausible is it to get supervised labels like these for the words in our vocabulary?
  - “Sentiment” classification task gave us  $p(\text{sentiment}|\text{word})$  as a side effect of a simple linear model
- Can the “meaning” of a word be characterized by a vector of features about that word, its morphology, its roles, etc.?

# A History of “Meaning” for AI in NLP

- Symbol Grounding
- Chomsky, Syntax, and Semantics
- The borders of written words and formalisms
- Taxonomic knowledge for text and vision

# A Brief Introduction to Grounding Natural Language

- Stevan Harnad's Symbol Grounding Problem brings together two camps of 1990s AI
  - Symbolic AI
  - Connectionism

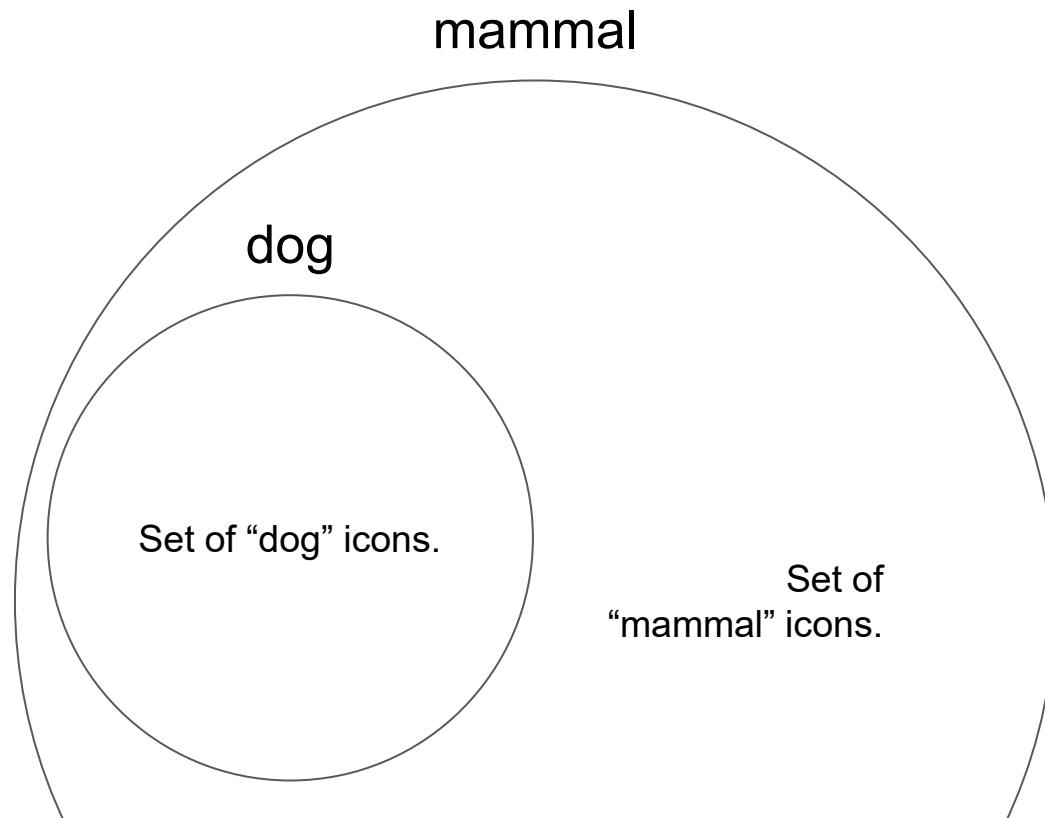
# Iconographic and Categorical Representations

- Iconographic Representation: a projection of a physical observation onto a “sensory surface”, e.g., the response of our retinas to images of dogs.
  - For simplicity, we'll call those images “instances”.
- Categorical Representation: “*invariant features* of the sensory projection that will reliably distinguish a member of a category from any nonmembers with which it could be confused.”
  - The invariant dog-ness of dogs across iconographic dogs

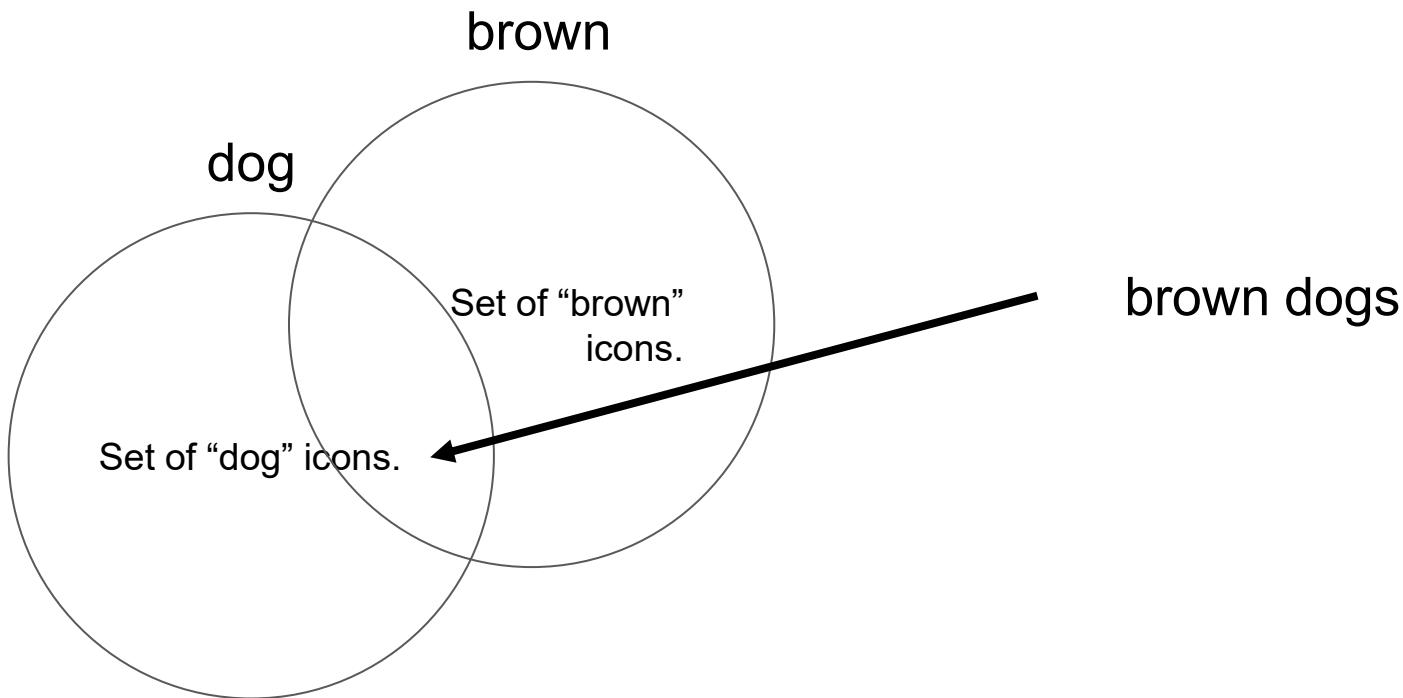
# Discrimination (Icons) versus Identification (Categories)



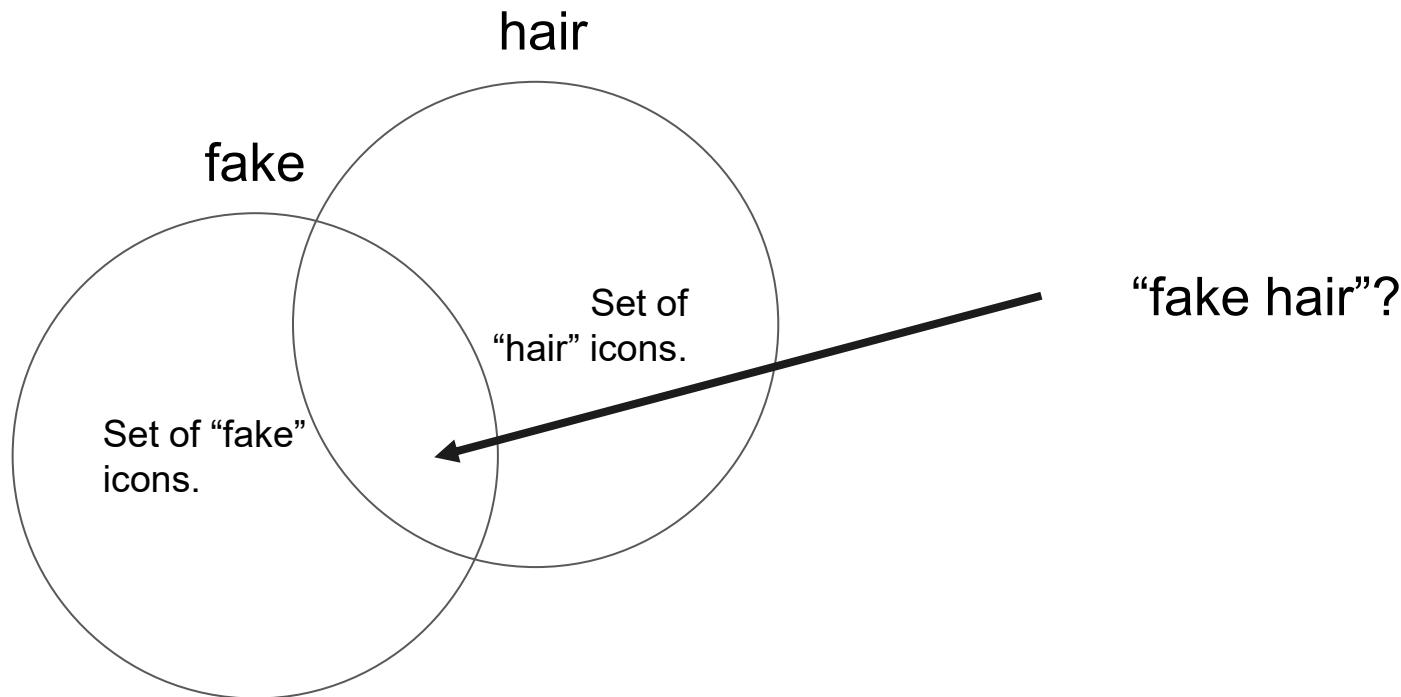
# Categorical Semantics Lite



# Categorical Semantics Lite

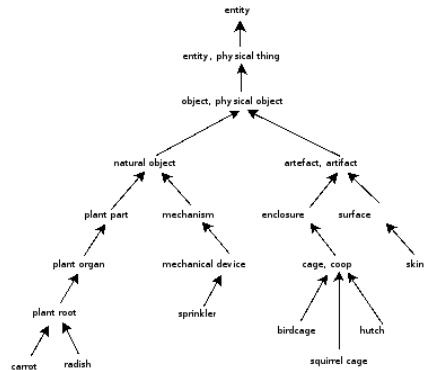


# Categorical Semantics Lite



# The Symbol Grounding Machine Harnad Envisioned

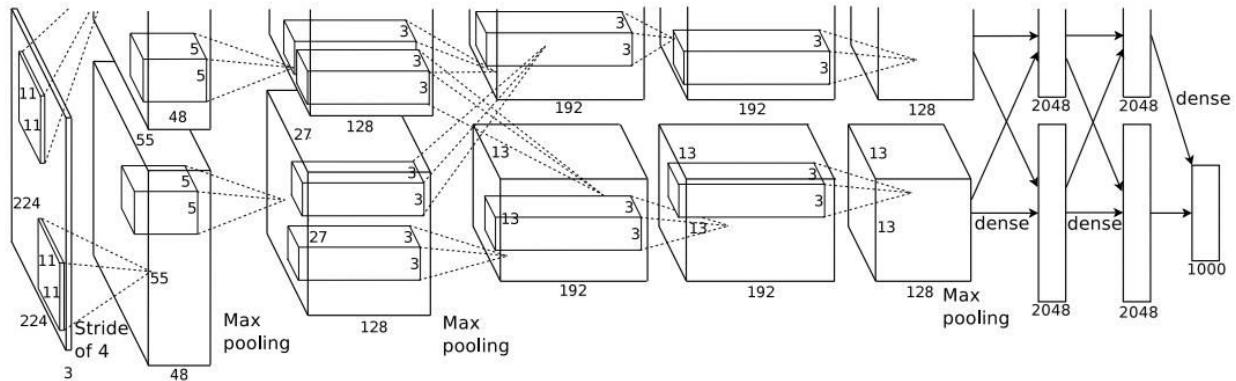
## WordNet (1995)



## ImageNet (2010)



## AlexNet (2012)



## The 1990 Prediction -> AlexNet

"Icons, **paired with feedback indicating their names**, could be processed by a connectionist network that learns to identify icons correctly..."

by dynamically adjusting the weights of the features and feature combinations that are reliably associated with the names in a way that (provisionally) resolves the confusion ...

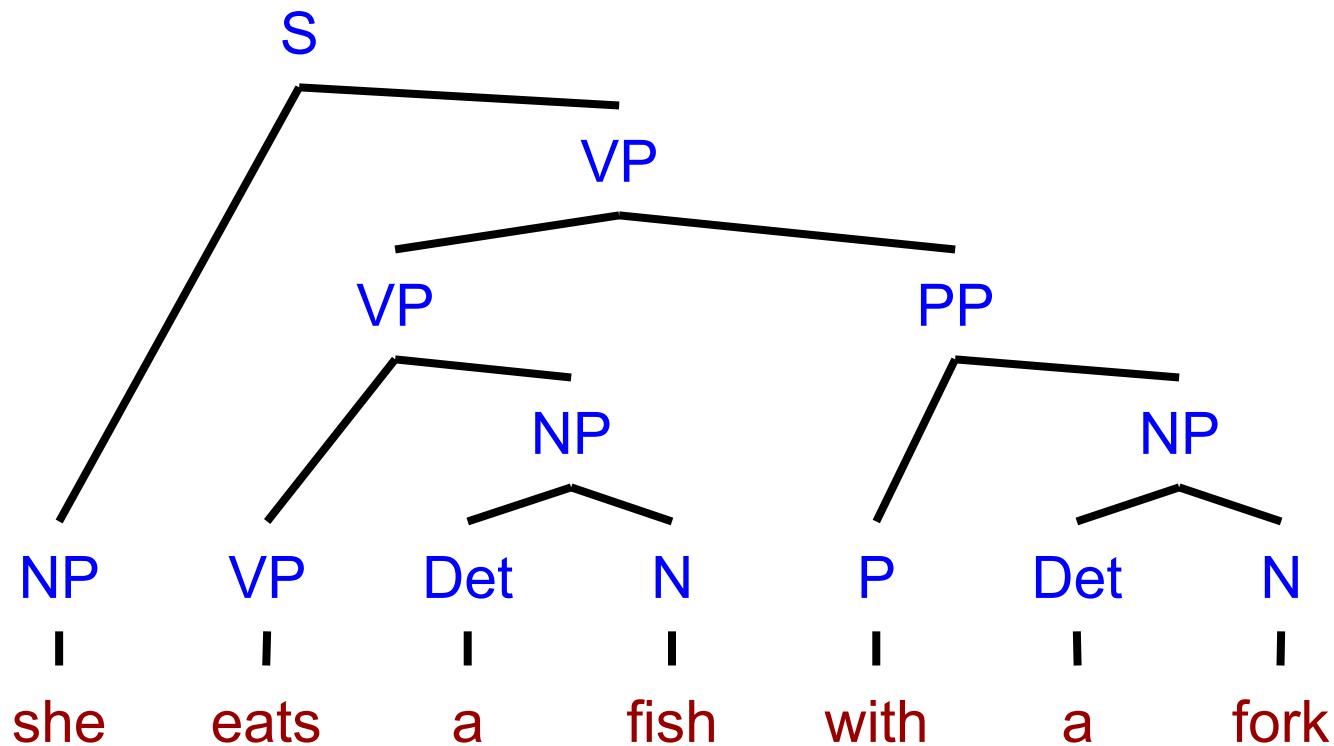
the "connection" between the names and the objects that give rise to their sensory projections and their icons would be provided by connectionist networks."

# Symbol Systems from Syntax and Semantics

- Chomsky: a biologically innate symbol system exists within humans from which we produce language.
  - “Universal Grammar”
    - (e.g., any person can learn any language)
  - “Generativism”: innate grammar leads to language systems whose rules describe all possible meaningful sentences.

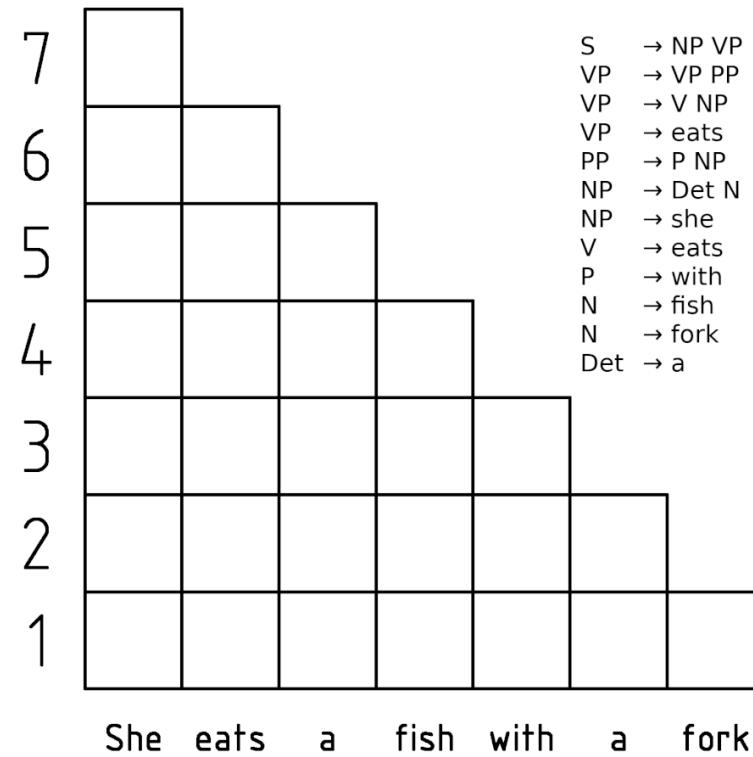
# Symbol Systems from Syntax and Semantics

S → NP VP  
VP → VP PP  
VP → V NP  
VP → eats  
PP → P NP  
NP → Det N  
NP → she  
V → eats  
P → with  
N → fish  
N → fork  
Det → a



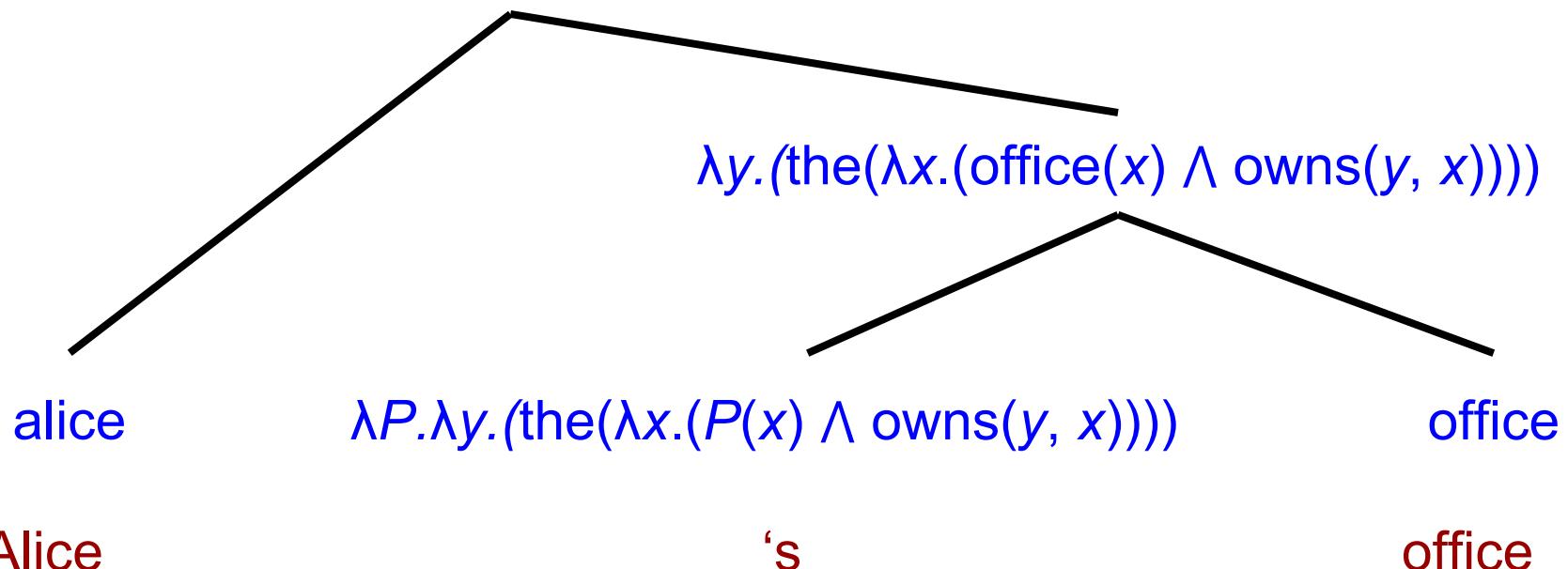
# Symbol Systems from Syntax and Semantics

S → NP VP  
VP → VP PP  
VP → V NP  
VP → eats  
PP → P NP  
NP → Det N  
NP → she  
V → eats  
P → with  
N → fish  
N → fork  
Det → a

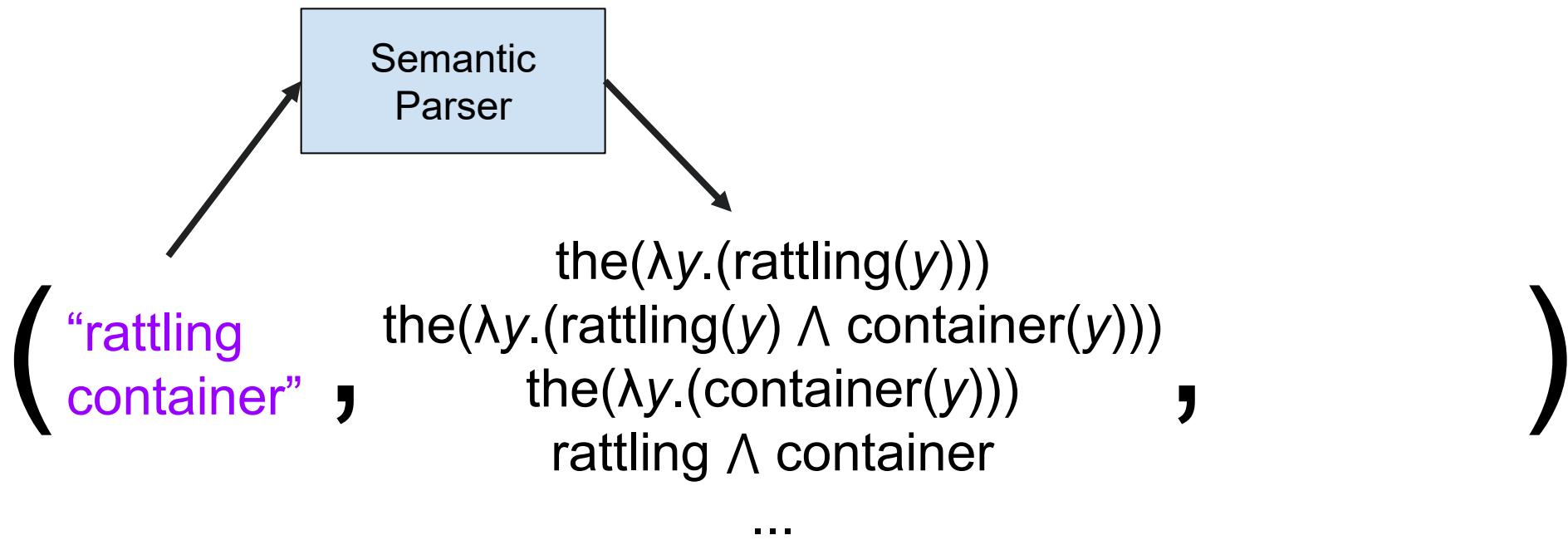


# Predicate Logic and Montague Grammar

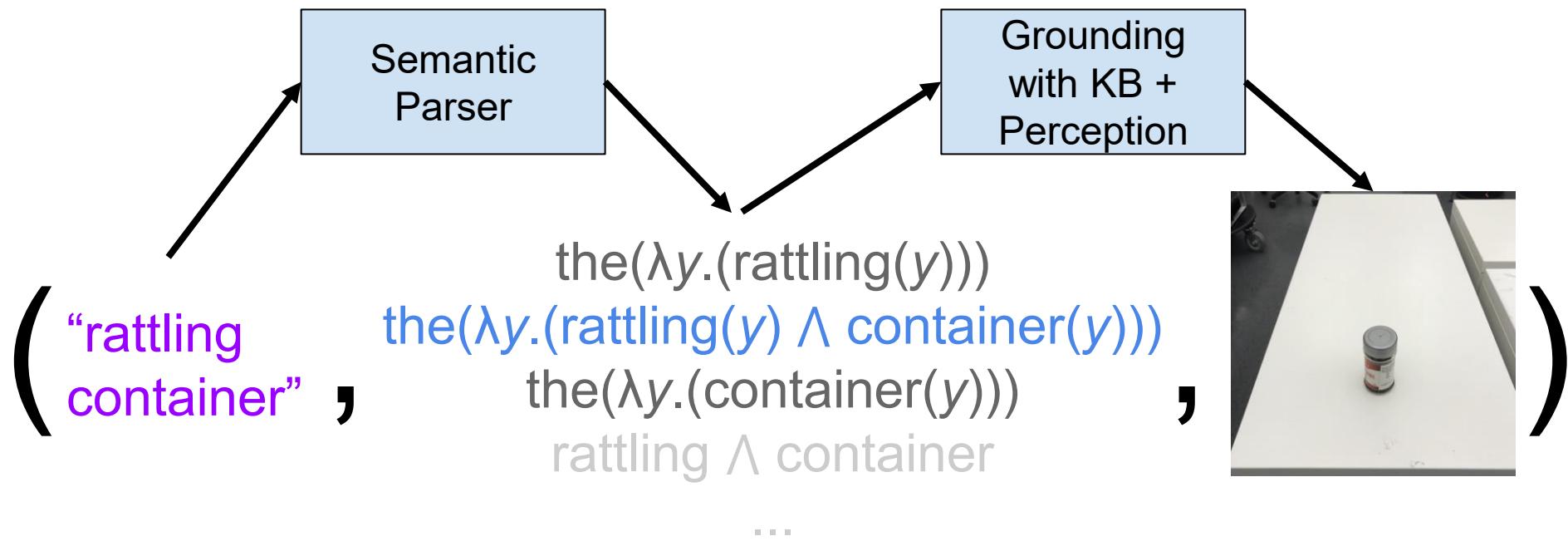
$\text{the}(\lambda x.(\text{office}(x) \wedge \text{owns(alice, } x)))$



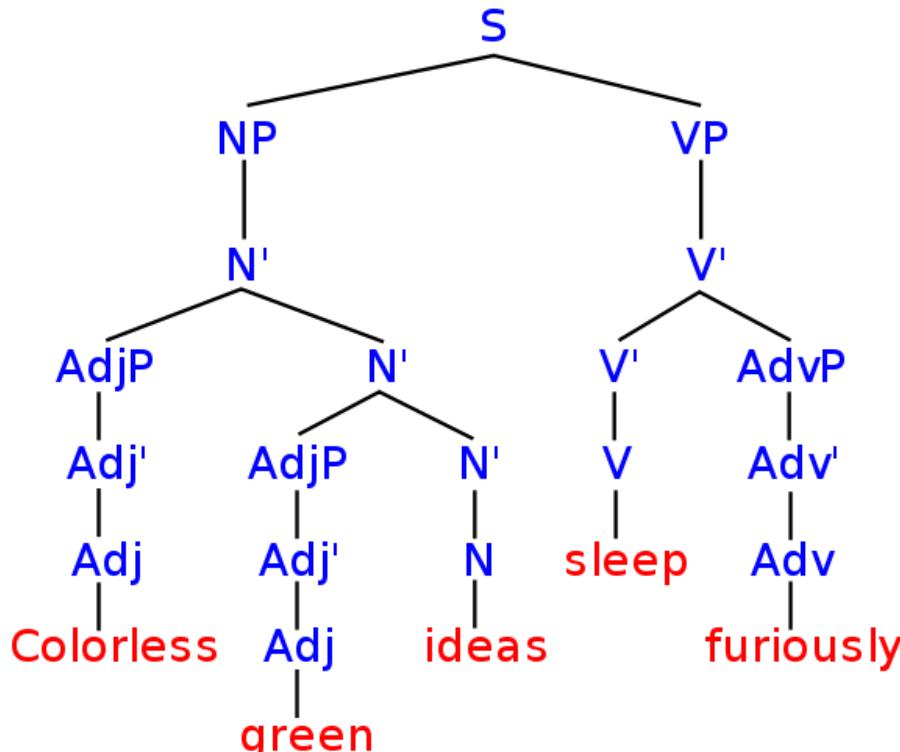
# Predicate Logic and Symbol Grounding



# Predicate Logic and Symbol Grounding



# Borders of Generative Grammars



Colorless green ideas sleep furiously.

- English grammar can certainly *produce* this sentence
- A statistical grammar could produce it.
- Would a person produce it on purpose?
- Does it have a “meaning”?

## *Colorless Green Ideas Sleep Furiously*

- A sentence has meaning if someone produces it.
- Humans are *very good* at forcing meanings.
- By posing this example, Chomsky basically guaranteed we'd find a way to make it seem less ridiculous.
- E.g., “colorless” can mean uninteresting or unremarkable; “green” can be “naive”; “sleep furiously” is almost “fester”
  - Unremarkable, naive ideas fester.
  - “He is a novice at statecraft and he knows it; his colorless green ideas sleep furiously.”

# Borders of Generative Grammars for *Text*

- She saw the man on the hill with the telescope.
  - PP attachment ambiguity / NN ambiguity
  - Prosody-based disambiguation
- Fetch me the light mug from the kitchen.
  - Polysemy; gesture disambiguation, potentially.

# Borders of Generative Grammars for Text

- Prosody-based emphasis can determine *pragmatic* meaning
- Lisa's not flying to Hawaii this weekend.
  - $\neg(\text{flying(lisa, hawaii, 22nd)})$
- LISA'S not flying to Hawaii this weekend.
  - $\neg(\text{flying(lisa, hawaii, 22nd)}) \wedge \exists x(\text{flying}(x, \text{hawaii}, 22nd))$
- Lisa's NOT flying to Hawaii this weekend.
  - $\neg(\text{flying(lisa, hawaii, 22nd)}) \wedge \exists x(\text{expects}(x, \text{flying(lisa, hawaii, 22nd)))}$
- Lisa's not FLYING to Hawaii this weekend.
  - $\neg(\text{flying(lisa, hawaii, 22nd)}) \wedge \exists P(P(\text{lisa, hawaii, 22nd}))$

# Borders of Generative Grammars for Text

- Lisa's not flying TO Hawaii this weekend.
  - $\neg(\text{flying}(\text{lisa}, \text{hawaii}, 22\text{nd})) \wedge \text{on}(\text{lisa}, \text{hawaii})$   
 $\wedge \exists x(\neg\text{hawaii}(x) \wedge \text{flying}(\text{lisa}, x, 22\text{nd}))$
- Lisa's not flying to HAWAII this weekend.
  - $\neg(\text{flying}(\text{lisa}, \text{hawaii}, 22\text{nd})) \wedge \exists x(\text{flying}(\text{lisa}, x, 22\text{nd}))$
- Lisa's not flying to Hawaii THIS weekend.
  - $\neg(\text{flying}(\text{lisa}, \text{hawaii}, 22\text{nd})) \wedge \exists x(\text{weekend}(x) \wedge \text{flying}(\text{lisa}, \text{hawaii}, x))$
- Lisa's not flying to Hawaii this WEEKEND.
  - $\neg(\text{flying}(\text{lisa}, \text{hawaii}, 22\text{nd})) \wedge \exists x(\text{flying}(\text{lisa}, \text{hawaii}, x) \wedge (x < 22\text{nd}))$

# Symbol Systems from Syntax and Semantics

- Chomsky: a biologically innate symbol system exists within humans from which we produce language.
  - “Universal Grammar”
    - (e.g., any person can learn any language)
  - “Generativism”: innate grammar leads to language systems whose rules describe all possible meaningful sentences.
    - Overscoped; also catches not meaningful sentences.
- Symbolic representations of the “meaning” of sentences.
  - While the “theory” accounts for prosody, most NLP does not!

# Predicate Logic and Montague Grammar

- Finch sees Doodle :  $\text{sees}(\text{Finch}, \text{Doodle})$
- A cat climbs a tree :  $\exists(x, y) ((\text{cat}(x) \wedge \text{tree}(y)) \wedge \text{climb}(x, y))$
- Cats chase birds :  $\forall(x, y) ((\text{cat}(x) \wedge \text{bird}(y)) \rightarrow \text{chase}(x, y))$
- Cats are mammals :  $\forall x (\text{cat}(x) \rightarrow \text{mammal}(x))$
- Doodle is a cat :  $\text{cat}(\text{Doodle})$
- Infer:  $\forall x (\text{bird}(x) \rightarrow \text{chase}(\text{Doodle}, x))$  : Doodle chases birds
- Infer:  $\text{mammal}(\text{Doodle}, x)$  : Doodle is a mammal

# Taxonomic Knowledge

- Hearst Patterns

(S1) The bow lute, such as the Bambara ndang, is plucked and has an individual curved neck for each string.

(1a)  $NP_0$  such as  $\{NP_1, NP_2 \dots, (\text{and} \mid \text{or})\} NP_n$

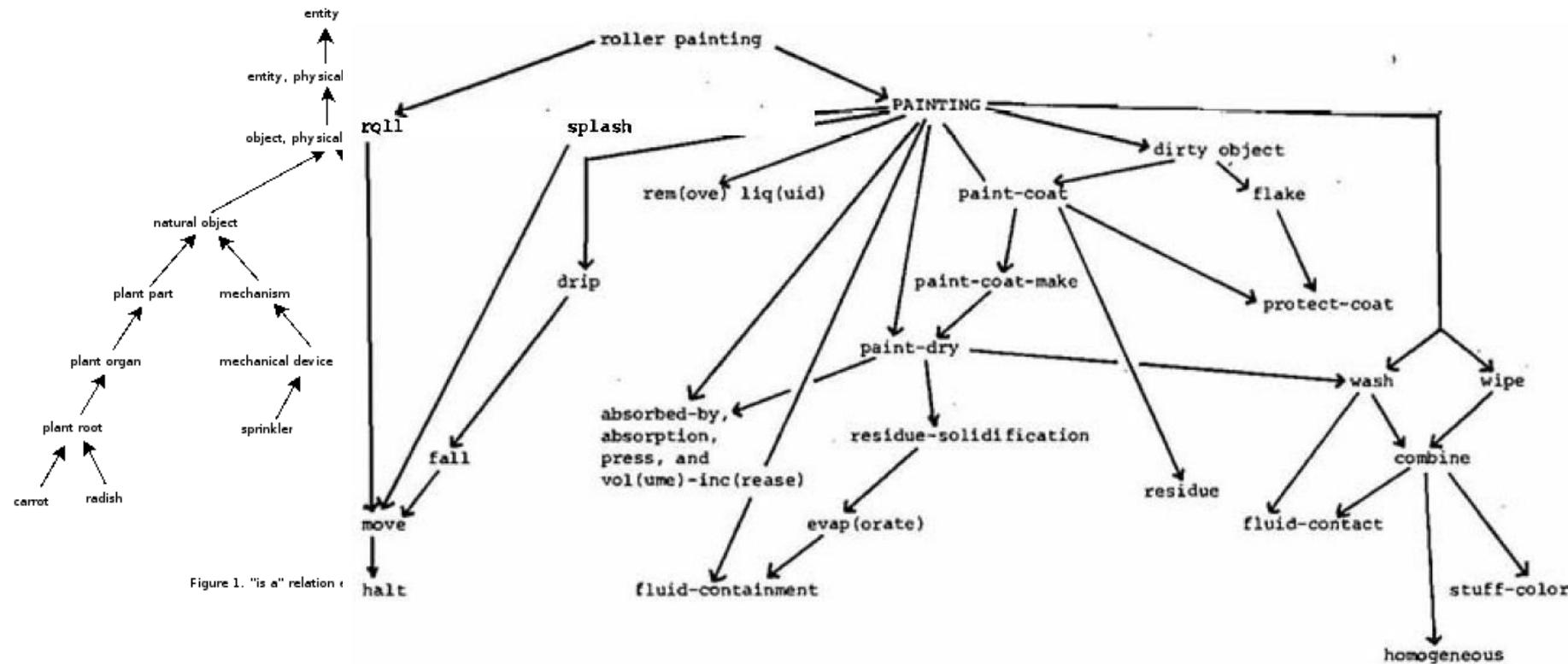
are such that they imply

(1b) for all  $NP_i$ ,  $1 \leq i \leq n$ ,  $\text{hyponym}(NP_i, NP_0)$

Thus from sentence (S1) we conclude

$\text{hyponym}(\text{"Bambara ndang"}, \text{"bow lute"}).$

# Knowledge Graphs and Knowledge Bases



# Knowledge Graph \*Nets

- [WordNet](#): hyponymy relationships
- [FrameNet](#): semantic frames; “slot-based” meaning
  - [\*\*VerbNet\*\*](#): more extensive coverage of verbs; only verbs
- [ConceptNet](#): Expansive knowledge graph between concepts, including motivations, causes and effects, etc.
  - [\*\*Related modern works like ATOMIC\*\*](#) are automatically curated and focus more on cause/effect
- With some asterisks, these resources are *hand crafted* by a mix of paid and expert annotators!

# Semantics with Frame Parsing

- Identify a frame that fits a sentence and assign all its constituents:
  - “The wolves devoured the carcass completely”
    - Frame: Ingestion
    - Ingestor: wolves
    - Ingestibles: carcass
    - Degree: completely

# CSCI: 699: History of Language and Computation

- If you're interested in really diving into historical paradigms for representing word and sentence meaning
- Reading and writing intensive course; optional programming via the course project
- Consider enrolling in Spring 2025 if piqued by the ideas in these last few slides and the development of NLP over the historical timeline more generally
  - Spoiler: Machine Translation drove almost everything!
- Spring 2024 syllabus:
  - [https://jessethomason.com/class\\_material/csci\\_699\\_history\\_of\\_language\\_and\\_computation\\_spring\\_2024.pdf](https://jessethomason.com/class_material/csci_699_history_of_language_and_computation_spring_2024.pdf)

# What features of tokens themselves can we consider?

- Can the “meaning” of a word be characterized by a vector of features about that word, its morphology, its roles, etc.?
- Maybe! But resources are limited, and the volume of *unannotated* text dwarfs *annotated* text and continues to grow
- We will discuss both *supervised* and *unsupervised* methods for representing word and sentence meaning in this course
  - In general, *supervised* methods have laid the groundwork for the intuitions of *unsupervised* methods in NLP
  - As data and compute grow, *unsupervised* methods can be deployed to approximate their *supervised* ancestors

# Why would we want “meaning” of a token?

- Reformulating our positive/negative classification task from Naive Bayes earlier, we got two properties of each token:
  - Frequency ( $p(\text{tkn})$ ) and chance of showing up indicating the review text is positive ( $p(\text{positive}|\text{tkn})$ )
- Say we have a few such numerical properties of each token, call it a *word vector*  $\mathbf{v}$  for each token  $v$
- What kind of model could we use to still do a classification task on such word vectors?
- We'd need a way to bring them together to form a meaning for the whole input  $x$ ... This will be the focus of Module 2!

# What features of tokens themselves can we consider?

- But for now, if we want “features” of each token, what features can we consider if all we get is input texts  $x \in X$  ?
- What if the presence or absence of every word in the vocabulary were considered a “label”?
- $p(w_i|w_j)$  — The posterior likelihood that token  $w_i$  is in my input text given that token  $w_j$  is in the input text
  - $p(\text{"dog"}|\text{"cat"}) > p(\text{"truck"}|\text{"cat"})$ ; features of “cat”
  - $p(\text{"compete"}|\text{"science"}) > p(\text{"study"}|\text{"science"})$
- These  $n$ -gram features provide a basis for both learning *word meaning* and for general *language modeling*; next time!

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 1: Lecture 1  
Text Processing & ML 101 for NLP

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 1: Lecture 2  
N-Grams & DL 101 for NLP

# What features of tokens themselves can we consider?

- What does it mean for two words to be “close”?
- Can we think of one word as being “between” two others?
- Valence (emotion), intensity, syntactic role, semantic role, gender, morphology, count, ...
- How plausible is it to get supervised labels like these for the words in our vocabulary?
  - “Sentiment” classification task gave us  $p(\text{sentiment}|\text{word})$  as a side effect of a simple linear model
- Can the “meaning” of a word be characterized by a vector of features about that word, its morphology, its roles, etc.?

# What features of tokens themselves can we consider?

- We want “features” of each token; what features can we consider if all we get is input texts  $x \in X$  ?
- What if the presence or absence of every word in the vocabulary were considered a “label”?
- $p(w_i|w_j)$  — The posterior likelihood that token  $w_i$  is in my input text given that token  $w_j$  is in the input text
  - $p(\text{"dog"}|\text{"cat"}) > p(\text{"truck"}|\text{"cat"})$ ; features of “cat”
  - $p(\text{"compete"}|\text{"science"}) > p(\text{"study"}|\text{"science"})$
- These  $n$ -gram features provide a basis for both learning *word meaning* and for general *language modeling*

# NLP FUNDAMENTALS: Word Embeddings

- A word embedding is a *vector representation* of a word
- Word embeddings can be learned from self-supervised objectives by leveraging the *distributional hypothesis*
- “**You shall know a word by the company it keeps**” - Firth
- The guiding principle of many learned word embeddings is that two words who share similar *context* should have embeddings that are close together in vector space
  - We’ll discuss notions of closeness later.

# Word Embeddings: Co-occurrence matrix

- When we wanted to represent  $|V|$  words used to classify IMDb sentiment, we estimated  $p(w|y)$  from data
  - Think of this as a one-dimensional word embedding
- We can do the something similar for the *entire vocabulary* as the “label” space, creating a size  $|V|$  word vector for each word
- “You shall know a word by the company it keeps”
- Let’s given each word  $w$  a vector of dimension  $|V|$  that represents the distribution of words that occur *near* to  $w$
- Near could be same document, came before, within  $k$  words...

# Word Embeddings: Co-occurrence matrix

$p(w_i|w_{i-1})$

	the	students	learned	about	word	vectors	today	in	class
the	0	0	0.05	0.1	0	0	0.05	0.1	0
students	0.1	0	0	0.1	0	0	0.05	0.05	0
learned	0	0.2	0	0	0.05	0.05	0.05	0	0.2
about	0	0.05	0.2	0	0	0.05	0.1	0.2	0.1
word	0.1	0	0.1	0.1	0	0	0.05	0.05	0
vectors	0.1	0	0.2	0.2	0.4	0	0.05	0.05	0
today	0	0.05	0.1	0	0	0	0	0.05	0.05
in	0	0.1	0.1	0.05	0.05	0.05	0.2	0	0.05
class	0.1	0	0.05	0.1	0.05	0	0	0.2	0

# NLP FUNDAMENTALS: Language Models

- A *language model* is an estimate of the likelihood of a string
- Tokenizing the string into component parts is necessary
- So we can consider a string  $s$  as a sequence of tokens  $w_1 \dots w_n$
- Then a language model estimates:
  - $p(s) = p(w_1 \dots w_n)$
- Term is overloaded; also used these days to mean:
  - $p(w_k | w_1 \dots w_{k-1})$  — (e.g., *auto-regressive*)
  - $p(w_k | w_1 \dots w_{k-1} w_{k+1} \dots w_n)$  — (e.g., *cloze / MLM*)

# Uses of a Language Model

- If we have a function  $p(s) = p(w_1 \dots w_n)$  that assigns probabilities to a sequence of word tokens, what can we use it for?
  - *Generation*: we could sample from this LM to get new strings that the LM deems plausible
    - Given “The cat climbed the”, what could come next?
  - *Scoring*: given a set of strings, we could use the LM to score their plausibility (for example, speech-to-text)
    - Given “The cat climbed the free” versus “The cat climbed the tree”, what’s more likely to have been said?

# $N$ -gram language models

- Let's focus on *left context* models (e.g., score likelihood of following words given previous words)
- To factor word by word, then:
  - $p(s) = p(w_1 \dots w_n)$
  - $= p(w_n | w_1 \dots w_{n-1})p(w_1 \dots w_{n-1})$ 
    - In other words, the probability of the next word is equal to its chance given the words so far compounded by the likelihood of that sequence
- What would it take to store  $p(w_n | w_1 \dots w_{n-1})$  on disk?

# $N$ -gram language models

- How could we efficiently estimate  $p(w_n | w_1 \dots w_{n-1})$ ?
- Does a word  $k$  away in the history really affect what word comes next that much? What if we assume:
  - $p(w_n | w_1 \dots w_{n-1}) \approx p(w_n | w_{n-k+1} \dots w_{n-1})$
- For example, if we say “a word 4 away in the history does not affect me”, then  $p(w_n | w_1 \dots w_{n-1}) \approx p(w_n | w_{n-3} w_{n-2} w_{n-1})$
- How big on disk?
  - $|V|^*|V|^*|V|^*|V| = |V|^4$

# Word Embeddings: $n$ -gram Co-occurrence matrix

$p(w_i|w_{i-1})$

	the	students	learned	about	word	vectors	today	in	class
the	0	0	0.05	0.1	0	0	0.05	0.1	0
students	0.1	0	0	0.1	0	0	0.05	0.05	0
learned	0	0.2	0	0	0.05	0.05	0.05	0	0.2
about	0	0.05	0.2	0	0	0.05	0.1	0.2	0.1
word	0.1	0	0.1	0.1	0	0	0.05	0.05	0
vectors	0.1	0	0.2	0.2	0.4	0	0.05	0.05	0
today	0	0.05	0.1	0	0	0	0	0.05	0.05
in	0	0.1	0.1	0.05	0.05	0.05	0.2	0	0.05
class	0.1	0	0.05	0.1	0.05	0	0	0.2	0

# $N$ -gram language models

- $k=3$ , trigram LM
  - $p(w_n|w_1\dots w_{n-1}) \approx p(w_n|w_{n-2}w_{n-1})$
  - $|V|^*|V|^*|V| = |V|^3$
- $k=2$ , bigram LM
  - $p(w_n|w_1\dots w_{n-1}) \approx p(w_n|w_{n-1})$
  - $|V|^*|V| = |V|^2$
- $k=1$ , unigram LM
  - $p(w_n|w_1\dots w_{n-1}) \approx p(w_n)$
  - $|V|$

# $N$ -gram language models

- What issues do we face as  $k$  decreases?  $p(w_n)$ 
  - Loss of information and context; model is dumber
- What issues do we face as  $k$  increases?  $p(w_n|w_{n-2}w_{n-1})$ 
  - Disk space is *exponential* in  $k$
  - *Sparsity*: we would assign a score of zero to an unseen sequence of three words
    - How common will unseen sequences be at test time?
      - “It depends”, but probably “very”
    - How could we handle this without doing much work?

# $N$ -gram language models

- As  $k$  increases, so does the sparsity of our model
- Distribute a probability  $\varepsilon$  uniformly across *each* row in our probability table; this is called *smoothing*
  - $p(w_n|w_{n-2}w_{n-1}) = (1-\varepsilon)[c(w_n|w_{n-2}w_{n-1}) / c(w_{n-2}w_{n-1})] + [\varepsilon / |V|]$
- Triplets that are never seen?
  - $c(w_n|w_{n-2}w_{n-1})=0$ , total probability  $p(w_n|w_{n-2}w_{n-1}) = \varepsilon / |V|$
- Triplets that are seen?
  - Just lose  $\varepsilon$  to be distributed out to unseen triplets.

# $N$ -gram language models: Scoring Strings; Perplexity

- We score a string  $T$  of tokens  $t_i$  using our LM based on a measure called *perplexity*: roughly, how “surprised” the model is by each subsequent token
- Surprise at token  $t_i$ :  $s(t_i) = -\log(LM(t_i))$ 
  - $s(t_i) = 0$  when  $LM(t_i)=1$
  - $s(t_i) = \infty$  when  $LM(t_i)=0$
- Surprise across the entire sequence? **Cross-Entropy!**
  - $H(T) = -\sum_{t_i \in T} p^*(t_i) \log(\sim p(t_i))$ ;  $p^*$  true,  $\sim p$  predicted
- *Perplexity* is defined as the base-2 exp of entropy:  $PP(T)=2^{H(T)}$

$$H(P^* | P) = - \sum_i \frac{P^*(i)}{\text{TRUE CLASS DISTRIBUTION}} \log \frac{P(i)}{\text{PREDICTED CLASS DISTRIBUTION}}$$

# $N$ -gram language models: Scoring Strings; Perplexity

- Surprise across the entire sequence? **Cross-Entropy!**
  - $H(T) = -\sum_{t_i \in T} p^*(t_i) \log(\sim p(t_i))$ ;  $p^*$  true,  $\sim p$  predicted
- *Perplexity* is defined as the base-2 exp of entropy:  $PP(T)=2^{H(T)}$
- How can we get  $p^*$ , the *true distribution of n-grams across the language*?
  - We can't!
- We can estimate the entropy as
  - $H(T) \approx -(1/|V|) \log(\sim p(T))$
- There is an information-theoretical reason for this ↓

# $N$ -gram language models: Scoring Strings; Perplexity

- *Perplexity* is defined as the base-2 exp of entropy:  $PP(T)=2^{H(T)}$ 
  - Recall that entropy is the number of bits  $N$  needed to efficiently represent which event from a probability distribution happened (do we need the whiteboard?)
  - Then  $2^n$  is the number of choices those bits can represent
  - Remember our examples;  $H+1$  when possibilities doubled

# $N$ -gram language models: Recap

- Represent the likelihood of the next word token given a history of the last  $n$  words ( $n=0$ , unigram;  $n=1$ , bigram; ...)
- Include a small but non-zero probability of transitioning to a word never seen at training time given history (*smoothing*)
- **Scoring:** given the language model, can calculate the *perplexity* of a given string to see how confusing it is to the LM
- **Generation:** given the language model, we can sample next tokens given previous tokens [much more on this later]
- Scoring and generation are tightly related to seq2seq; trust me

# Coding Activity: $n$ -Gram Language Models

- Let's explore  $n$ -gram language models using our books corpus
- We will investigate the relationship between a few hyperparameters and final perplexity:
  - $|V|$ , how many vocabulary tokens to represent
  - $n$ , how big a co-occurrence table to build
  - $\varepsilon$ , how much to smooth out probability mass
- Check the Coding Activities gDrive folder
  - There is also an editable slide which we'll collaborate to fill out

# Coding Activity: $n$ -Gram Language Models

- Before running the code, scan through and try to answer the following questions:
  - Q1) How are strings turned into sequences of tokens? How will that affect our language model?
  - Q2) What is the algorithm the code uses to create  $p(w_i|w_{i-1} \dots w_{i-k})$ ? What assumptions are made?
  - Q3) Is it clear yet why we wouldn't want to tokenize with both our train+val data? What would be “lost” to the model if we were to do that?

# Coding Activity: $n$ -Gram Language Models

- A sample run:  $|V|=1,000$ ;  $n=2$ ;  $\epsilon=0.001$ 
  - Perplexity: 1.0502
- High Perplexity:
  - <start> coming into the room till both his arms were round me and he was <end> [1.0998]
  - <start> not wish anyone to stay with me a day or two of rest will set me up <end> [1.1050]
  - <start> that who the man is why he died how he died are <unk> that do not <end> [1.1029]
- Low Perplexity:
  - <start> <unk> <end> [1.0076]
  - <start> <unk> <end> [1.0076]
  - <start> <unk> <end> [1.0076]

# Coding Activity: $n$ -Gram Language Models

Varying Vocab Size

Vocab $ V $	Perplexity

$n=2; \epsilon=0.001$

Varying  $n$ -Gram Len

$n$ -Gram	Perplexity
1	
2	
3	

$|V|=10,000; \epsilon=0.001$

1	
2	
3	

$|V|=1,000; \epsilon=0.001$

Varying Smoothing

$\epsilon$	Perplexity

$|V|=10,000; n=2$

# Coding Activity: $n$ -Gram Language Models

- What if  $\varepsilon=1$ ?
- Perplexity:  $PP(T)=2^{H(T)}$ 
  - $H(T) = -(1/|V|)\log(\sim p(T))$
- If  $\varepsilon=1$ ,  $p(t_i)=1 / |V|$ 
  - $H(T) = -(1/|V|)\log((1 / |V|)^L)$
- Say  $|V|=1,000$ . In our code  $L=21$ , then
  - $H(T) = -(1/1000)\log((1 / 1000)^{21})$
  - $H(T) = 0.209$
- $PP(T) = 1.156$ 
  - This perplexity doesn't seem that bad! Why?

# Coding Activity: *n*-Gram Language Models

- Hints for what comes next...
- How would you *generate text* from the n-gram model here?
- Are any pieces missing?

# Coding Activity: *n*-Gram Language Models

- What's *unsatisfying* about this n-gram model?
  - Using book “lines” instead of “sentences”
  - Is there something better we can do than smoothing probabilities for, for example, an unseen trigram?
    - Hint: what if we haven’t seen (ABC) but we’ve seen (AB) and (BC)?
  - Take a look at the output “Prob table size … zero entries”
    - Maybe words aren’t the right co-occurrence? POS?
    - Maybe there’s a *latent subspace* words can embed in…

# Word Embeddings: $n$ -Gram Model

- An  $n$ -gram model represents the context of each word  $w$  as its co-occurrence with other words 1-, 2-, ...,  $n-1$ -steps away
- $n$ -Grams are based on rigid, explicit counting, plus some smoothing to relax that rigidity
- How else could we relax  $p(w_i|w_{i-1}w_{i-2}\dots w_{i-n+1})$ 
  - Previously, we kept a  $|V|$  count for each distance  $i-k$ 
    - We could just keep a  $|V|$  count that merges distances
    - “Bag of Words”-based model

# Word Embeddings: Bag-of-Words Model

- Estimate  $p(w_i|w_{i-1}w_{i-2}\dots w_{i-n+1})$  as distribution of  $w_i$  given that  $w_{i-1}w_{i-2}\dots w_{i-n+1}$  occur *in any order* left of  $w_i$ 
  - Pros:
    - Space to store this model is  $|V|^2$
    - Model will be considerably less sparse
  - Cons:
    - Loss of order information will hurt POS guessing / syntax generally
  - Because the *context* from which we predict  $w_i$  is now a single vector, what other interesting information is here?

# Word Embeddings: Bag-of-Words Model

	<b>&lt;S&gt;</b>	<b>a</b>	<b>b</b>	<b>&lt;e&gt;</b>
<b>&lt;S&gt;</b>	0	0	0	0
<b>a</b>	2	0	4	0
<b>b</b>	2	4	4	0
<b>&lt;e&gt;</b>	0	2	2	0

- Context vectors:
  - s: <4, 0, 0, 0, 0>
  - a: <0, 2, 0, 4, 0>
  - b: <0, 2, 4, 4, 0>
  - e: <0, 0, 2, 2, 0>
- Corpus:
  - <s>abba<e>
  - <s>bbab<e>
  - <s>babb<e>
  - <s>abba<e>
- What does the *magnitude* of the context vector show?
- What does the *direction* of the context vector show?

# Word Embeddings: Bag-of-Words Model

- Context vectors:

- s: <4, 0, 0, 0, 0>
- a: <0, 2, 0, 4, 0>
- b: <0, 2, 4, 4, 0>
- e: <0, 0, 2, 2, 0>

- Corpus:

- <s>abba<e>
- <s>bbab<e>
- <s>babb<e>
- <s>abba<e>

- Magnitude corresponds to word frequency

- Direction represents the *context* in  $\mathbb{R}^{|V|}$

- Cosine similarity measures  
Directional distance

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

# Word Embeddings: Bag-of-Words Model

- Context vectors:
  - s: <4, 0, 0, 0, 0>
  - a: <0, 2, 0, 4, 0>
  - b: <0, 2, 4, 4, 0>
  - e: <0, 0, 2, 2, 0>
- Corpus:
  - <s>abba<e>
  - <s>bbab<e>
  - <s>babb<e>
  - <s>abba<e>
- $\cos(s, a) = 0; \cos(s, b) = 0; \cos(s, e) = 0$
- $\cos(a, b) = 0.745$
- $\cos(a, e) = 0.632$
- $\cos(b, e) = 0.943$

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

# Word Embeddings: Bag-of-Words Model

- Estimate  $p(w_i|w_{i-1}w_{i-2}\dots w_{i-n+1})$  as distribution of  $w_i$  given that  $w_{i-1}w_{i-2}\dots w_{i-n+1}$  occur *in any order* left of  $w_i$
- Because the *context* from which we predict  $w_i$  is now a single vector, what other interesting information is here?
  - We've been thinking about LM from just one perspective
    - “What *next token* is likely given the *context* so far?”
  - We could ask something else...
    - “What *context* will make a *particular token* most likely?”

# Word Embeddings: Bag-of-Words Model

- We've been thinking about LM from just one perspective
  - “What *next token* is likely given the *context* so far?”
    - Gives us a way to reason about what comes next given the *context* so far
- We could ask something else...
  - “What *context* will make a *particular token* most likely?”
    - Gives us a sense of which tokens are *similar* based on sharing similar favored *contexts* that produce them!
    - “You shall know a word by the company it keeps.”

# Word Embeddings

- What *context* will make a *particular token w=a* most likely?
  - $\text{argmax}_{w(i-k) \in V} (p(w_i=a | w_{i-1} w_{i-2} \dots w_{i-n+1}))$
- Let's apply Bayes' Theorem:
  - $p(a | w_{i-1} w_{i-2} \dots w_{i-n+1})$
  - $= p(w_{i-1} w_{i-2} \dots w_{i-n+1} | a) p(a) / p(w_{i-1} w_{i-2} \dots w_{i-n+1})$
- Then we want:
  - $\text{argmax}_{w(i-k) \in V} (p(w_{i-1} w_{i-2} \dots w_{i-n+1} | a) p(a) / p(w_{i-1} w_{i-2} \dots w_{i-n+1}))$
  - $\text{argmax}_{w(i-k) \in V} (p(w_{i-1} w_{i-2} \dots w_{i-n+1} | a) / p(w_{i-1} w_{i-2} \dots w_{i-n+1}))$
- The *context* that maximizes the chance token *a* comes next is the likelihood of the context given *a* by the marginal probability of the context itself

# Word Embeddings

- The *context* that maximizes the chance token  $a$  comes next is the likelihood of the context given  $a$  by the marginal probability of the context itself
  - $\text{argmax}_{w(i-k) \in V} (p(w_{i-1} w_{i-2} \dots w_{i-n+1} | a) / p(w_{i-1} w_{i-2} \dots w_{i-n+1}))$
- Let's consider a bigram model:
  - $\text{argmax}_{w(i-1) \in V} (p(w_{i-1} | a) / p(w_{i-1}))$
  - Intuitively: the word that frequently occurs before the target word  $a$ , and seldom occurring without  $a$
- If our *context* is a *multi-hot* vector instead of a *one-hot* vector, the intuition is the same, just for the last few words

# Word Embeddings

- If our *context* is a *multi-hot* vector instead of a *one-hot* vector, the intuition is the same, just for the last few words
- Could we have reasoned about likely *context* in our  $n$ -gram models that considered word order?
  - Yes! Just even less tractable.  $|V|^{n-1}$  places to reason about.

# Word Embeddings

- The core unsatisfying thing about  $n$ -grams:
  - Sparsity!
- In our coding exercise, our  $|V|^n$  matrix was *mostly* zeros
  - I mean >99% of the entries, in almost every parameter setting ( $n, |V|$ )
  - Smoothing function doing most of the “heavy lifting”
- Zero entries dominate even more as context ( $n$ ) grows
- Using a multi-hot context of size  $|V|$  for previous  $n$  words (our bag-of-words alternative) *does not resolve this fundamental issue* because it’s based on co-occurrence counts yet

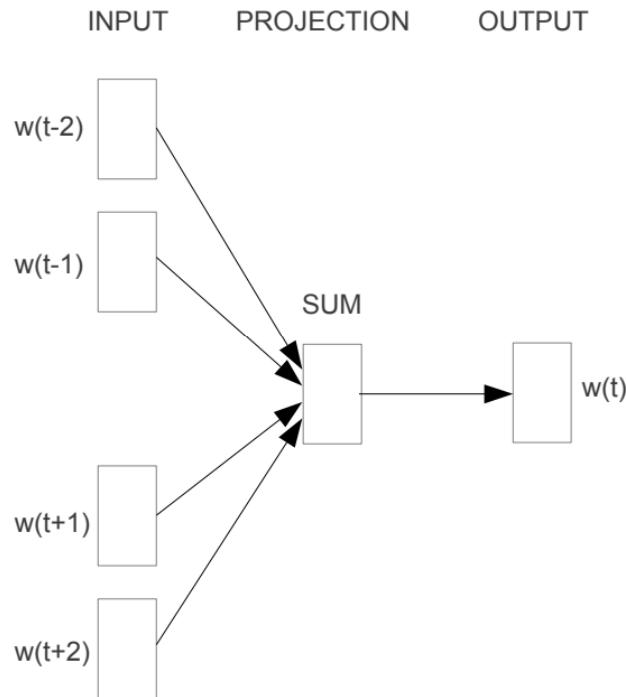
# Word Embeddings

- What is the standard remedy for sparsity in a data structure?
  - **Compression!**
- What we would *like to achieve* is to compress all the co-occurrence information in our big table of word/context counts in fewer than  $|V|^n$  dimensions
- Luckily, we can
  - The first *neural* approaches to achieving this goal stored  $|V|=30k$ ,  $n=10$  with bidirectional context in 600d vectors
    - Raw co-occurrence:  $30,000^{10}$
    - Three hundred trillion entries in co-occ matrix

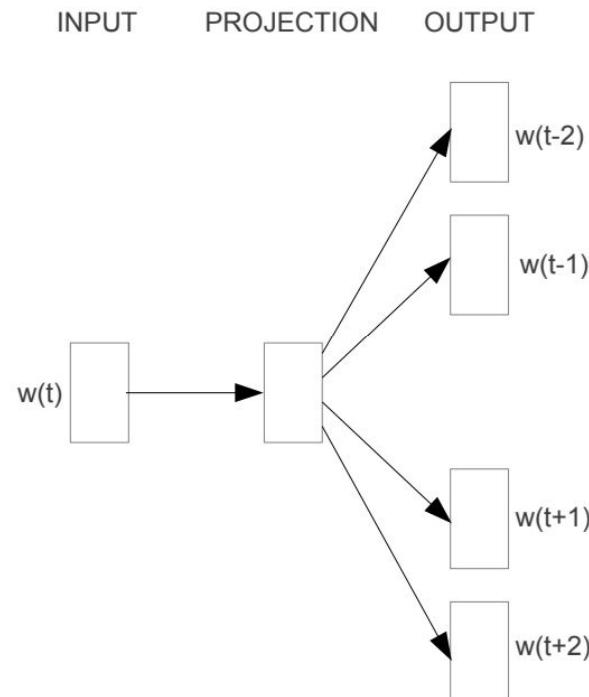
# Word Embeddings: A dense alternative to sparse n-grams

- The first *neural* word embeddings learning paper to gain traction:
  - [Efficient Estimation of Word Representations in Vector Space](#). Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. International Conference on Learning Representations (ICLR), 2013.
- Popularly known as?
  - [Word2Vec](#)

# Word2Vec Proposed Two Models



**CBOW**



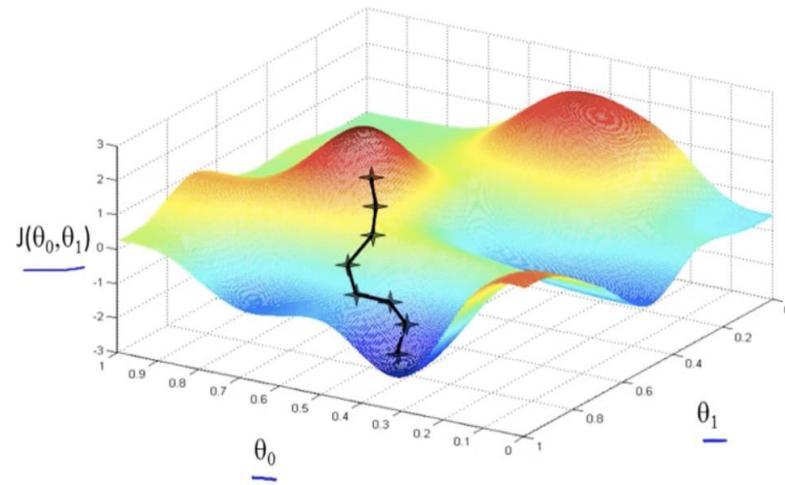
**Skip-gram**

# ML FUNDAMENTALS: Neural Network Training

- A neural network is, commonly, a composition of matrix multiplications and non-linear transformations
- A neural network can perform *non-linear function approximation* through repeated linear layers and non-linear transformations
- Optimization:  $M(\phi(x), \theta) = y$  for pairs  $(x, y)$  in data by making adjustments to  $\theta$
- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**

# ML FUNDAMENTALS: Stochastic Gradient Descent

- Gradient Descent
- “Step”  $\varepsilon \nabla L[M(\phi(x), \theta)]$ 
  - Go a little bit in the direction of the gradient (derivative) towards a local minima in the loss
- “Loss”: measure of how poorly the model did, e.g., the cross-entropy between predicted and true classes



$$H(P^* | P) = - \sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTRIBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTRIBUTION}}$$

# ML FUNDAMENTALS: Stochastic Gradient Descent

- “Step”  $\varepsilon \nabla L[M(\phi(x), \theta)]$
- What’s  $\varepsilon$ ?
  - Learning rate
- Can we actually get  $\nabla$  over all of  $D$ ?
  - Sometimes! Usually “no”, and so we use *stochastic* GD
- Could estimate  $\nabla_x$  with a single sample  $(x, y)$ . Why not?
- Could estimate  $\nabla_{x_i \dots x_j}$  with, e.g., as many examples as we can fit in RAM at a time  $x_i \dots x_j$ . What do we call these?
- Each subset of  $D$  used to estimate gradient is a **mini-batch**

# ML FUNDAMENTALS: Stochastic Gradient Descent

- “Step”  $\varepsilon \nabla_{x_i \dots x_j} L[M(\phi(x), \theta)]$
- **Learning rate:**  $\varepsilon$
- **Mini-batch:**  $(x, y)_i \dots (x, y)_j$
- **Iteration:**  $\theta := \theta - \varepsilon \nabla_{x_i \dots x_j} L[M(\phi(x), \theta)]$ 
  - Move parameters in direction of gradient estimated for a mini-batch
- **Epoch:** Completed each time we’ve seen all of  $(x, y) \in D$  again

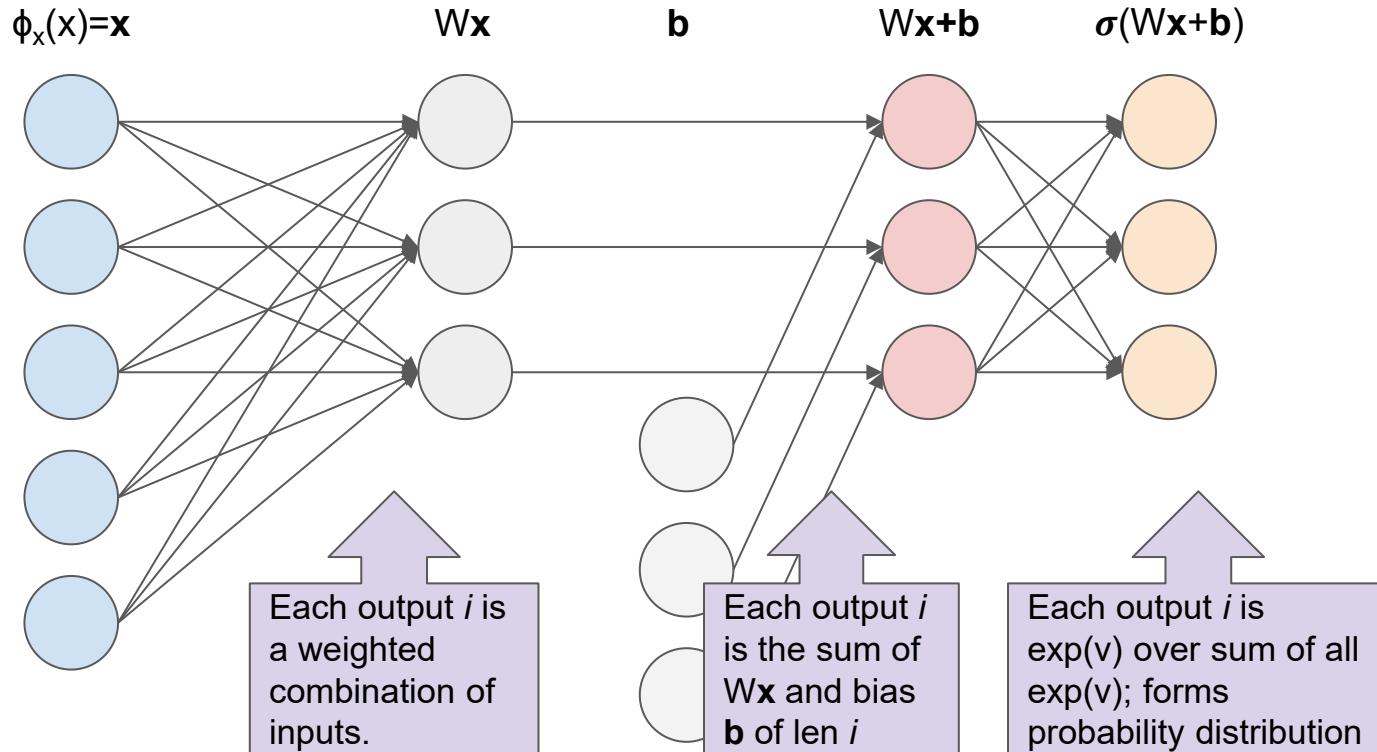
# ML FUNDAMENTALS: Neural Network Training

- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**
- What do we need?
  - Pass over data for multiple **epochs**
  - Sample **mini-batches** of our data
  - Step the optimization algorithm to complete an **iteration**
- Let's step through a motivating example to see how we arrive at training deep neural networks for function approximation

# Linear Image Classification



$$f(\mathbf{x}; \theta) = \sigma(W\mathbf{x} + \mathbf{b})$$



# Linear Image Classification

- What does a good  $f(x; W, b)$  look like?

0.1	1.5
0.07	2.32

0.3	0.15
1.5	1.91

1	3
0.37	0.8

0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

$W$

$f(x; W, b)$

0.15
-0.1
-0.2

$b$

# Linear Image Classification

- What does a good  $f(x; W, b)$  look like?



0.1	1.5
0.07	2.32



0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

 $W$ 

0.1
1.5
0.07
2.32

 $x$ 

+

0.15
-0.1
-0.2

 $b$ 

=

0.631
0.969
0.71

 $f(x; W, b)$ 

Cat

Car

Airplane

# Linear Image Classification

- What does a good  $f(x; W, b)$  look like?



0.3	0.15
1.5	1.91



0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

$W$

0.3
0.15
1.5
-0.2
1.91

$x$

+

0.15
-0.1
1.5
-0.2

$b$

=

0.73
0.598
0.161

$f(x; W, b)$

Cat

Car

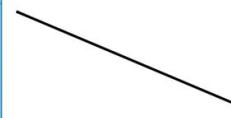
Airplane

# Linear Image Classification

- What does a good  $f(x; W, b)$  look like?



1	3
0.37	0.8



0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

$W$

1
3
0.37
0.8

$x$

+

0.15
-0.1
-0.2

$b$

=

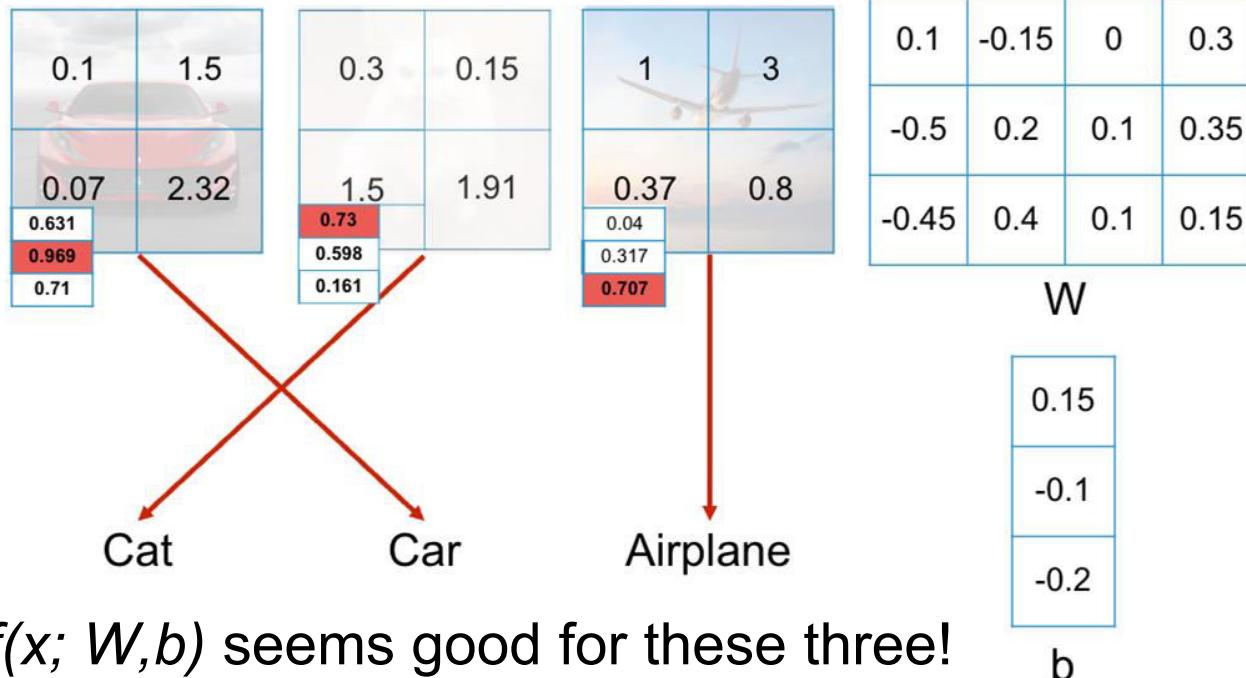
0.04
0.317
0.707

Cat  
Car  
Airplane

$f(x; W, b)$

# Linear Image Classification

- What does a good  $f(x; W, b)$  look like?



# Linear Image Classification

- Same images, new candidate weights  $W$  and bias vector  $b$

0.1	1.5
0.07	2.32
0.37	0.8
0.35	0.1

0.3	0.15
1.5	1.91
0.37	0.8
0.35	0.1

1	3
0.37	0.8
0.35	0.1
0	0

-0.25	0.1	0.05	0.175
0.05	-0.075	0	0.15
-0.225	0.2	0.05	0.075

$W$

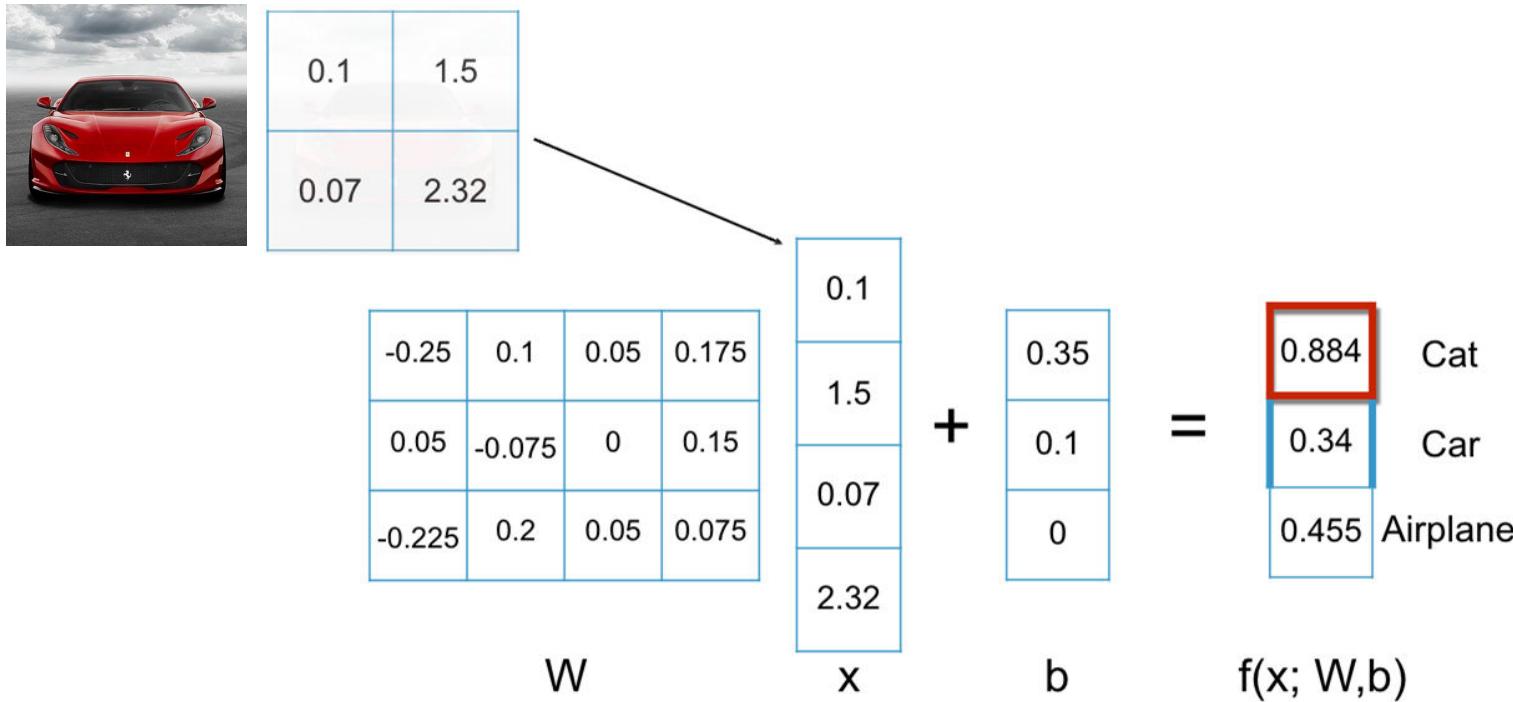
0.35
0.1
0

$b$

- Let's check our predictions for these...

# Linear Image Classification

- Same images, new candidate weights  $W$  and bias vector  $b$



# Linear Image Classification

- Same images, new candidate weights  $W$  and bias vector  $b$



0.3	0.15
1.5	1.91

-0.25	0.1	0.05	0.175
0.05	-0.075	0	0.15
-0.225	0.2	0.05	0.075

$W$

$x$

$b$

+

=

0.699
0.39
0.181

Cat

Car

Airplane

$f(x; W, b)$

# Linear Image Classification

- Same images, new candidate weights  $W$  and bias vector  $b$



1	3
0.37	0.8

-0.25	0.1	0.05	0.175
0.05	-0.075	0	0.15
-0.225	0.2	0.05	0.075

$W$

1
3
0.37
0.8

$x$

+

0.35
0.1
0

$b$

=

0.558
0.04
0.453

$f(x; W, b)$

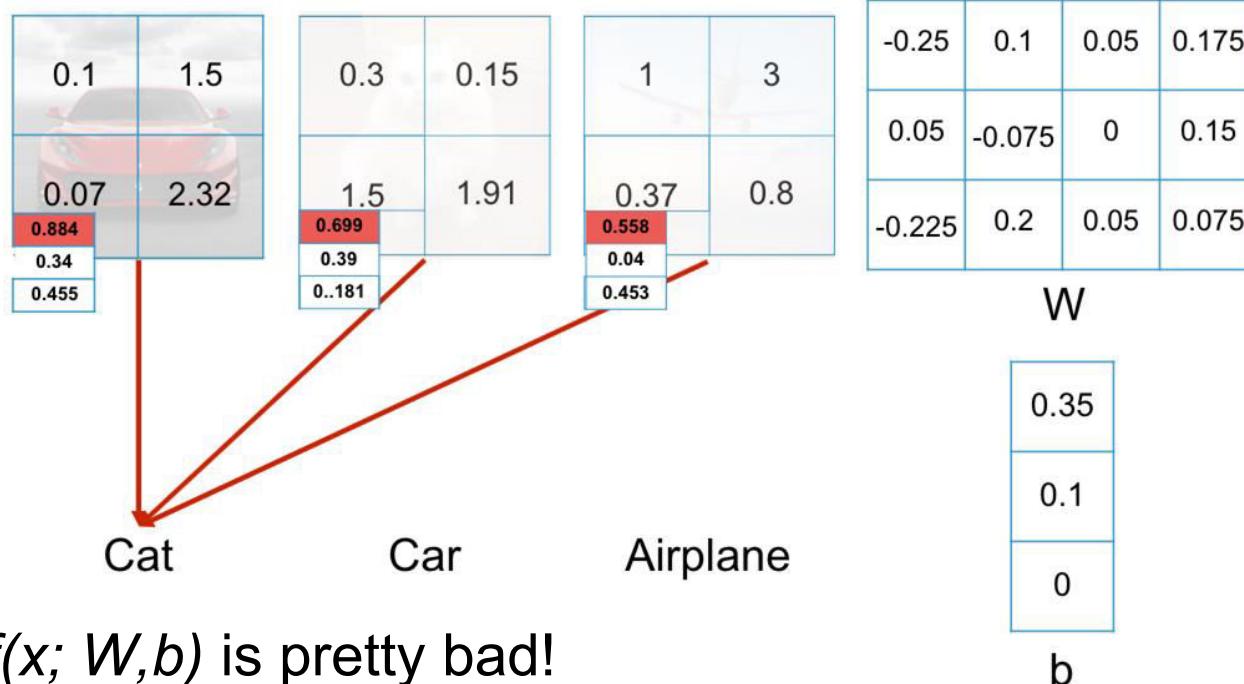
Cat

Car

Airplane

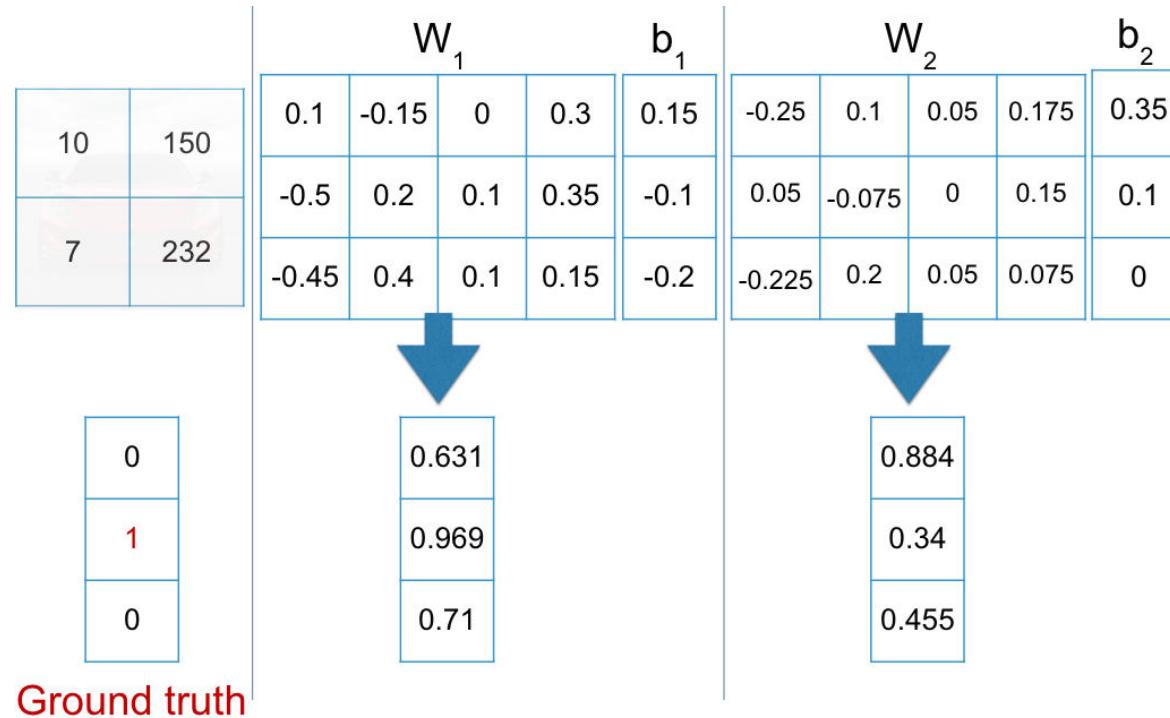
# Linear Image Classification

- Same images, new candidate weights  $W$  and bias vector  $b$



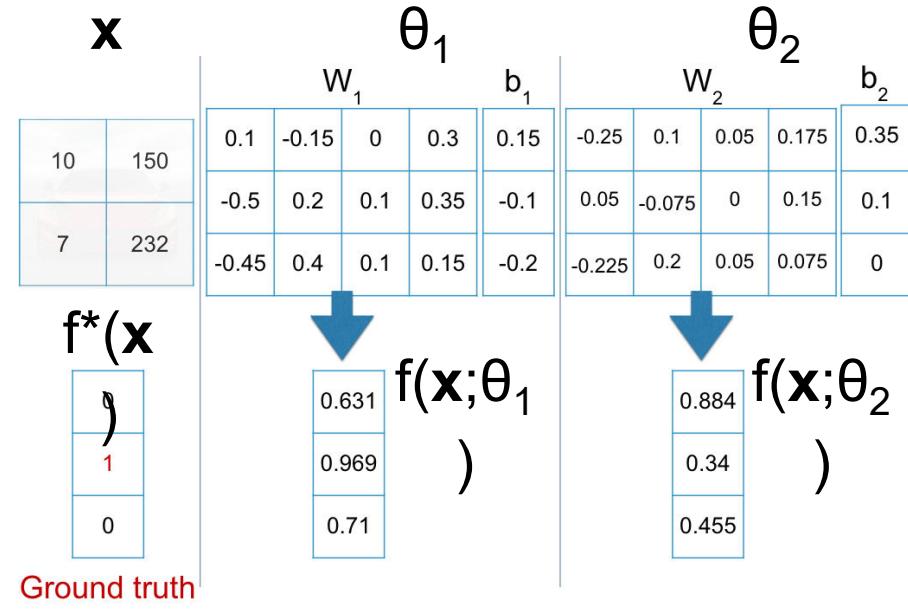
# Linear Image Classification

- How do we measure the *fit* of candidate parameters?



# How Do We Measure the Quality of $f(\mathbf{x}; \theta)$ ?

- How do we measure the *fit* of candidate parameters?
- Target vector  $f^*(\mathbf{x})$  if we knew the exact function,  $f^*$ :
  - <0.00, 1.00, 0.00>
- Prediction of  $f(\mathbf{x}; \theta_1)$ :
  - <0.63, 0.97, 0.71>
- Prediction of  $f(\mathbf{x}; \theta_2)$ :
  - <0.88, 0.34, 0.46>
- We want a way to penalize *distance* between  $f^*(\mathbf{x})$  and predicted vectors



# How Do We Measure the Quality of $f(\mathbf{x};\theta)$ ?

- How do we measure the *fit* of candidate parameters?
- Want to formulate an **optimization problem** to minimize the distance  $\{f^*(\mathbf{x}) - f(\mathbf{x};\theta)\}$  across all  $\mathbf{x}_i \in X$  in a dataset of  $(X, Y)$  pairs
- We call the function we want to minimize, subject to the available training data, the **loss function**
- What are some possible ways to measure such a distance penalty? We want “good”  $\theta$  to have less loss than “bad”  $\theta$

# How Do We Measure the Quality of $f(\mathbf{x};\theta)$ ?

- We could turn our scores into a probability distribution

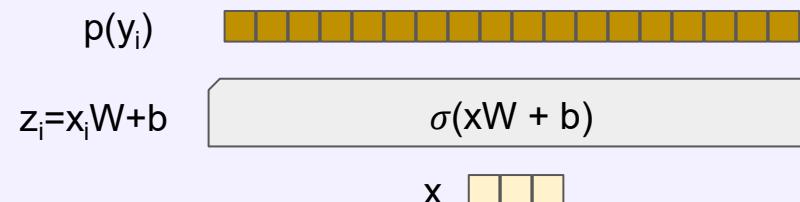
$$Loss = -\frac{1}{N} \sum_{i=1}^N \hat{y}_i \log(P(y_i = j | x_i)) \quad P(y_i = j | x_i) = \frac{e^{f(x_i; W, b)}}{\sum_{j=1}^C e^{f(x_j; W, b)}}$$

- This loss function is called **Cross-Entropy Loss**
- For every example  $x_i$ , the loss approaches zero as the score for true class  $y_i$  dominates the sum of all scores, and approaches infinity (slowly!) as score  $y_i$  is dominated by other class scores

# ML FUNDAMENTALS: Softmax Function

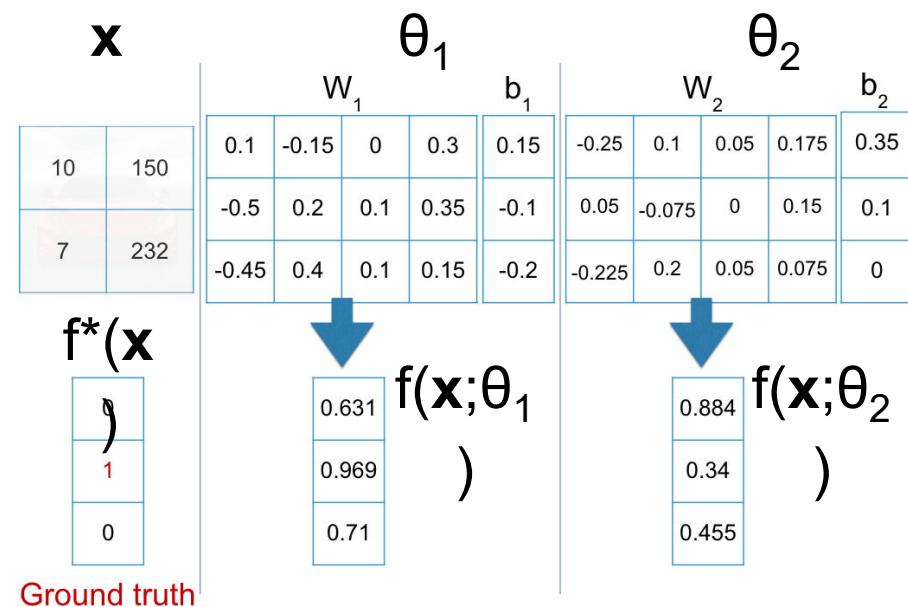
- Sometimes we want a probability distribution over possible outputs from our model
- We can induce a probability distribution from a set of output logits (“activations” / “energies” / “scores”)
- For  $z_i$ , the logit of class  $i$  in the prediction space, the softmax probability is given by:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



# How Do We Measure the Quality of $f(\mathbf{x}; \theta)$ ?

- How do we measure the *fit* of candidate parameters?
- We can calculate the **loss** of parameter options  $\theta_1$  versus  $\theta_2$  across a dataset  $(X, Y)$
- Now that we can measure the *quality* of a set of parameters, we can consider how to *discover* those parameters



# How Do We Find Good Parameters $\theta$ for $f(\mathbf{x};\theta)$ ?

- Backing up:
  - $f(\mathbf{x};\theta)$  is our notation for any general function we'd like to learn
  - $f(\mathbf{x};\theta)$  maps from  $x \in X$  to  $y \in Y$
  - Vector  $\mathbf{x}$  is obtained by applying representation function  $\phi(x) = \mathbf{x}$  to input
  - $\theta$  parameterizes  $f$ ;  $f$  is a class of functions that can vary according to the selection of  $\theta$
  - For example,  $\theta = \{W, b\}$  for  $f(\mathbf{x};\theta) = W\mathbf{x} + b$

# How Do We Find Good Parameters $\theta$ for $f(\mathbf{x};\theta)$ ?

- $f(\mathbf{x};\theta) = \mathbf{W}\mathbf{x} + b; \theta = \{\mathbf{W}, b\}$
- In our running image classification example, how do we *define* what are good parameters?
- We pick a loss function  $L$ !
  - $\theta^* = \min_{\theta \in \Theta} L(f(\mathbf{x};\theta), y)$
- $L$  is calculated on an  $(x, y)$  basis, so we really want to find good parameters for an entire *dataset*, not just one example!
  - $\theta^* = \min_{\theta \in \Theta} (\sum_{(x,y) \in D} L(f(\mathbf{x};\theta), y))$
  - For  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$  our training dataset

# How Do We Find Good Parameters $\theta$ for $f(\mathbf{x};\theta)$ ?

- $f(\mathbf{x};\theta) = \mathbf{W}\mathbf{x} + b; \theta = \{\mathbf{W}, b\}$
- $\theta^* = \min_{\theta \in \Theta} (\sum_{(x,y) \in D} L(f(x;\theta), y))$ 
  - For  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$  our training dataset
- Three basic classes of approach:
  - Analytical solution
    - Find where  $\frac{d}{d\theta} (\sum_{(x,y) \in D} L(f(x;\theta), y)) = 0$
  - Heuristic search
    - Initialize random candidate  $\theta$ 's and check  $L$  value
  - Numerical approach
    - Estimate better  $\theta$ 's given current ones

# How Do We Find Good Parameters $\theta$ for $f(\mathbf{x};\theta)$ ?

- $f(\mathbf{x};\theta) = \mathbf{W}\mathbf{x} + b; \theta = \{\mathbf{W}, b\}$
- $\theta^* = \min_{\theta \in \Theta} (\sum_{(x,y) \in D} L(f(x;\theta), y))$ 
  - For  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$  our *training dataset*
- Numerical approach
  - Estimate better  $\theta$ 's given current ones
- Best of both worlds: cheap (heuristic) + informed (analytical)
  - Finding where  $\frac{d}{d\theta} (\sum_{(x,y) \in D} L(f(x;\theta), y)) = 0$  is hard
  - Finding good  $\theta$  through random/population search is hard
- Calculating  $\frac{d}{d\theta_i} (\sum_{(x,y) \in D} L(f(x;\theta), y))$  for dimension  $\theta_i$  is easy...

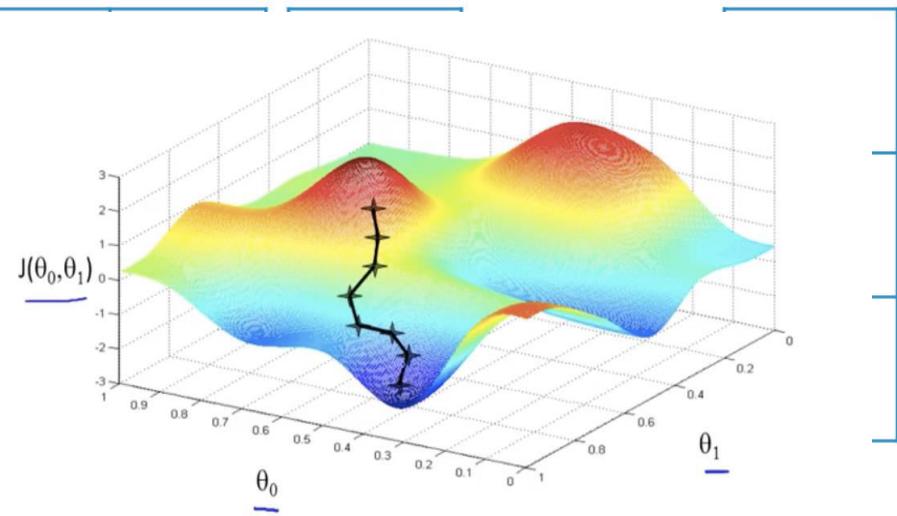
# Gradient Descent

0.1	1.5
0.07	2.32

X

?	?
?	?
?	?

W



- We want  $\theta$  minimizing  $\sum_{(x,y) \in D} L(f(x;\theta), y)$
- Calculating  $\frac{d}{d\theta_i} (\sum_{(x,y) \in D} L(f(x;\theta), y))$  for dimension  $\theta_i$  is easy
- For a candidate  $\theta$ , we can calculate the *gradient* in each dimension of  $\theta_i$  that could lead us towards the minimum

# Gradient Descent

0.1	1.5
0.07	2.32

x

?	?	?	?	?
?	?	?	?	?
?	?	?	?	?

W

?
?
?

b

0
1
0

y

- Let's consider just  $W \in \theta$  for the above  $(x, y)$  datum
- Recall the definition of a derivative for a generic function  $f$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

# Gradient Descent

0.1	1.5
0.07	2.32

x

?	?	?	?	?
?	?	?	?	?
?	?	?	?	?

W

?
?
?

b

0
1
0

y

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

- We will take the derivative of  $L(f(x;\theta), y)$  with respect to each individual dimension of  $\theta$
- That vector of partial derivatives is called the **gradient** wrt  $(x,y)$

# Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

$W$

<b>-0.1+ 0.001</b>	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

$W+h$

?	?	?	?
?	?	?	?
?	?	?	?

gradient ( $dW$ )

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

- For  $L(f(x;\theta_1), y)$ , we calculate  $L$  before and after adding a tiny perturbation  $h$  to  $\theta_1$ , giving  $f(x+h)$  versus original  $f(x)$
- (Remember, here  $f$  is our loss  $L$  and derivative is against  $\theta_1$ )

# Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

$$L(W) = 1.6688$$

<b>-0.1+ 0.001</b>	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

$$L(W+h) = 1.6693$$

?	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

$$dW_1 = (1.6693 - 1.6688) / 0.001$$

$$= 0.5$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

# Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

$$L(W) = 1.6688$$

<b>-0.1 + 0.001</b>	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

$$L(W+h) = 1.6693$$

<b>0.5</b>	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

$$\begin{aligned} dW_1 &= (1.6693 - 1.6688) \\ &\quad / 0.001 \end{aligned}$$

$$= 0.5$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

# Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

$$L(W) = 1.6688$$

-0.1+ 0.001	<b>0.2+</b> <b>0.001</b>	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

$$L(W+h) = 1.6697$$

0.5	<b>0.9</b>	?	?
?	?	?	?
?	?	?	?

gradient (dW)

$$\begin{aligned} dW_2 &= (1.6697 - 1.6688) \\ &\quad / 0.001 \end{aligned}$$

$$= 0.9$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

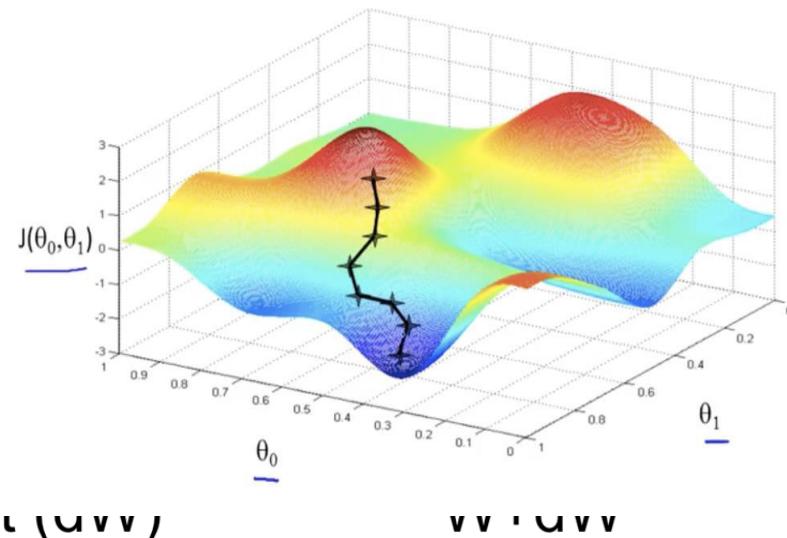
# Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

0.5	0.9
-0.13	0.17
-0.55	.92

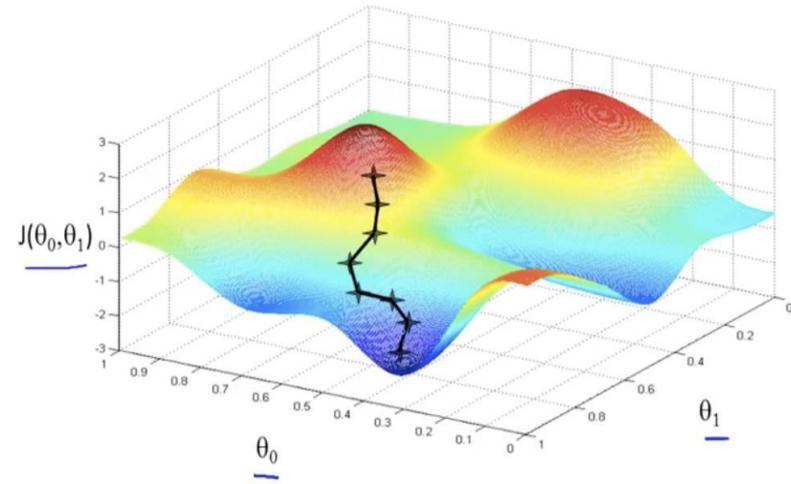
gradient (uvv)



- Gradient update step: add estimated gradient to existing parameters to take a “step” towards parameter values that are closer to minimizing the objective function
- What’s missing?

# Gradient Descent

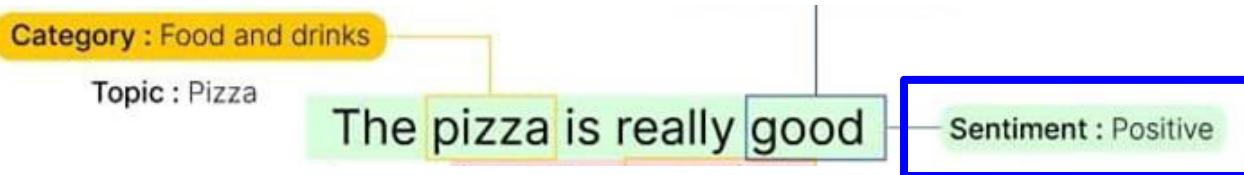
- $f(\mathbf{x}; \theta) = \mathbf{W}\mathbf{x} + \mathbf{b}$ ;  $\theta = \{\mathbf{W}, \mathbf{b}\}$
- $\theta^* = \min_{\theta \in \Theta} (\sum_{(x,y) \in D} L(f(x; \theta), y))$
- In our example, we calculated the gradient with respect to just one  $(x, y)$  pair;  $\nabla_{(x,y)}$
- We actually want the gradient for  $\sum_{(x,y) \in D} L(f(x; \theta), y)$ , which requires repeating our process for every  $(x, y) \in D$
- $\nabla_D = (1/|D|) \sum_{(x,y) \in D} \nabla_{(x,y)}$
- What's the problem?



# Stochastic Gradient Descent

- Storing  $\nabla_D = (1/|D|) \sum_{(x,y) \in D} \nabla_{(x,y)}$  is expensive
- Depending on the size of your dataset  $D$ , it's likely not possible to store the partial derivatives for every  $(x,y)$  for every parameter in your entire model in the GPU at once
- In *stochastic* gradient descent, we take just a slice (*batch*) of  $D$  at a time; in practice, as much as we can fit on the GPU
- What sounds best?
  - Randomly sample batch from  $D$  and repeat
  - Shuffle  $D$ , sample slices as batches, repeat
  - Shuffle  $D$ , sample slices as batches, reshuffle  $D$  and repeat

# Where Does Deep Learning Come In?



- Input space  $X$ ?
  - Review tokens  $T$
  - $X=T^N$  for max length  $N$
- Linear function:  $f(\mathbf{x}, \theta) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- What loss function could we use for this  $n$ -ary classification problem?
  - Cross entropy
- Non-linear transformation:

$$Loss = -\frac{1}{N} \sum_{i=1}^N \hat{y_i} \log(P(y_i = j | x_i)) \quad P(y_i = j | x_i) = \frac{e^{f(x_i; W, b)}}{\sum_{j=1}^C e^{f(x_j; W, b)}}$$

# Where Does Deep Learning Come In?

- Linear function:  $f(\mathbf{x}, \theta) = \mathbf{W}\mathbf{x} + b$
- What loss function could we use for this  $n$ -ary classification problem?
  - Cross entropy
- Non-linear transformation:

$$Loss = -\frac{1}{N} \sum_{i=1}^N \hat{y_i} \log(P(y_i = j | x_i)) \quad P(y_i = j | x_i) = \frac{e^{f(x_i; W, b)}}{\sum_{j=1}^C e^{f(x_j; W, b)}}$$

- Our function can only capture the degree to which each dimension of the input,  $\mathbf{x}_i$ , contributes to the likelihood of each class  $y_j$ 
  - "... brought out the ice cold food."
  - "... brought out the ice cold gazpacho."
  - "... Amazing service all around. Yeah right! 😊"

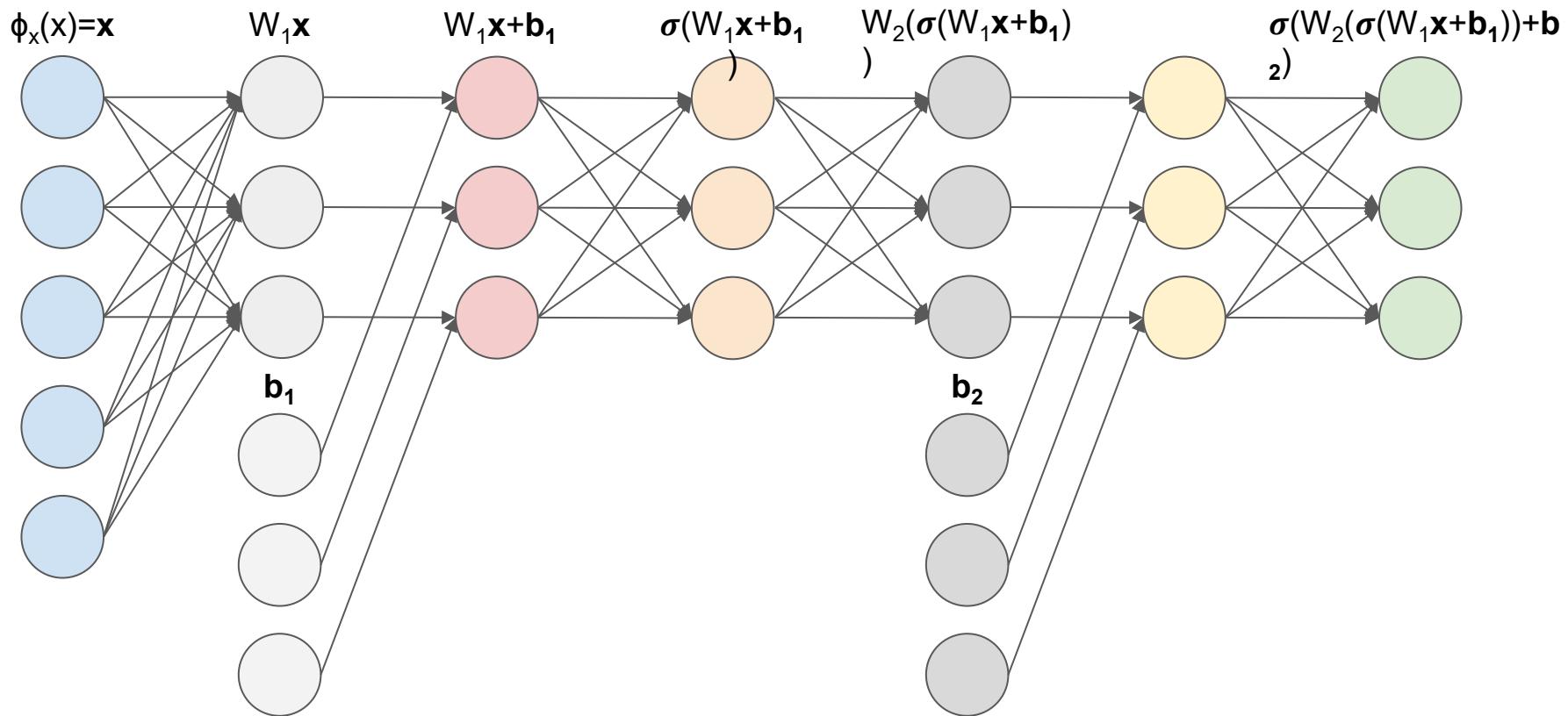
# Where Does Deep Learning Come In?

- $f(\mathbf{x}, \theta) = \sigma(W\mathbf{x} + b)$ 
  - Linear function with nonlinearity
- If we need to make decisions based on  $\mathbf{x}_i$  input features, what can we do?
- $f(\mathbf{x}, \theta) = \sigma(W_2(\sigma(W_1\mathbf{x} + b_1)) + b_2)$ 
  - First layer *latent variables*  $\sigma(W_1\mathbf{x} + b_1)$  are each activated by linear combination of values of  $\mathbf{x}_i$  input variables
  - Now  $W_2, b_2$  can mix signals from the first *layer*; the final outputs  $\sigma(W_2(\sigma(W_1\mathbf{x} + b_1)) + b_2)$  are each activated by linear combinations of the latent variables from layer one
  - So our class logits can now consider combinations of input words like “**cold gazpacho**” and “**Amazing ... Yeah right!**”

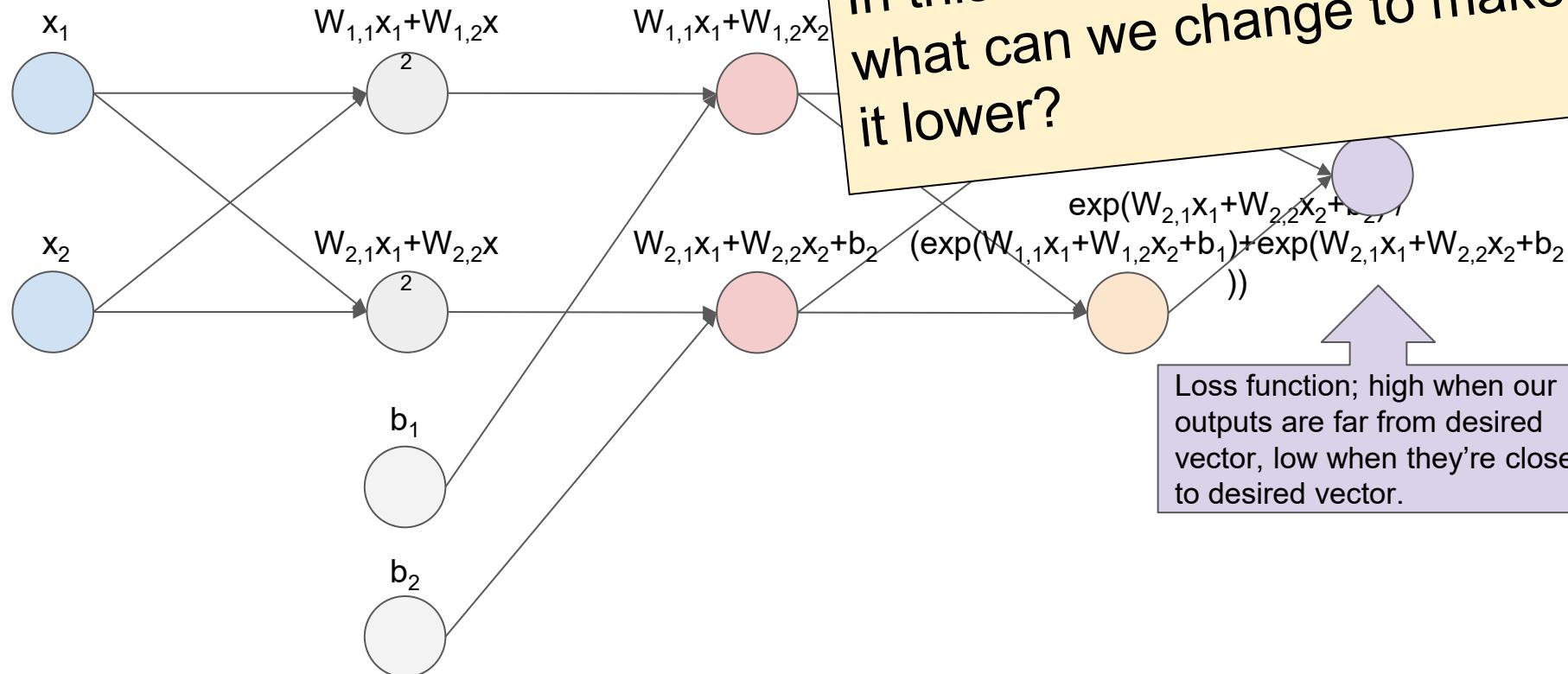
Quick terminology note,  
these basic transformations  
are called perceptron layers  
or fully-connected layers

$$f(\mathbf{x}, \theta) = \sigma(W_2(\sigma(W_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2)$$

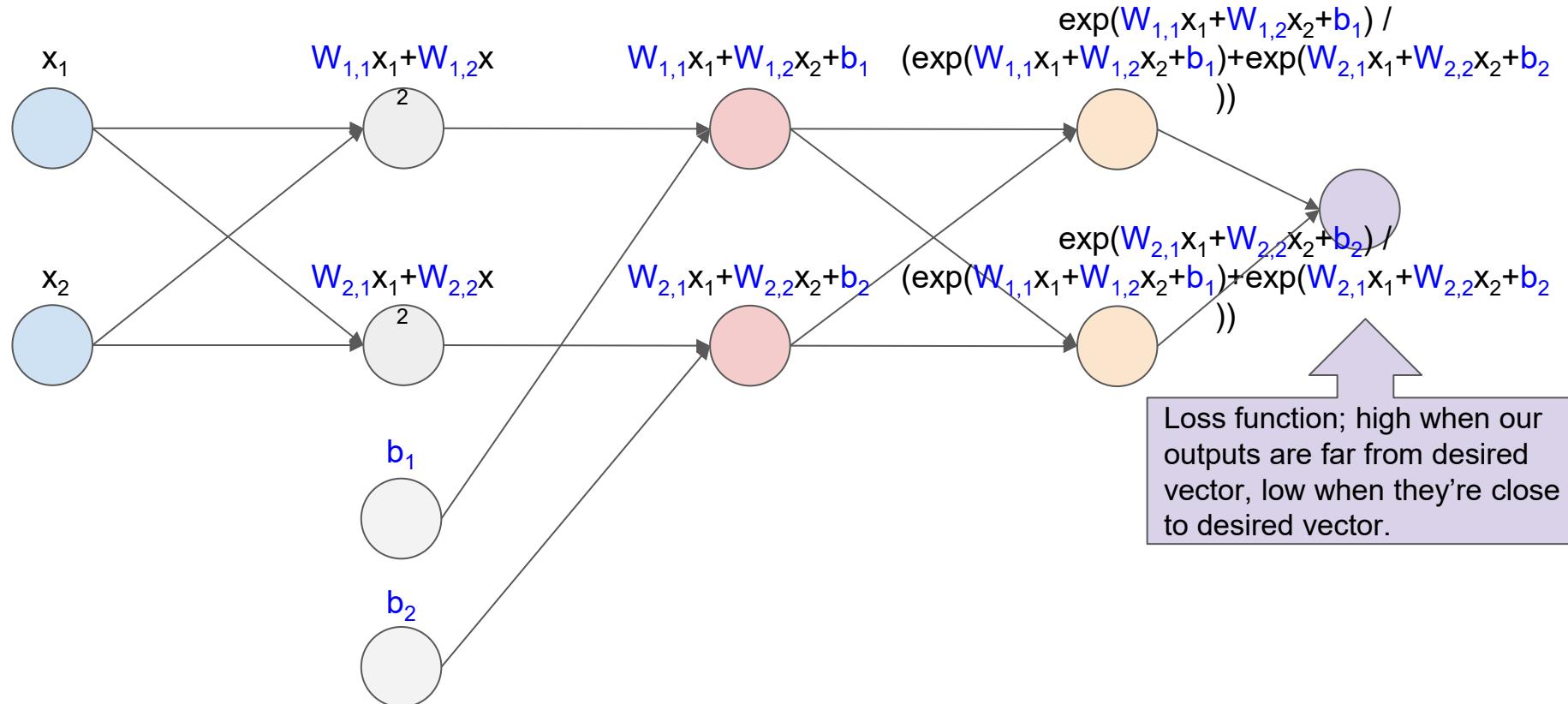
$$W_2(\sigma(W_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2$$



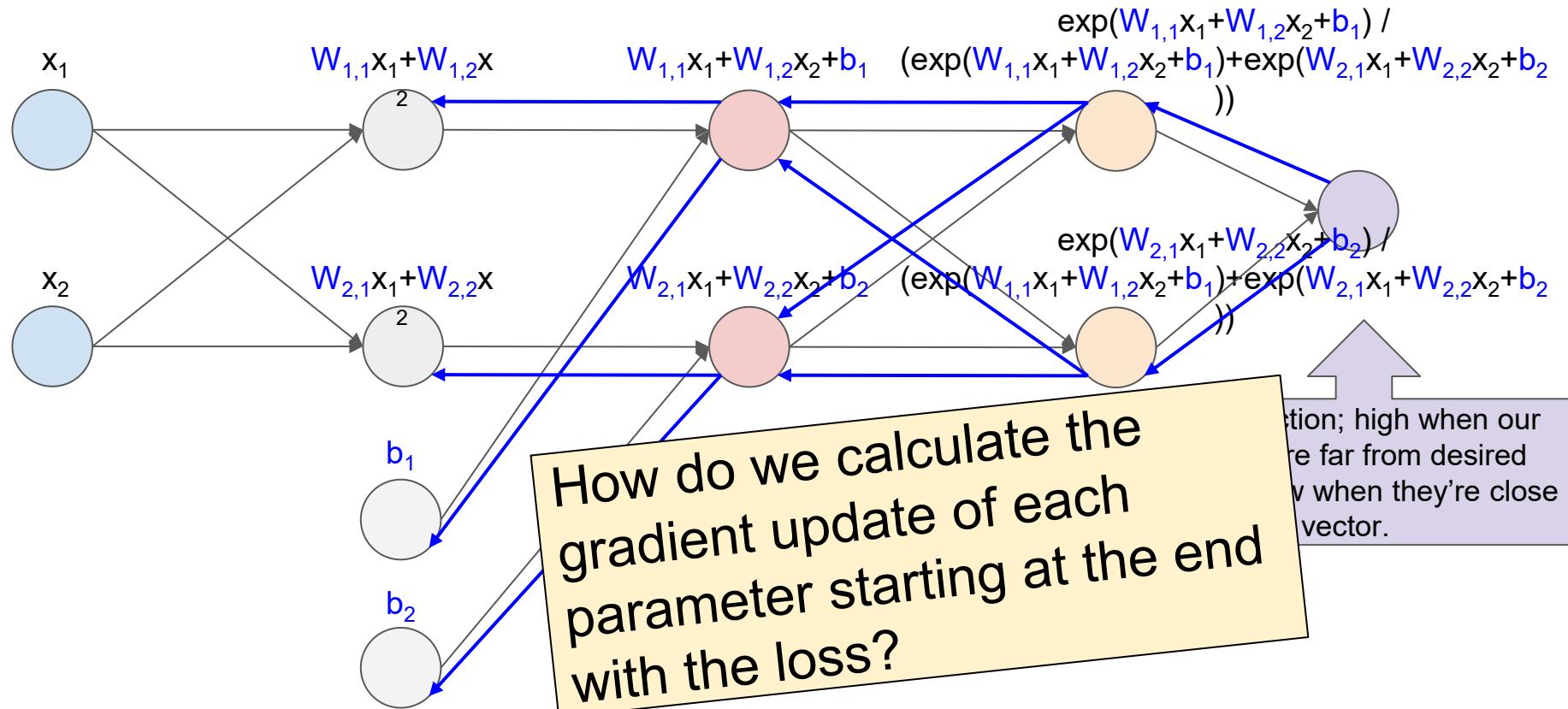
# Forward Pass



# NNs are Parameterized, Compositional Functions



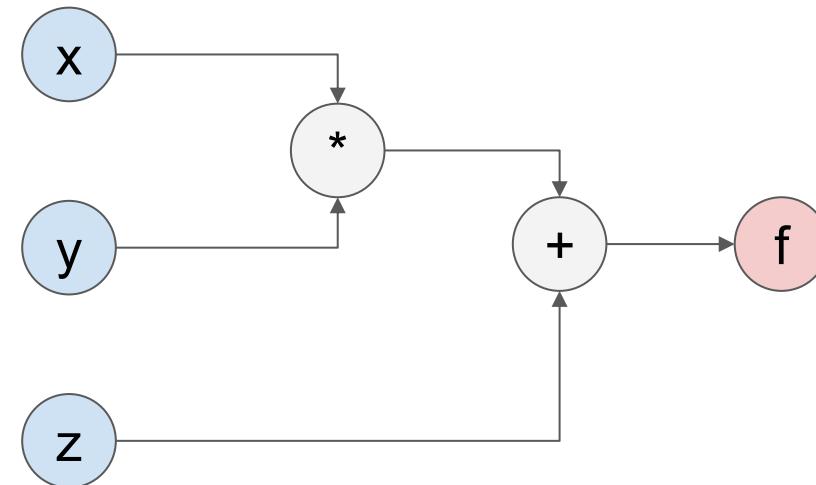
# Backwards Pass



# We Can Employ a Computation Graph

Consider function:

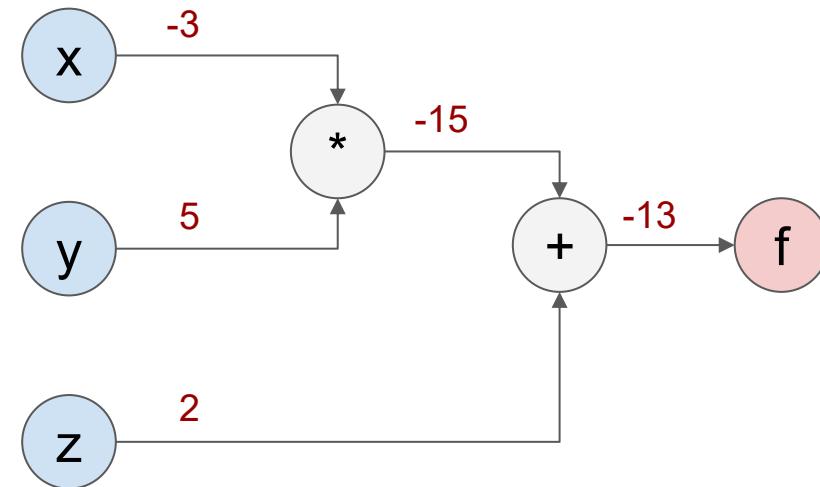
$$f(x,y,z) = x^*y + z$$



# Employ a Computation Graph

Consider function:

$$f(x,y,z) = x^*y + z$$



We need to be able to take partial derivatives

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

<b>-0.1+ 0.001</b>	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

<b>0.5</b>	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

$$L(W) = 1.6688$$

$$L(W+h) = 1.6693$$

$$\begin{aligned} dW_1 &= (1.6693 - 1.6688) \\ &\quad / 0.001 \end{aligned}$$

$$= 0.5$$

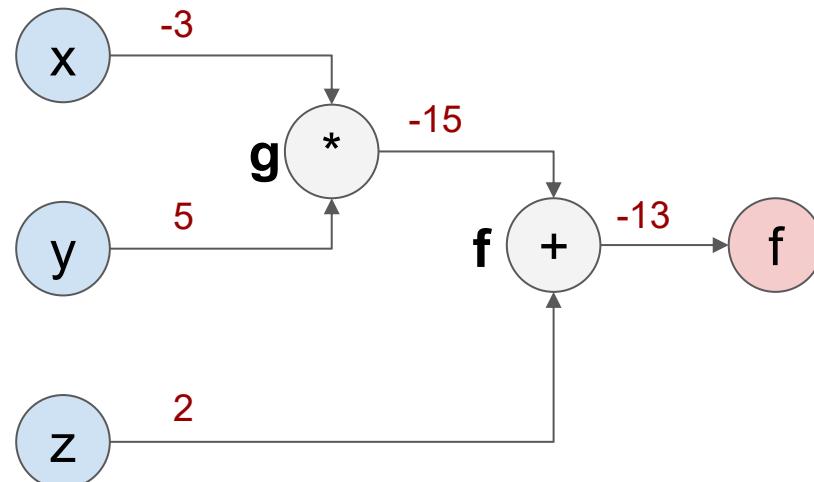
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

# Backpropagation via Computation Graph

Consider function:

$$f(x,y,z) = x^*y + z$$

- NNs are compositions of functions
- Let's define:
  - $g(x,y) = x^*y$
  - $f(g,z) = g + z$



# Backpropagation via Computation Graph

$$f(x,y,z) = x^*y + z$$

$$g(x,y) = x^*y; f(g,z) = g+z$$

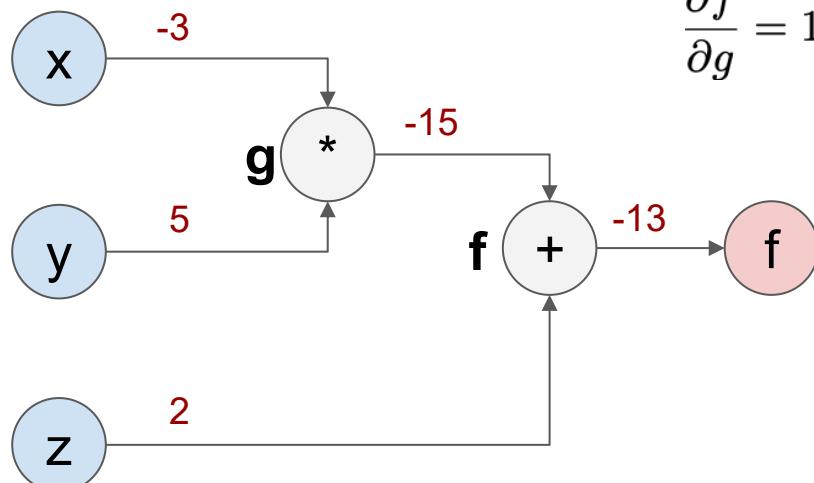
- Let  $x, y, z$  be *parameters* we want to update

- Then we need:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

$$\frac{\partial g}{\partial x} = y \quad \frac{\partial g}{\partial y} = x$$

$$\frac{\partial f}{\partial g} = 1 \quad \frac{\partial f}{\partial z} = 1$$



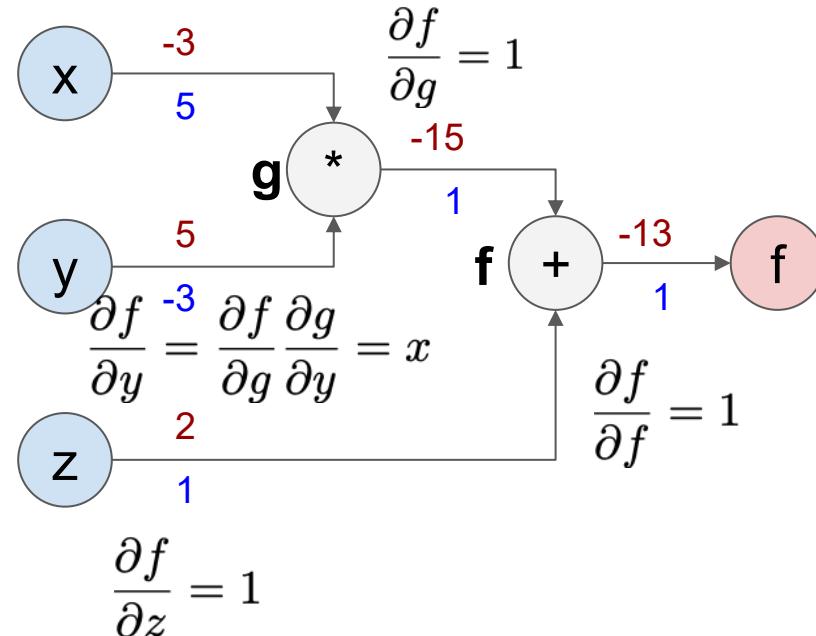
# Backpropagation via Computation Graph

$$f(x,y,z) = x^*y + z$$

$$g(x,y) = x^*y; f(g,z) = g+z$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} = y$$

- Let  $x, y, z$  be *parameters* we want to update
- Then we need:  
 $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Backpropagation via Computation Graph

$$f(x,y,z) = x^*y + z$$

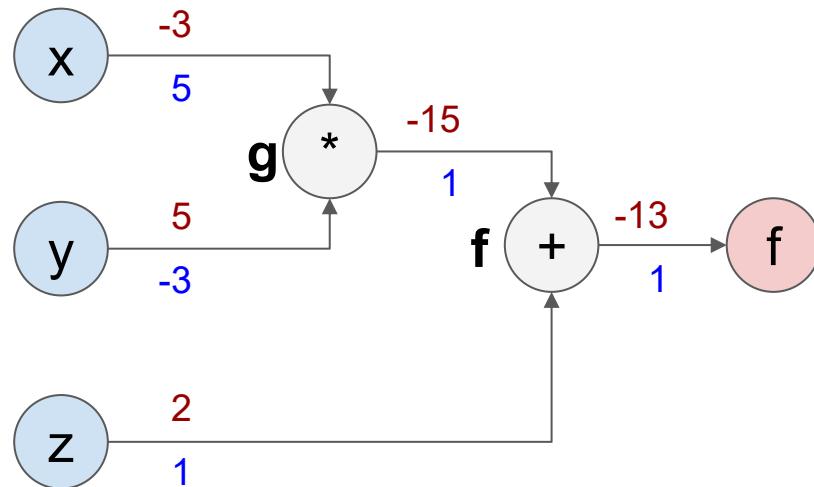
$$g(x,y) = x^*y; f(g,z) = g+z$$

- Then we get:

$$\frac{\partial f}{\partial x} = 5$$

$$\frac{\partial f}{\partial y} = -3$$

$$\frac{\partial f}{\partial z} = 1$$

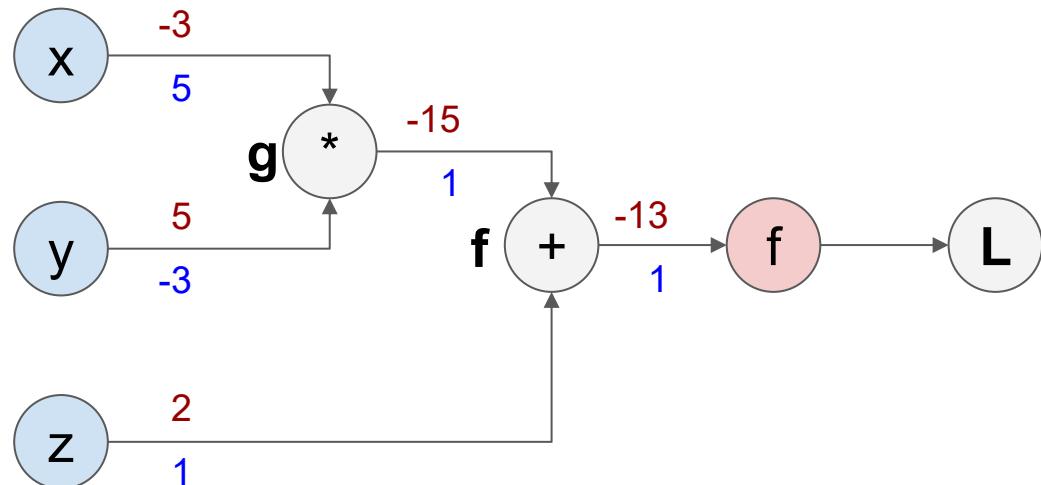


# Backpropagation with a Loss Function

$$f(x,y,z) = x^*y + z$$

$$g(x,y) = x^*y; f(g,z) = g+z$$

- If  $f$  is just our predictor, we can add a loss on top, call it function  $L$
- $L(f)$



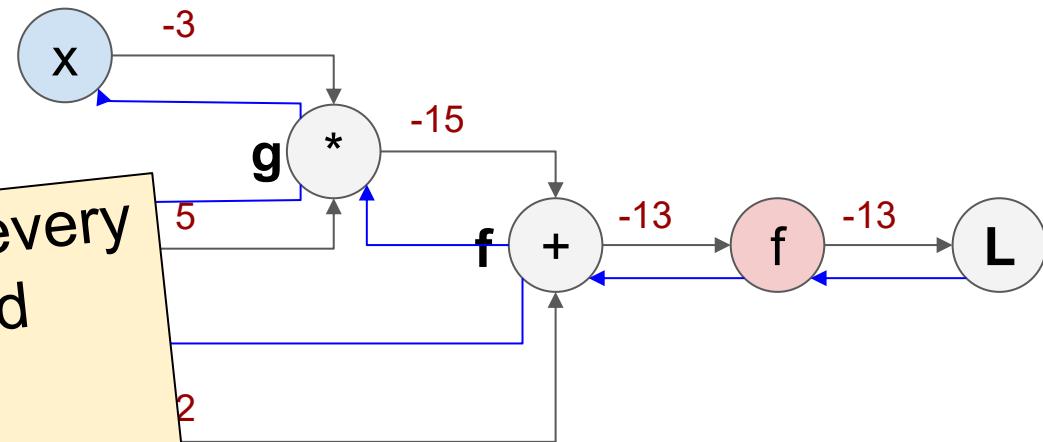
# Backpropagation with a Loss Function

$$f(x,y,z) = x^*y + z$$

$$g(x,y) = x^*y; f(g,z) = g+z$$

- Forward pass
- Backwards pass

The loss function  $L$  and every component of the forward function  $f$  need to be differentiable for backprop



# Where Does Deep Learning Come In?

- $f(\mathbf{x}, \theta) = \sigma(W_2(\sigma(W_1\mathbf{x} + b_1)) + b_2)$ 
  - This derivative is more computationally intense to take, since we need it with respect to every parameter in both weight matrices and bias vectors
  - Modern DL software handles these computations automatically through *autograd* software for efficient derivative estimation
  - So what's stopping us from....
- $f(\mathbf{x}, \theta) = \sigma(W_4(\sigma(W_3\sigma(W_2(\sigma(W_1\mathbf{x} + b_1)) + b_2) + b_3)) + b_4)$
- $f(\mathbf{x}, \theta) = \sigma(W_8(\sigma(W_7\sigma(W_6(\sigma(W_5\sigma(W_4(\sigma(W_3\sigma(W_2(\sigma(W_1\mathbf{x} + b_1)) + b_2) + b_3)) + b_4) + b_5)) + b_6) + b_7)) + b_8)$
- Nothing!?
  - Well, realistically, our GPU memory and data constraints
  - For fixed  $D$ , quality of estimated parameters  $\theta$  as  $|\theta|$  grows?

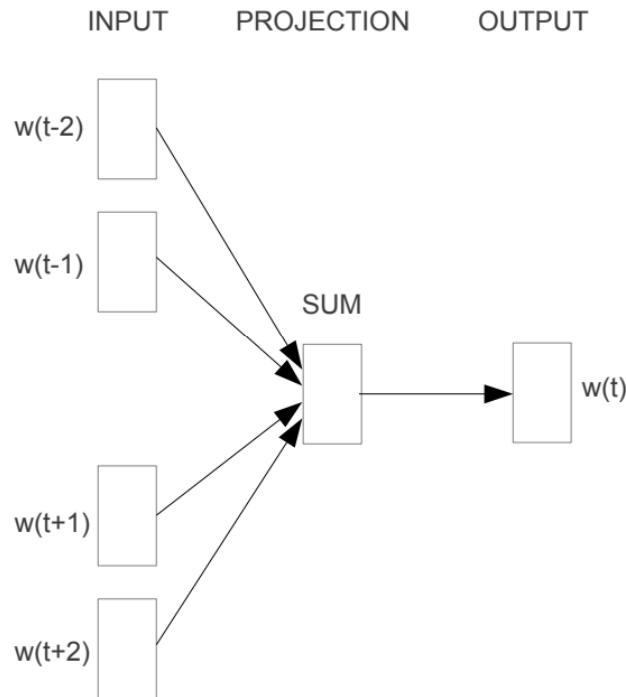
# Activity: Formulate a Multi-Layer Perceptron for Sentiment

- Whiteboard!
- Define inputs during training
- Define function to turn input  $x$  into feature vector  $\mathbf{x}$
- Define function to turn output  $y$  into output vector  $\mathbf{y}$
- Define compositional function to be applied during forward pass
- Define non-linearities
- Ensure your model functions are all differentiable

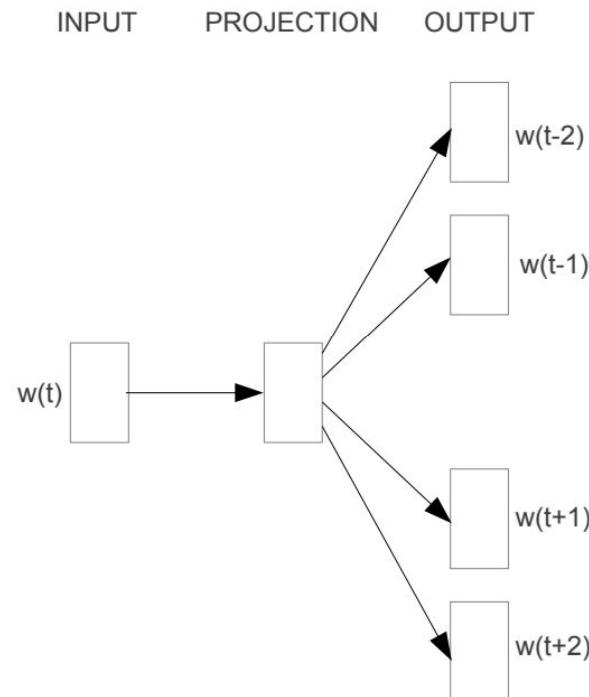
# Word Embeddings: A dense alternative to sparse n-grams

- The first *neural* word embeddings learning paper to gain traction:
  - [Efficient Estimation of Word Representations in Vector Space](#). Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. International Conference on Learning Representations (ICLR), 2013.
- Popularly known as?
  - [Word2Vec](#)

# Word2Vec Proposed Two Models



**CBOW**



**Skip-gram**

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 1: Lecture 2  
N-Grams & DL 101 for NLP

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 1: Lecture 3  
Distributional Semantics and Lexical Embeddings

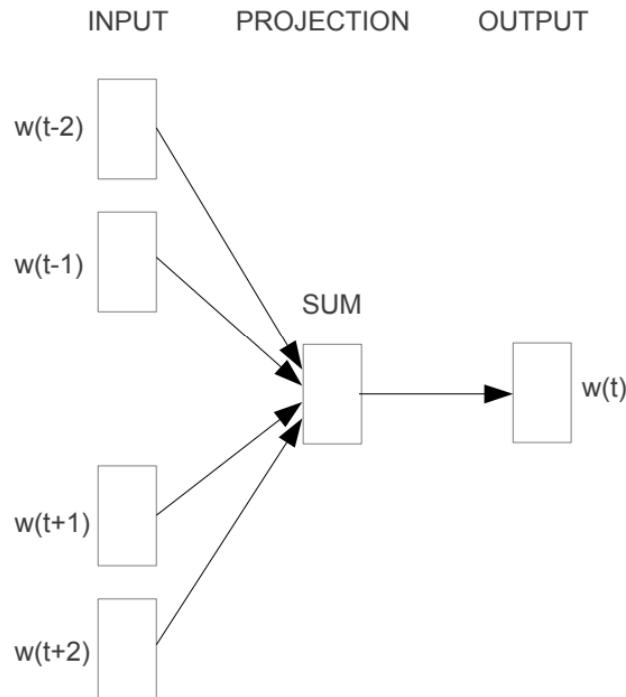
# Whiteboard Activity: Word Embeddings as Compression

- Given a co-occurrence matrix  $\mathbf{A}^{|V| \times |V|}$ , calculate the smallest integer  $k$  such that decomposition matrices  $\mathbf{B}^{|V| \times K}$  and  $\mathbf{S}^{K \times |V|}$  give  $\mathbf{A}^{|V| \times |V|} = \mathbf{B}^{|V| \times K} \mathbf{S}^{K \times |V|}$  (*problem example + volunteer*)
- What about  $\mathbf{A}$  determines the value of  $K$ ? (*open question*)
- Given  $\mathbf{A}^{|V| \times |V|}$  and a specified  $k$ , how can we *approximate*  $\mathbf{A}$  such that  $\mathbf{A}^{|V| \times |V|} \approx \mathbf{B}^{|V| \times k} \mathbf{S}^{k \times |V|}$ ? (*volunteer formulates*)
  - $\operatorname{argmin}_{\theta \in \Theta} d(\mathbf{A}_{[i,:]} - \sum_j \mathbf{S}_{[i,j]} \mathbf{B})$
  - Can let  $d(w, v) = \sqrt{\sum (w_i - v_i)^2}$  (L2 norm; Euclidean dist)
- If we have an objective to optimize, we can apply DL!
- What are the learnable parameters in this formulation?

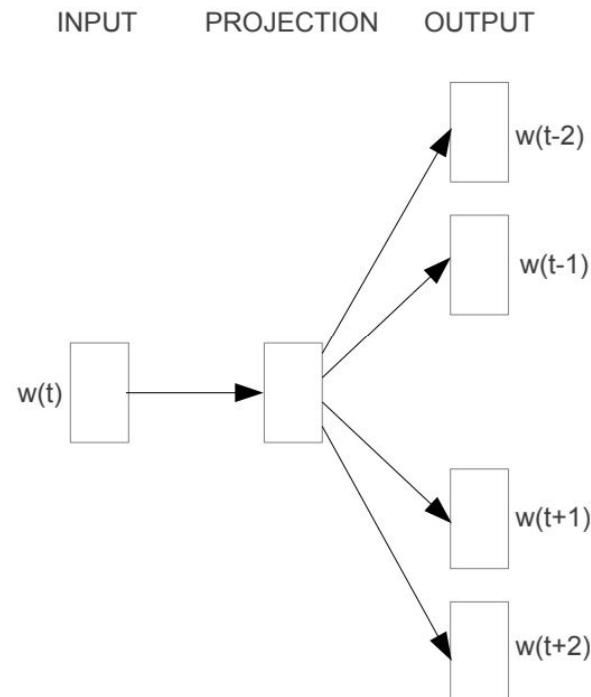
# Word Embeddings: A dense alternative to sparse n-grams

- The first *neural* word embeddings learning paper to gain traction:
  - [Efficient Estimation of Word Representations in Vector Space](#). Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. International Conference on Learning Representations (ICLR), 2013.
- Popularly known as?
  - [Word2Vec](#)

# Word2Vec Proposed Two Models



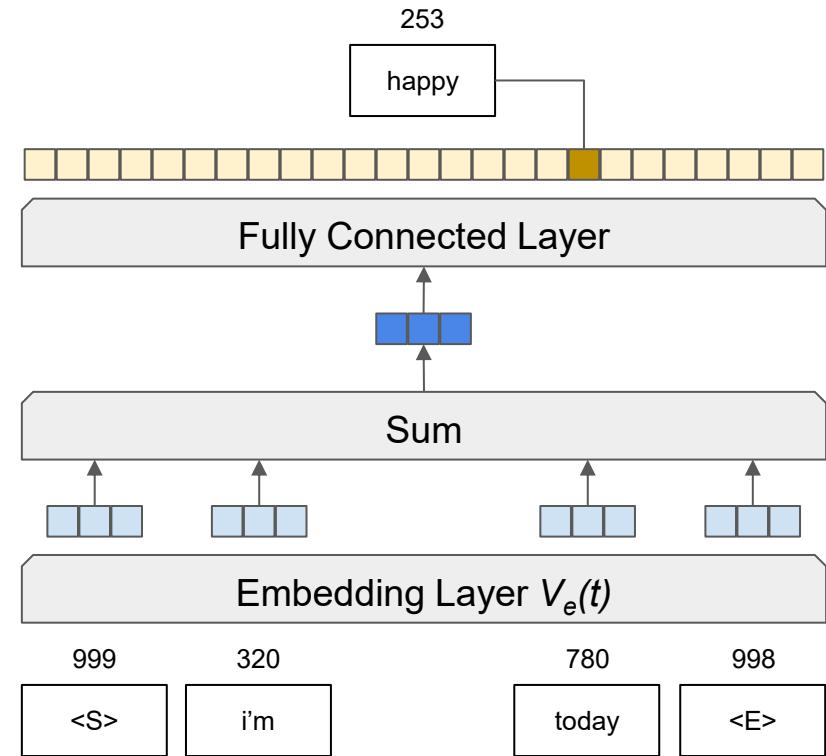
**CBOW**



**Skip-gram**

# Word2Vec: Continuous Bag-of-Words (CBOW)

- Let's unpack on whiteboard
- What probability does the FC layer estimate?
  - $p(w_i|w_{i-2}w_{i-1}w_{i+1}w_{i+2})$
- What assumptions?
  - Word order ignored
- What's being learned?
  - FC + Embeddings
  - A, B, S from exercise??



# Word2Vec: Continuous Bag-of-Words (CBOW)

- What kind of context? (left/right/bidirectional)
  - Bidirectional
- Thinking in terms of  $n$ -grams,  $n=9$ !
  - Look 4 words back and 4 words ahead

# Word2Vec: Evaluation

Table 1: Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

- (Athens, Greece, Oslo, Norway) -> (ABCD)
- “A is to B as C is to D”
- Then:
  - $v(A) - v(B) = v(C) - v(D)$
  - $v(D) = v(C) + v(B) - v(A)$
- Evaluation:
  - Estimate  $v(w)$  via learning method
  - Calculate accuracy on analogies by checking whether  $v(D)$  is nearest to  $[v(C)+v(B)-v(A)]$  among all vocabulary
  - ... excluding A,B,D

# Word2Vec: Evaluation

Table 1: Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

“Semantic” Analogies

“Syntactic” Analogies

# Word2Vec: Evaluation

Table 4: Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.

Existing RNN Methods {  
Paper RNN Implementation {

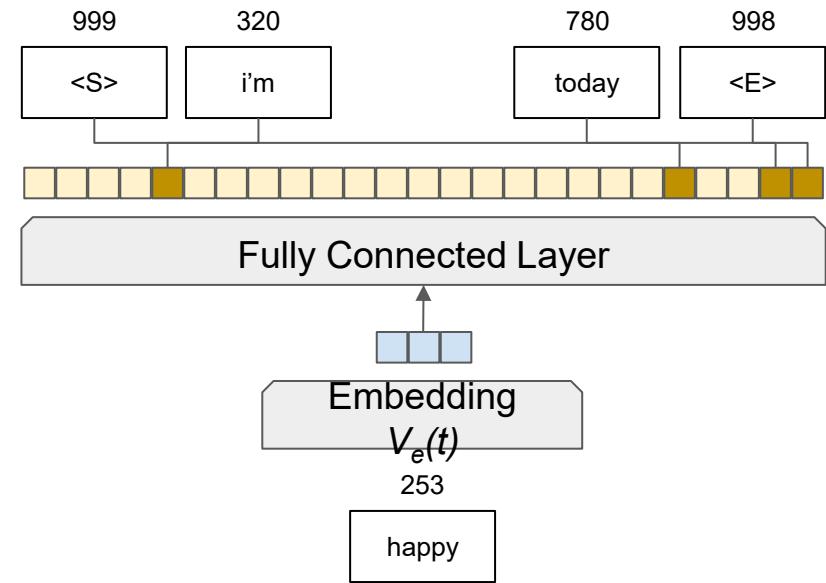
Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	<b>64.5</b>	50.8
CBOW	300	783M	15.5	53.1	36.1

# Word Embeddings: Bag-of-Words Model

- Estimate  $p(w_i|w_{i-1}w_{i-2}\dots w_{i-n+1})$  as distribution of  $w_i$  given that  $w_{i-1}w_{i-2}\dots w_{i-n+1}$  occur *in any order* left of  $w_i$
- Because the *context* from which we predict  $w_i$  is now a single vector, what other interesting information is here?
  - We've been thinking about LM from just one perspective
    - “What *next token* is likely given the *context* so far?”
  - We could ask something else...
    - “What *context* will make a *particular token* most likely?”

# Word2Vec: Continuous Skip-Gram Model

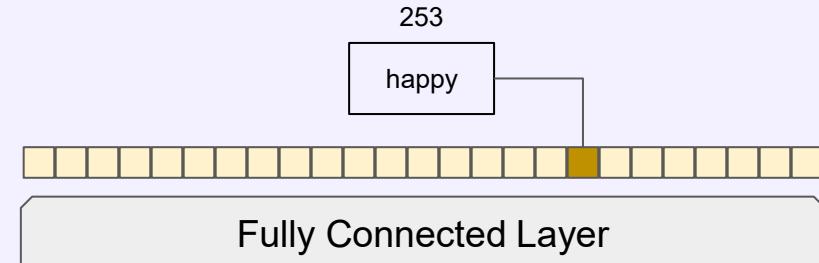
- What probability does the FC layer estimate?
  - $p(w_{i-2}|w_i)p(w_{i-1}|w_i)^*$
  - $p(w_{i+1}|w_i)p(w_{i+2}|w_i)$
- What assumptions?
  - Word order ignored
- What's being learned?
  - FC + Embeddings
  - A, B, S from exercise?



# ML FUNDAMENTALS: Cross Entropies

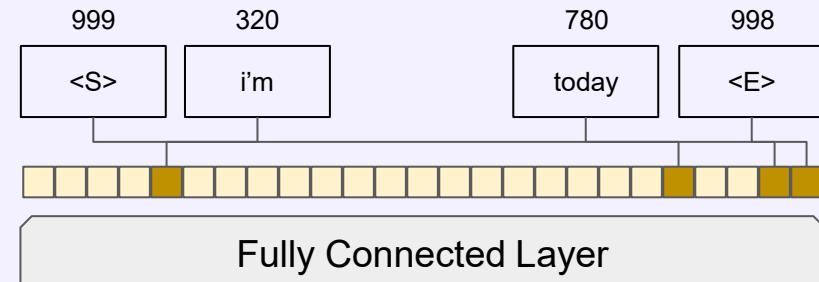
- Categorical cross entropy:
  - For  $F(x) \rightarrow |C|$ , say which one of  $|C|$  labels  $x$  exhibits  $c^* \in C$ ;  $F(x)$  estimates  $p(c^*|x)$ , and the loss encourages  $p(c^*|x)=1$

Say which one of  $|C|$  labels  $x$  exhibits



- Binary cross entropy:
  - For  $F(x) \rightarrow |C|$ , say whether  $x$  exhibits it; a combination of labels  $c_k \in C$ ;  $F(x)$  estimates  $p(c|x)$ , and the loss encourages  $p(c_k|x)=1$

For each  $|C|$  say whether  $x$  exhibits it

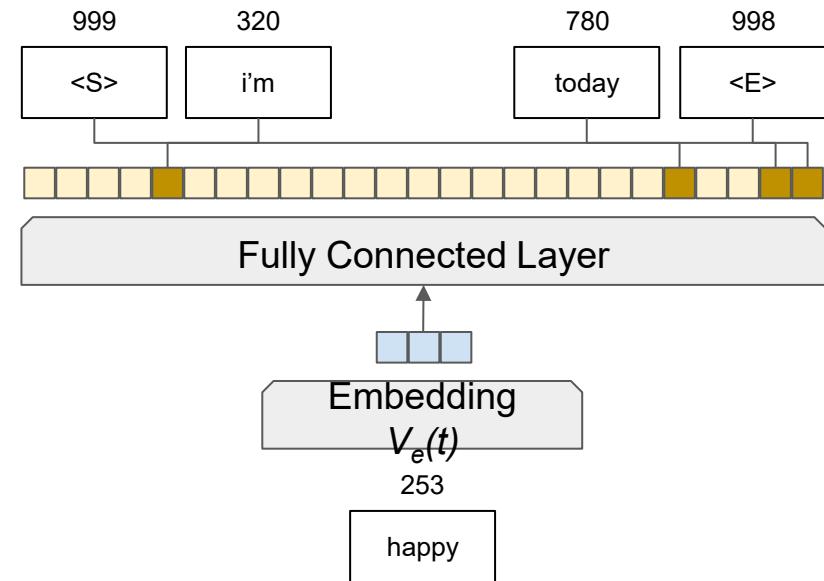


# Word2Vec: Continuous Skip-Gram Model

- What kind of context? (left/right/bidirectional)
  - Bidirectional
- Thinking in terms of  $n$ -grams,  $n=11$ !
  - Look to *up to 5 words back and 5 words ahead*
- Authors hypothesize that words farther away should have less influence; what's one strategy we could use to enforce that?
- When forming training data; they actually *sample n* from [3, 11], looking either at just the left/right tokens ( $n=3$ , trigram) up to the full window ( $n=11$ )

# Word2Vec: Continuous Skip-Gram Model

- How does this architecture allow sampling the context window size?
  - Independent prediction of all targets
  - No architecture component cares about # of targets; only the loss does
- Why does this “work”?



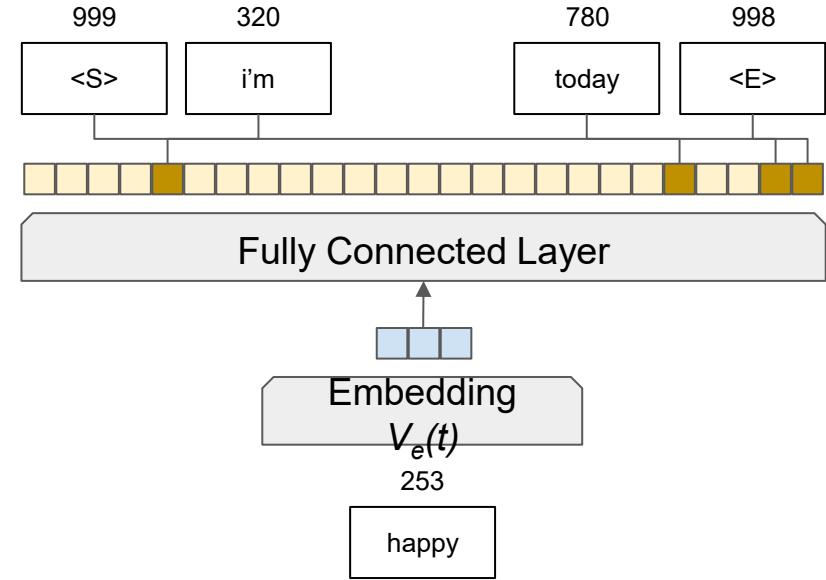
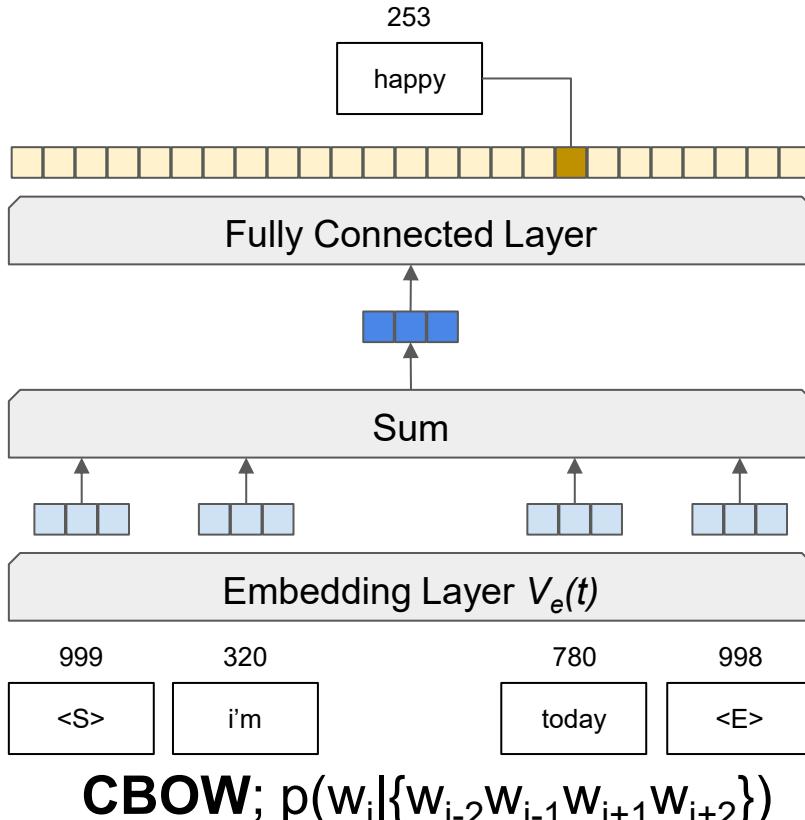
# Word2Vec: Evaluation

Table 4: Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.

Existing RNN Methods {  
Paper RNN Implementation {

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	<b>64.5</b>	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	<b>50.0</b>	55.9	<b>53.3</b>

# Word2Vec: Recap



**Skip-Gram;**  
 $p(w_{i-2}|w_i)p(w_{i-1}|w_i)p(w_{i+1}|w_i)p(w_{i+2}|w_i)$

# Evaluating Learned Word Embedding Vectors

- *In vitro* evaluation
  - “outside the living body”
  - In ML: a controlled task setting with no surprises
  - For CBOW?
    - Masked language modeling (accuracy / surprisal / perplexity of target word)
  - For skip-gram?
    - Accuracy / surprisal / perplexity of target context words

# Evaluating Learned Word Embedding Vectors

- *In vivo* evaluation
  - “within the living body”
  - In ML: a task set out in the wild for which the model was not explicitly trained
  - Analogy evaluation: ABCD
    - Words in eval need to have learned vectors
    - Evaluates whether vector space arranged itself to satisfy the geometric properties we find intuitive
  - Text classification; machine translation; ...

# Analogy Evaluation Metrics

- Given input analogy ABCD and learned embeddings  $\mathbf{V}$ 
  - Evaluate how well  $\mathbf{V}(D)$  matches  $v^{\sim} = [\mathbf{V}(C) + \mathbf{V}(B) - \mathbf{V}(A)]$
  - Cosine distance? Euclidean distance?
    - Won't tell us if there's a word with  $\mathbf{V}(Z)$  closer than D
  - Nearest neighbor?
    - Could test if  $\mathbf{V}(D)$  is *the* closest in vocab to  $v^{\sim}$
    - No partial credit in this case
  - Reciprocal Rank
    - Create ordered list  $\mathbf{Z}(v^{\sim})$  over vocabulary by proximity to  $v^{\sim}$  and give score  $[1 / (\mathbf{Z}(v^{\sim}).\text{index}(D) + 1)]$

# Analogy Evaluation Metrics

- Given input analogy ABCD and learned embeddings  $\mathbf{V}$ 
  - Evaluate how well  $\mathbf{V}(D)$  matches  $v^{\sim} = [\mathbf{V}(C) + \mathbf{V}(B) - \mathbf{V}(A)]$
  - Reciprocal Rank
    - Create ordered list  $\mathbf{Z}(v^{\sim})$  over vocabulary by proximity to  $v^{\sim}$  and give score  $[1 / (\mathbf{Z}(v^{\sim}).\text{index}(D) + 1)]$
  - What computation is needed to create  $\mathbf{Z}(v^{\sim})$ ?
    - Cosine sim between all  $|\mathbf{V}|$  candidate words and  $v^{\sim}$
    - Then also a sort
    - Floating point operations?
      - Sim:  $O(|\mathbf{V}|h)$ ; sort  $O(|\mathbf{V}|\log(|\mathbf{V}|))$

# Vocabulary Size Woes

- Every algorithm, evaluation, metric, etc. in NLP tasks is bottlenecked by  $|V|$ 
  - Softmax:  $O(|V| * h)$  per input
  - Nearest neighbor:  $O(|V| * h)$  per input
  - *The Curse of Dimensionality*
- How could we estimate these on the cheap?
  - *Negative Sampling*: select a subset of  $k < |V|$  words
  - Problem?
    - Unless the  $k$  words are already similar, makes tasks too easy! Model won't need to learn. We'll cover this more.

# Global Vectors (GloVe)

- Rather than iterating over the corpus and estimating  $p(w|c)$  [CBOW] or  $p(c|w)$  [skip-gram], directly estimates vectors from the co-occurrence matrix
- Some people will fight about whether word2vec or GloVe is better for a particular task or in general
  - Estimates of the same underlying information
  - GloVe has fewer hyperparams; harder for large  $|V|$  and big corpora because of needing to reason with co-occ matrix
- Consider this choice a hyperparameter for pretraining

# Why Learn Word Embeddings from Language Modeling?

- The techniques we've shown in this lecture learn word embeddings  $\mathbf{V}$  as a side-effect of:
  - estimating  $p(w|\mathbf{c})$ , which is itself a way to estimate:
    - $p(s)$ , the fundamental function of a language model
- How else could we learn word embeddings?
  - Any task! E.g., in text classification we also had a learnable embedding layer
- So why do we *bother* learning word embeddings via LM?
- **DATA:** LM objectives are unsupervised! Only  $\mathbf{x}$  given; we construct “fake” tasks out of  $\mathbf{x}$  to create pairs

# Why Learn Word Embeddings from Language Modeling?

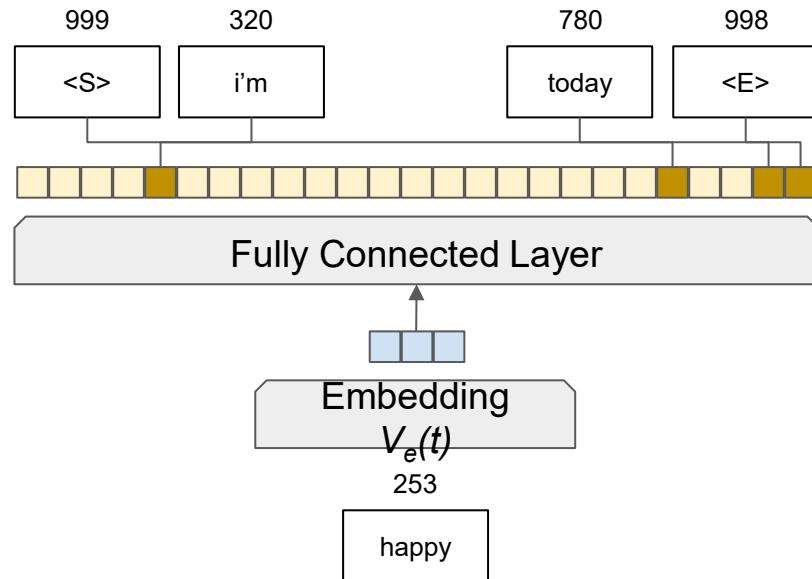
- Why would the word embeddings we learn as a side-effect of language modeling be beneficial downstream?
- The key is in the bane of NLP:
  - **The curse of dimensionality**
- Word embeddings are a form of dimensionality reduction: instead of reasoning about  $|V|$  (or worse, all “possible” words), we reason about vectors of fixed dimension
- It’s easier to learn when your data isn’t extremely sparse
- Word embeddings make co-occurrence less sparse for estimating the *mutual information* between inputs and outputs

# Word Embeddings: What are We Actually Learning?

- Language model estimation methods:
  - “What *next token* is likely given the *context* so far?”
    - $p(w_k|c)$ , where **c** is words left/right/both
    - CBOW, estimates as  $p(w_k|\{w_{k-1} w_{k-2} \dots\})$
    - n-gram models, estimate as  $p(w_k|w_{k-1} w_{k-2} \dots)$
  - “What *context* will make a *particular token* most likely?”
    - $p(c|w_k)$
    - Skip-gram, estimates as  $\dots p(w_{k-1}|w_k) p(w_{k+1}|w_k) \dots$
- In estimating latent representations that give us *co-occurrence* information, we tap into *mutual information*

# Word Embeddings: What are We Actually Learning?

- A language model is ultimately performing *compression* of the co-occurrence matrix
- Consider, what would we expect for each case?



- Embedding vector size:
  - $|V|$  ?
    - *Indicator features*
  - $|V|-1$  ?
    - Most similar two words (synonyms) share
  - $|V|/2$  ?
  - $|V|^{\frac{1}{2}}$  ?

# Word Embeddings: Pointwise Mutual Information

- In estimating latent representations that give us *co-occurrence* information, we tap into *mutual information*
- Pointwise mutual information between two events  $x$  and  $y$ :

$$\text{pmi}(x; y) \equiv \log_2 \frac{p(x, y)}{p(x)p(y)}$$

- Ratio of the likelihood that  $(x, y)$  co-occur to the likelihood that they occur independently
  - PMI(*spongebob*; *squarepants*)?
  - PMI(*the*; *a*)?

# Word Embeddings: Pointwise Mutual Information

$$\text{pmi}(x; y) \equiv \log_2 \frac{p(x, y)}{p(x)p(y)} = \log_2 \frac{p(x|y)}{p(x)} = \log_2 \frac{p(y|x)}{p(y)}$$

- What do these PMI formulations start to look like?
- Estimate of most likely context given a target word:
  - $\text{argmax}_{w(i-k) \in V} (p(w_{i-1} w_{i-2} \dots w_{i-n+1} | a) / p(w_{i-1} w_{i-2} \dots w_{i-n+1}))$
  - $p(w_{i-1} w_{i-2} \dots w_{i-n+1} | a) \rightarrow$  co-occurrence of context and target
  - $p(w_{i-1} w_{i-2} \dots w_{i-n+1}) \rightarrow$  likelihood of context showing up in docs
- Estimate of the most likely target word given a context:
  - $\text{argmax}_{a \in V} (p(a | w_{i-1} w_{i-2} \dots w_{i-n+1}) / p(a))$
  - Co-occ of  $a$  and context over independent likelihood of  $a$

# NLP FUNDAMENTALS: Word Embeddings

- A word embedding is a *vector representation* of a word
- Word embeddings can be learned for specific tasks (e.g., our book classification task) or from self-supervised objectives
- “**You shall know a word by the company it keeps**” - Firth
- The guiding principle of many learned word embeddings is that two words who share similar *context* should have embeddings that are close together in vector space
- **Word embedding learning algorithms estimate PMI between words and one another (their “company”)**

# Discussion Activity: Exploring Word Embeddings

```
> import gensim  
> wv = gensim.models.KeyedVectors.load_word2vec_format('~/Downloads/GoogleNews-  
vectors-negative300.bin', binary=True)  
  
> wv.most_similar(positive=['her', 'husband'], negative=['him'], topn=10)  
[('mother', 0.6206774711608887), ('daughter', 0.6129351258277893),  
('hus_band', 0.5745669603347778), ('wife', 0.5739557147026062),  
('hers', 0.5729579329490662), ('fiancé', 0.5609115958213806),  
('eldest_daughter', 0.5527141094207764), ('she', 0.5503020286560059),  
('fiance', 0.5470099449157715), ('huband', 0.5298539400100708)]  
  
> wv.most_similar(positive=['princess', 'baron'], negative=['prince'], topn=10)  
[('magnate', 0.592768669128418), ('tycoon', 0.5915622711181641),  
('barons', 0.5287919044494629), ('mogul', 0.521348237991333),  
('heiress', 0.4887126088142395), ('magnates', 0.46558016538619995),  
('industrialist', 0.4555717706680298), ('billionaire', 0.43505433201789856),  
('millionaire', 0.4347013831138611), ('tycoons', 0.4303288757801056)]
```

# Whiteboard Activity: Lexical Word Embeddings

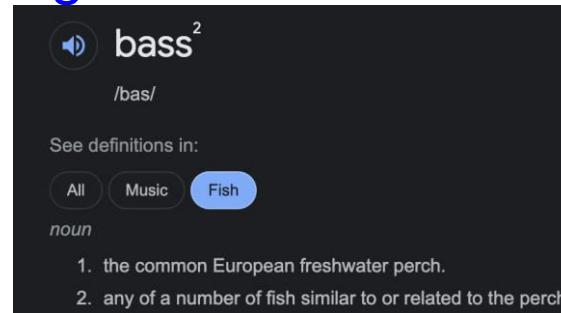
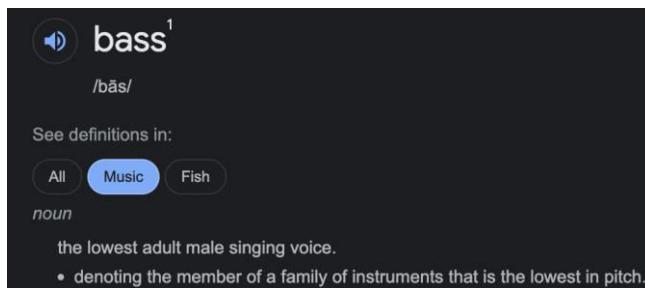
- Our word embedding map  $\mathbf{V}$  takes in a token and returns a vector of real numbers of dimension  $h$
- What problematic assumption does that formulation make?
  - *money is to bank as child is to daycare* [make true in 2d]
  - *road is to shoulder as river is to bank*
  - *leg is to hip as arm is to shoulder*
  - *cool is to hot as lame is to hip*
  - *abled is to lame as frantic is to cool* [🍺]
- Lexical embeddings: each token gets a single representation
- Does not take into account ***polysemy***

# Lexical Word Embeddings

- Lexical embeddings: each token gets a single representation
- Does not take into account ***polysemy***
- What will the vector representation of *bank* be?
  - $v(bank_{\text{finance}})p(bank_{\text{finance}}) + v(bank_{\text{river}})p(bank_{\text{river}})$
- Word sense vectors effectively averaged based on frequency
- What should we do about it?
  - Use a dictionary resource to see sense descriptions?
  - What's the problem with a dictionary-based approach?
    - brat ; bet ; ate ; slay ; ...
- [this is an instance of a much bigger hurdle: language isn't stationary ]

# Word Sense Disambiguation

- A whole core NLP task formed around this problem
  - Given a *dictionary* of senses for each polysemous word, identify which sense is being used in each sentence



- x= “she launched into a bass solo” ; y=1
- x= “the line went taut as the bass pulled”; y=2
- x= “the power of the bass broke the string”

# PREVIEW: Contextual Word Embeddings

- What do we *really* want if we're processing these sentences?
- $\mathbf{X} = \{\text{"she launched into a bass solo"}, \text{"the line went taut as the bass pulled"}, \text{"the power of the bass broke the string"}\}$
- We could preprocess the sentence to dynamically select *which* sense vector to choose for each word:
  - Learn individual sense vector for every word, then select which is right based on surrounding *context*
- Or we could supplant that two-step idea and learn a function that combines *input* lexical embeddings from each word to produce *output* **contextual** embeddings that consider surrounding words

## PREVIEW: Contextual Word Embeddings

- Learn a function that combines *input* lexical embeddings from each word to produce *output* **contextual** embeddings that consider surrounding words
- Requires processing a *sequence* of such input embeddings
- Module 2: Sequence-to-Sequence modeling!

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 1: Lecture 3  
Distributional Semantics and Lexical Embeddings

# CSCI 444: Natural Language Processing

Jesse Thomason

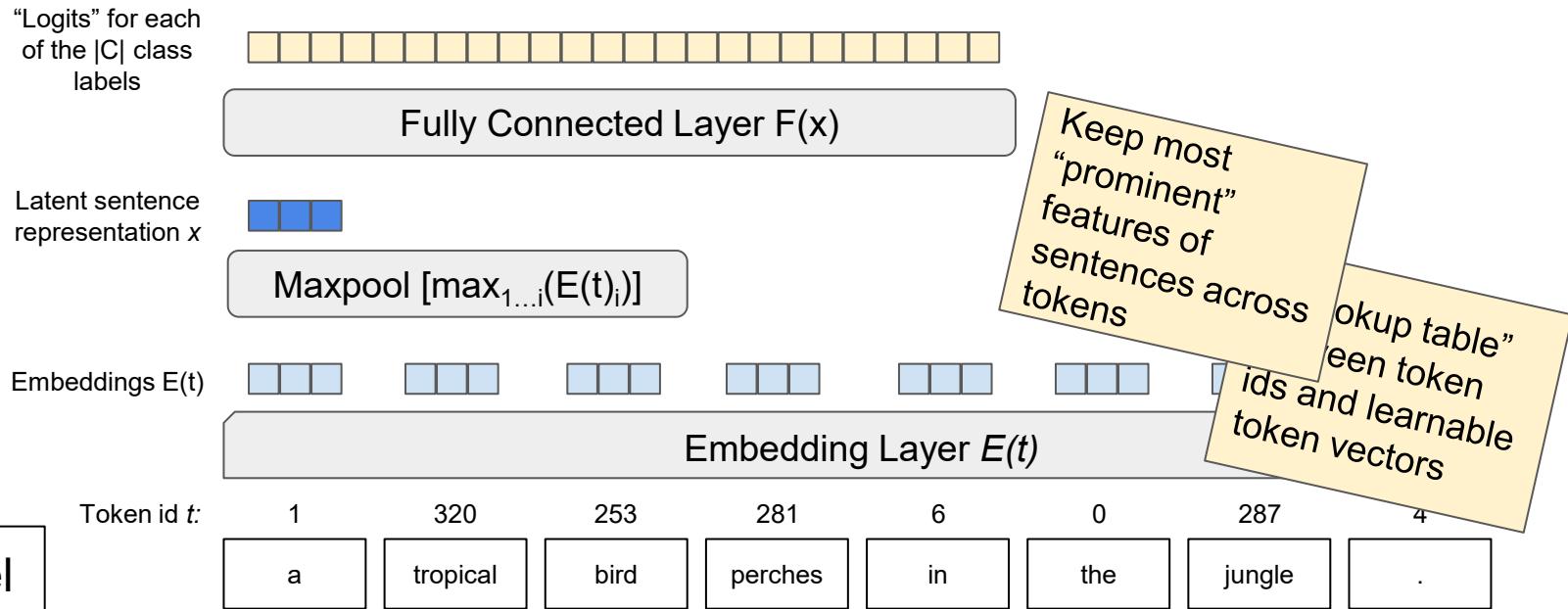
Module 2: Lecture 1  
Sequence to Sequence Modeling

<https://forms.gle/VS6yy2g9byE99n1h7>

# Text Classification

- Represent the *input* text of instances of  $D$ 
  - In fact, we need to define an *encoding function*,  $\phi(d)=x$
  - Our encoding function will map raw text to *input features* for the model  $M$
- Learn parameters of  $M$  that reliably map those input representations to the correct output classes
  - $M$  is parameterized by a set of parameters  $\theta$
  - Many learning algorithms exist to nudge  $\theta$  in directions that increase how often  $M(\phi(d)) = l$  for pairs  $(d, l)$

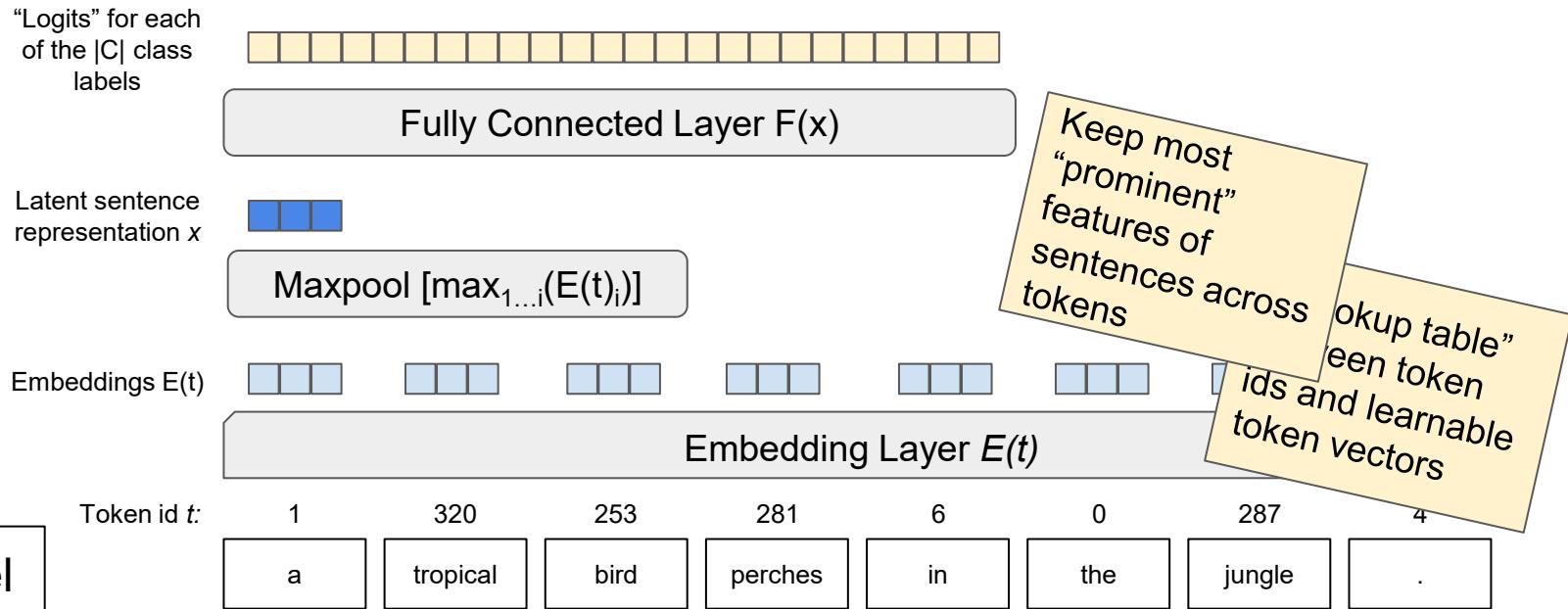
# Text Classification with a Neural Network



# ML FUNDAMENTALS: Neural Network Training

- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**
- What do we need?
  - Pass over data for multiple **epochs**
  - Sample **mini-batches** of our data
  - Step the optimization algorithm to complete an **iteration**
- Let's see how this looks for our proposed neural network

# Text Classification with a Neural Network



# Text Classification with a Neural Network

- For model  $M$ , let's say we have 1000 tokens  $T$  we will learn embeddings for, and mapping to 100 class labels
- Let's set our *word embedding dimension* to 128.
- Then we need parameters  $\theta$  for:
  - Embedding layer  $E(t)$  representing 1000 different 128-dimensional word vectors accessed by token id
  - Fully connected layer  $F(x)$  mapping 128-dimensional sentence vectors to the 100 output class labels

# Text Classification with a Neural Network

- $t$  token ids; each input has length  $L$  (e.g.,  $L$  tokens)
- What's our final function look like?

- Embedding:  $E(t_1) \dots E(t_L)$ 
  - size:  $128 * L$  [token emb \* len]
- Maxpool:  $\langle \max_{i=1 \dots L} (E(t_{li})) \rangle_{i=1 \dots 128}$ 
  - size:  $128$  [token emb]
- Fully-connected:  $F(\langle \max_{i=1 \dots L} (E(t_{li})) \rangle_{i=1 \dots 128})$ 
  - size:  $100$  [output]

# ML FUNDAMENTALS: Neural Network Layers

- What parameters  $\theta$  are in each of our proposed NN layers?
- Embedding layer:
  - Matrix  $E^{1000 \times 128}$ ; 1000 rows representing each token id, with each token having 128 learnable embedding params
- Maxpool layer:
  - No parameters! This is just an operation.
- Fully-connected layer:
  - Matrix  $W_F^{128 \times 100}$ ; a *weight* for how each latent dimension contributes to each output class
  - Vector  $b_F^{100}$ ; a *bias* on each output class

# ML FUNDAMENTALS: Neural Network Layers

- Fully-connected layer:
  - Matrix  $W_F^{128 \times 100}$ ; a *weight* for how each latent dimension contributes to each output class
  - Vector  $b_F^{100}$ ; a *bias* on each output class
  - $y = xW + b$  for sentence vec  $x$  that's 128d
- Finally, we could add a non-linearity to squeeze the dimensions of  $y$ . Sigmoid, arc-tangent, rectified linear units, etc...
  - $y = \sigma(xW + b)$

# Text Classification with a Neural Network

- $t$  token ids; each input has length  $L$  (e.g.,  $L$  tokens)
- What's our final function look like?
  - Fully-connected:  $F(\langle \max_{l=1 \dots L}(E(t_{li})) \rangle_{i=1 \dots 128})$ 
    - $y = \sigma(xW + B)$
    - $\sigma(\langle \max_{l=1 \dots L}(E(t_{li})) \rangle_{i=1 \dots 128} W_F + b_F)$
- So what are we taking the gradient of?
- $P(i) = \sigma(\langle \max_{l=1 \dots L}(E(t_{lj})) \rangle_{j=1 \dots 128} W_F + b_F)_i$

**Cross entropy loss:** a measure of how well the true label distribution compares to the predicted distribution

$$H(P^* | P) = - \sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTRIBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTRIBUTION}}$$

# Training our Text Classifier

## Mini-Batch

1	320	253	281	6	0	287	4
a	tropical	bird	perches	in	the	jungle	.

1	490	322	163	4	
a	small	white	fox	.	?

0	330	261	8	1	1000	741	18
the	orange	spider	on	a	UNK	leaf	is

82	189	
big	elephant	?

:

# Training our Text Classifier

## Mini-Batch

1	320	253	281	6	0	287	4
a	tropical	bird	perches	in	the	jungle	.
1	490	322	163	4	999	999	999
a	small	white	fox	.	PAD	PAD	PAD
0	330	261	8	1	1000	741	18
the	orange	spider	on	a	UNK	leaf	is
82	189	999	999	999	999	999	999
big	elephant	PAD	PAD	PAD	PAD	PAD	PAD

# Padding Tokens

# Training our Text Classifier

## Sequence Start/End Tokens

Mini-Batch

997	1	320	253	281	6	0	287	4	998
START	a	tropical	bird	perches	in	the	jungle	.	END
997	1	490	322	163	4	998	999	999	999
START	a	small	white	fox	.	END	PAD	PAD	PAD
997	0	330	261	8	1	1000	741	18	475
START	the	orange	spider	on	a	UNK	leaf	is	sitting
997	82	189	4	998	999	999	999	999	999
START	big	elephant	.	END	PAD	PAD	PAD	PAD	PAD

Padding Tokens

# Training our Text Classifier

- $t$  token ids; each input has length  $L$  (e.g.,  $L$  tokens)
- For minibatch size  $b$ , our input is a matrix with  $b$  rows of  $L$  columns each with a value in 0 to  $t$ 
  - How can we represent the input using only binary values?

997	320	253	281	998
997	490	322	163	4
997	330	1000	1	998
997	189	82	999	998
997	1	741	18	745

A tensor is an N-dimensional array of data



## Coding Activity: Book Identification

- Starter and data in gDrive in “Activities” folder
- You won’t need to run the code to ponder these questions.
  - How is the data split into train/val? What will trained model performance on val tell us? What won’t it tell us?
  - What does the model look like? What does the code tell you about the encoding of the data?
  - How are mini-batches sampled?
  - Can you identify where the gradient is estimated based on a mini-batch? Where is an *iteration* completed?
  - What might go well in this task? What might go wrong?

## Coding Activity: Book Identification

- Let's run the code and consider more questions based on the output we get.
  - Is the model performing well?
  - What could we do to mitigate what's happening?
  - What else could we do?
  - Is the model doing well now?
  - What's, realistically, the best thing we can hope the model would learn?
  - What's the dumbest thing we can hope the model will learn that still works?

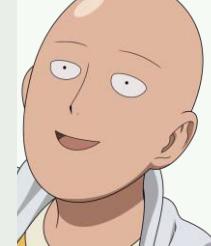
## Coding Activity: Book Identification

- Term Frequency Inverse Document Frequency (TF-IDF) word representations
- A  $|C|$  dimensional word vector where each element is that word's *relative frequency* in class  $c$

**128d Emb+Pool+FC**



**TF-IDF Vectors**

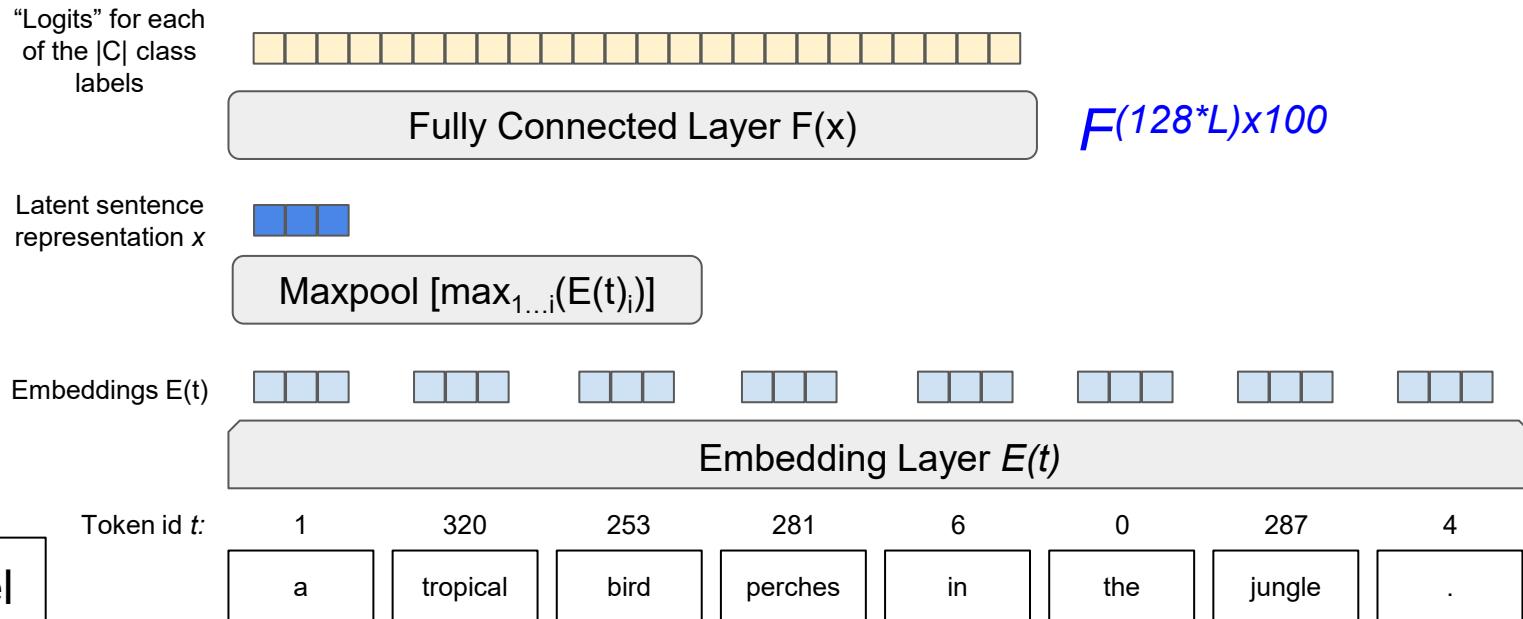


# ML FUNDAMENTALS: Running List of Lessons

- Always look at your data! What will the model see?
- Always implement a strong baseline!
  - Strong does not always mean “clever”
  - Strong *never* means “fancy” or “complicated”
  - <https://twitter.com/seanjtaylor/status/1550326602105466880?s=20&t=0ZjH63z3S22vwAdG9l5D0g>

# Text Classification with a Neural Network

- What could be better?



# Text Classification with a Neural Network

- What could be better?

“Logits” for each  
of the  $|C|$  class  
labels



Fully Connected Layer  $F(x)$

$F(128*L) \times 100$

If  $L$  is short, we cut off info. If  $L$  is long, we have huge wasted parameters.

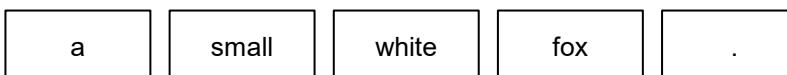
Embeddings  $E(t)$



Embedding Layer  $E(t)$

Token id  $t$ :

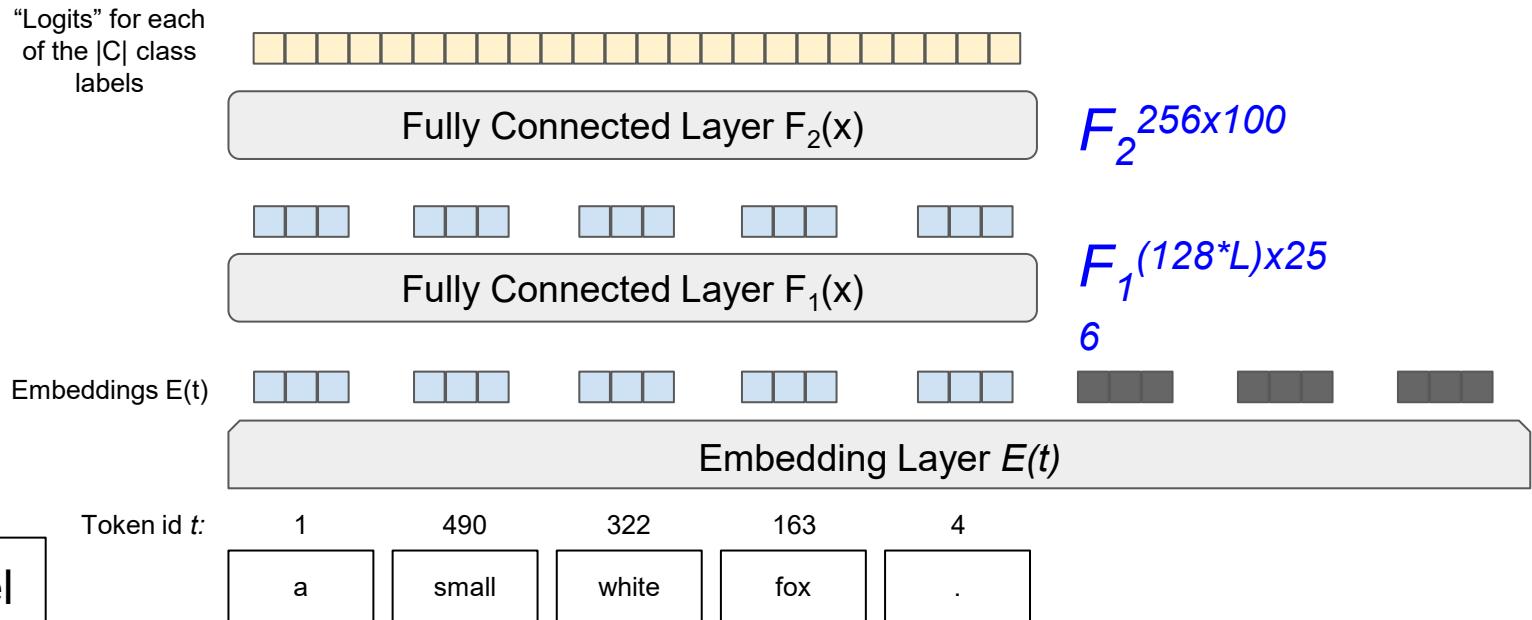
1      490      322      163      4



Word-level

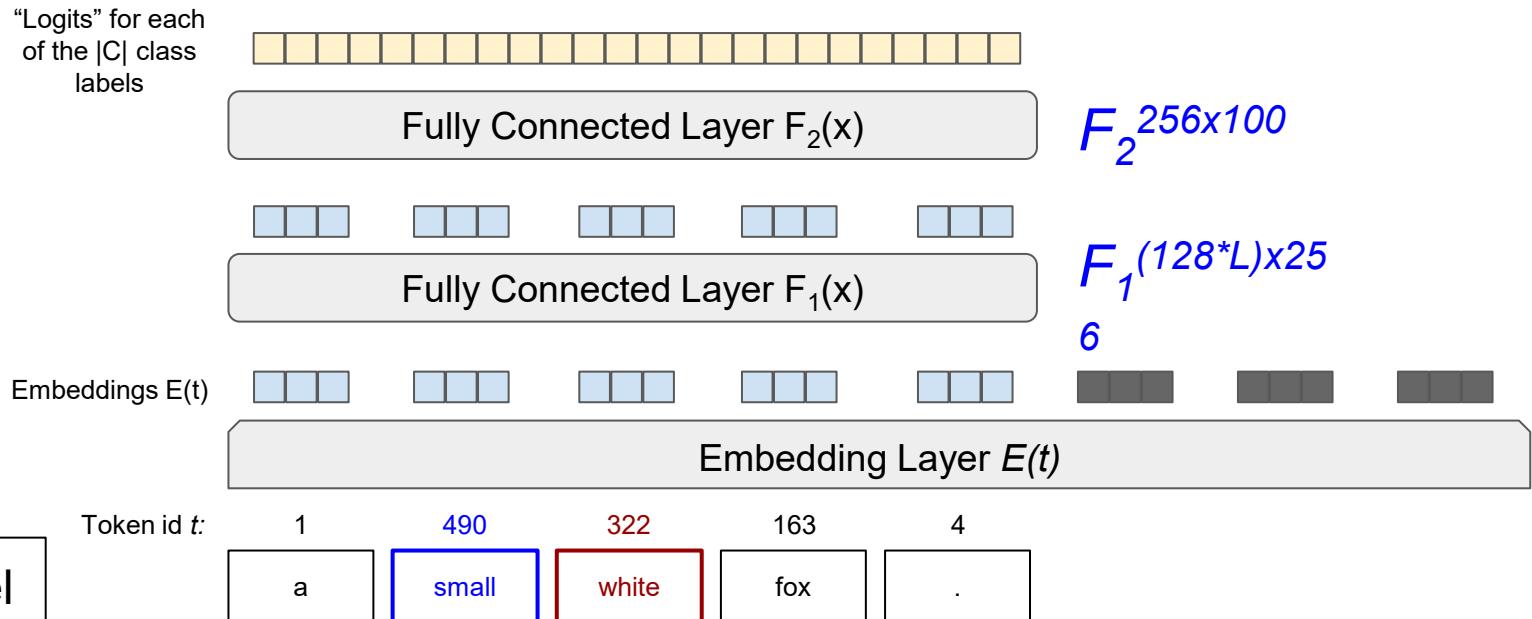
# Text Classification with a Neural Network

- What could be better?



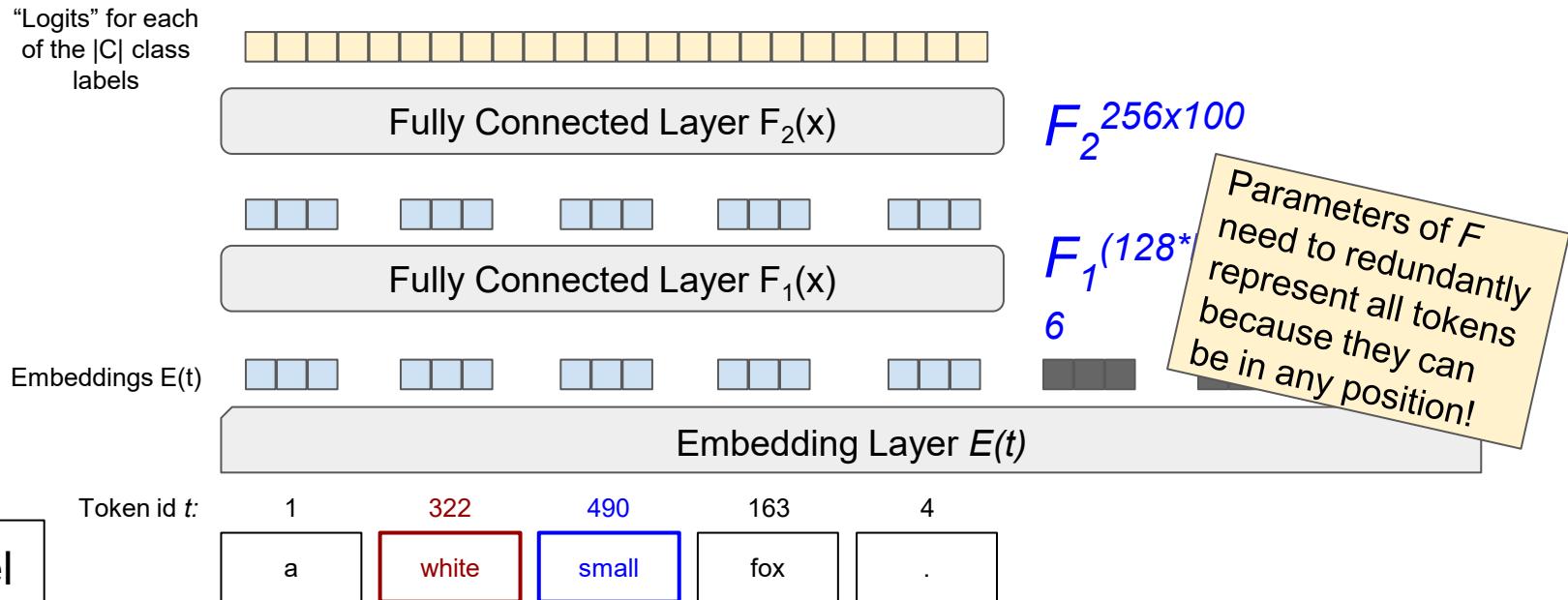
# Text Classification with a Neural Network

- What could be better?



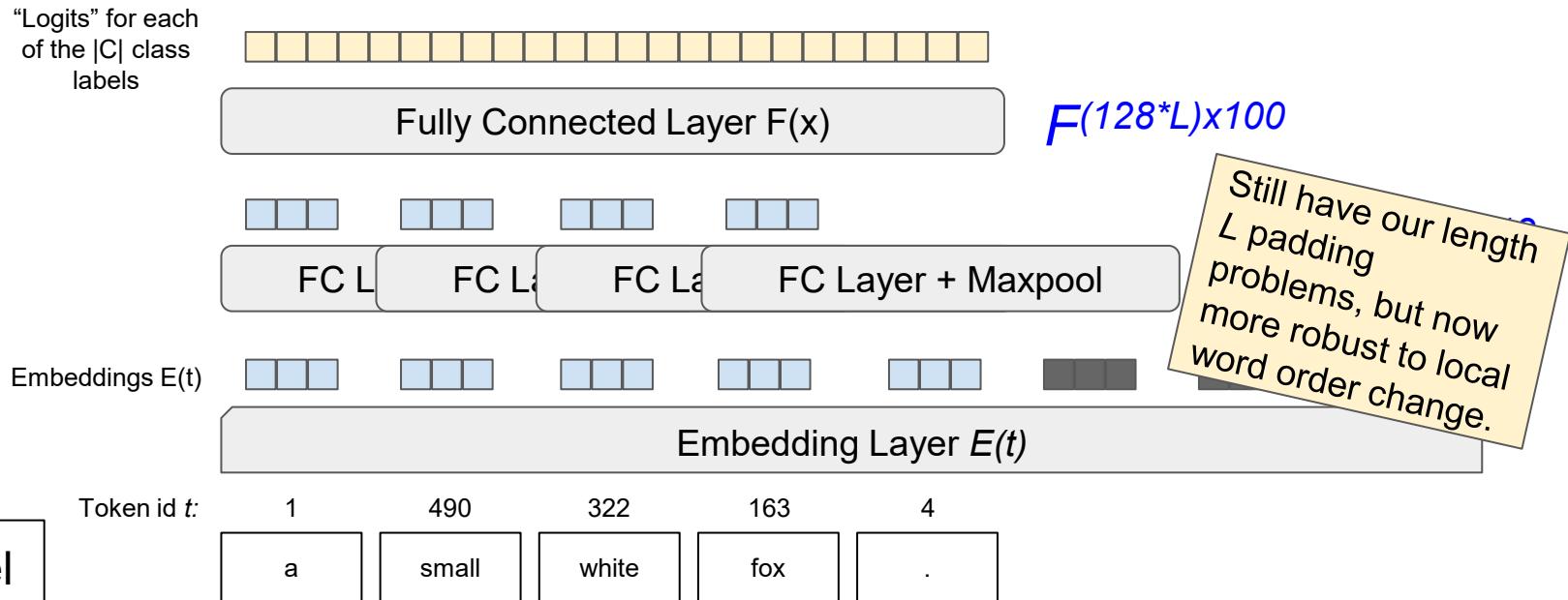
# Text Classification with a Neural Network

- What could be better?



# Text Classification with a Neural Network

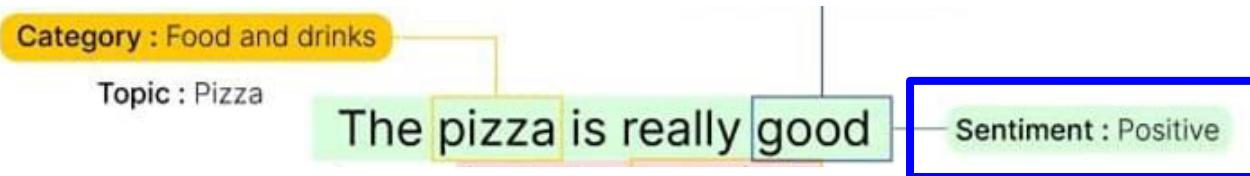
- What could be better?



# Text Classification with a Neural Network

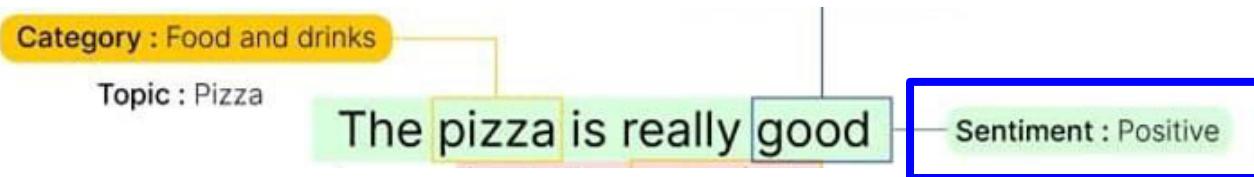
- The *maxpool* layer in our original network eliminated word order information
  - On the plus side: fewer parameters
  - On the downside: word order often does actually matter
- A single fully connected layer preserves order but has to memorize tons of redundant information
- A “sliding window” approach would take words 3-4 at a time and combine them
- What if we could learn how to encode **one word at a time** while iteratively building up our sentence representation  $x$ ?

# Text Classification



- Input space  $X$ ?
  - Review *tokens*  $T$
  - $X=T^N$  for max length  $N$
- Output space  $Y$ ?
  - Possible sentiments  $S$
  - $Y=S$
- Classification via a multi-layer perceptron:
  - Represent  $x$  as a multi-hot “bag of words” vector that tells us which words are in the review and which aren’t
- Weaknesses of an MLP model?
  - No ordering information
  - “cold ice cream and warm pizza”
  - “warm ice cream and cold pizza”

# Text Classification



- Input space  $X$ ?
  - Review *tokens*  $T$
  - $X=T^N$  for max length  $N$
- Output space  $Y$ ?
  - Possible sentiments  $S$
  - $Y=S$
- Classification via a CNN (sliding window)
  - Represent  $x$  as a sequence of token embedding vectors
  - Filters can combine words close together
- Weaknesses?
  - Words may have long-range dependencies that are hard to see with CNNs that favor neighborhood information
  - E.g., Main verb of "The horse raced past the barn fell"?

# Machine Translation



- Input space  $X$ ?
  - Input tokens  $T$
  - $X=T^N$  for max length  $N$
- Output space  $Y$ ?
  - Output tokens  $V$
  - $X=V^M$  for max length  $M$
- Would an MLP even make sense here?
  - What would our output classification be?
- Would a CNN even make sense here?
  - What might the output architecture look like?
- Probably we need an entirely different approach!

# Classes of Sequence-to-Sequence Problems

- One-to-one
  - One input produces one output
    - Image classification

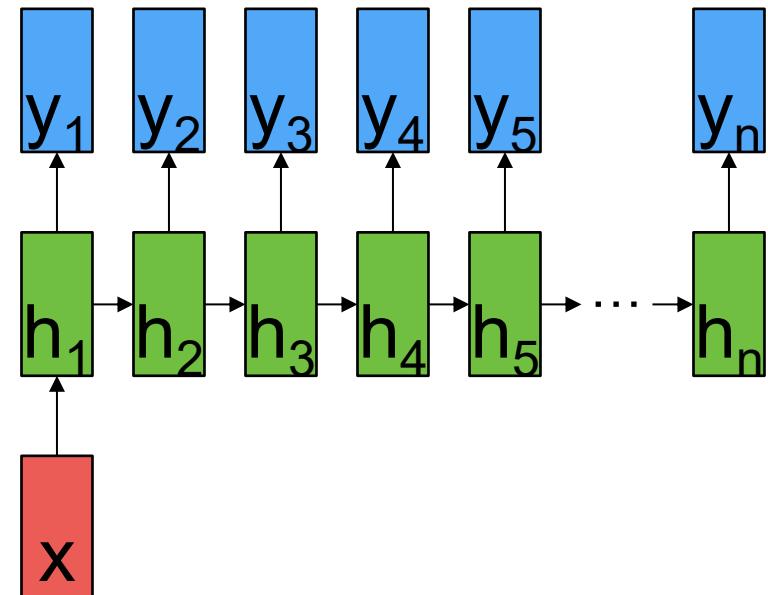


Person



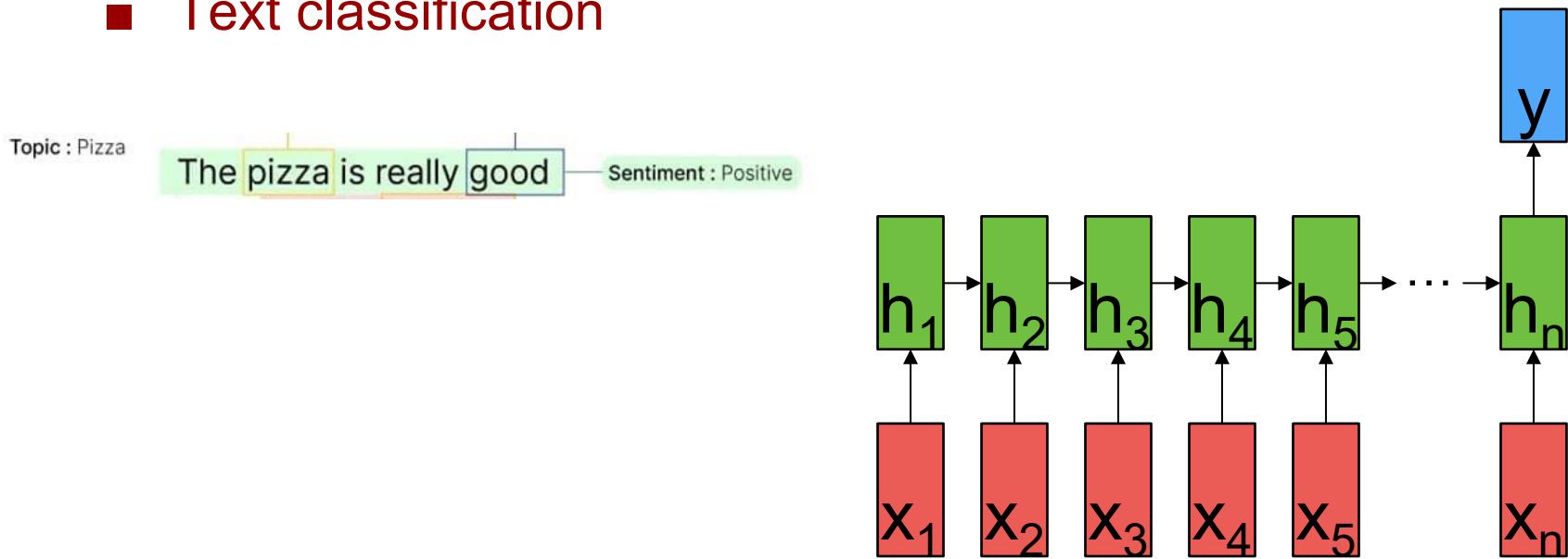
# Classes of Sequence-to-Sequence Problems

- One-to-many
  - One input produces a sequence of outputs
    - Image captioning



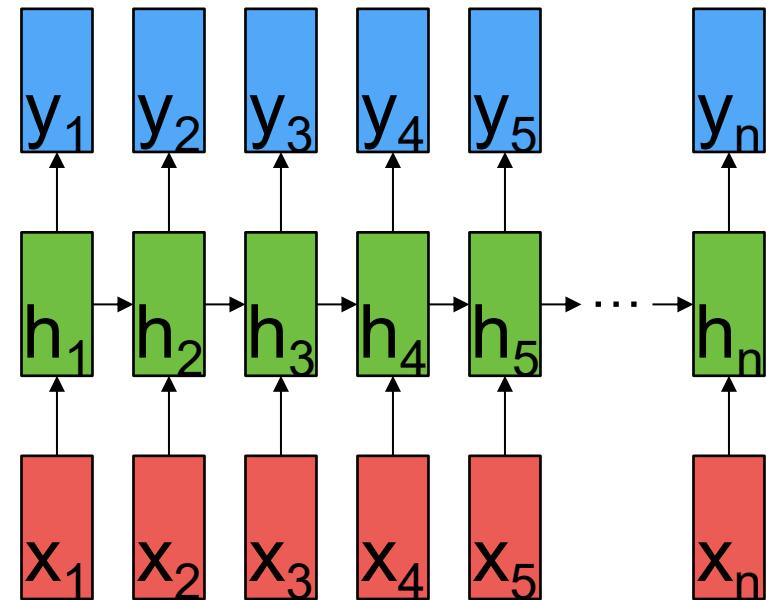
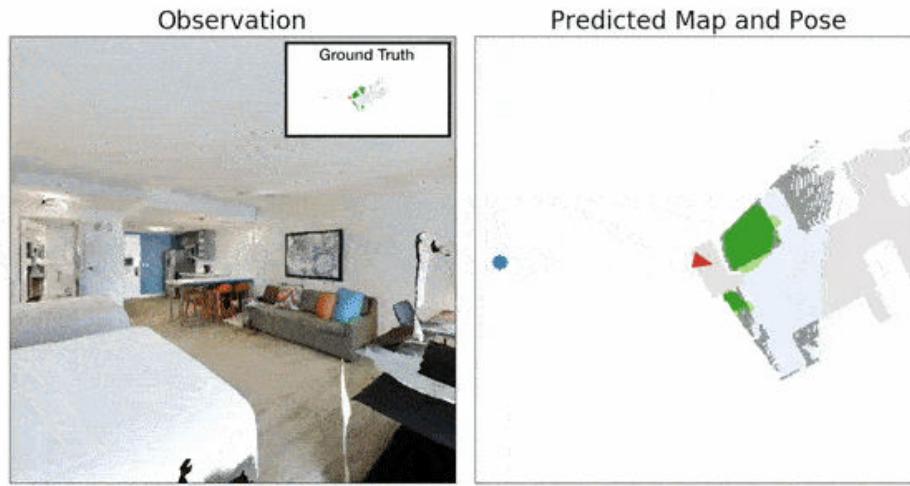
# Classes of Sequence-to-Sequence Problems

- Many-to-one
  - Sequence of inputs produces a single output
    - Text classification



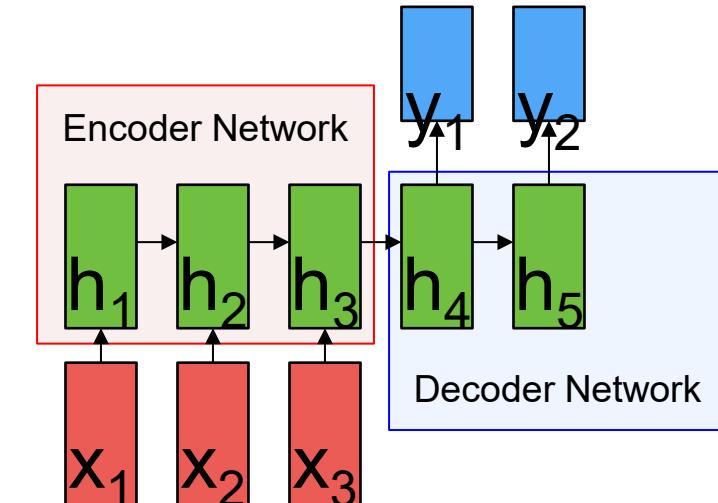
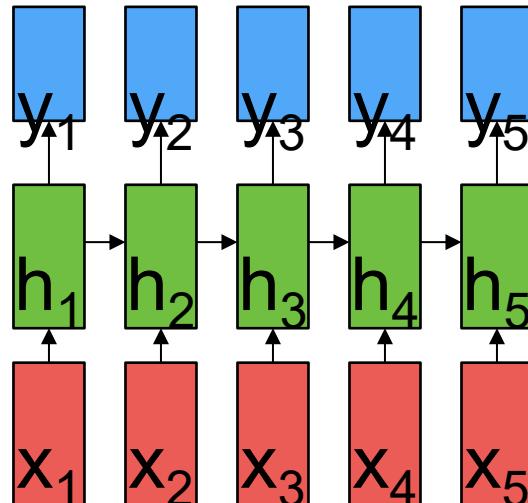
# Classes of Sequence-to-Sequence Problems

- Many-to-many
  - A sequence of inputs produces a sequence of outputs
    - Robot Actions



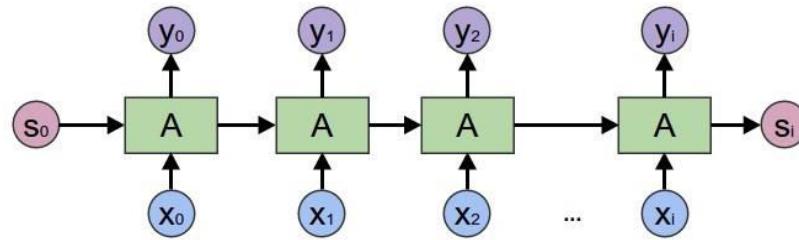
# Classes of Sequence-to-Sequence Problems

- Many-to-many variants:
  - Sequential one-to-one
    - Robot actions
  - Encoder-decoder
    - Machine Translation



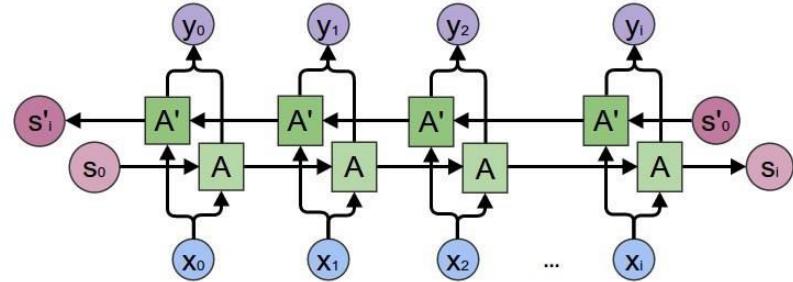
# Unidirectional and Bidirectional Seq2Seq Models

Forward RNN



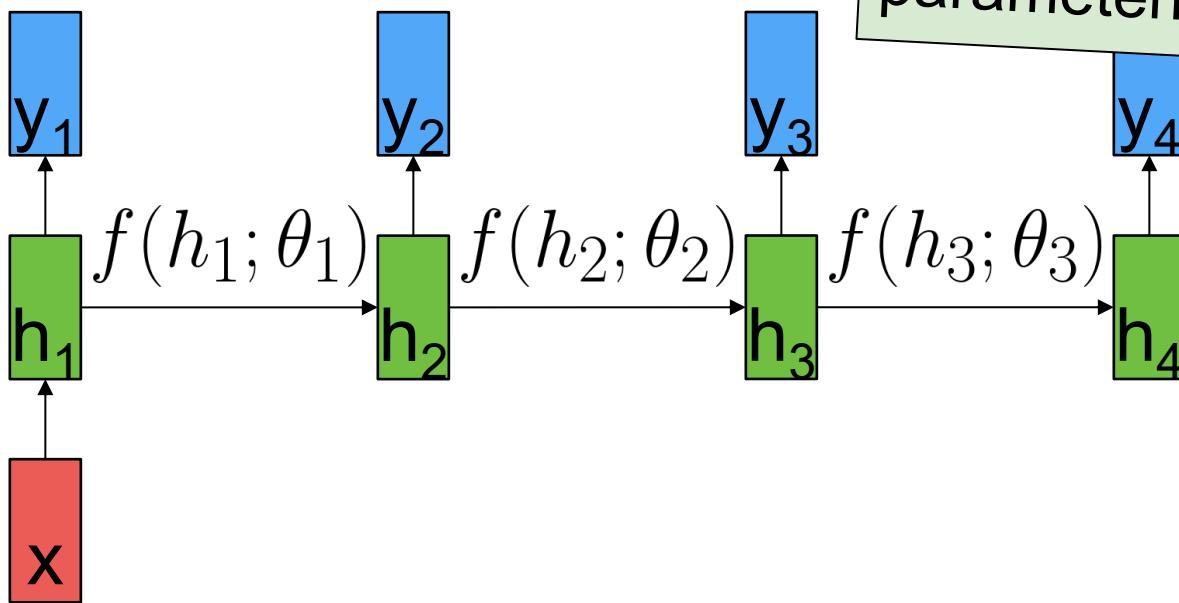
- Output  $y_t$  depends only on inputs so far  $x_{0:t-1}$ 
  - E.g., robot actions

Bidirectional RNN



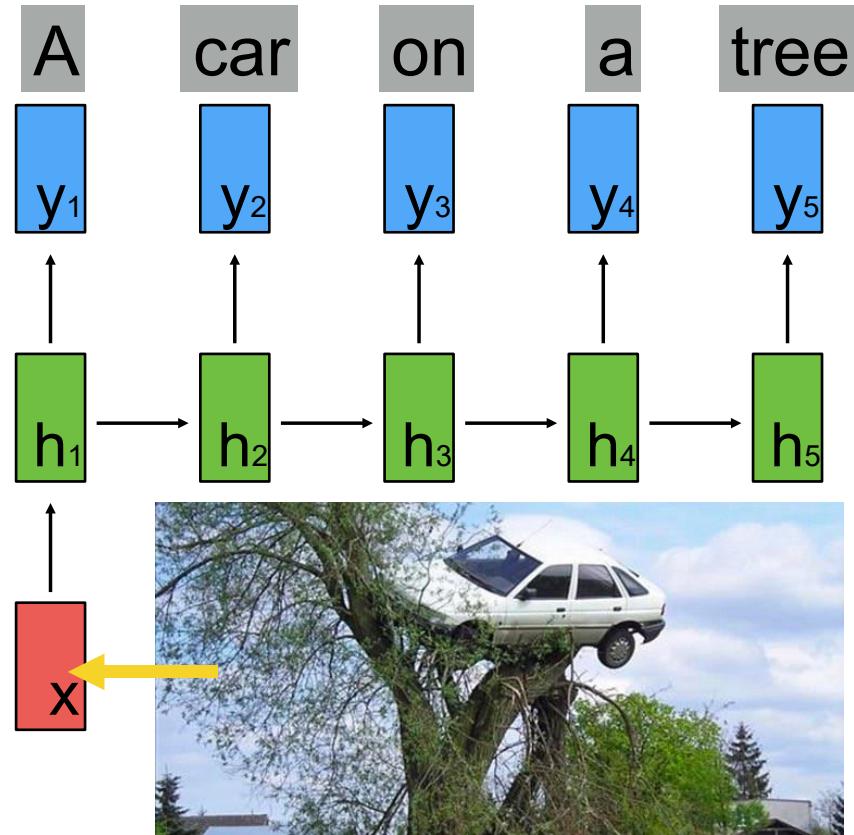
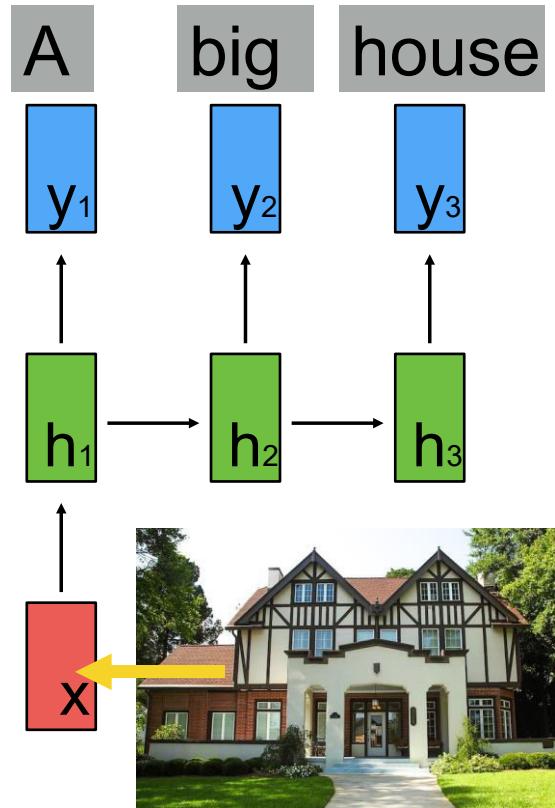
- Output  $y_t$  depends on all inputs  $x_{0:L}$ 
  - E.g., machine translation

# Sequence Modeling with Learned Neural Functions

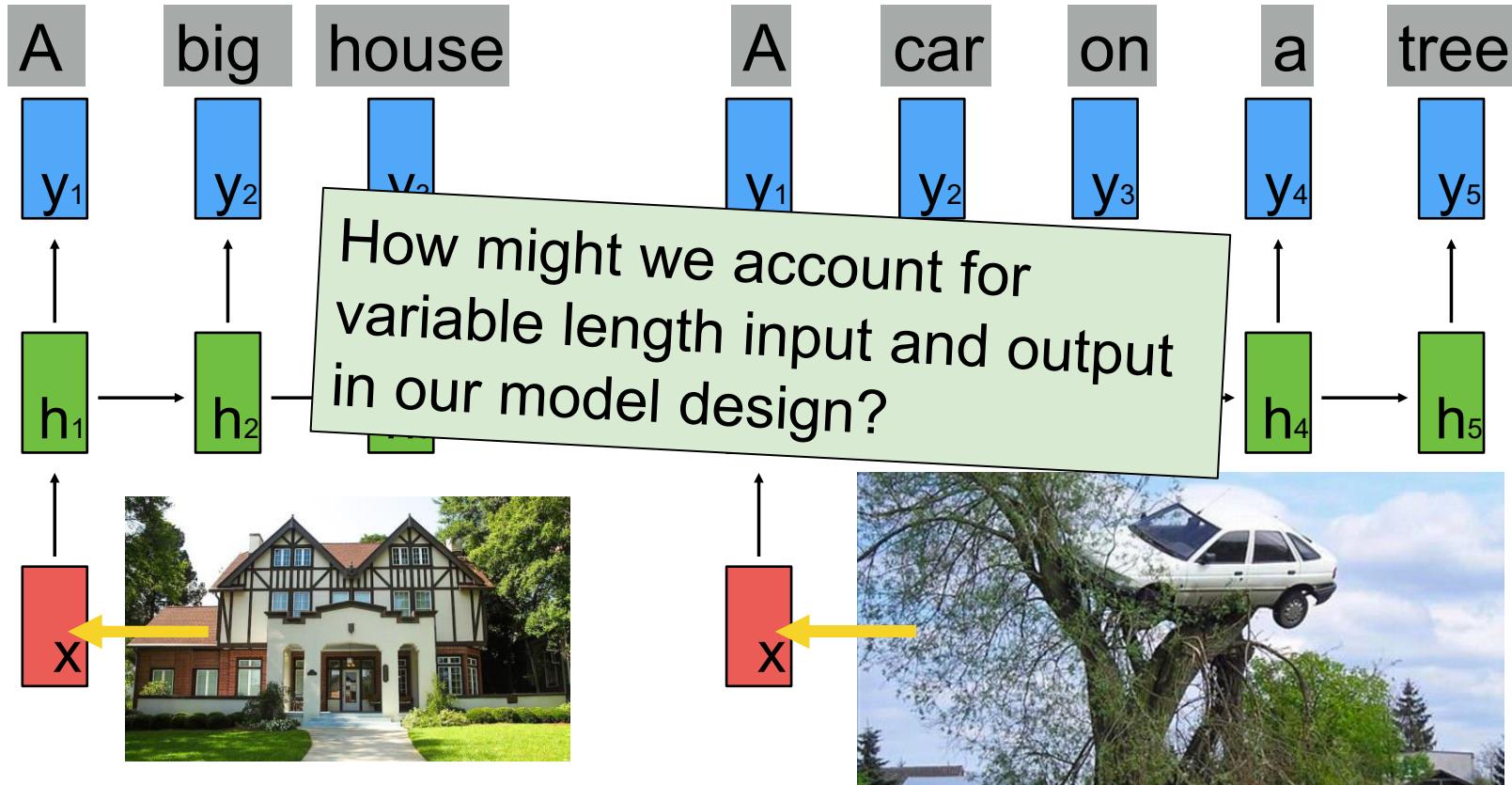


What's a weakness of this parameterization?

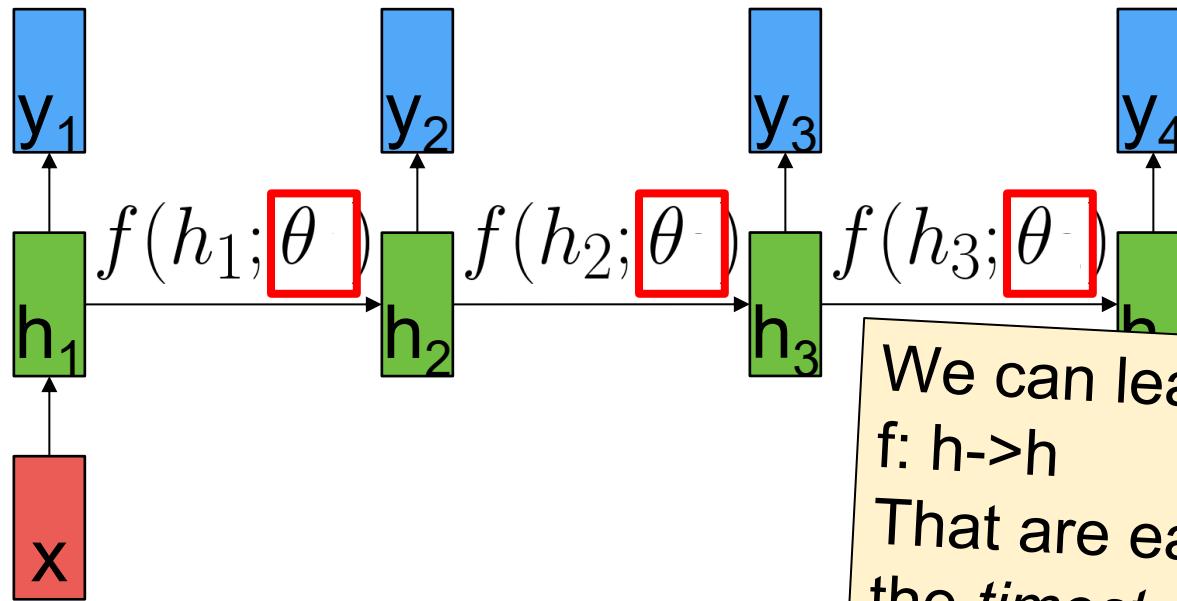
# Sequence Modeling with Learned Neural Functions



# Sequence Modeling with Learned Neural Functions



# Sequence Modeling with Learned Neural Functions

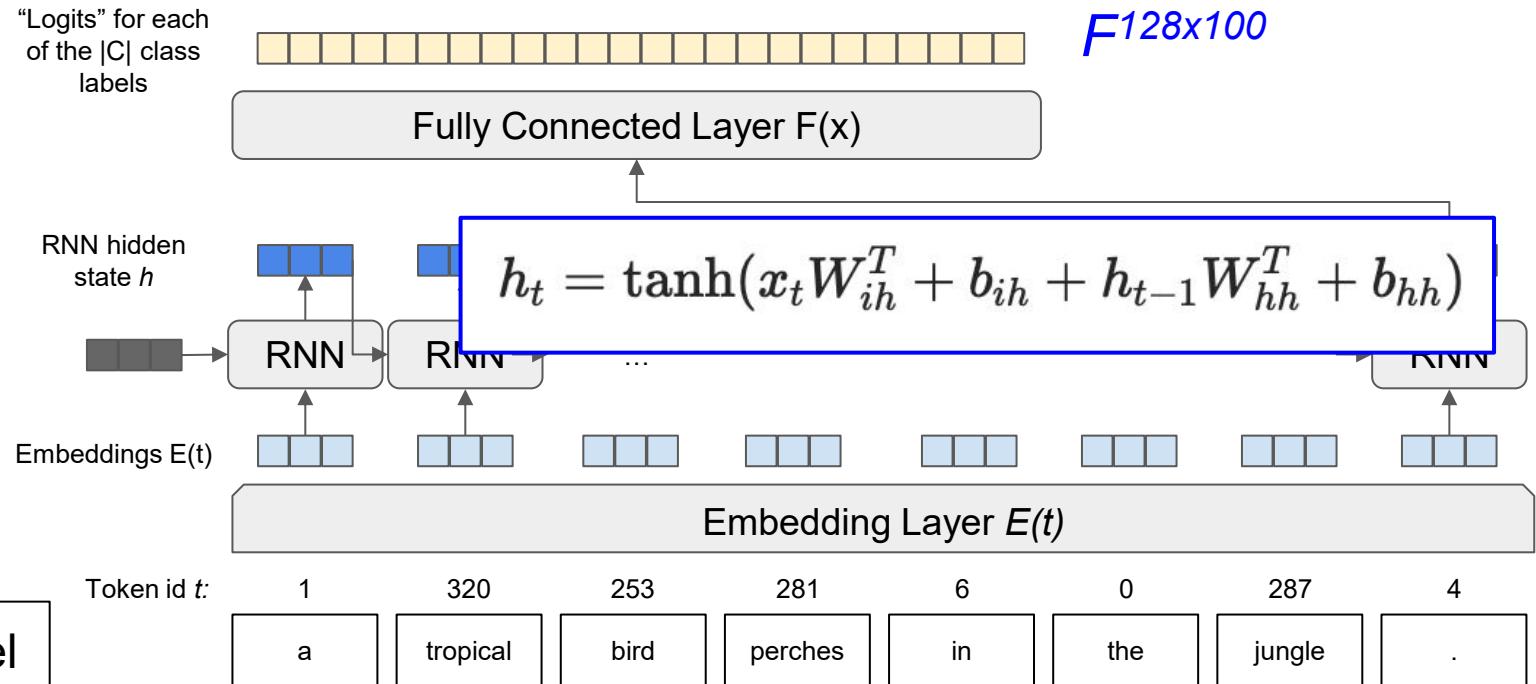


We can learn functions:  
 $f: h \rightarrow h$        $f: x \rightarrow h$        $f: h \rightarrow y$   
That are each independent of  
the *timestep* dimension.

# Text Classification with Recurrent Neural Networks

- What if we could learn how to encode **one word at a time** while iteratively building up our sentence representation  $x$ ?
- We can! **Recurrent Neural Network (RNN)**
- An RNN layer processes inputs one at a time conditioned on a *hidden state*, and outputs a new hidden state at each step.
- The hidden state is the product of a *non-linearity* which serves to represent non-linear functions and to keep the hidden state vector in check.

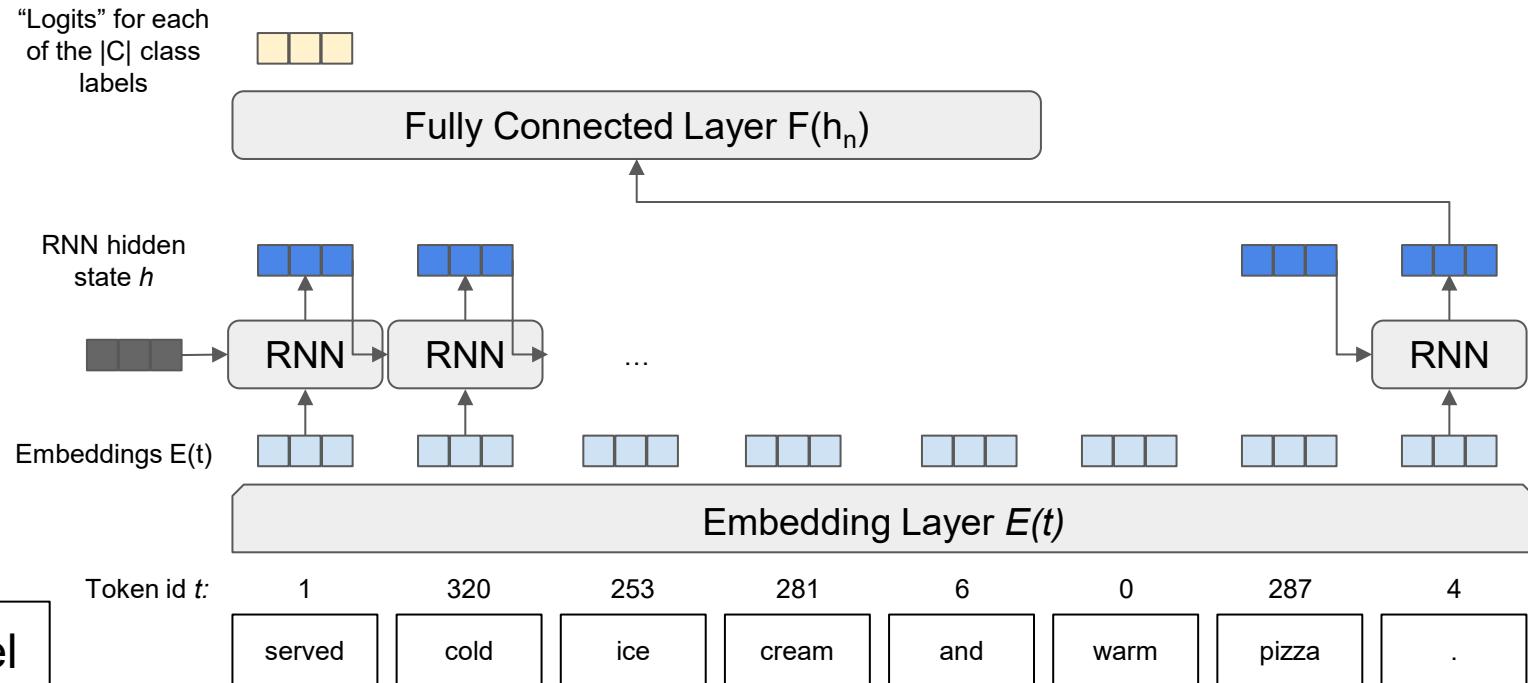
# Text Classification with a Recurrent Neural Network



# Text Classification with Recurrent Neural Networks

- 1000 tokens  $T$  we will learn 128-dim embeddings for, and we're classifying to 100 class labels.
- Let's use 128-dim for the RNN hidden state too.
- Then our parameters  $\theta$  are:
  - Embedding layer  $E^{1000 \times 128}$ , a matrix of 1000 different 128-dimensional word vectors accessed by token id (row)
  - RNN layer  $R^{(128+128) \times 128}$ , which takes in its own previous hidden state (128d) and the next word embedding (128d)
  - And finally  $F^{128 \times 100}$ , a linear layer to our output classes.

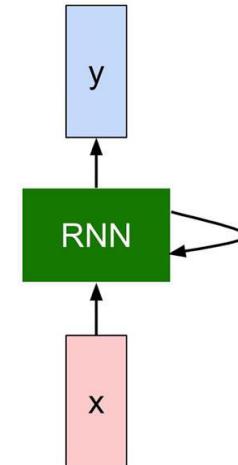
# Text Classification with a Recurrent Neural Network



# Recurrence In Neural Networks

- We can process a sequence of inputs one input at a time by defining a *recurrence function*
- Recurrence function takes two arguments: the next input and a state representing aggregated information from past inputs

$$h_t = f_W(h_{t-1}, x_t)$$

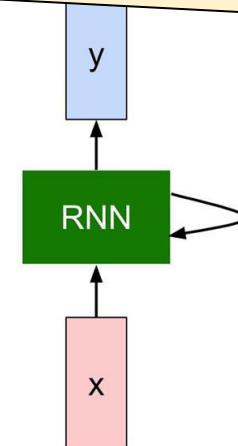


# Recurrence In Neural Networks

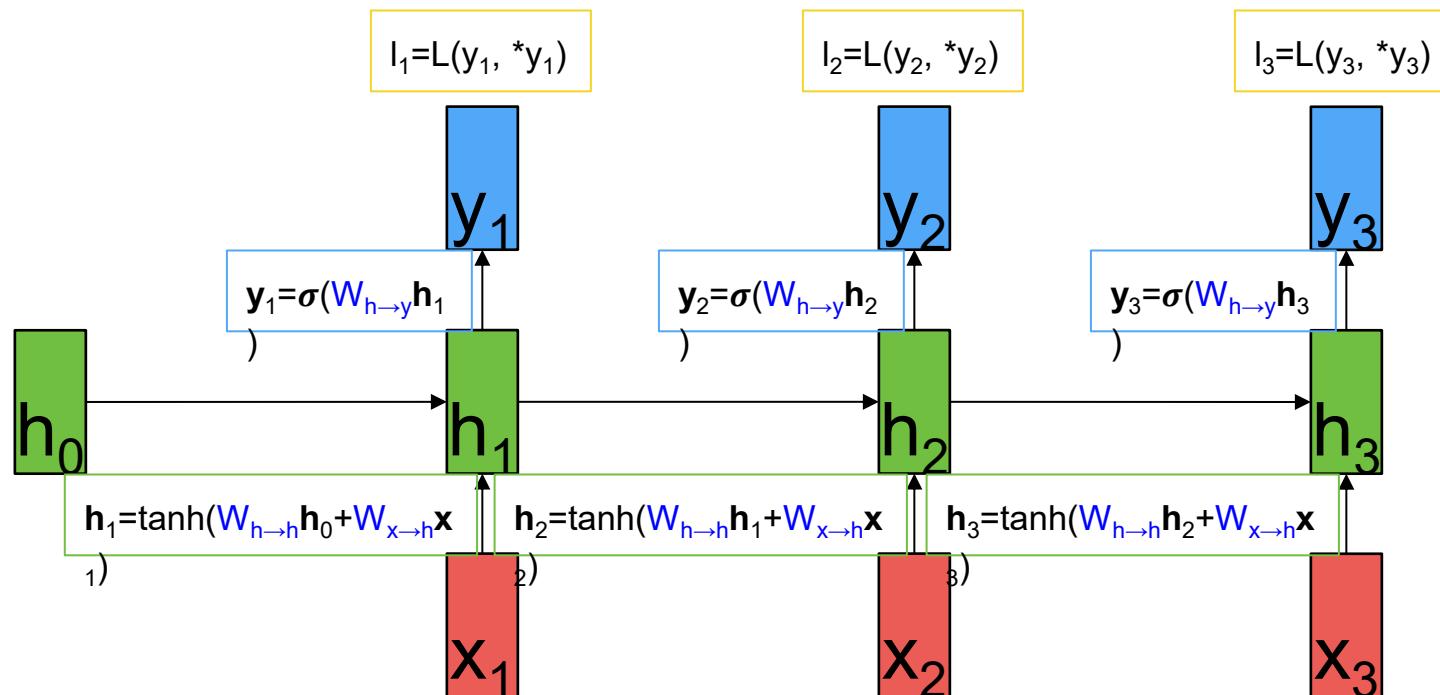
- Remember how CNN filters were just little linear layers?
- Same story for how we define  $f_W$ !
  - Vanilla RNN:
    - $\mathbf{h}_t = \tanh(\mathbf{W}_{h \rightarrow h} \mathbf{h}_{t-1} + \mathbf{W}_{x \rightarrow h} \mathbf{x}_t)$
    - $\mathbf{y}_t = \sigma(\mathbf{W}_{h \rightarrow y} \mathbf{h}_t)$

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

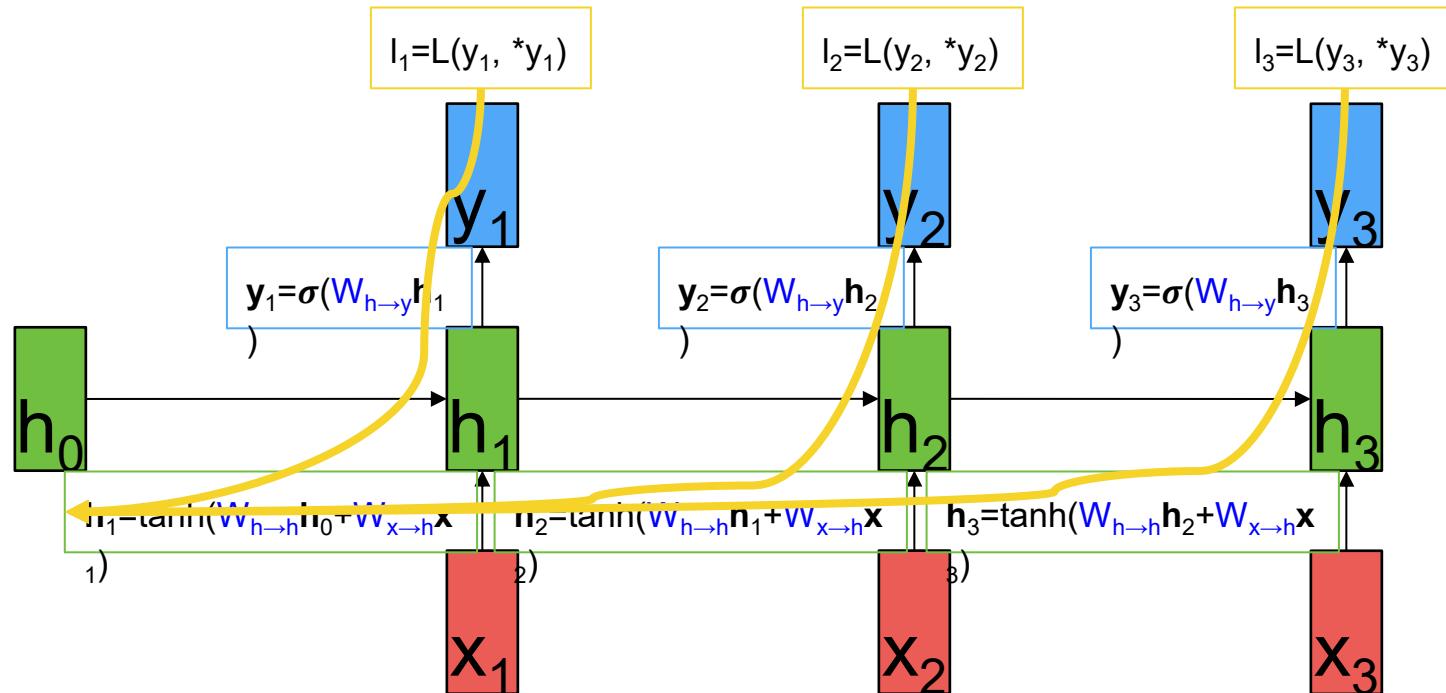
Note that to compress notation we frequently drop bias vectors.



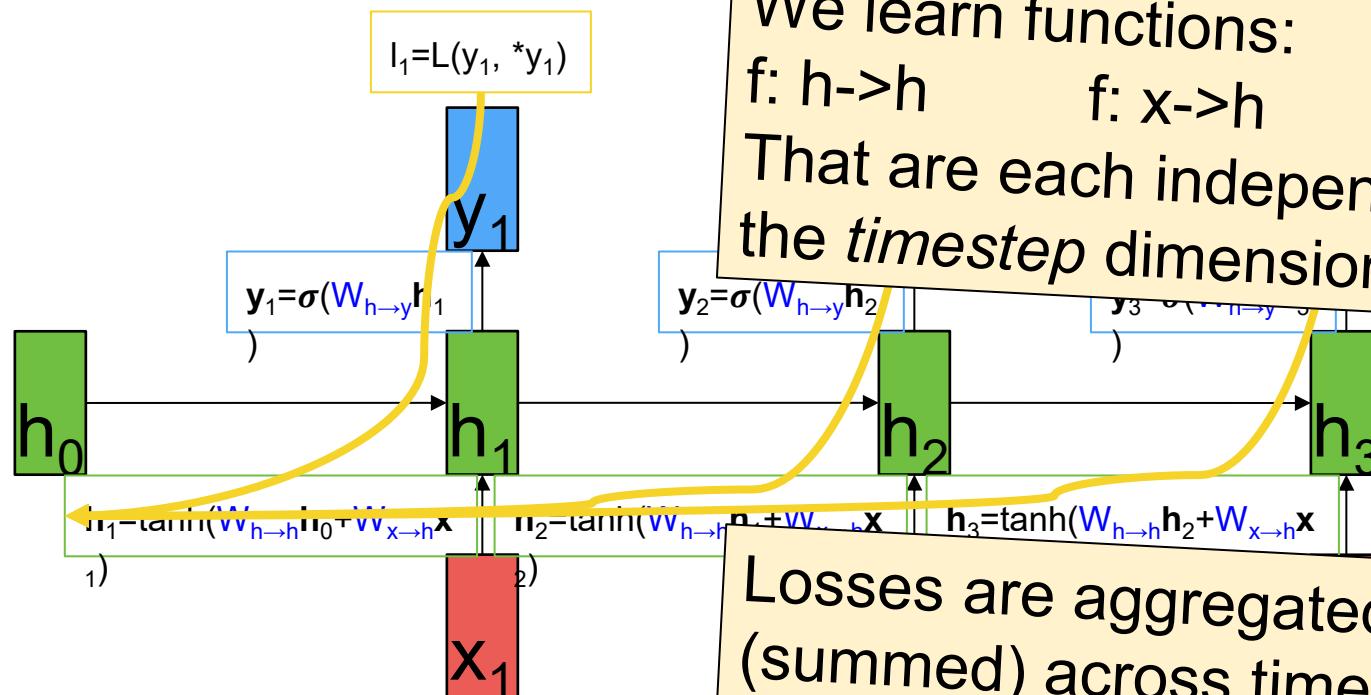
# High Level Computation Graph for RNN



# Backpropagation for RNN



# Backpropagation for RNN



We learn functions:

$f: h \rightarrow h$

$f: x \rightarrow h$

$f: h \rightarrow y$

That are each independent of the *timestep* dimension.

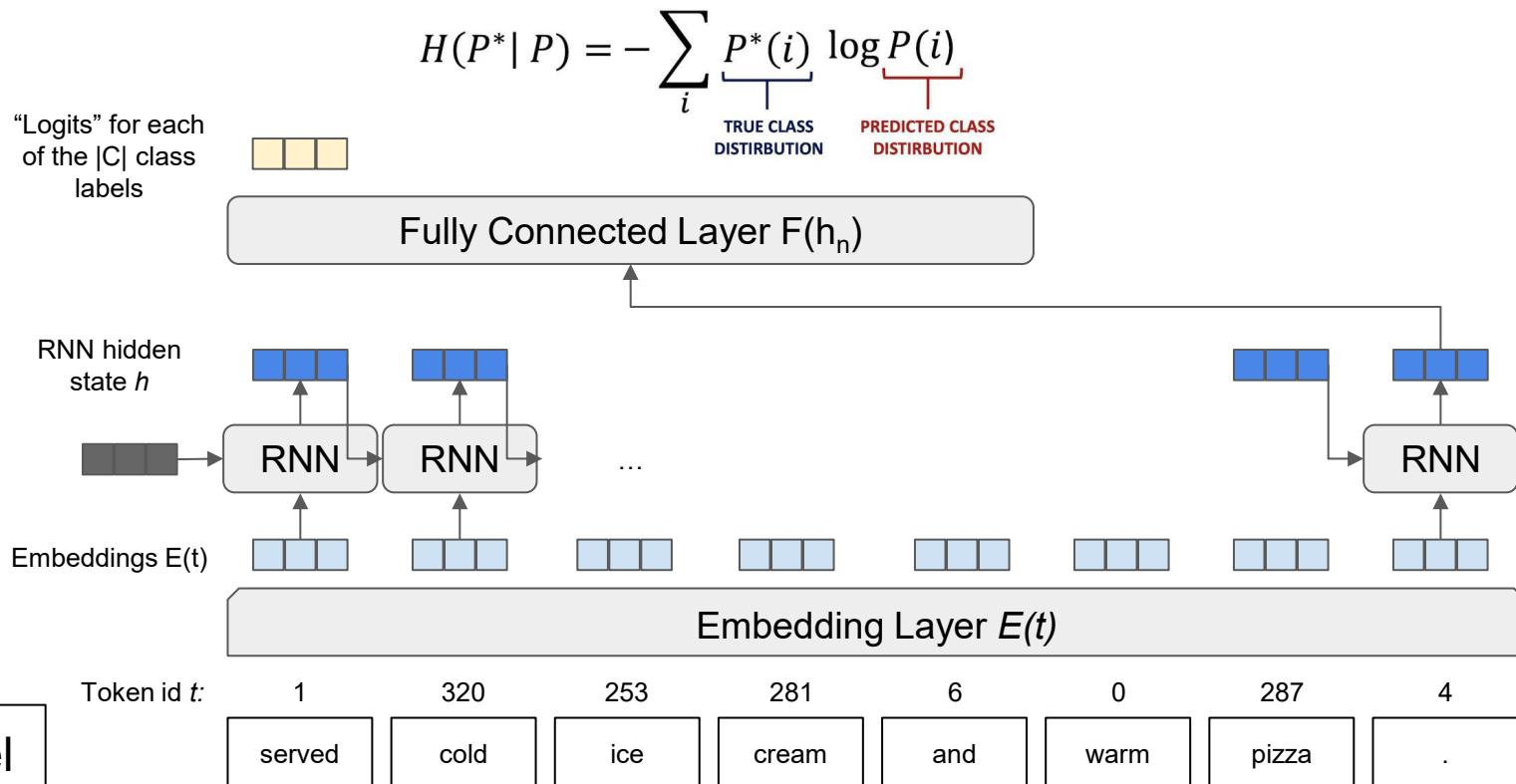
Losses are aggregated (summed) across timesteps.

# Backpropagation for RNNs

- But we... don't use RNNs?
- I know Transformers are all the rage, but even when RNNs were common we stopped using vanilla RNNs. Why?
- Let's look at our gradient step again:
  - $\theta := \theta - \varepsilon \nabla_{d_i \dots d_j} L[M(\phi(d), \theta)]$
  - Need the derivative of, in this example, cross entropy
- So  $L$  is  $H(P^*|P)$
- $P(i) = M(\phi(d), \theta)$
- $M$  is our fancy new RNN

$$H(P^* | P) = - \sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTRIBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTRIBUTION}}$$

# Text Classification with a Recurrent Neural Network



# The Slide Where RNNs Lose

- $P(i) = M(\phi(d), \theta)$ ;  $R$  the RNN layer we're learning
  - ...  $E(t_1) \dots E(t_L) \dots$
  - ...  $R(E(t_1), 0) \dots$
  - ...  $R(E(t_2), R(E(t_1), 0)) \dots$
  - ...  $R(E(t_L), \dots R(E(t_4), R(E(t_3), R(E(t_2), R(E(t_1), 0)))) \dots$
  - $P(i) = F(R(E(t_L), \dots R(E(t_4), R(E(t_3), R(E(t_2), R(E(t_1), 0)))) \dots))$
- Just thinking about the size of  $\nabla$ , it'll be biggest at  $F$
- For every nested call to  $R$ , the gradient *vanishes* further; remember that we're non-linearly squeezing with each  $R$  too

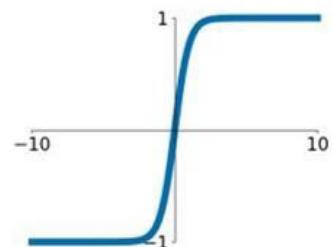
# Addressing the Vanishing Gradient Problem

- It turns out there are a lot of ways to mitigate this problem
  - None of them are satisfying, some of them are simple
- Long short-term memory (LSTM)
  - A dumb name for some complex cell architecture
  - The details of LSTMs have taken a backseat in NLP lately
  - Upshot: preserve more gradient by keeping more of the hidden state around between updates
- Gated Recurrent Units (GRU) are in the same family of attempts to get around this issue with sequence problems

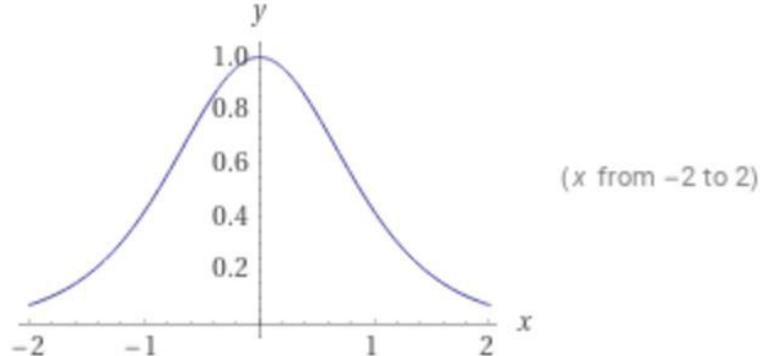
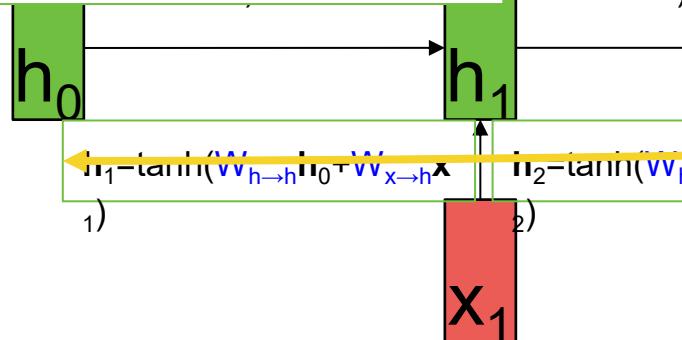
# The Vanishing Gradient Problem

**tanh**

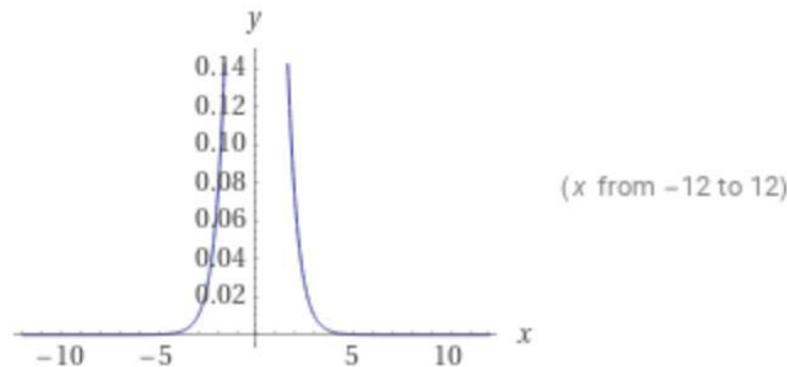
$\tanh(x)$



$$\frac{d}{dx} \tanh(x) = \text{sech}^2(x)$$

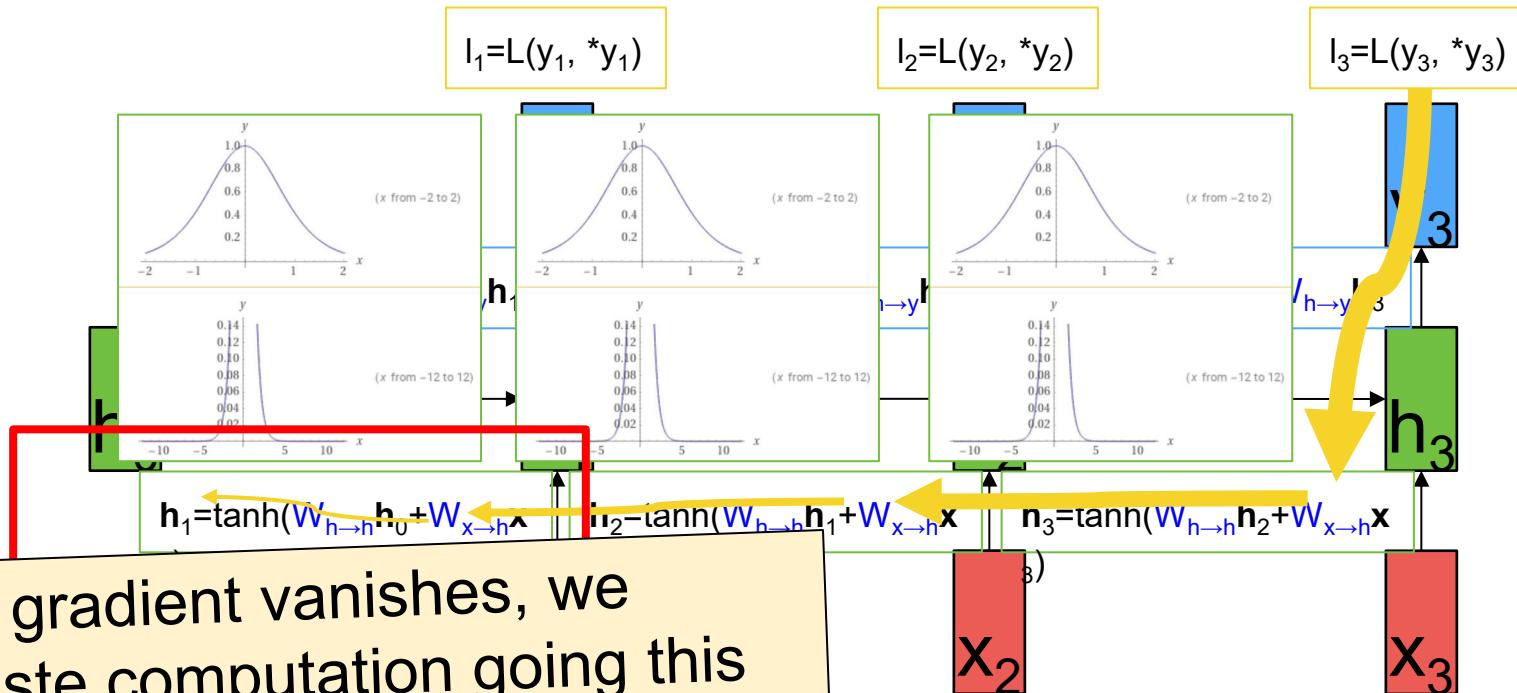


( $x$  from -2 to 2)

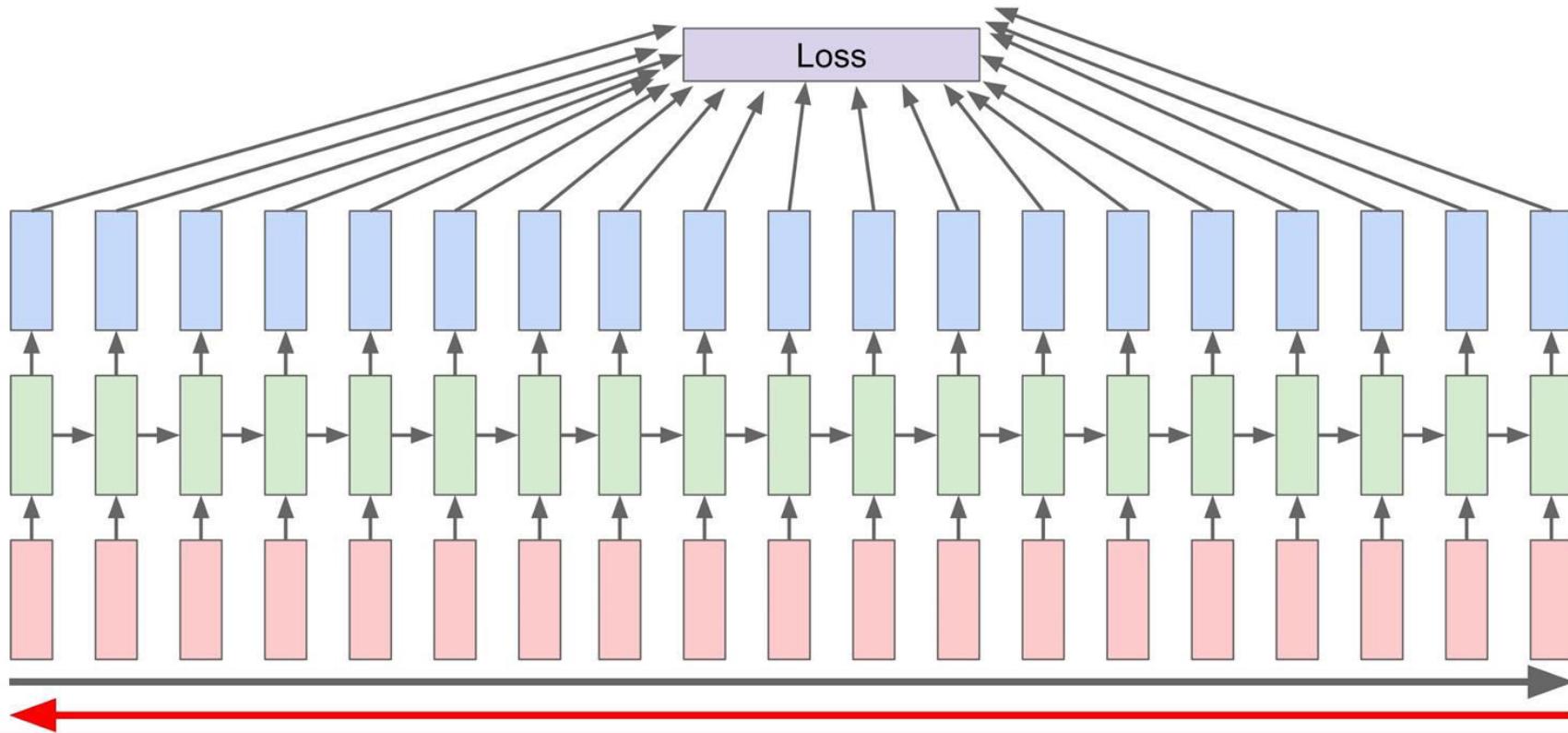


( $x$  from -12 to 12)

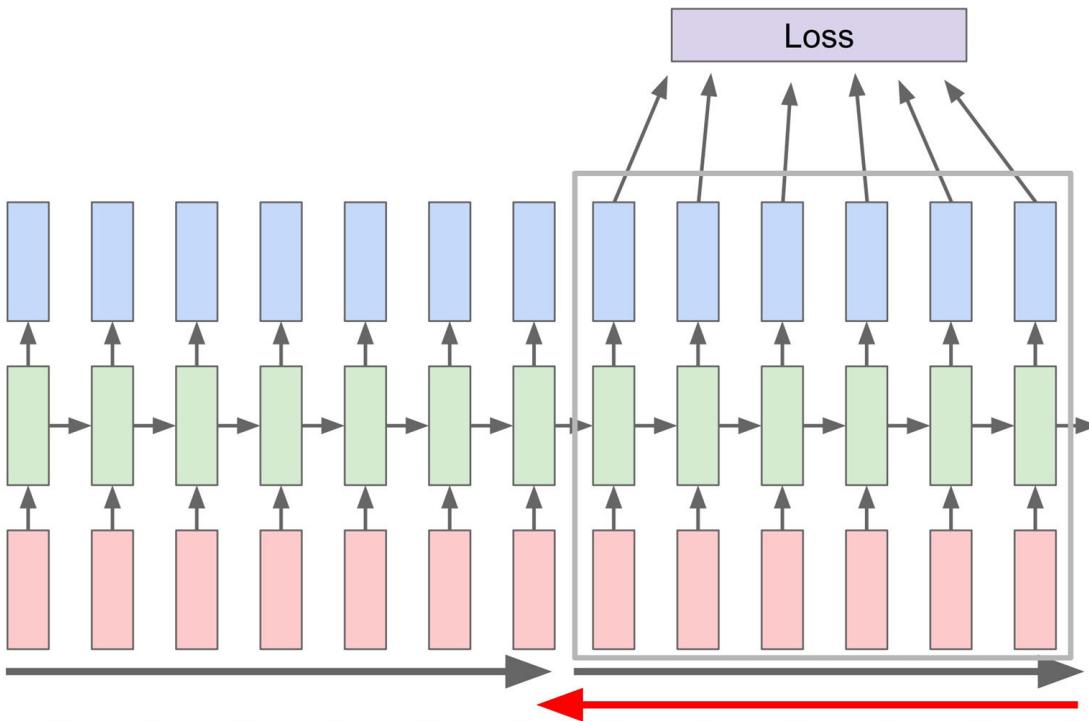
# The Vanishing Gradient Problem



# The Vanishing Gradient Problem

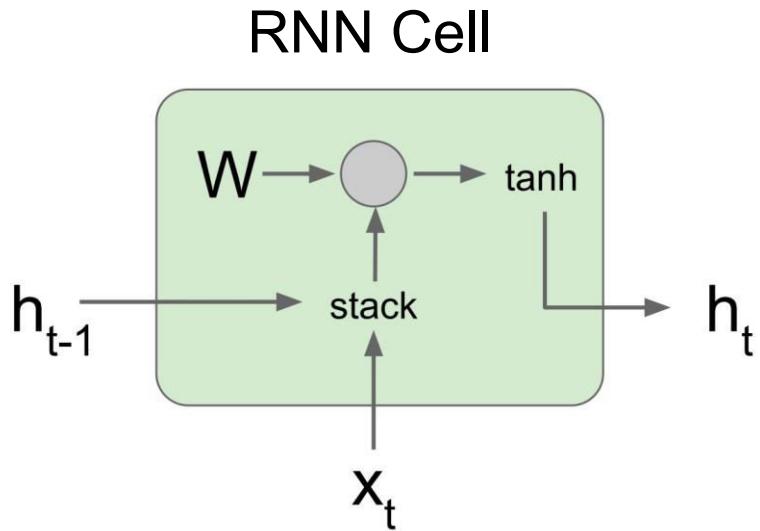


# Truncated Backpropagation



- Run forward and backward through chunks of the sequence instead of the whole sequence
- Carry hidden states forward through chunks
- Truncate backprop after a few steps

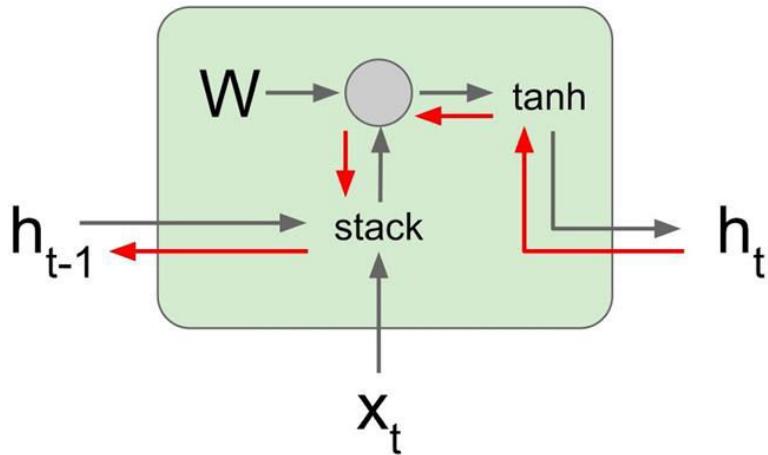
# The Vanishing Gradient Problem



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left( \begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

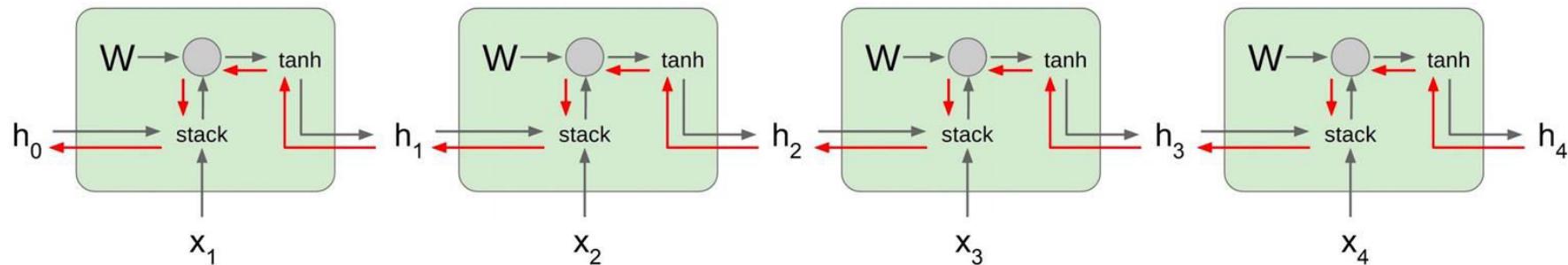
# The Vanishing Gradient Problem

Backpropagation from  $h_t$   
to  $h_{t-1}$  multiplies by  $W$   
(actually  $W_{hh}^T$ )



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left( \begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

# The Vanishing Gradient Problem



Computing gradient of  $h_0$  involves many factors of  $W$  (and repeated tanh)

**Gradient clipping:** scale gradient if its norm is too big  
To not squash downstream gradients, we need a new architecture.

# Long short-term memory

- A dumb name for some complex cell architecture
  - The details of LSTMs have taken a backseat in NLP lately
  - Upshot: preserve more gradient by keeping more of the hidden state around between updates
- Gated Recurrent Units (GRU) are in the same family of attempts to get around this issue with sequence problems
- In many settings, the lower parameter count of LSTMs and GRUs makes them preferable to Transformers
- So we will still learn LSTMs!

# Long short-term memory

- Intuition of LSTMs is to “flip” the default recurrence behavior from “squeeze out the history” to “mostly consider the history”

**Vanilla RNN**

$$h_t = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

**LSTM**

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

# Long short-term memory

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

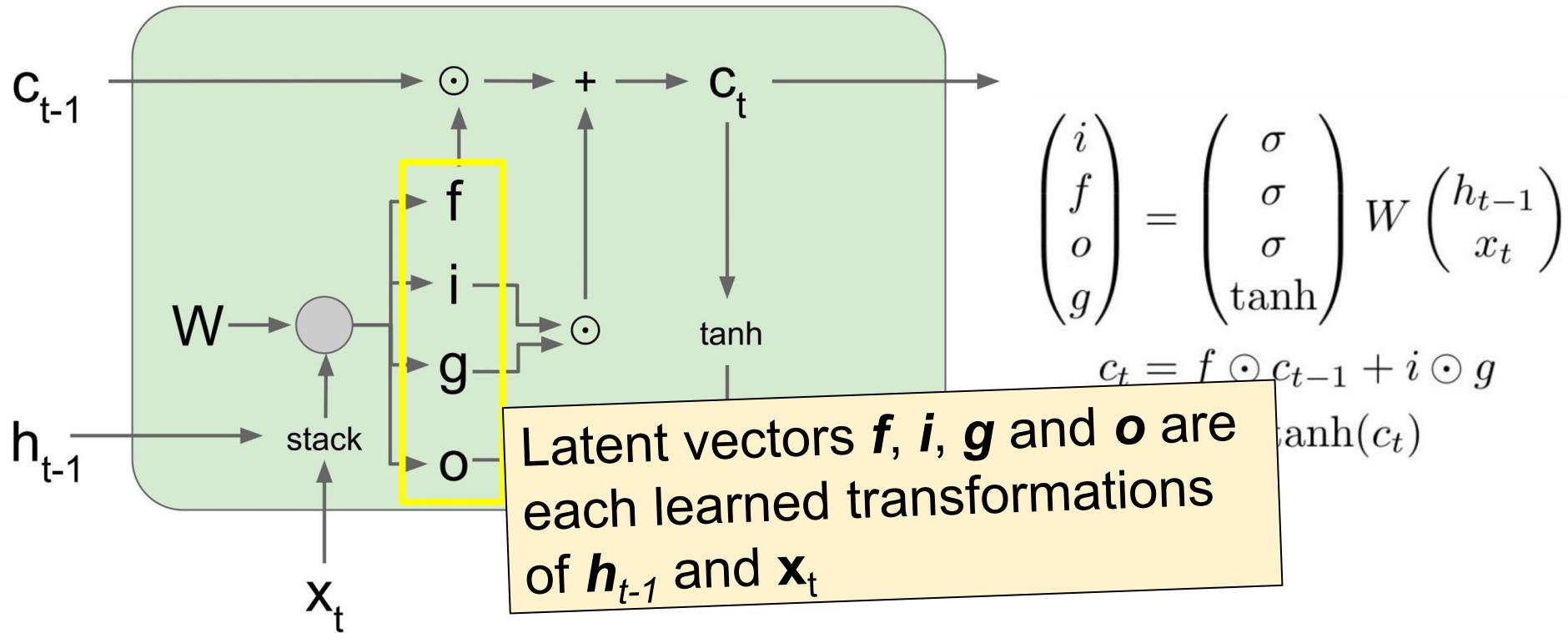
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

$$i = \sigma(W_i \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$
$$f = \sigma(W_f \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$
$$o = \sigma(W_o \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$
$$g = \tanh(W_g \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$
$$tanh(W$$

# Long short-term memory

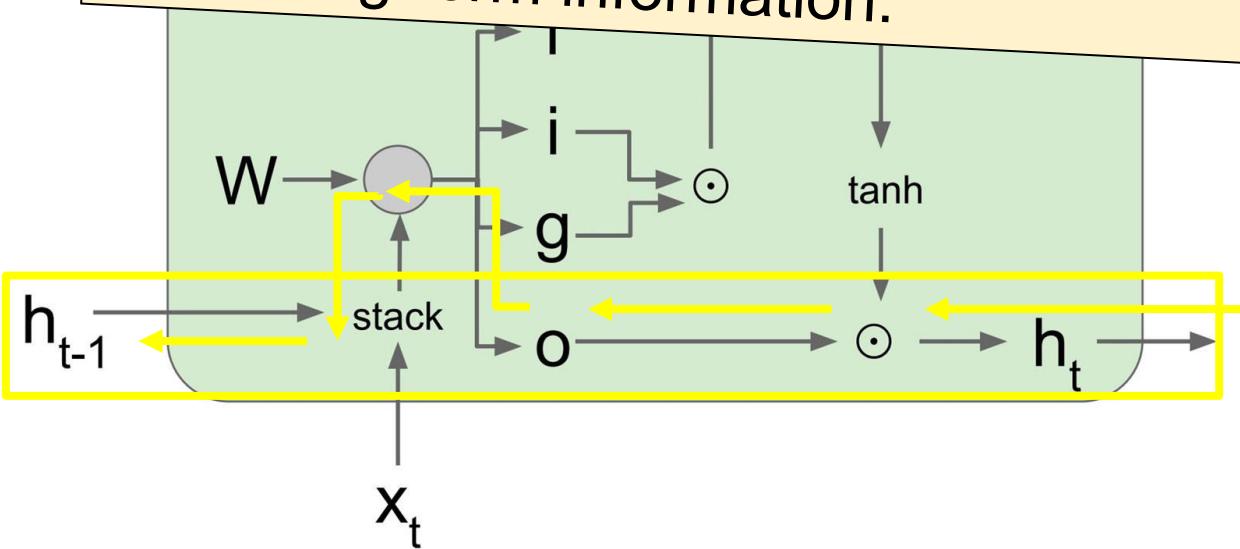
## LSTM Cell



# Long short-term memory

## LSTM Cell

$c_t$  Hidden state  $h_t$  is a function of both  $(x_t, h_{t-1})$  and cell state  $c_t$ . The cell state holds long-term information.



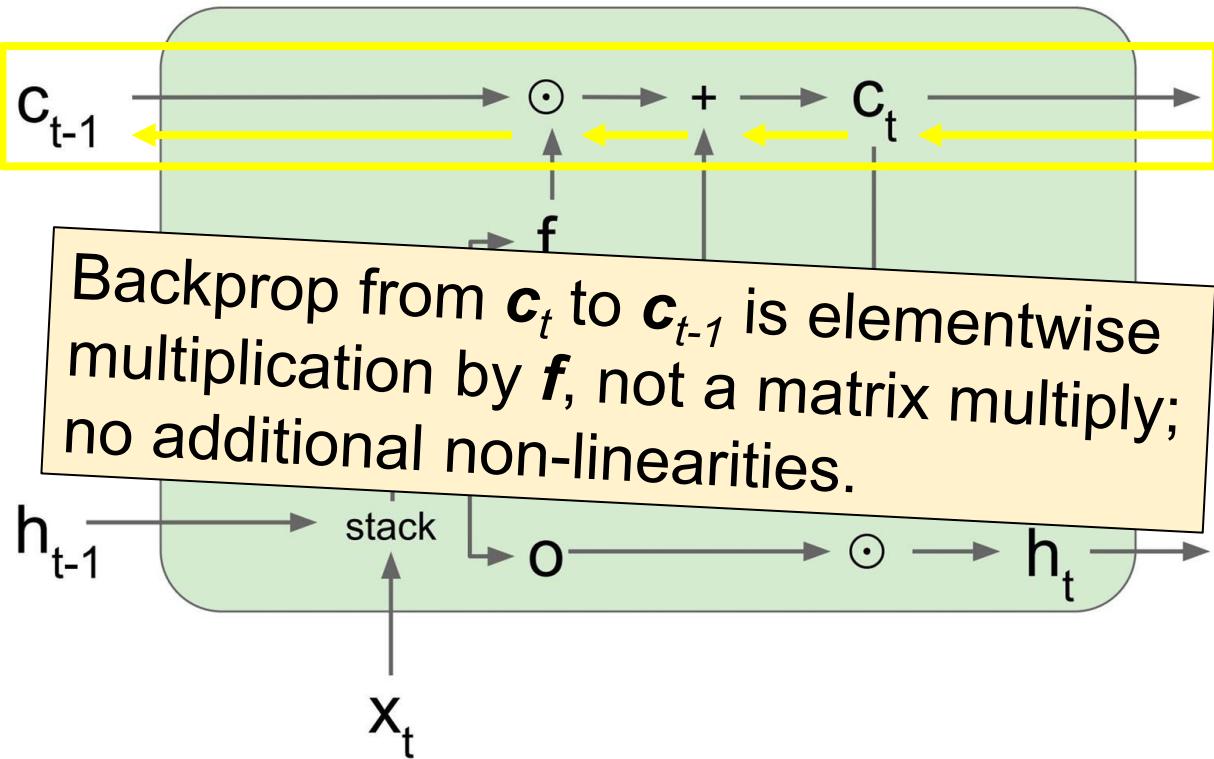
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

# Long short-term memory

## LSTM Cell

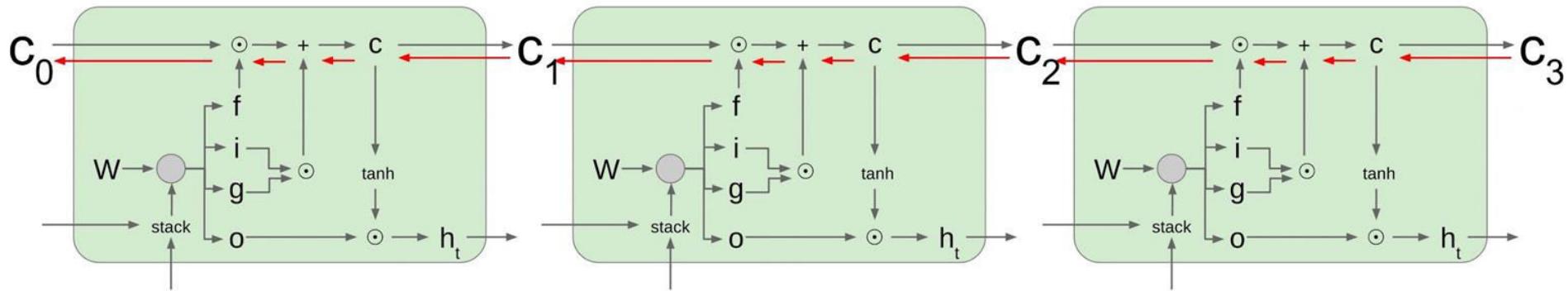


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

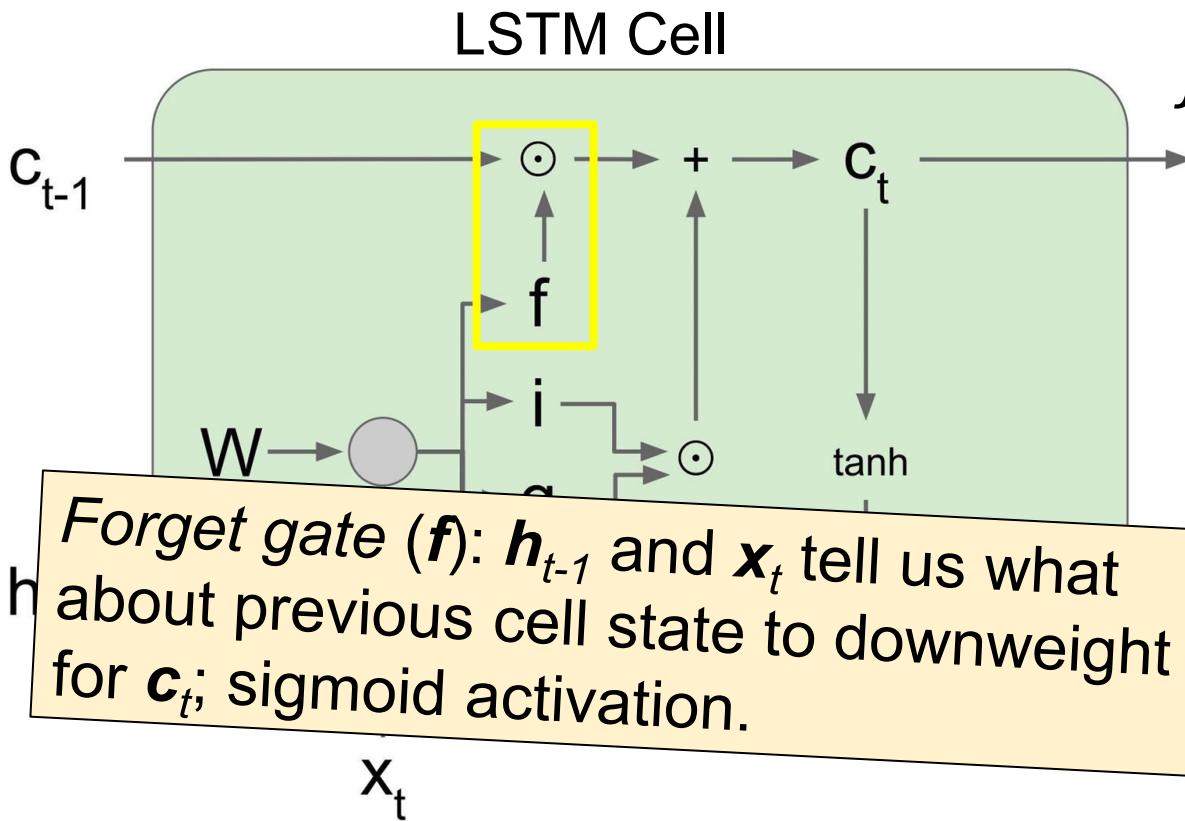
# Long short-term memory



$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

# Long short-term memory



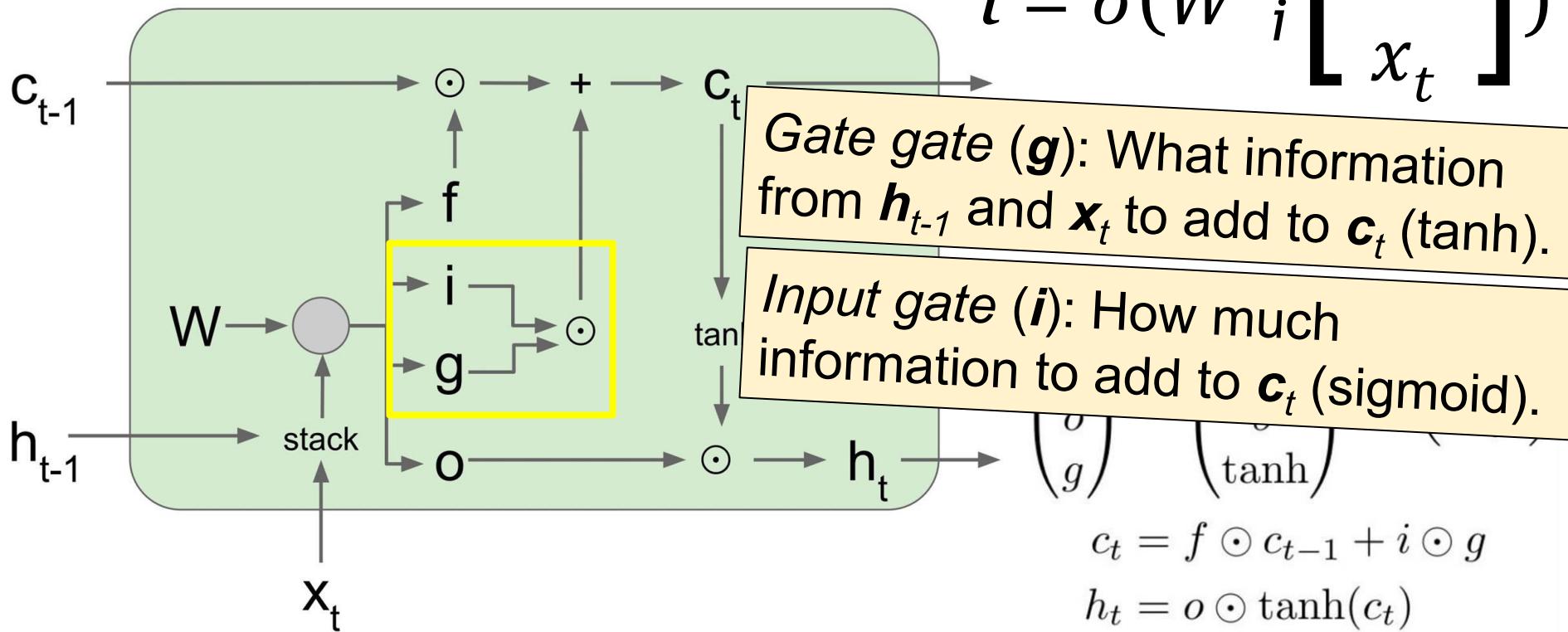
$$f = \sigma(W_f \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix})$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

# Long short-term memory

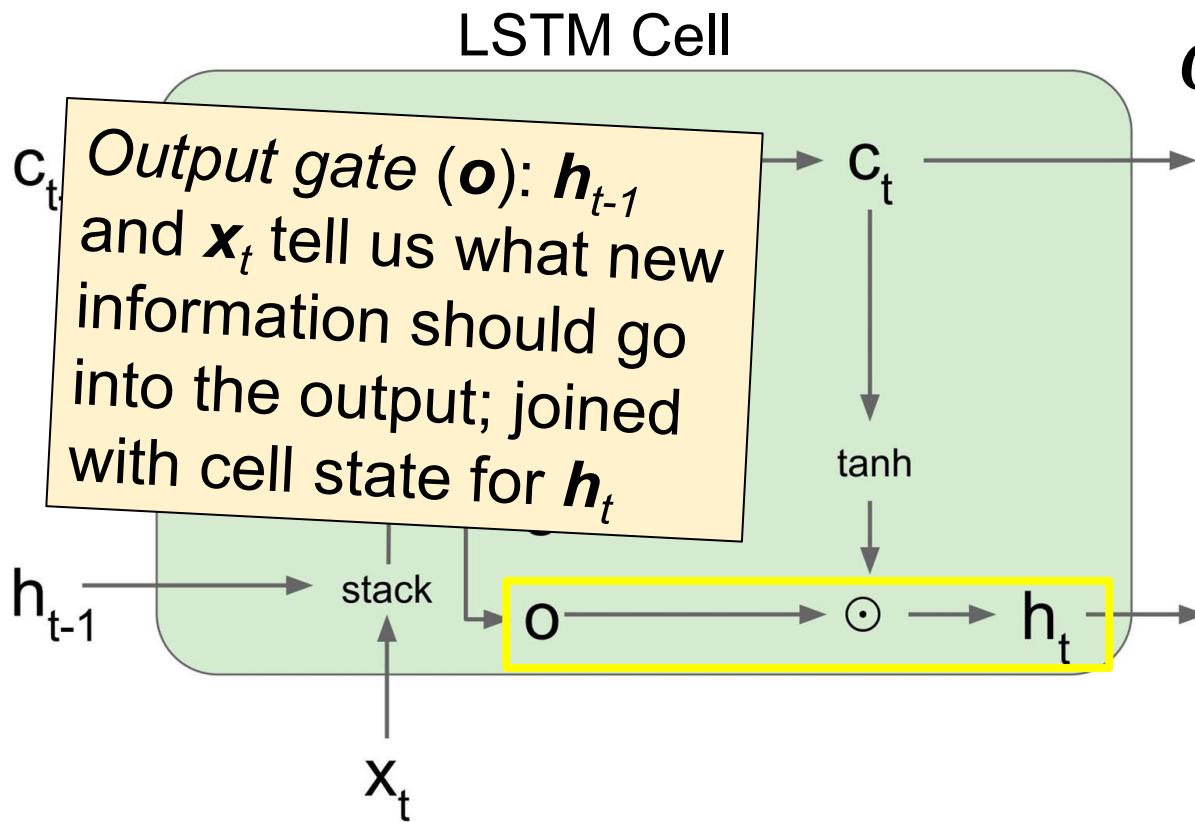
## LSTM Cell



**Gate gate ( $g$ ):** What information from  $h_{t-1}$  and  $x_t$  to add to  $c_t$  ( $\tanh$ ).

**Input gate ( $i$ ):** How much information to add to  $c_t$  ( $\text{sigmoid}$ ).

# Long short-term memory



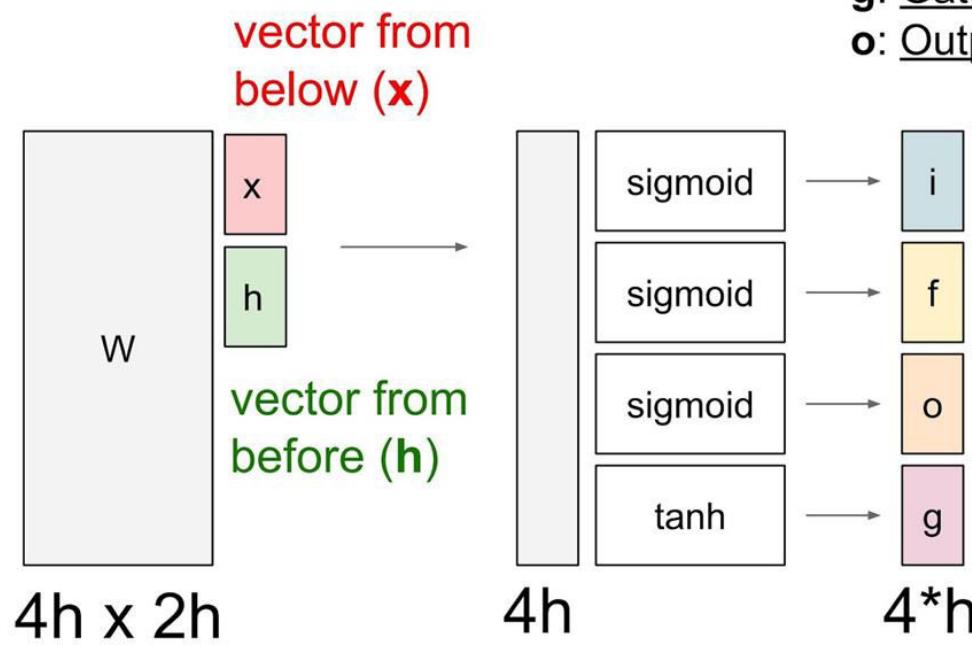
$$o = \sigma(W_o [h_{t-1} \ x_t])$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

# Long short-term memory

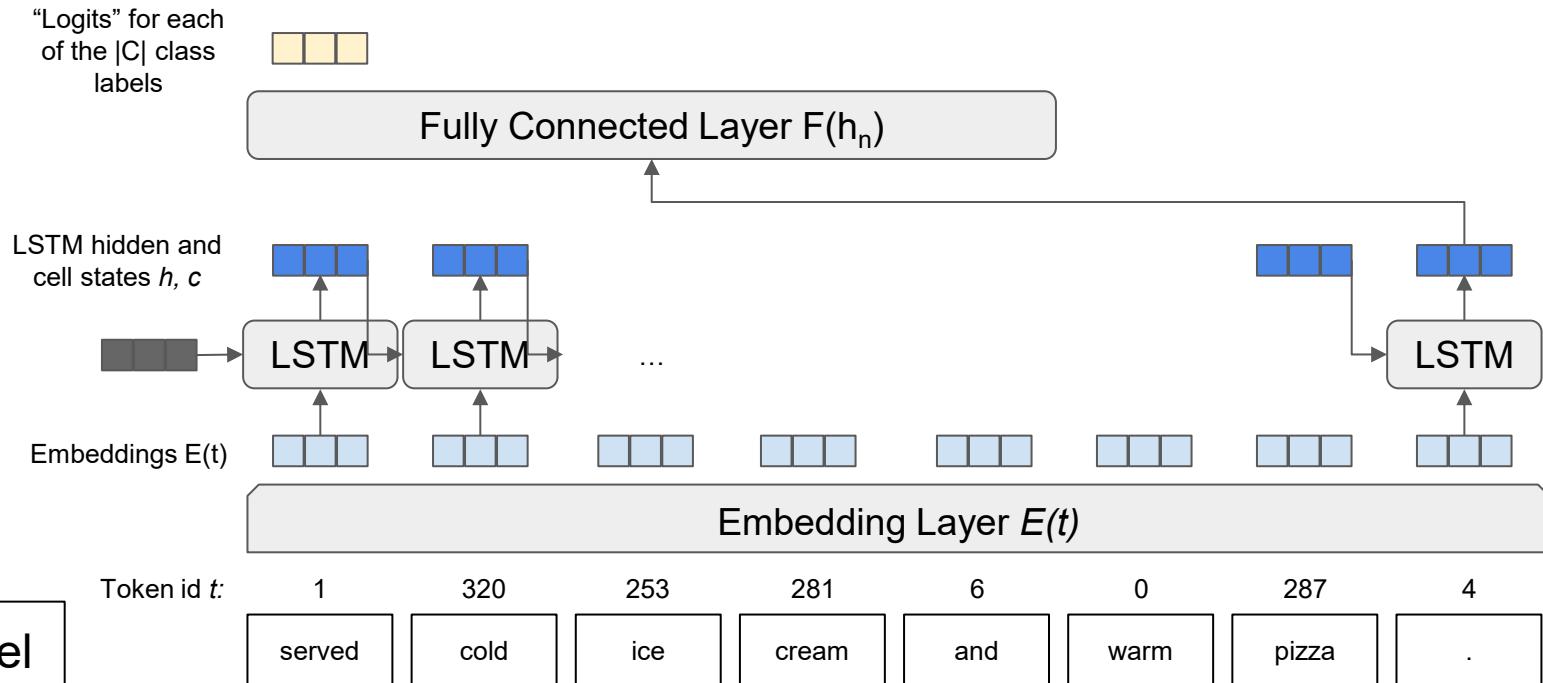


- f: Forget gate, Whether to erase cell
- i: Input gate, whether to write to cell
- g: Gate gate (?), How much to write to cell
- o: Output gate, How much to reveal cell

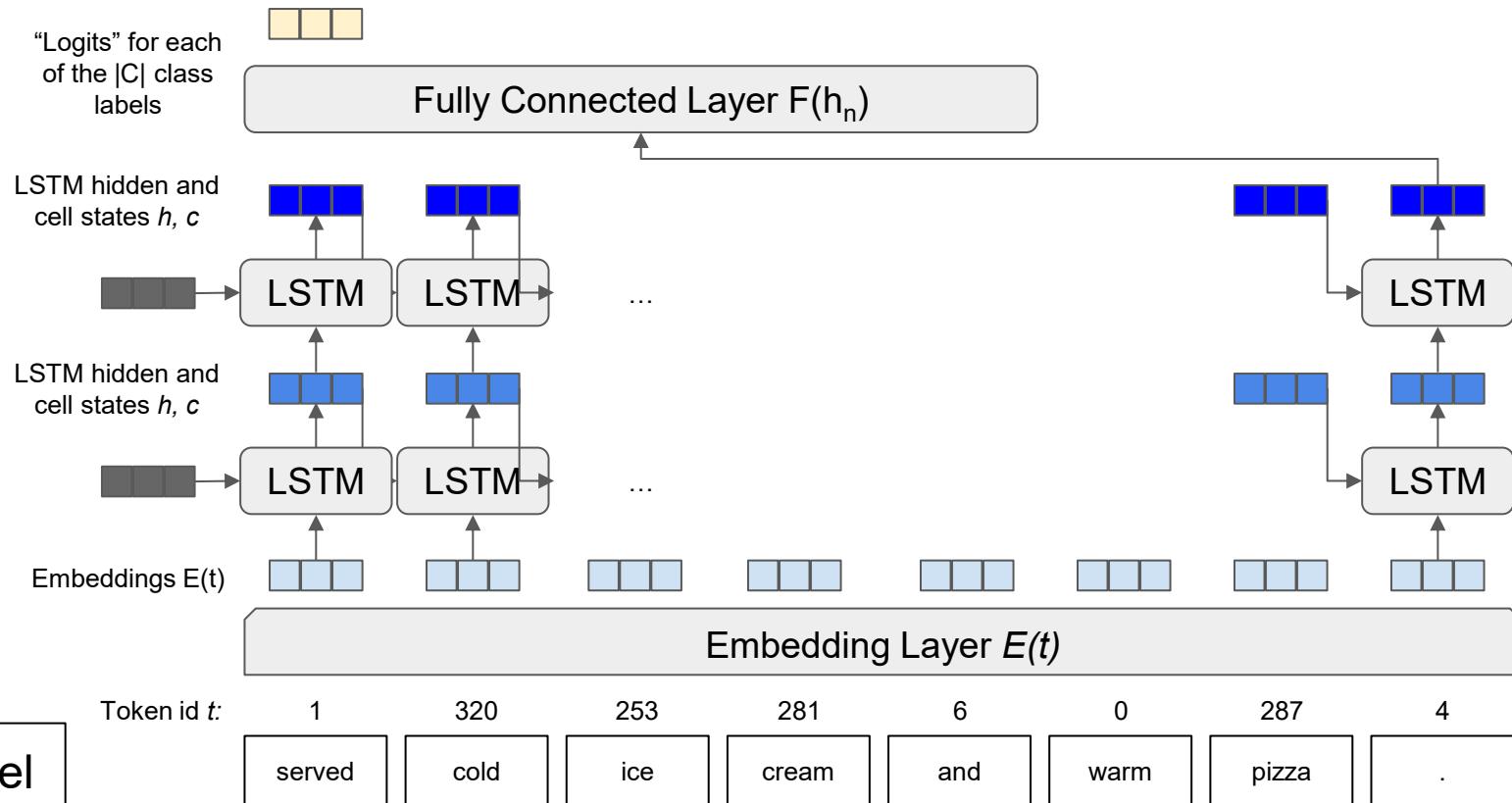
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

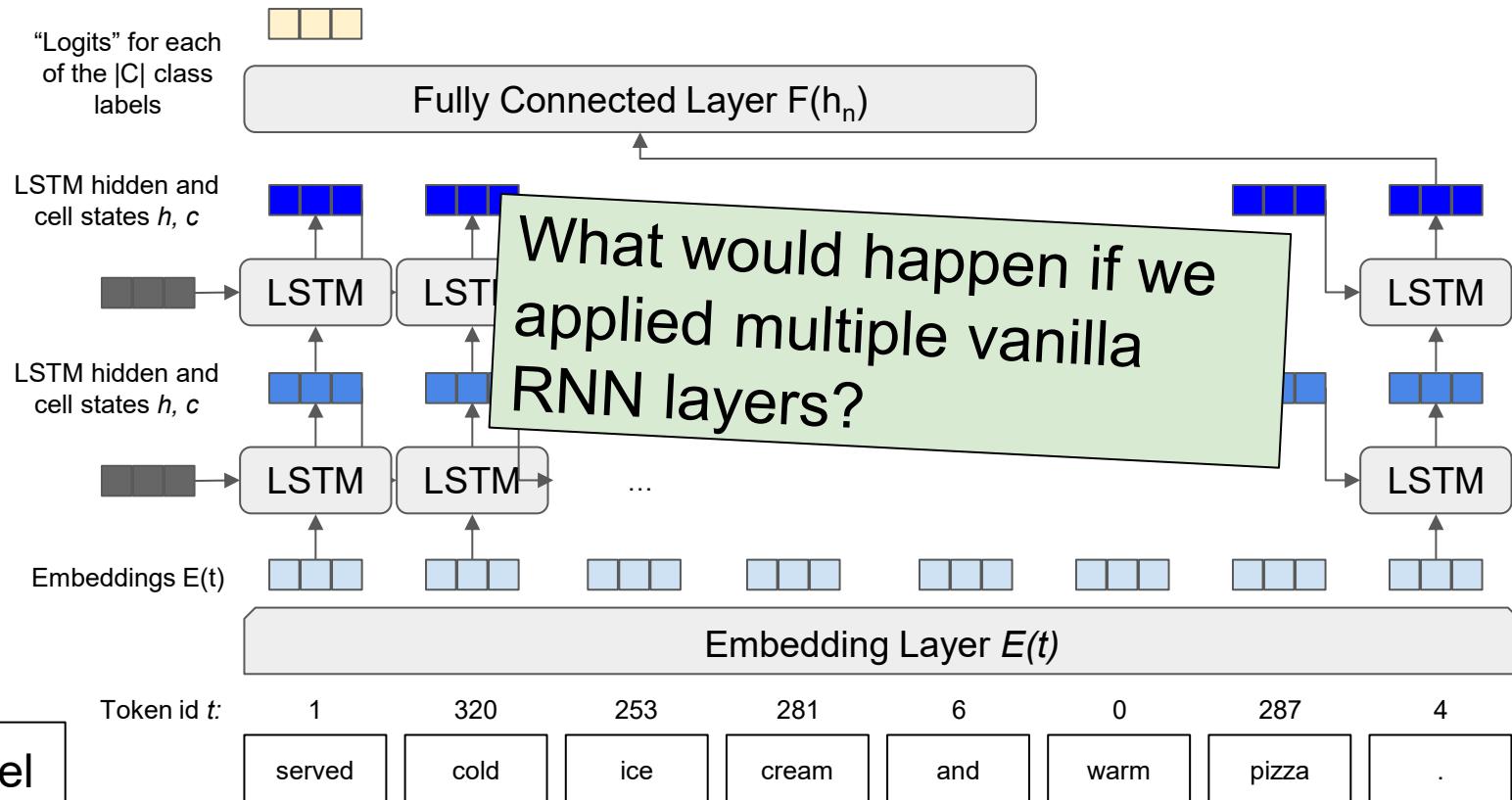
# Text Classification with an LSTM



# Text Classification with a Multi-Layer LSTM



# Text Classification with a Multi-Layer LSTM



# Addressing the Vanishing Gradient Problem

- What do you need to know?
  - **LSTMs > vanilla RNNs basically all the time**
  - Multi-layer LSTM makes sense; multi-layer vanilla RNN will have such bad gradient vanishing it probably won't train
  - LSTMs  $\approx$  GRUs; think of this choice as a hyperparameter you can tune on your validation data
  - Using an RNN/LSTM/GRU is *different* from using convolutional layers to process text (e.g., short segments combined with some kind of pooling layer)

# Recurrent Neural Network Applications

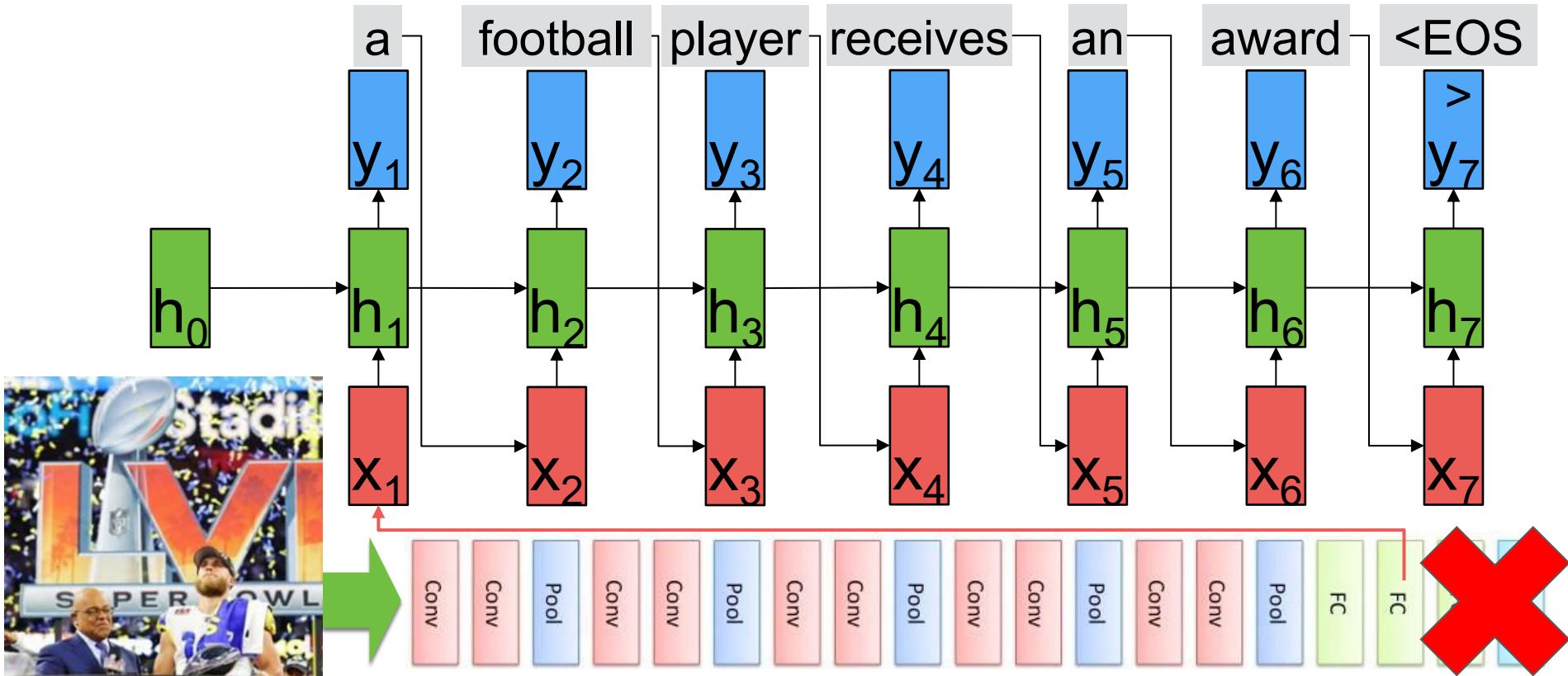
- Image Captioning
- Question Answering
- Visual Question Answering
- Speech Recognition
- Action Recognition in Videos
- Text Parsing
- Machine Translation
- Genomic Sequence Classification

# Image Captioning with Sequential Decoder



A football player receives an award.

# Image Captioning with Sequential Decoder



# Question Answering with Encoder-Decoder

Text

Mary moved to the bathroom.

John went to the hallway.

Question

Where is Mary?

Answer

bathroom

# Question Answering with Encoder-Decoder

Text

Mary moved to the bathroom.  
John went to the hallway.



Text embedding

Question

Where is Mary?



Question embedding

# Question Answering with Encoder-Decoder

Text

Mary moved to the bathroom.  
John went to the hallway.

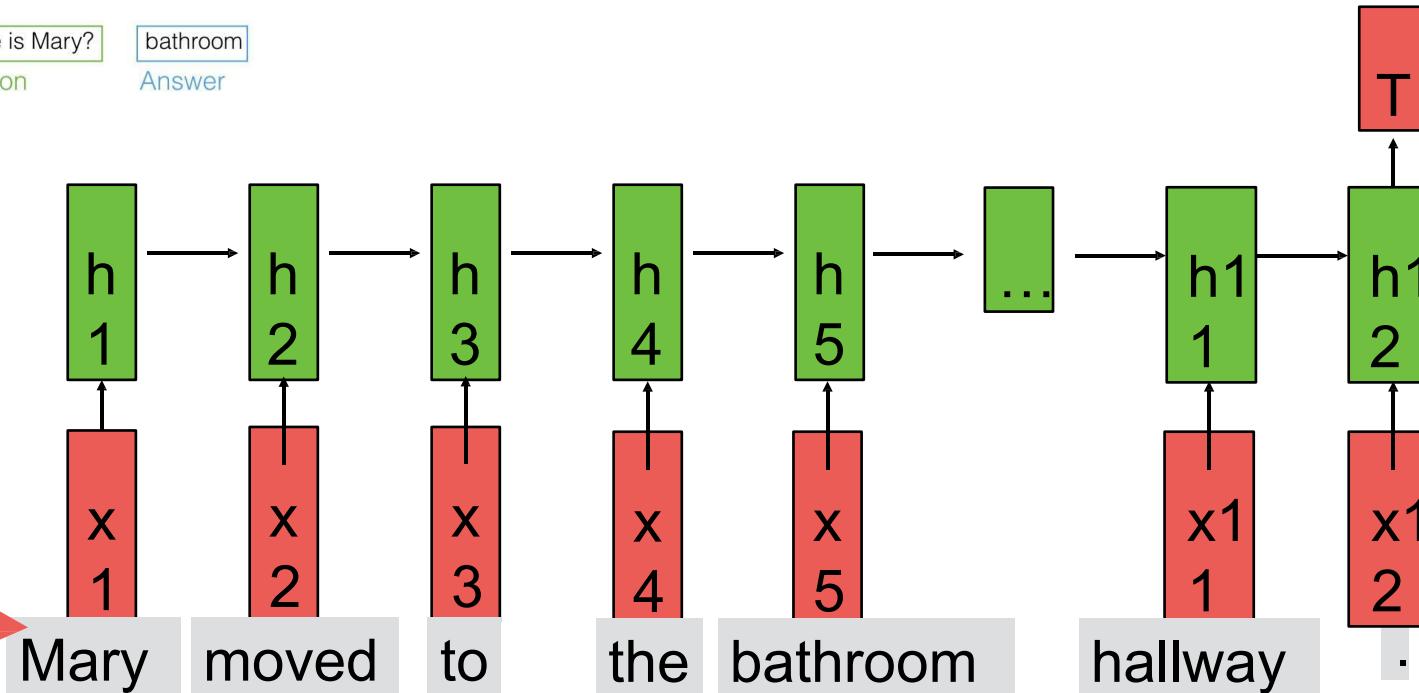
- Where is Mary?

bathroom

Question

Answer

Text embedding



# Question Answering with Encoder-Decoder

Text

Mary moved to the bathroom.  
John went to the hallway.

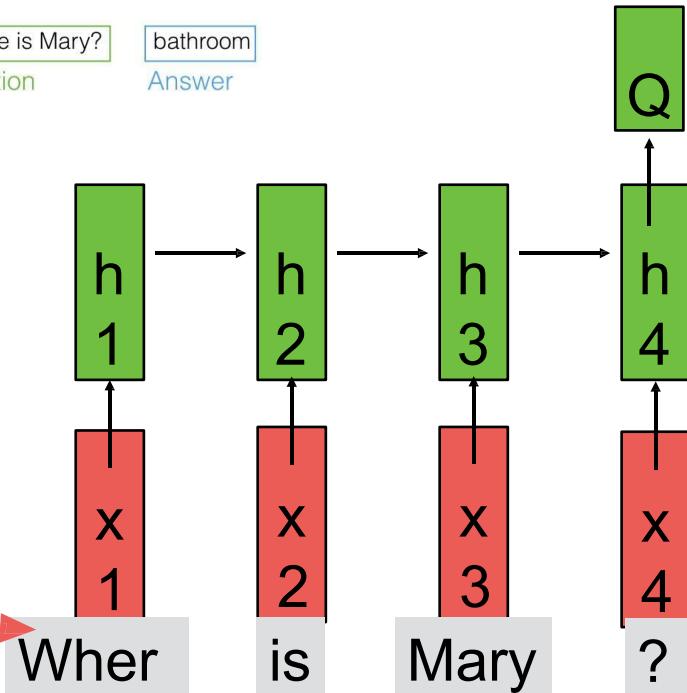
- Where is Mary?

Question

bathroom

Answer

Question embedding



# Question Answering with Encoder-Decoder

Text

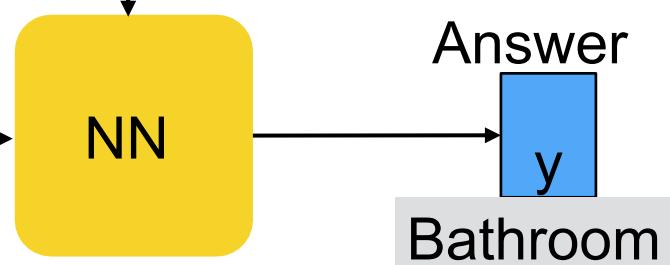
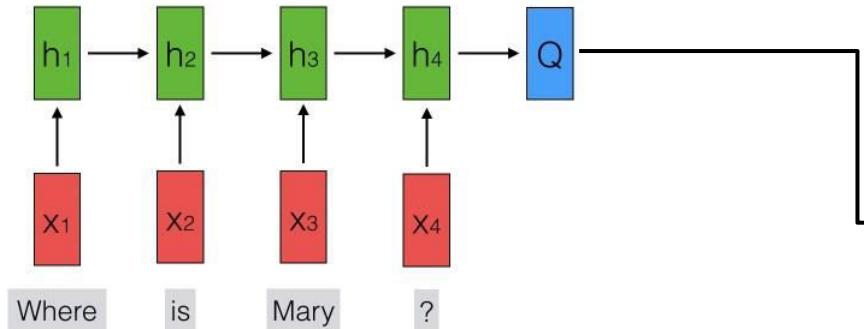
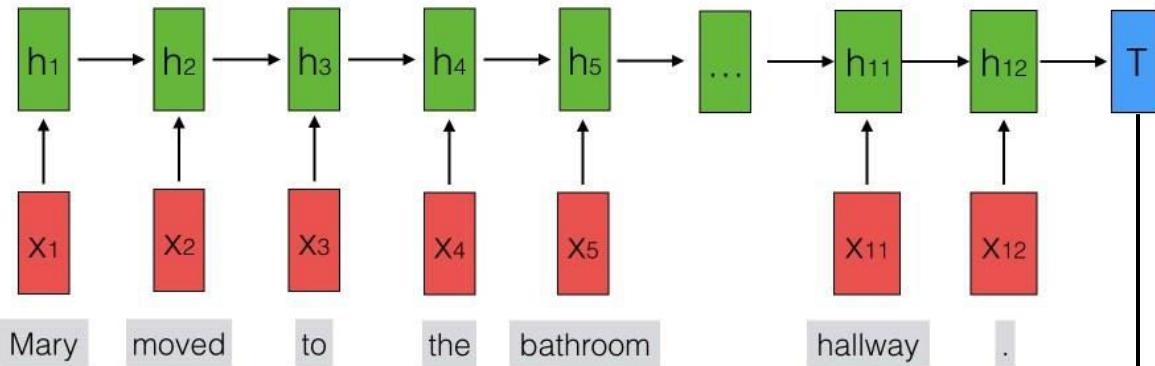
Mary moved to the bathroom.  
John went to the hallway.

- Where is Mary?

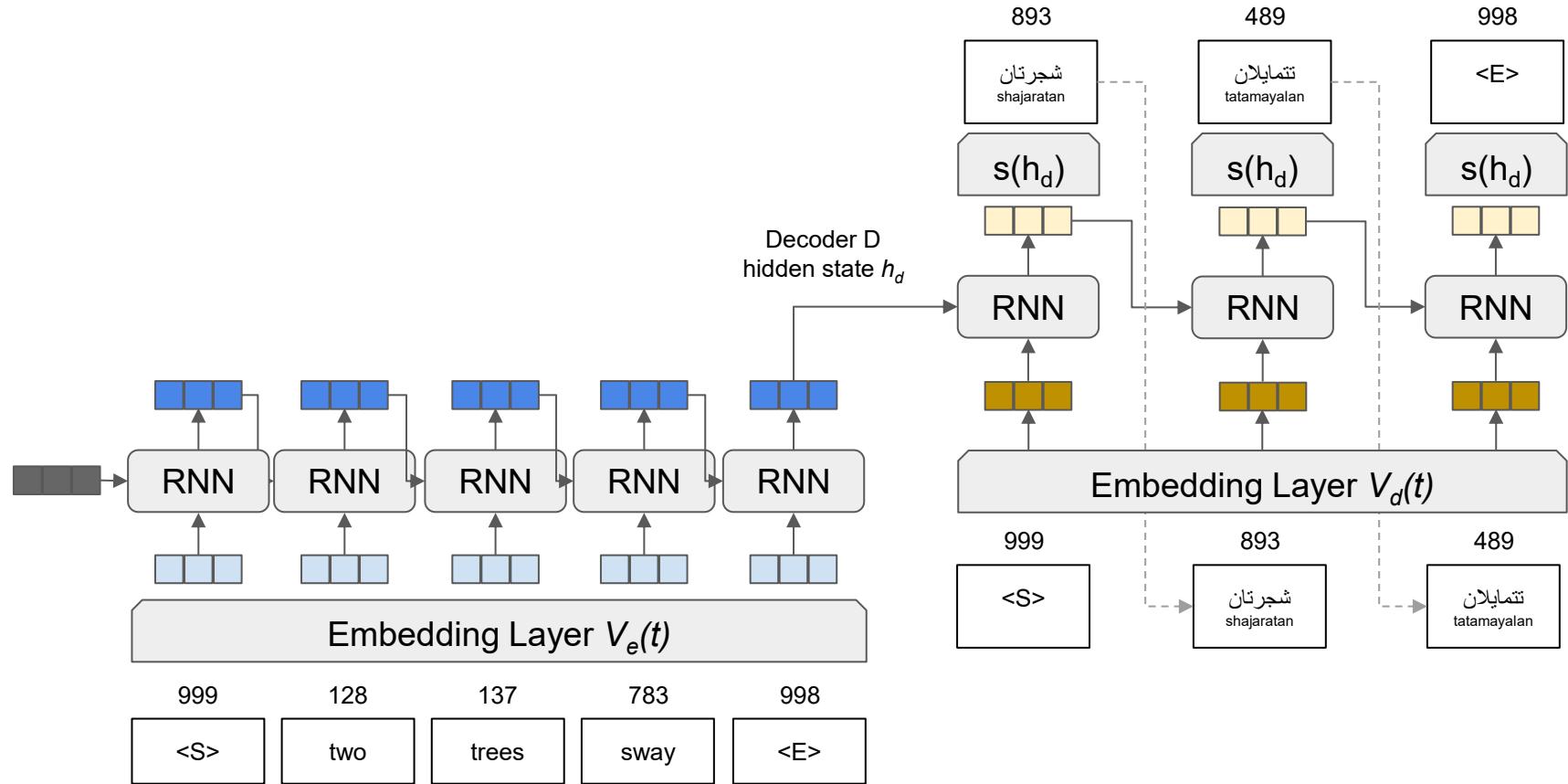
bathroom

Answer

Question



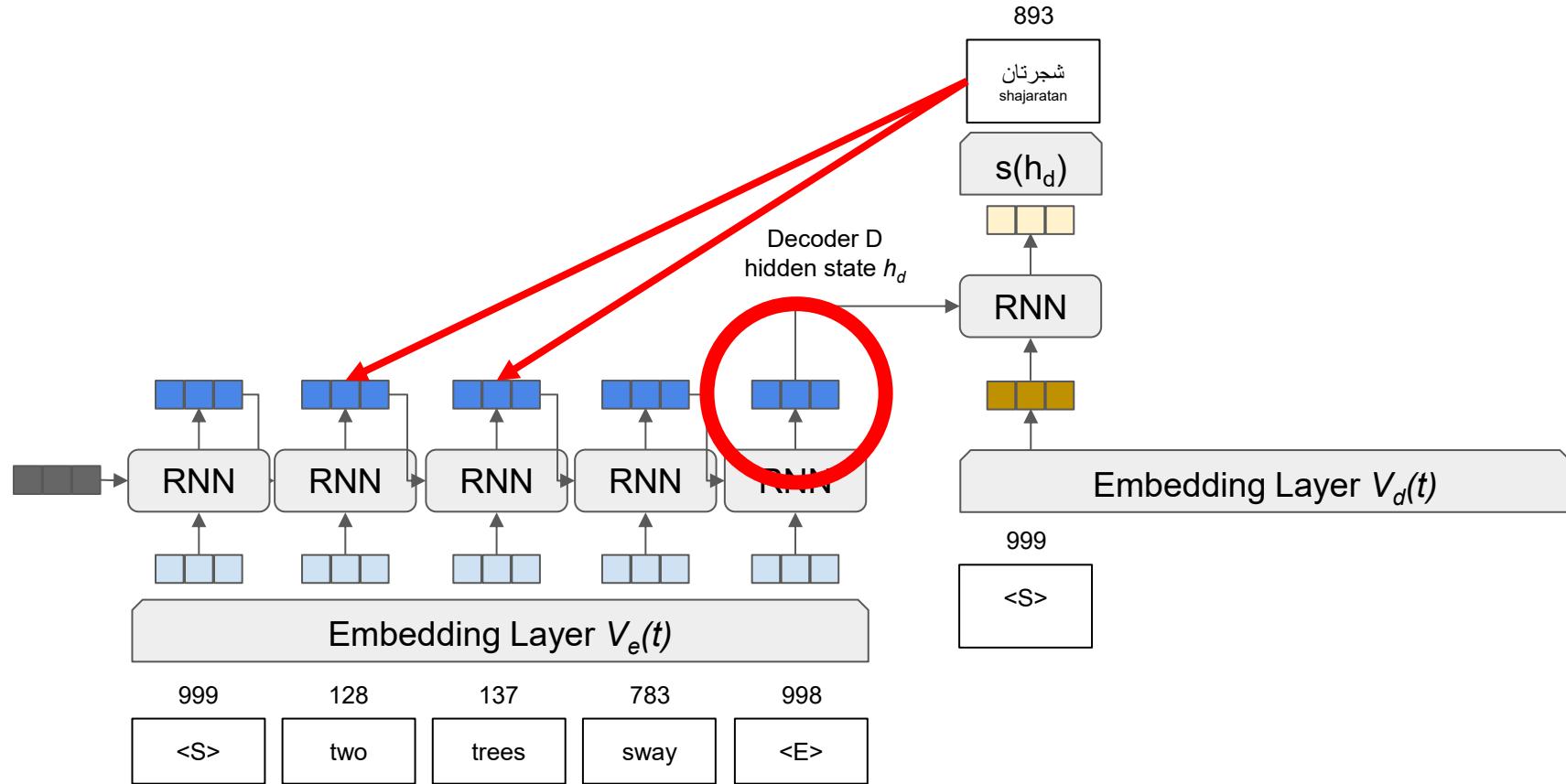
# Machine Translation and The Case for Contextual Info



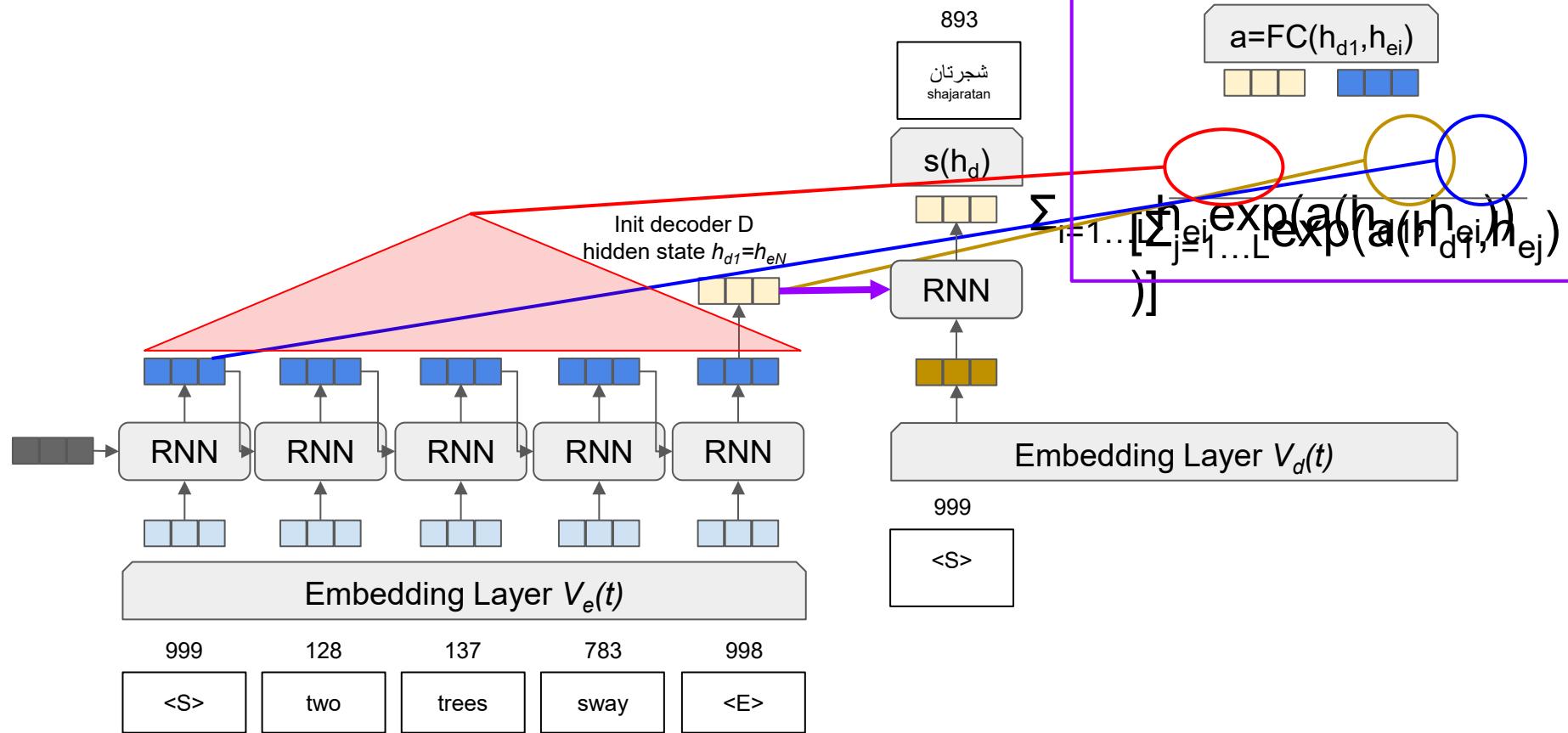
# The Case for Contextual Information

- English to Arabic translation
- Arabic has a few things English doesn't have, including the *dual* case, a noun- and verb- form for *pairs*
  - A tree sways
    - شجرة تتأرجح [shajarat tata'arjah]
  - Two trees sway
    - شجرتان تتمايلان [shajaratan tatamayalan]
  - The trees sway
    - الأشجار تتأرجح [al'ashjar tata'arjah]

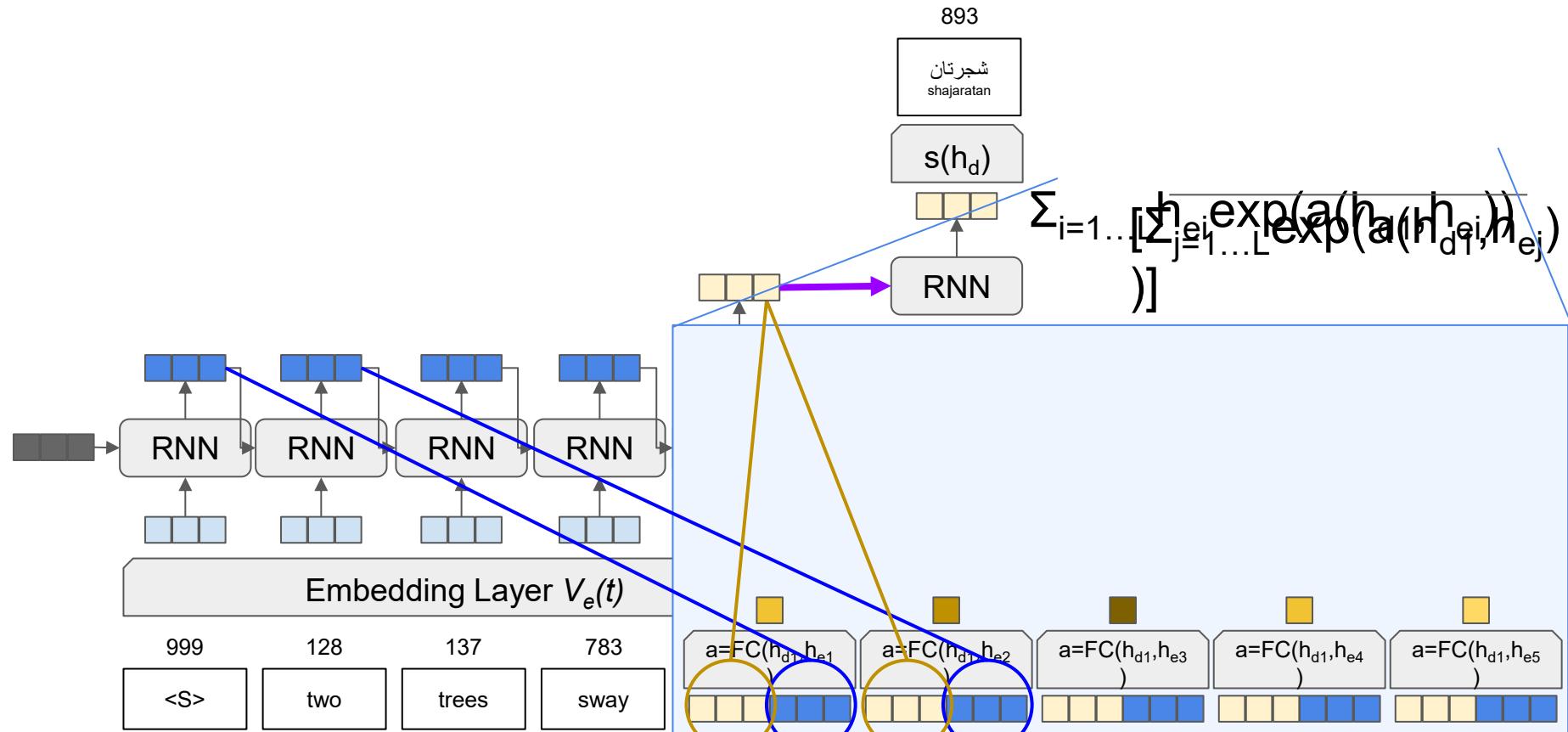
# Machine Translation and The Case for Contextual Info



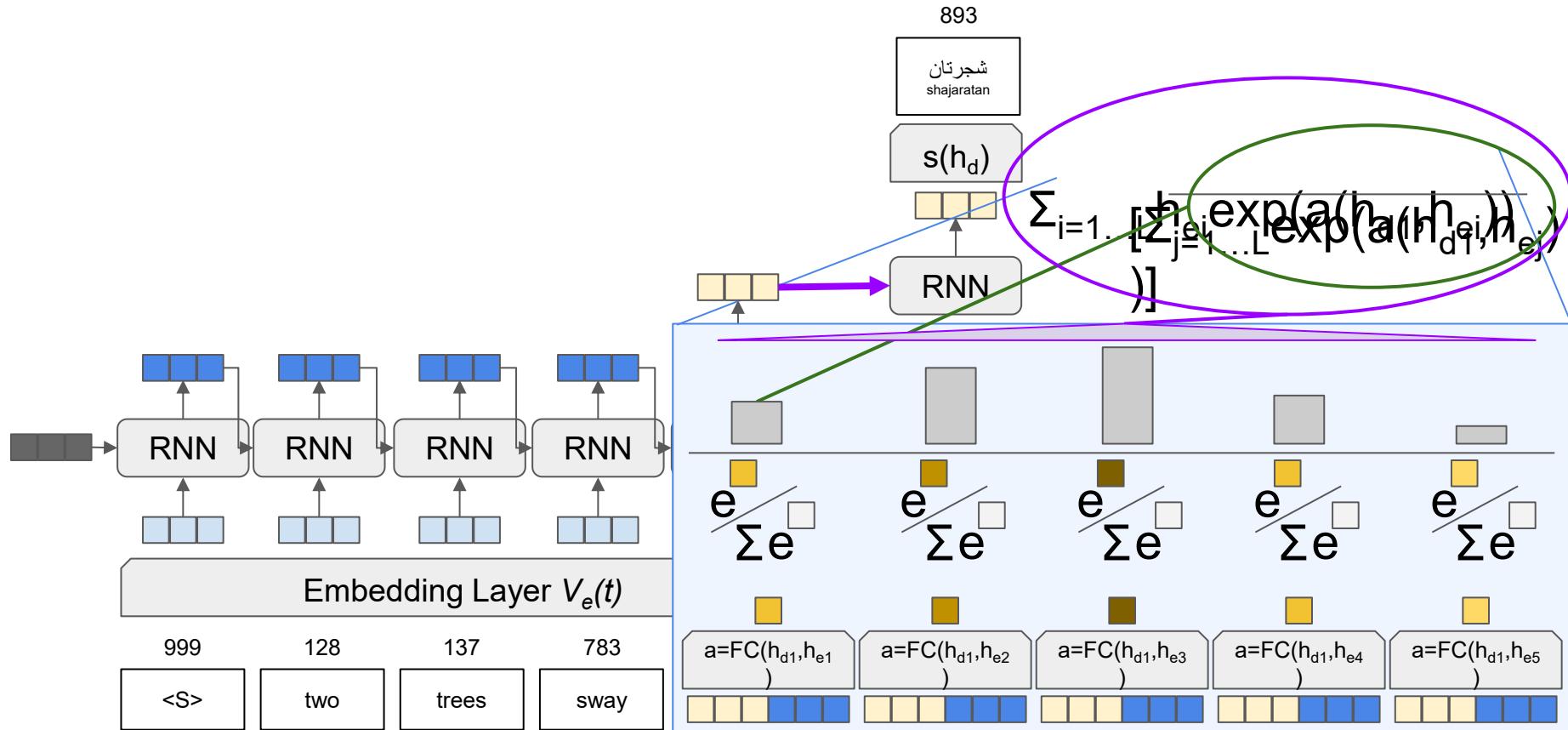
# Use Attention to Rewrite RNN State Input



# Attention



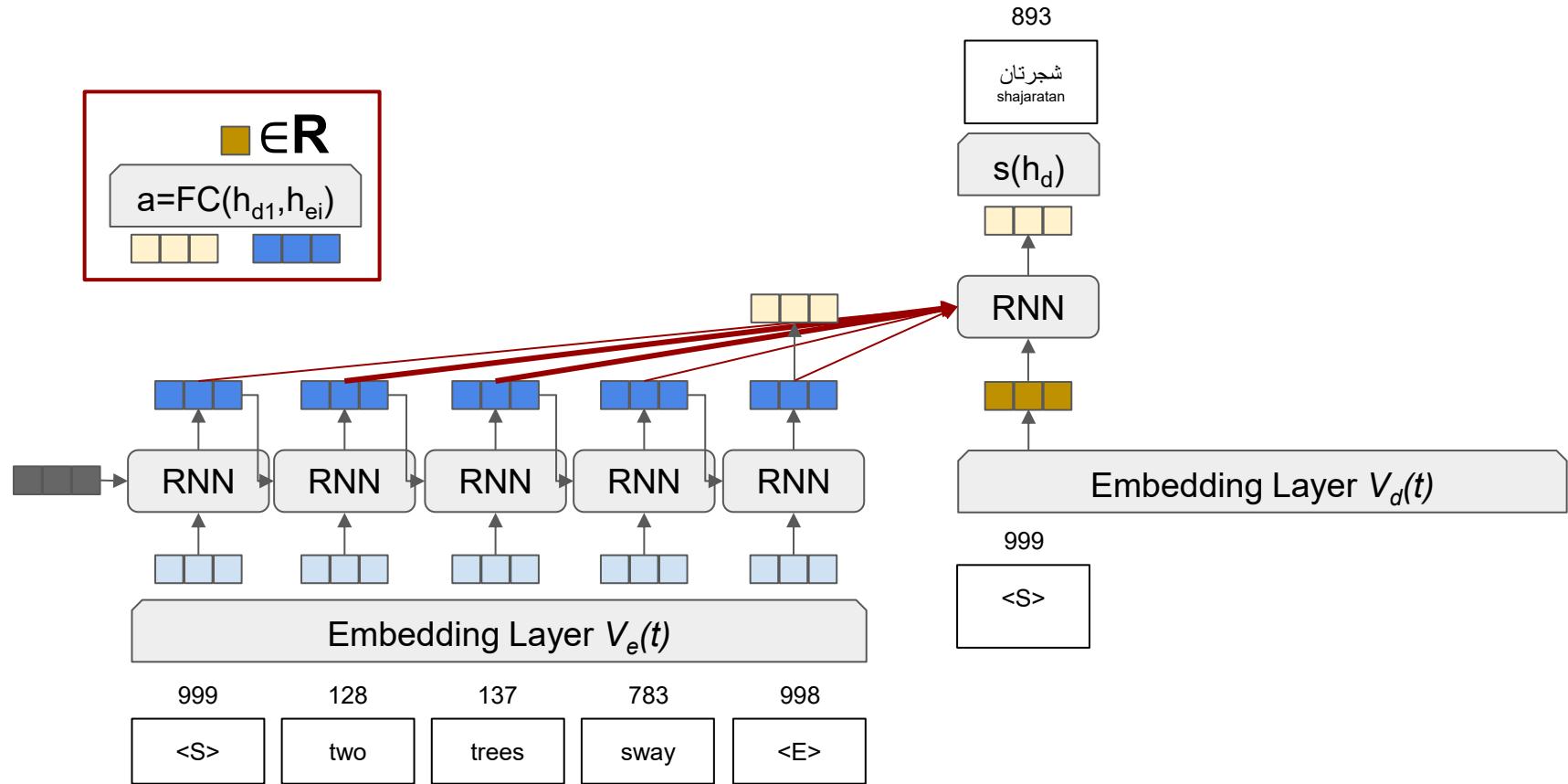
# Attention



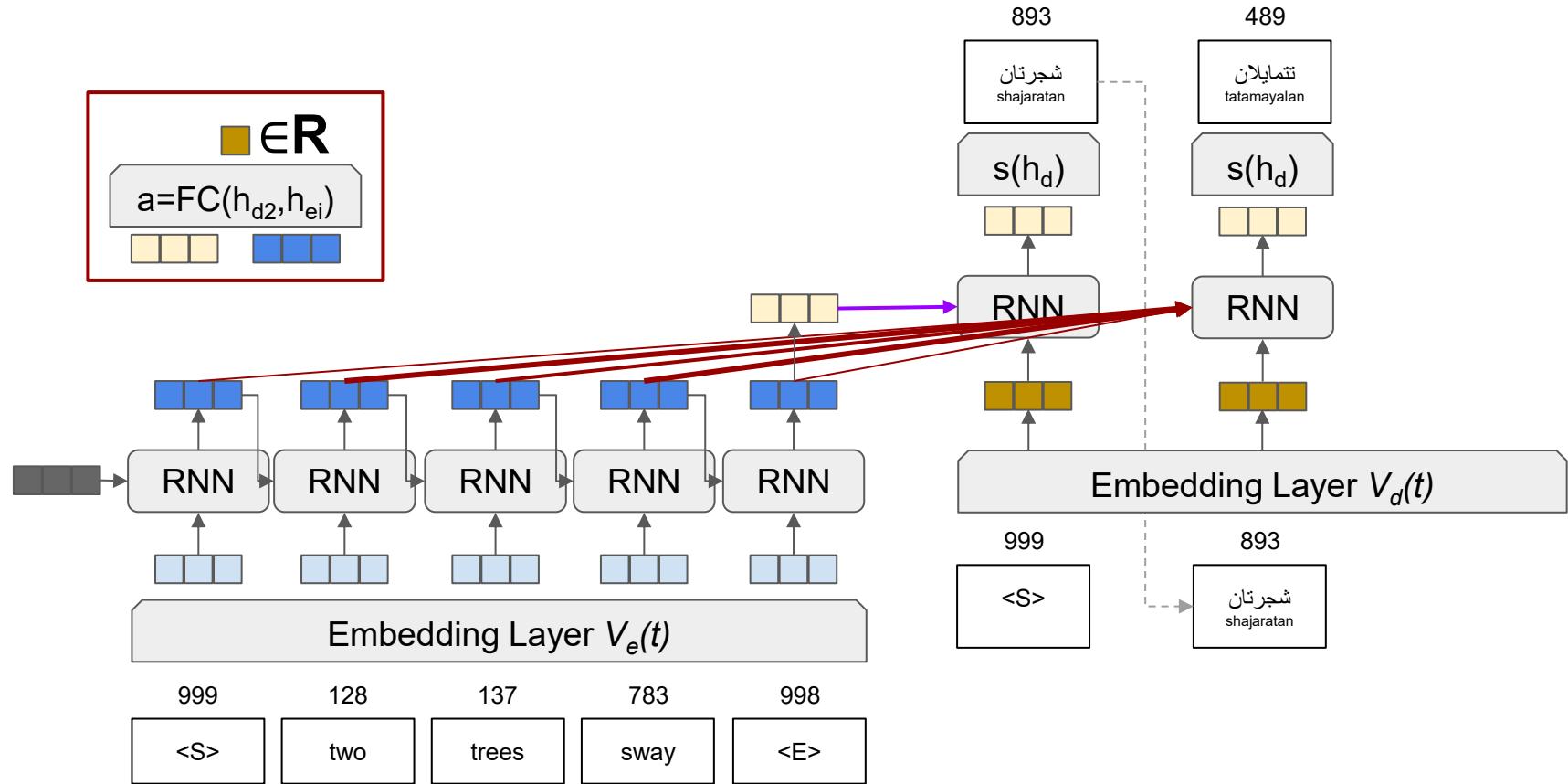
# Encoder-Decoder Attention

- The final hidden state of an encoder may not carry enough information about previous tokens to perform decoding
- *Attention* reweights encoder states  $h_{e[1:L]}$  as a function of the decoder hidden state  $h_{d[t-1]}$  at each timestep
  - $$\mathbf{h}_{d[t]} = [\sum_{i=1 \dots L} \mathbf{h}_{e[i]} \exp(a(\mathbf{h}_{d[t-1]}, \mathbf{h}_{e[i]}))] / [\sum_{j=1 \dots L} \exp(a(\mathbf{h}_{d[t-1]}, \mathbf{h}_{e[j]}))]$$
- A learned attention *head* determines the matching score between the decoder's last output state and each candidate encoder hidden state
- An attention *head* has shared weights for all encoder states, so it's cheap to add, parameter-wise

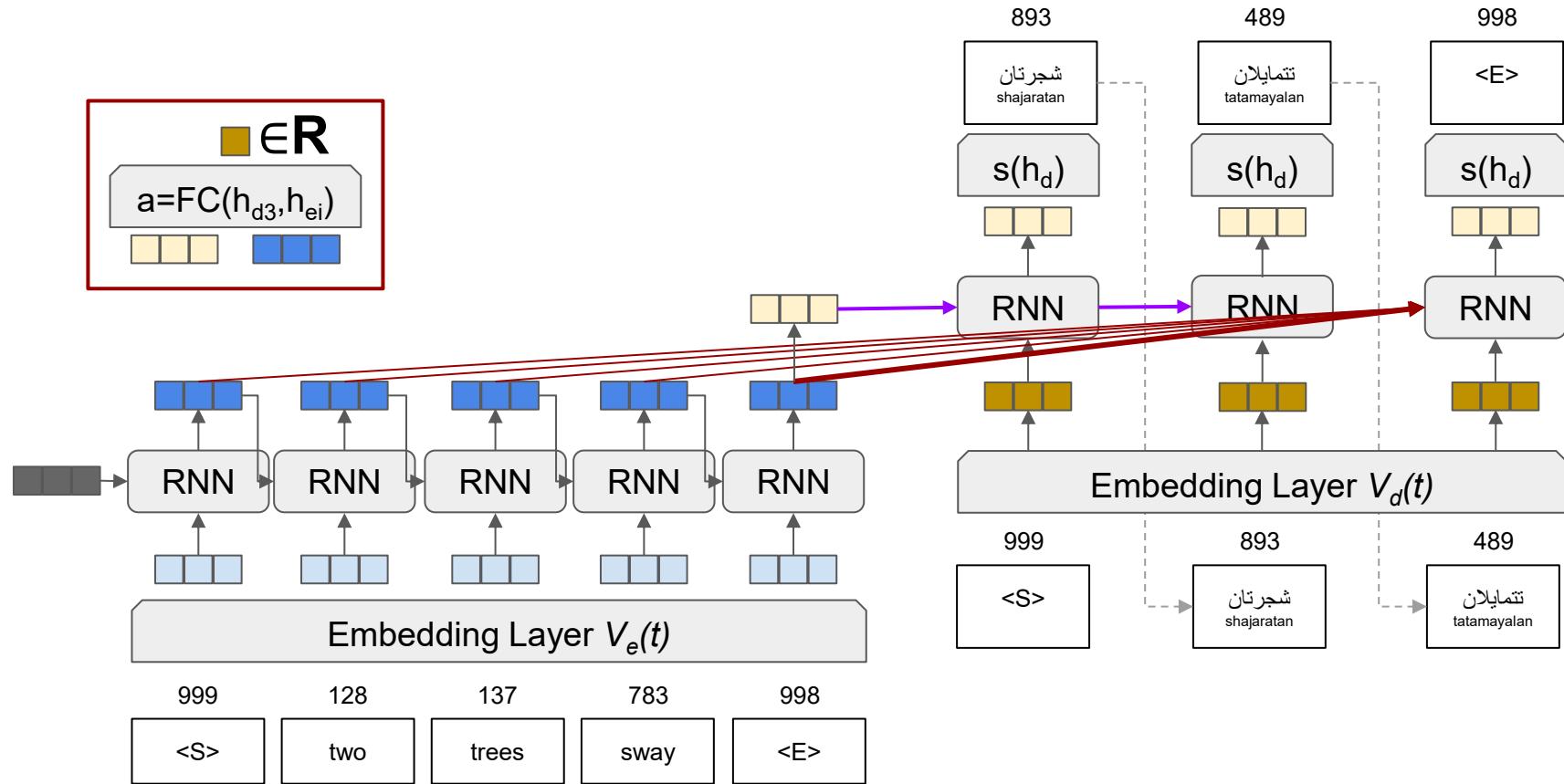
# Encoder-Decoder Attention



# Encoder-Decoder Attention



# Encoder-Decoder Attention



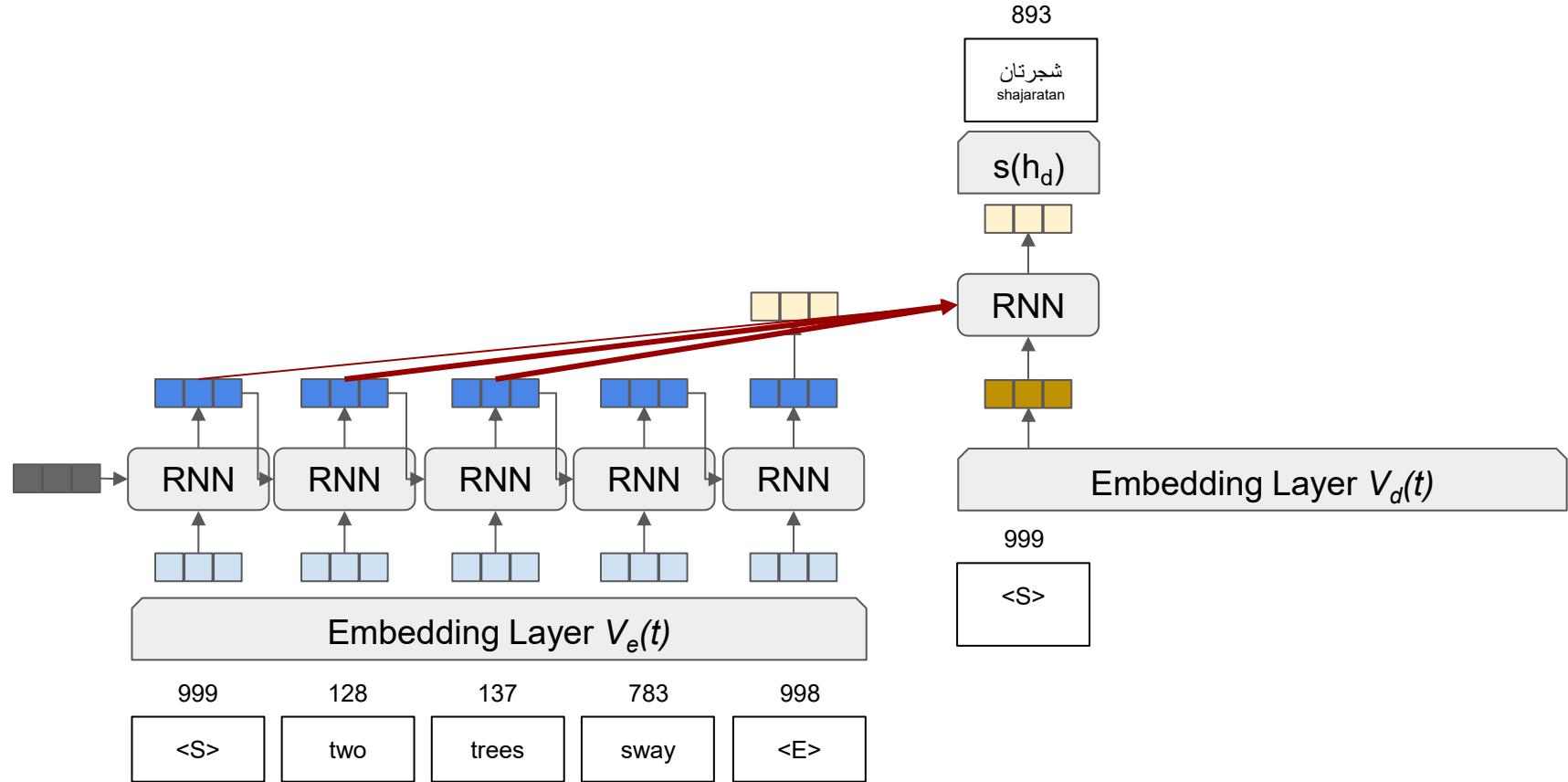
# Encoder-Decoder Attention

- What are some pitfalls or weaknesses we can anticipate with the attention mechanism described so far?
- *Pitfalls:*
  - We probably still don't need the entire sequence for every decoding step
  - A lot of our weights will be near zero, probably
  - The shared parameters of the attention head might only learn to pick out certain dependencies (e.g., noun-verb agreement, determiner-gender agreement, ...)

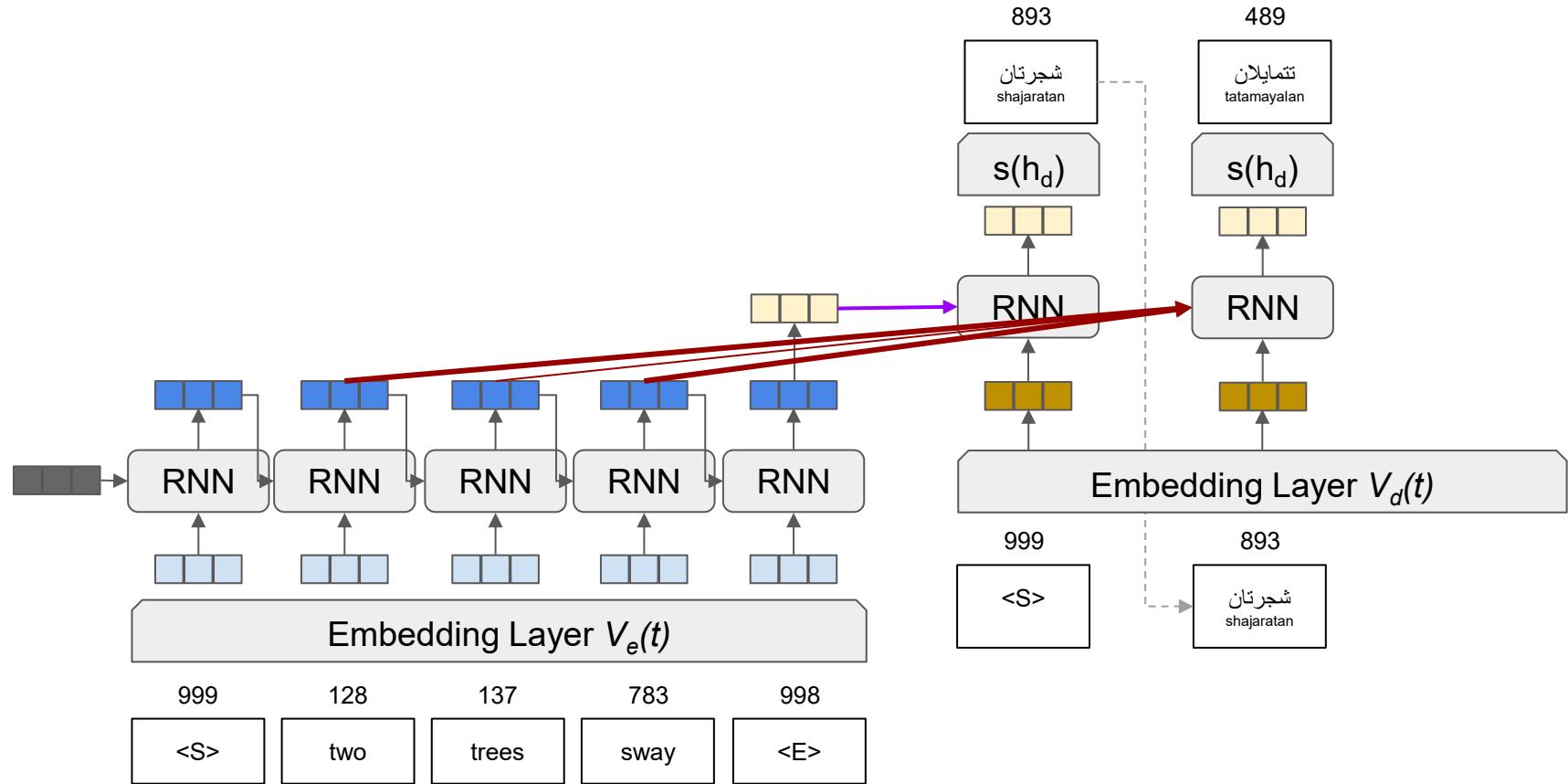
# Encoder-Decoder Attention: Global Versus Local

- We probably still don't need the entire sequence for every decoding step
- Can limit attention to a window of  $k$  around current decoding index to limit computation

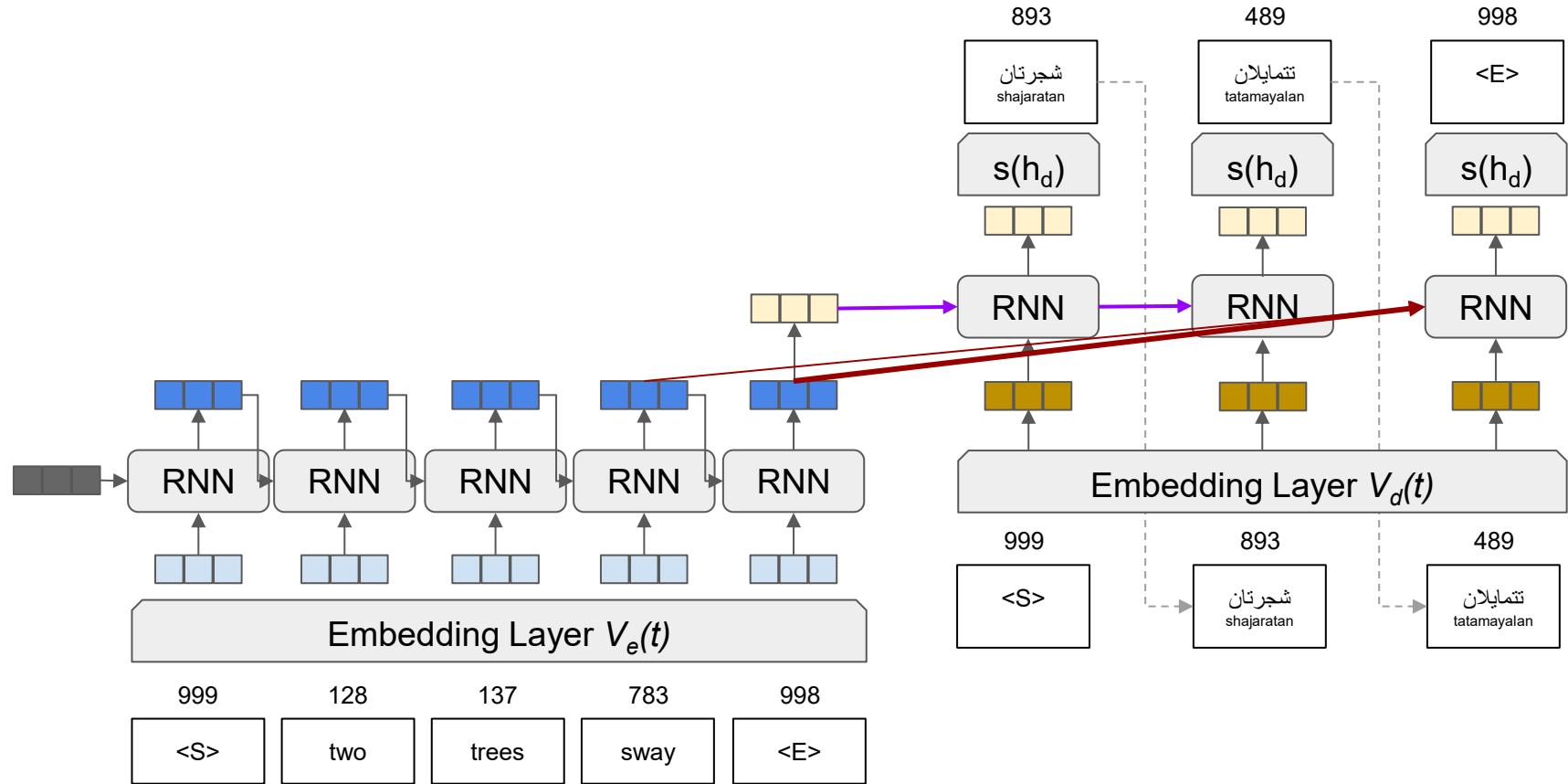
# Encoder-Decoder Attention: Global Versus Local



# Encoder-Decoder Attention: Global Versus Local



# Encoder-Decoder Attention: Global Versus Local



# Encoder-Decoder Attention: Soft versus Hard

- A lot of our weights will be near zero, probably
- We could instead just use the *most relevant* hidden state

## Soft Attention

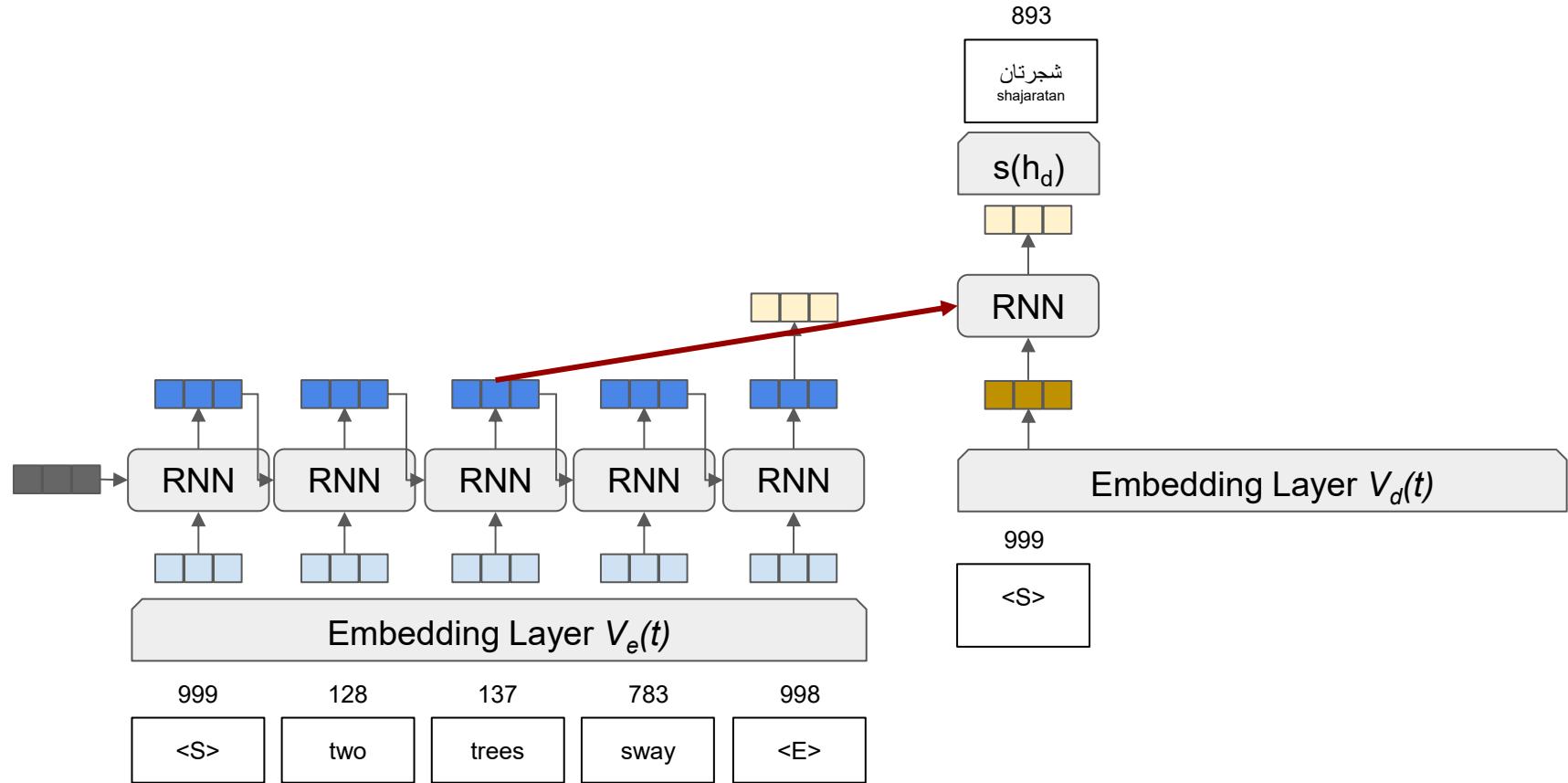
$$\sum_{i=1 \dots L} \left[ \frac{\exp(a(h_d, h_e))}{\sum_{j=1 \dots L} \exp(a(h_d, h_e))} \right]$$

## Hard Attention

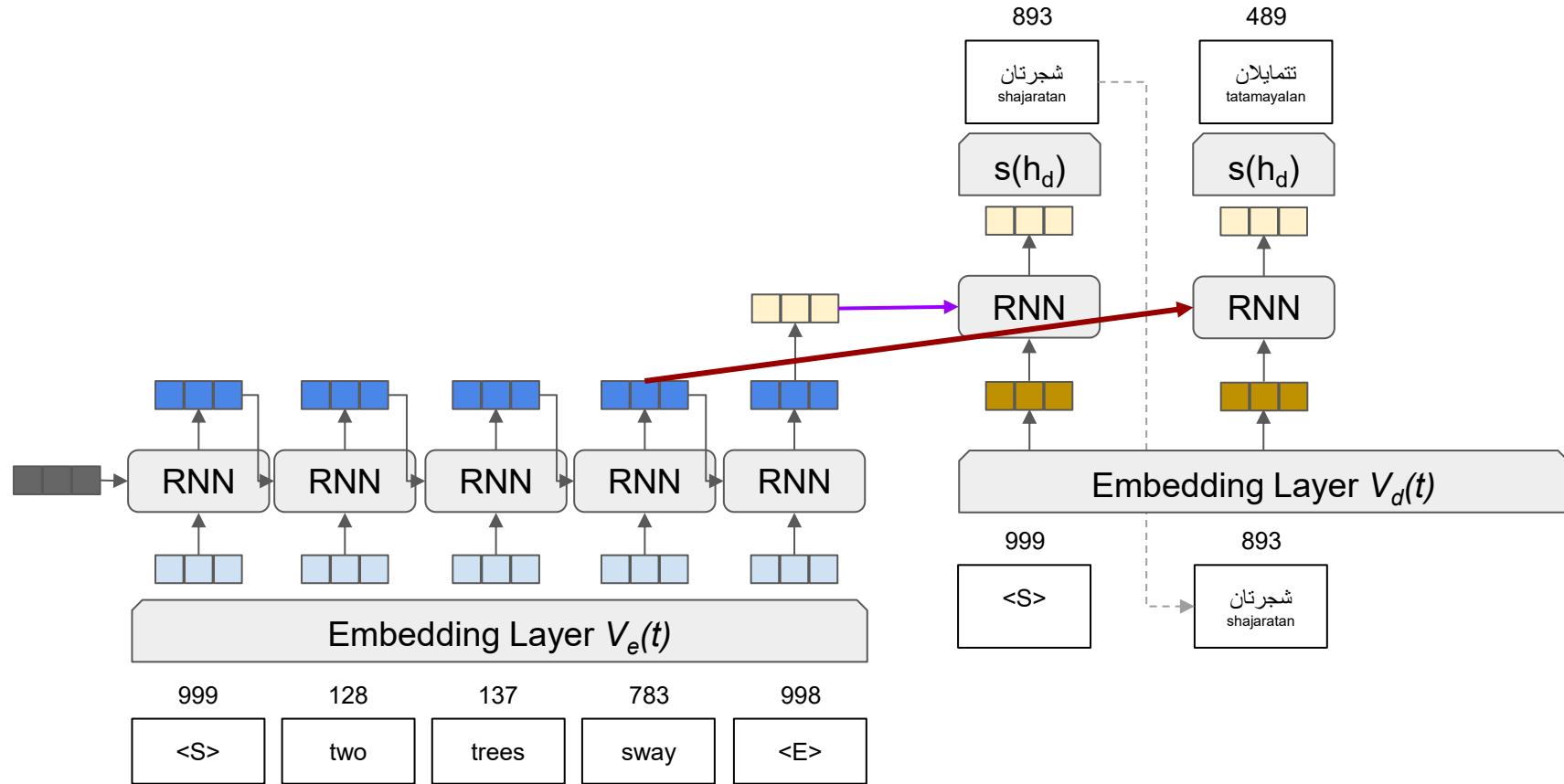
$$\text{argmax}_{h_e, i=1 \dots L} \frac{\exp(a(h_d, h_e))}{[\sum_{j=1 \dots L} \exp(a(h_d, h_e))]}$$

\* with some tricks to ensure the whole pipeline stays differentiable

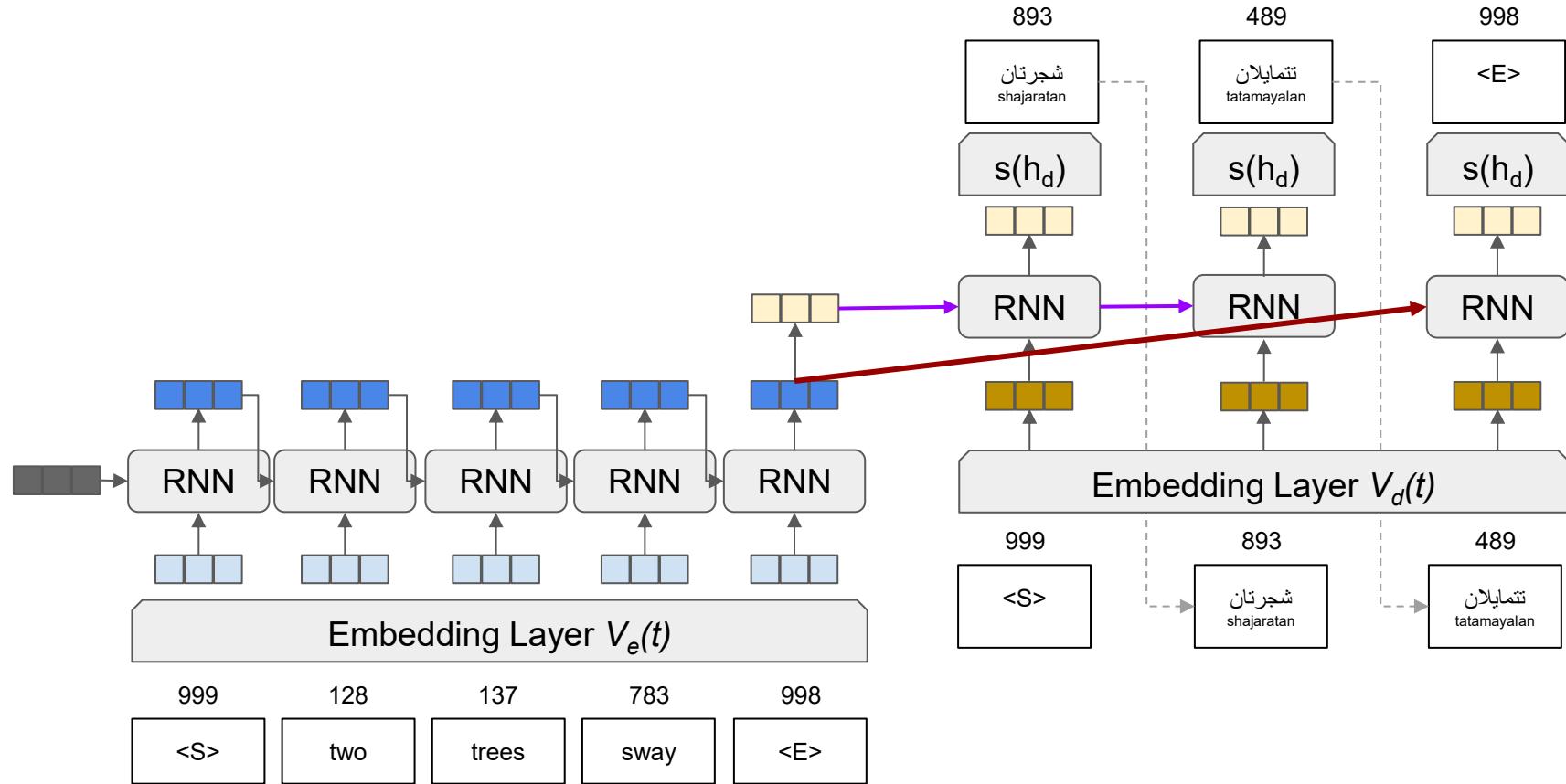
# Encoder-Decoder Attention: Hard Versus Soft



# Encoder-Decoder Attention: Hard Versus Soft



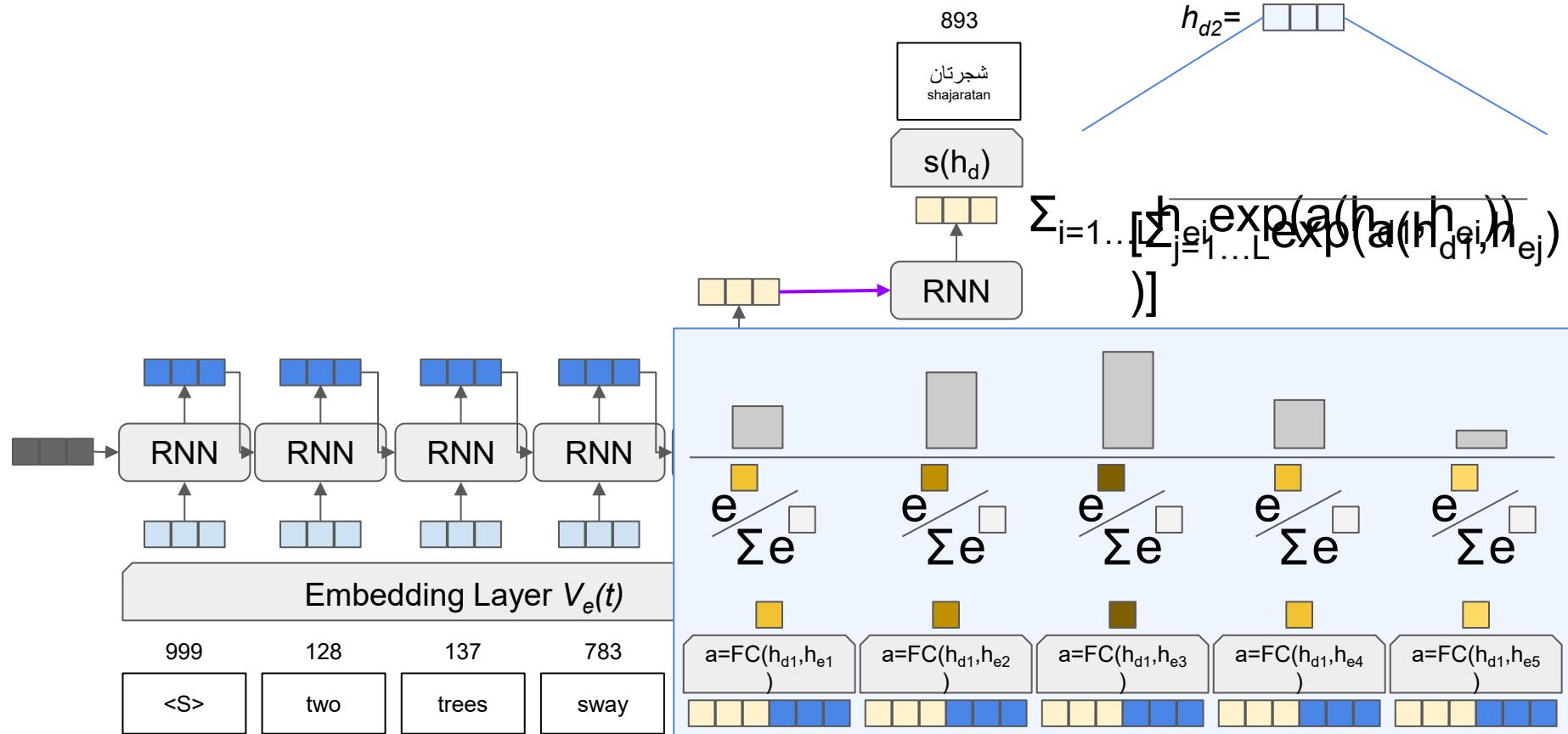
# Encoder-Decoder Attention: Hard Versus Soft



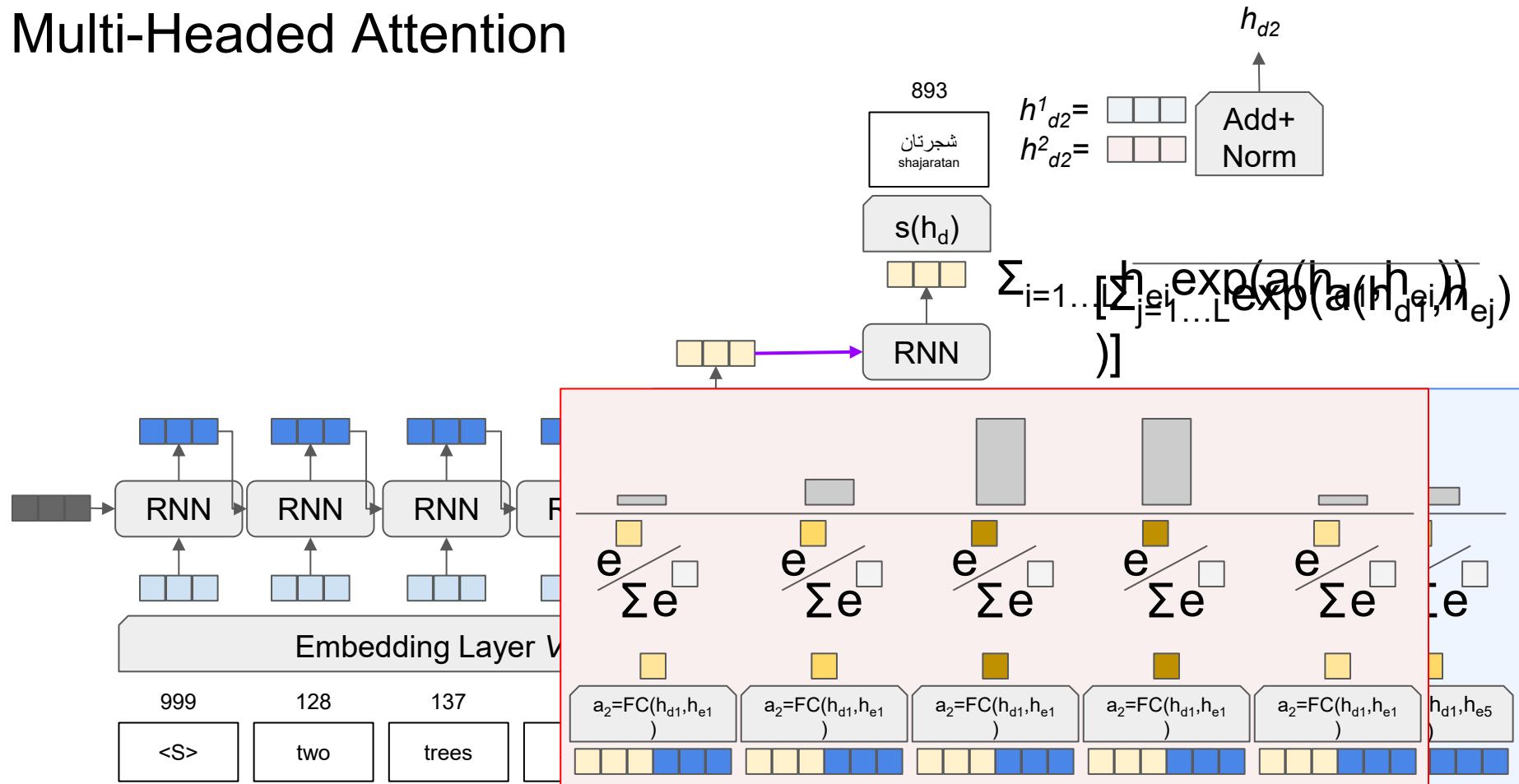
## Encoder-Decoder Attention: Multi-headed Attention

- The shared parameters of the attention head might only learn to pick out certain dependencies (e.g., noun-verb agreement, determiner-gender agreement, ...)
- We can create *multi-headed attention*, where we choose a number of attention heads  $k$  to learn independent of one another

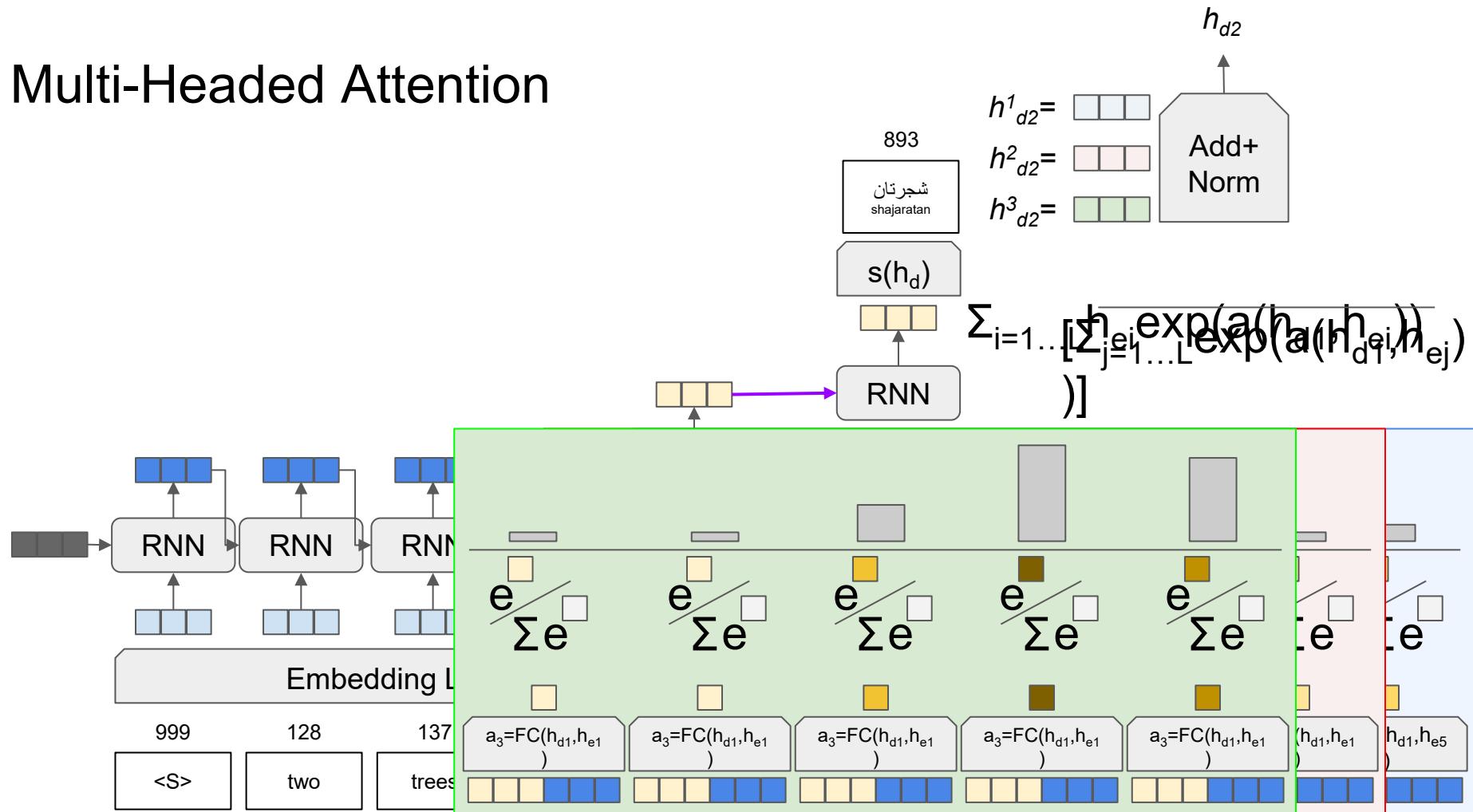
# Multi-Headed Attention



# Multi-Headed Attention



# Multi-Headed Attention



# Encoder-Decoder Attention: Recap

- The final hidden state of an encoder may not carry enough information about previous tokens to perform decoding
- *Global Soft Attention* creates a new conditional decoding vector  $x$  that is a weighted sum of all encoder hidden states, rather than only the final hidden state only
  - Subset of states: “Local” attention
  - Max score instead of weighted: “Hard” attention
- *Multi-headed Attention* learns multiple such scoring functions and sums the final vectors from each as the input hidden state

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 2: Lecture 1  
Sequence to Sequence Modeling

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 2: Lecture 2  
LM Decoding and Transformer Networks

# Lecture Outline

- Language Model Decoding
- Encoder-Decoder Attention Recap
- Transformer Networks and the Attention Mechanism
- Using Transformers for Language Modeling
- Zoom Out

# Lecture Outline

- **Language Model Decoding**
- Encoder-Decoder Attention Recap
- Transformer Networks and the Attention Mechanism
- Using Transformers for Language Modeling
- Zoom Out

# NLP FUNDAMENTALS: Language Models

- A *language model* is an estimate of the likelihood of a string
- Tokenizing the string into component parts is necessary
- So we can consider a string  $s$  as a sequence of tokens  $w_1 \dots w_n$
- Then a language model estimates:
  - $p(s) = p(w_1 \dots w_n)$
- Term is overloaded; also used these days to mean:
  - $p(w_k | w_1 \dots w_{k-1})$  — (e.g., *auto-regressive*)
  - $p(w_k | w_1 \dots w_{k-1} w_{k+1} \dots w_n)$  — (e.g., *cloze / MLM*)

# Uses of a Language Model

- If we have a function  $p(s) = p(w_1 \dots w_n)$  that assigns probabilities to a sequence of word tokens, what can we use it for?
  - *Scoring*: given a set of strings, we could use the LM to score their plausibility (for example, speech-to-text)
  - *Generation*: we could sample from this LM to get new strings that the LM deems plausible

# Discussion Activity: Generating from an $n$ -Gram LM

- Given the last one word, write down a grammatically correct and thematically plausible next word on the next line
  - E.g., “she”, “sells” as context, “seashells” / “lemonade” are reasonable follow up words
- *Fold* the paper back between the top word and second word, such that the top-most word is hidden, and pass the paper
- While that one sheet is going, a second with the last four words will also travel
  - Same rules for the four word context as the two word context

# Discussion Activity: Generating from an *n*-Gram LM

- Bigram (one word context) results:
  - [20223] upon playing football scored goals next, we go down the rabbithole went through stuff into our kitchen stove is hot
  - [20243] the picture showed me you are a bottle
- 5-gram (four word context) results:
  - [20223] call me Ishmael because I like to run and read. Then, go to the monolith and fed animals themselves.
  - [20243] a foul ball went into the net and saw her picture
- Co-occurrence:
  - Left context / Right context / bidirection context?
  - Context length:
    - Shorter context: word vectors favor syntax
    - Longer context: word vectors favor semantics/topic

# Generating Text from an n-Gram Language Model

- LM gives us an estimate,  $p^*(w_k|c)$  for *context* c, for example, the last word, the last two words, the last word and its POS, ...
- If we encode all our sentences with <start> and <end>, then we can *generate* a string that our LM finds likely in the language by:
  - $w_1 = \text{argmax}_w(p^*(w_1|\text{<start>}))$
  - $w_2 = \text{argmax}_w(p^*(w_2|w_1, \text{<start>}))$
  - $w_3 = \text{argmax}_w(p^*(w_3|w_2, w_1, \text{<start>})) \dots$
- When would we stop sampling?

# Generating Text from an n-Gram Language Model

- In our books corpus, what does argmax get us?
- $|V| = 1000$ ,  $n=2$ ,  $\epsilon=0.01$ 
  - <start> ah yes sir replied monte cristo pointed towards heaven knows exactly forward seized hold sacred sire replied [monte cristo pointed towards heaven knows exactly forward seized hold sacred sire replied]\*\* [1366.5914]
- $|V| = 1000$ ,  $n=3$ ,  $\epsilon=0.01$ 
  - <pad> <start> ah yes thank god for tonight nothing be lost each single <unk> fact above <unk> feet high even then beyond the worth of iron dantès would have been able to bear upon any occasion be <unk> unless indeed heaven took into <unk> street died almost in every particular but i cant bear your brave hand stranger <end> [243.3984]

# Generating Text from an n-Gram Language Model

- $|V| = 1000$ ,  $n=3$ ,  $\epsilon=0.01$ 
  - How do we generate the beginning of a sentence for  $n$  gram models with  $n>2$ ?
  - If  $n=2$ , we only need to look back one token;  $\langle s \rangle$  is “given”
- Can we get  $p(w_1|\langle s \rangle)$  from  $p(w_k|w_{k-1}w_{k-2})$ ?
  - Prove it!
- How did we handle counting for  $n>2$  in code?
  - $p(w_1|\langle s \rangle < \_ >)$
  - What happens if we *always* see  $< \_ >$  before  $\langle s \rangle$ ?

# Generating Text from an n-Gram Language Model

- If we take the *argmax* to generate the next token  $w_k$  at every timestep, are we guaranteed to get the *most likely string*?
  - Let's think step by step
- $n$ -Gram likelihood of a string  $s$  with tokens  $w_1 \dots w_n$ :
  - $p(w_1 \dots w_n) \approx p(w_2|w_1)p(w_3|w_2)\dots p(w_n|w_{n-1})$
- What will happen as the number of tokens in  $s$  increases?
  - $p(s)$  wrt #tkns is *monotonically non-increasing*
- So if we're generating an entire string token by token, what's our best move to maximize  $p(s)$ ?

# Strategies for Next Token Inference

- $n$ -Gram likelihood of a string  $s$  with tokens  $w_1 \dots w_n$ :
  - $p(w_1 \dots w_n) \approx p(w_2|w_1)p(w_3|w_2)\dots p(w_n|w_{n-1})$
- How to maximize  $p(s)$  during generation?
  - Just predict  $\langle\text{end}\rangle$ ?
    - Counter-case:  $p(\langle\text{end}\rangle|\langle\text{start}\rangle)=0$
  - Argmax at each token?
    - Counter-case:  $b=\text{argmax}(p(w|a))$  &  $a=\text{argmax}(p(w|b))$
  - Argmax at each token plus max length  $L$  to force  $\langle\text{e}\rangle$ ?
    - **Discussion Activity:** Explore this one on the board

# Strategies for Next Token Inference

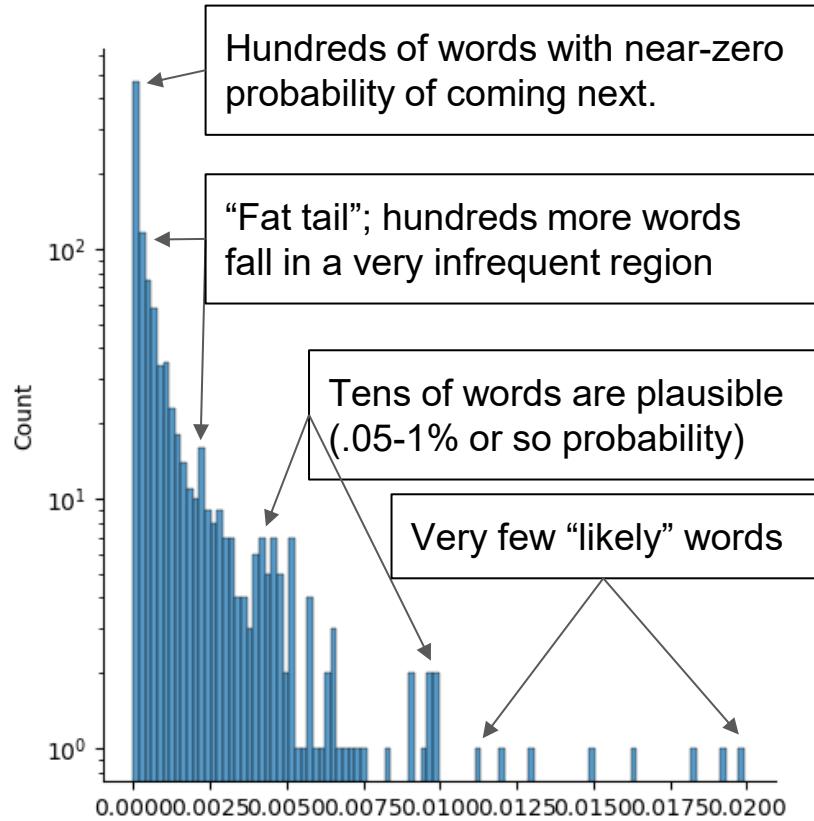
- $n$ -Gram likelihood of a string  $s$  with tokens  $w_1 \dots w_n$ :
  - $p(w_1 \dots w_n) \approx p(w_2|w_1)p(w_3|w_2)\dots p(w_n|w_{n-1})$
- How to maximize  $p(s)$  during generation?
  - Rather than *argmax*, what if we *sampled* from  $p(w_n|w_{n-1})$  at each timestep?
  - Are we likely to end up with a high probability string this way?
  - Need to consider “natural” distributions we’re likely to see in text corpora, rather than toy example

# Generating Text from an n-Gram Language Model

- In our books corpus, what does raw sampling get us?
- $|V| = 1000$ ,  $n=2$ ,  $\epsilon=0.01$ 
  - <start> maximilian morrel son next instant standing beside him whose race still live by many moments touched whether anything except act has slain achilles laid write again just made charles poor fellow curious thing impossible fancy makes following evening papers together often received written here as easily gold watch standing beside him instantly <end> [330.1306]
  - <start> inquired caderousse set forth as usual country called proud walls eight oclock found beauchamp keep there did have happened during those boats boats were moved small surprise each other divine beyond youth troy already passed away just told our fury cast its good sort since we please mr lorry **caught fire** others by people understood french law does **dr van helsing** shook his bright red room determined spirit might think charles taking six months ago \_ ever seen five oclock tomorrow beauchamp looked at our story **right whales** \_ our visit paris write sitting beside van helsing turned pale **blue ocean** through its least send any purpose fly along queequeg gave us return as a figure just below here indeed strange sense rather find terrible story however four walls eight minutes past seven oclock struck eight years since last struck down taking into sleep or whether pleasure because during dinner danglars whose story before quite true character although thats **strange creature** formed has placed beside her country where any land close upon my seat placed beneath stern ajax hector lies slain hundred thousand francs into light answer elizabeth therefore fixed point of late early account wonderful spirit their wall around them into tears filled with human race divine everything past fifty years before dantès your train and passed across the corner also arrived andrea ought one knows nothing particular state whom queequeg as good creature whom has written long passage where oer earth divine achilles hear none or sight others entered on seeing valentine shook itself seeing neither ought always seem better far way possible interest ever known remember desire that part tonight **dorian gray** vain while oer come forward child during dinner need be alone bore down among people remained silent state visit greece giving beowulf happened between elizabeth answered beauchamp repeated anne anne elliot cannot understand replied anne wanted ten minutes sleep beneath into greater beauty appear proud glance around ye my looking sky full front hair over such feelings dreadful ajax saw anyone madame defarge looking sky hear me ready money led through darkness closed behind us oer hector himself upon him gentlemen mr elliot pray sit up against fire around him quite engaged captain fine countenance remained silent they may once seized hold **sacred earth** who are all is bad race we become madame defarge stopped took mine already for some strange conversation instead memory brought these words tomorrow change indeed wonder stood near giving bad fortune everything else thats true replied **darcy cried** why stop heavy hours together no sense could bring down below strong sons married woman strong desire information today except just run before stubb caught hold most wonderful spirit and long journey **monte cristo** since beneath vast stone bold heroes dare return home grief **project gutenbergtm** electronic works professor stood [2651.6025]

# Strategies for Next Token Inference

- Need to consider “natural” distributions we’re likely to see in text corpora, rather than toy example
- Here’s the distribution of word tokens in  $p(w_k | \langle s \rangle, \langle \_ \rangle)$  in our books corpus for  $|V|=1000$ ,  $n=3$



# Strategies for Next Token Inference

- To *sample* the next token, we could sample over the entire distribution of  $|V|$ 
  - Pros: all strings stay possible to infer
  - Cons: “fat tail”; most probability mass is on unlikely words
- We could sample from the *top-k* highest probability words
  - Pros: increases likelihood of plausible string; balance between whole distribution sampling and argmax
  - Cons: throws away information in the tail, which might have been real; many strings now impossible to decode; need to pick a  $k$  that reflects the “right” amount of mass

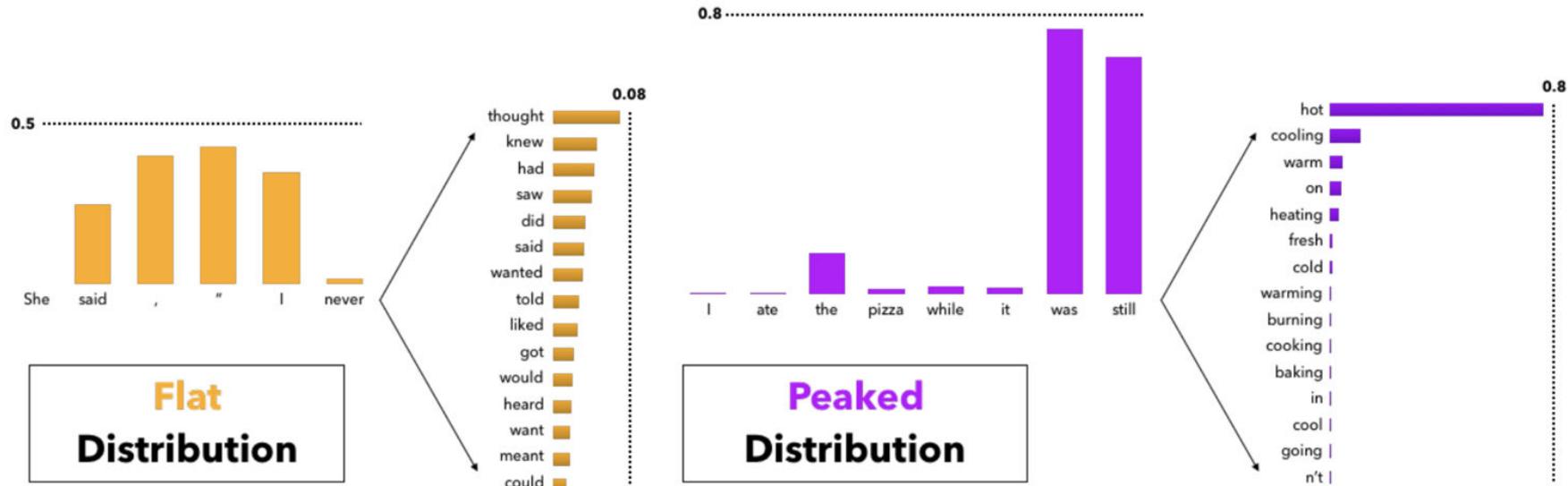
# Generating Text from an n-Gram Language Model

- In our books corpus, what does top- $k$  sampling get us?
- $|V| = 1000$ ,  $n=2$ ,  $\epsilon=0.01$ ,  $k=10$

- - <start> ah yes sir replied villefort drew back again closed within half filled another word monsieur defarge moved away sons arm round table turning pale blue skies o sacred age however yesterday doctor lived almost equal fury dare say anything wrong fair play began looking at last night journey across his hand over everything ought to pay off fernand looked round forced smile starbuck exclaimed turning towards ali turned towards heaven beside fernand became necessary therefore fixed glance around us sit down beside their usual state visit paid yourself tonight dr van helsing walked slowly rose high wall moved across four oclock tomorrow probably never forget what did not think ill try to meet achilles dreadful field proud achilles stood fixed point whether noirtier de morcerf drew near except myself bore safe enough happened since thinking over everything around suddenly stopped laid hold hard upon us copyright law means divine sent mademoiselle danglars immediately rose mighty chief attention doctor wanted anything else besides although debray continued franz remained therefore quickly understood french gentlemen turning over france between france between perfectly calm attention paid within fifty years afterwards quickly written words courage therefore wait wait longer lived without giving battle wait wait longer lived among persons attention suddenly stopped at least expected thee early opportunity returned morcerf turning pale trembling queen elizabeth began again jonathan since jonathan leaving paris murmured holmes caught sight beneath vast gray threw herself giving bad fortune remained perfectly natural thoughts unless indeed charge anything about five oclock today taking care giving noirtier cast mans society except myself whether bingley met instead of course rich gentleman doing something wrong tonight because instead of heaven exclaimed morrel seized him today because human nature try tonight for dinner party probably feel proud hopes nor sacred floor where hector glory save thy aid high command yourself safe through fair opportunity ahab starbuck taking different lives dark ahab touch unless among these words ran down beneath caught sight appear impossible impossible for yourself safe through mere beauty sense since leaving london road between them both rich gentleman doing wrong tonight since yesterday sitting sad fate rather than seven hundred times touched nor seek pity mine host command hardly help feeling curious chance placed between france between them both heroes aid neither jane united states copyright law means easy matter maximilian tried to meet van helsing instead of mine also cast around us copyright law itself seeing persons living body else could scarcely less fury cast around us copyright notice cavalcanti repeated defarge looked around us talk tomorrow beauchamp thank god o daughter whom jove rest bore down on earth nor sacred main glad surprise yesterday murmured dorian gray answered caderousse sitting beside fernand became clear gray threw myself entirely free tonight dr van helsing forced tone sit near beowulf happened [1902.6866]

# Strategies for Next Token Inference

- Choosing  $k$  well is really tricky



# Strategies for Next Token Inference

- We could sample from the  $top-p$  highest probability masses
  - This strategy is called *nucleus sampling*
  - Pros: increases likelihood of plausible string; balance between whole distribution sampling and argmax; just need to pick a probability threshold
  - Cons: throws away information in the tail, which might have been real; many strings now impossible to decode

# Generating Text from an n-Gram Language Model

- In our books corpus, what does nucleus sampling get us?
- $|V| = 1000, n=2, \epsilon=0.01, p=0.1$ 
  - <start> ah yes sir replied franz remained silent till late memory makes neither jane leaving london road below beneath vast deal better than usual calm tone exactly forward seized hold sacred floor instantly ran across four boats crew instantly ran across four boats crew seized hold sacred sire replied monte cristo drew near enough evil eye caught sight appear proud hopes remain till late memory makes neither jane instantly ran across four boats crew instantly rose quickly understood french gentlemen added caderousse rising wind shook hands together formed hopes remain till late years ago \_ <end> [300.5456]
  - <start> ah yes sir replied monte cristo stopped suddenly [stopped suddenly]\*\* [1244.4627]
- $|V| = 1000, n=2, \epsilon=0.01, p=0.2$ 
  - <start> thank god knows nothing else besides mere chance watch over london road below beneath caught sight appear proud troy seized hold sacred light hair stood beside fernand whose character set forth oer safe fifty thousand francs each other things are sure swift glance around us copyright law does anyone else thats right whale line sacred sire murmured starbuck giving way through several times except those who lives \_ <end> [227.9336]

# Strategies for Next Token Inference

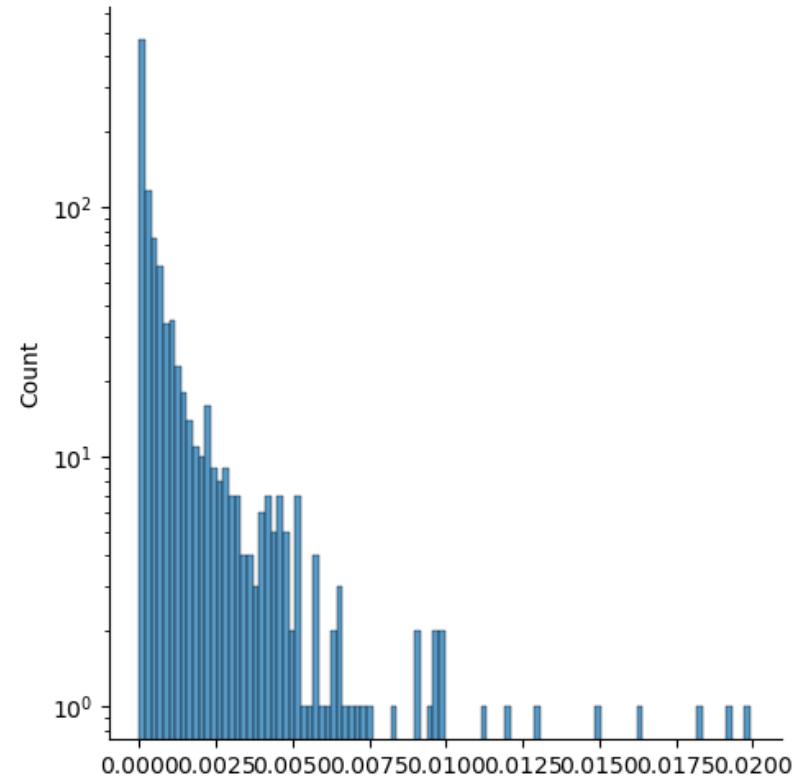
- $n$ -Gram likelihood of a string  $s$  with tokens  $w_1 \dots w_n$ :
  - $p(w_1 \dots w_n) \approx p(w_2|w_1)p(w_3|w_2)\dots p(w_n|w_{n-1})$
- How to maximize  $p(s)$  during generation?
  - Exact inference?
    - Enumerate all possible next characters at each timestep and keep a running score
    - Complexity if there's no fixed length  $L$ ?
      - Infinite!
    - Complexity for  $n$ -gram model of fixed length  $L$ ?
      - $|V|^L$ ; for  $|V|=1000$  and  $L=3$ , that's already a billion

# Strategies for Next Token Inference

- $n$ -Gram likelihood of a string  $s$  with tokens  $w_1 \dots w_n$ :
  - $p(w_1 \dots w_n) \approx p(w_2|w_1)p(w_3|w_2)\dots p(w_n|w_{n-1})$
- How to maximize  $p(s)$  during generation?
  - What do we always do when exact inference is intractable?
  - *Heuristic search!*
- *Beam search* over the output space
  - Two possible ways to cap our beam
    - Hint: full search is  $|V|^L$
  - We could only look  $k$  steps ahead [limit  $L$ ]
  - Or only look at the top  $k$  words [limit  $|V|$ ]

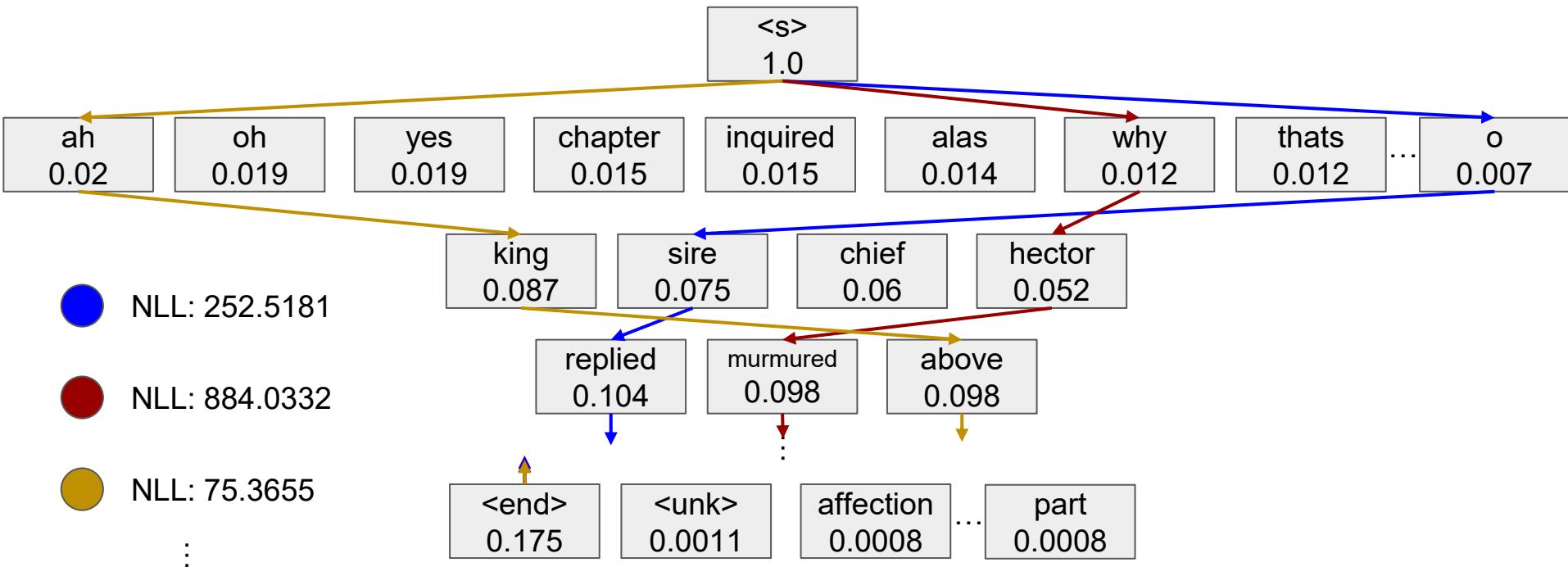
# Re-ranking Generation Outputs with Beam Search

- *Beam search over the output space*
  - We could only look  $k$  steps ahead [limit L]
  - Or only look at the top  $k$  words [limit  $|V|$ ]
- What kind of beam is likely to get us farthest given that we're working with language data?
- Tokens! Look at all that waste:

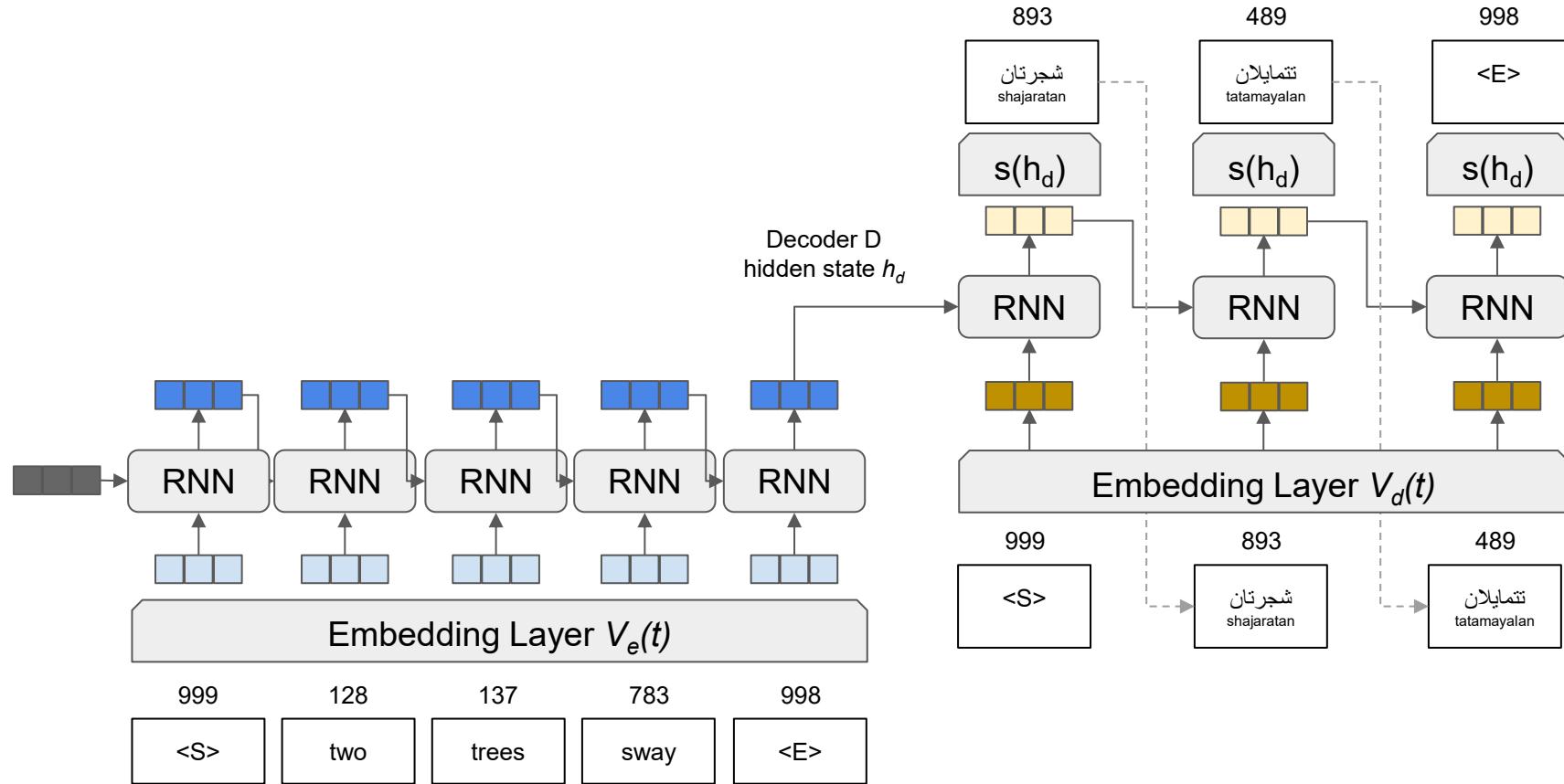


# Re-ranking Generation Outputs with Beam Search

<start> o sire replied franz merely wished sometimes sit down beside ali bring tears filled with equal rage hector comes back again lived mercédès although unhappy boy continued caderousse rising sun rise beyond expression fixed despair seized hold forth oer heavens sake dont want anything else appear proud troy walls trembling voice divine achilles slain companion shook hands touched stone walls eight oclock struck cold water darkness oer heavens sake dont want anything else besides \_ <end>



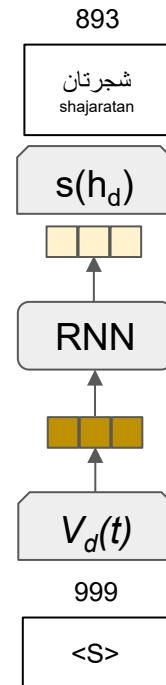
# Machine Translation and The Case for Contextual Info



# Decoding the Next Token

- Let's set the embedding dimension to  $m$  and the hidden dimension also to  $m$
- That is,  $|V_d(t)| = |h_d|$
- We can set  $s(h_d)$  to be the *product* between possible tokens and the hidden state:
  - Let  $z_i = V_d(t) \cdot h_d = \sum_{k=1 \dots m} \{V_d(t)\}_k \{h_d\}_k$

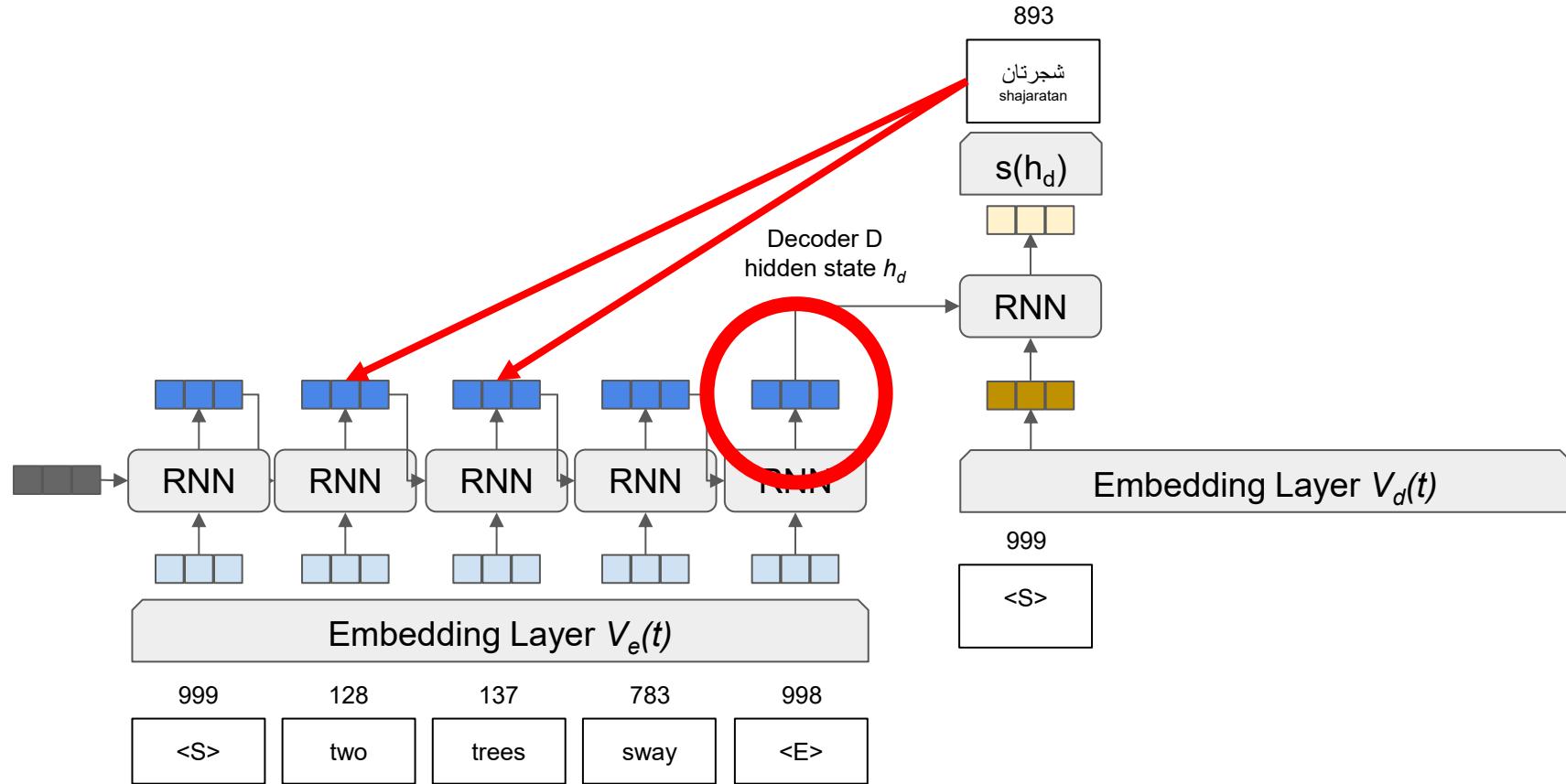
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



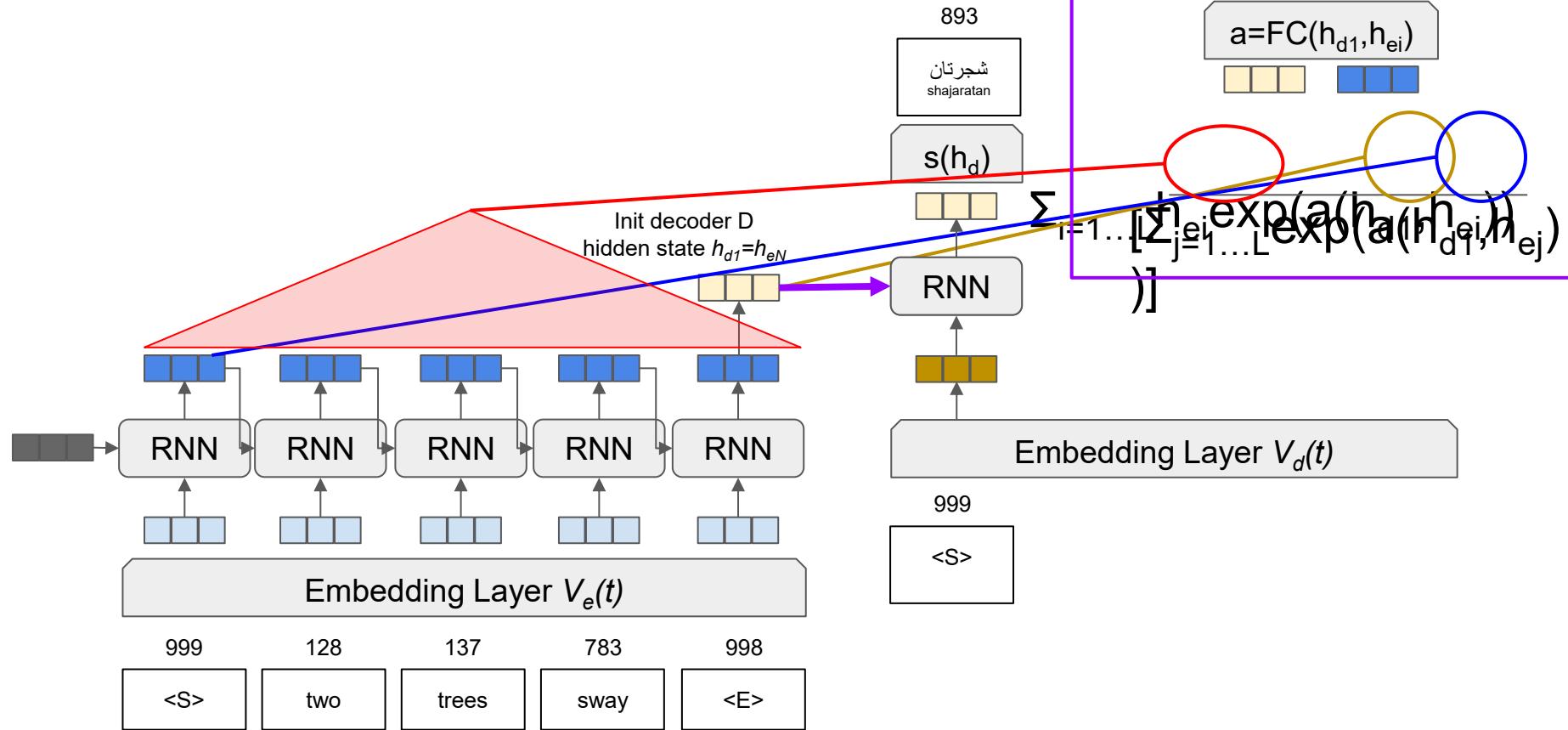
# Lecture Outline

- Language Model Decoding
- **Encoder-Decoder Attention Recap**
- Transformer Networks and the Attention Mechanism
- Using Transformers for Language Modeling
- Zoom Out

# Machine Translation and The Case for Contextual Info



# Use Attention to Rewrite RNN State Input



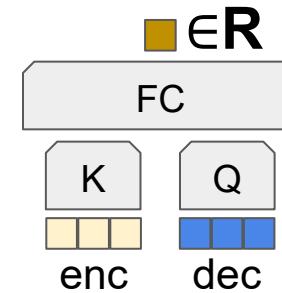
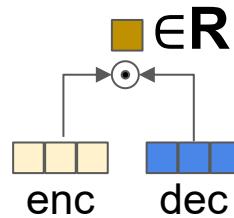
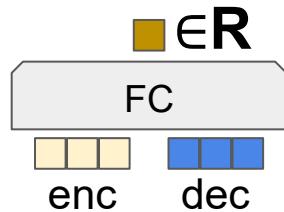
# Encoder-Decoder Attention

- The final hidden state of an encoder may not carry enough information about previous tokens to perform decoding
- *Attention* reweights encoder states  $h_{e[1:L]}$  as a function of the decoder hidden state  $h_{d[t-1]}$  at each timestep
  - $$h_{d[t]} = [\sum_{i=1 \dots L} h_{e[i]} \exp(a(h_{d[t-1]}, h_{e[i]}))] / [\sum_{j=1 \dots L} \exp(a(h_{d[t-1]}, h_{e[j]}))]$$
- A learned attention *head* determines the matching score between the decoder's last output state and each candidate encoder hidden state
- An attention *head* has shared weights for all encoder states, so it's cheap to add, parameter-wise

# Encoder-Decoder Attention: Recap

- The final hidden state of an encoder may not carry enough information about previous tokens to perform decoding
- *Global Soft Attention* creates a new conditional decoding vector  $x$  that is a weighted sum of all encoder hidden states, rather than only the final hidden state only
  - Subset of states: “Local” attention
  - Max score instead of weighted: “Hard” attention
- *Multi-headed Attention* learns multiple such scoring functions and sums the final vectors from each as the input hidden state

# Attention Variants



- $\text{FC}(h_{enc}, h_{dec})$
- $\text{dot}(h_{enc}, h_{dec})$
- $\text{FC}(Kh_{enc}, Qh_{dec})$
- Key matrix
- Query matrix

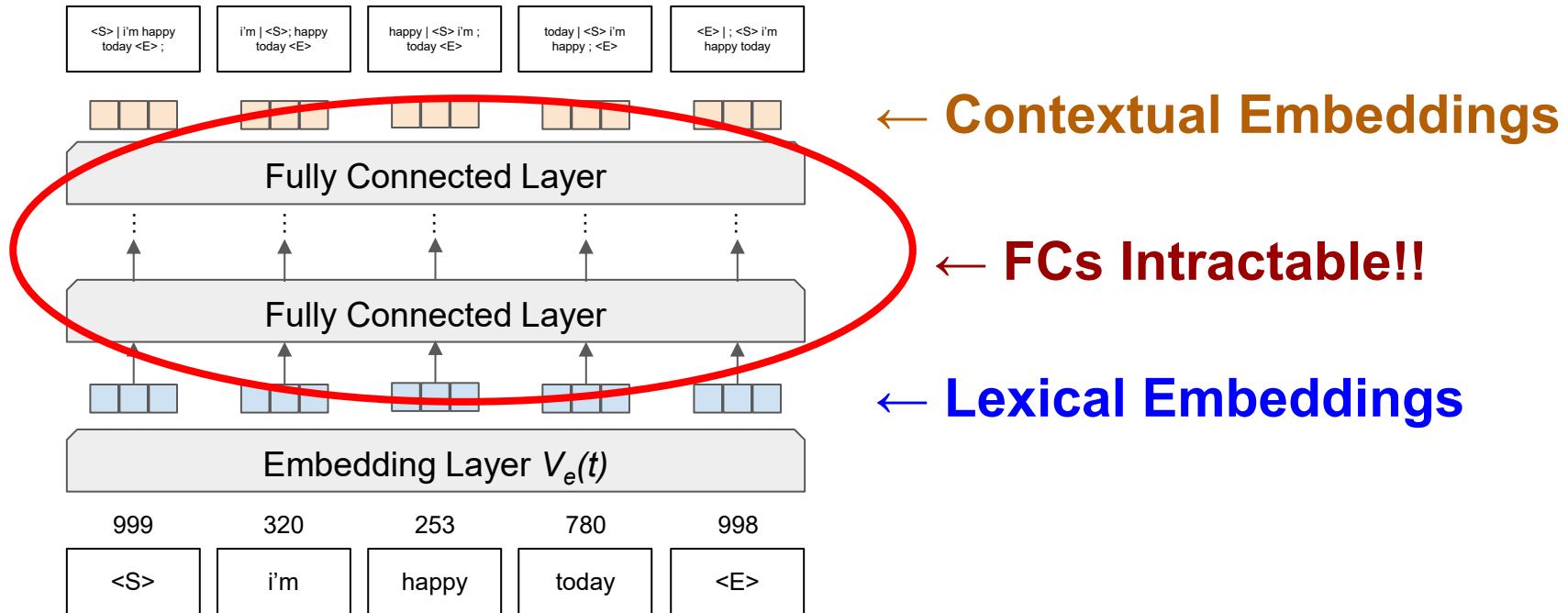
# Lecture Outline

- Language Model Decoding
- Encoder-Decoder Attention Recap
- **Transformer Networks and the Attention Mechanism**
- Using Transformers for Language Modeling
- Zoom Out

# Contextual Word Embeddings

- Consider the word vectors in the following sentences:
  - $\mathbf{X} = \{\text{"she launched into a bass solo"}, \text{"the line went taut as the bass pulled"}, \text{"the power of the bass broke the string"}\}$
- Many words have multiple senses, or meanings; even worse for BPE tokens!
- Polysemy is a key weakness of so-called *lexical* embeddings
- We can instead learn a function that combines *input* lexical embeddings from each word to produce *output* **contextual** embeddings that consider surrounding words

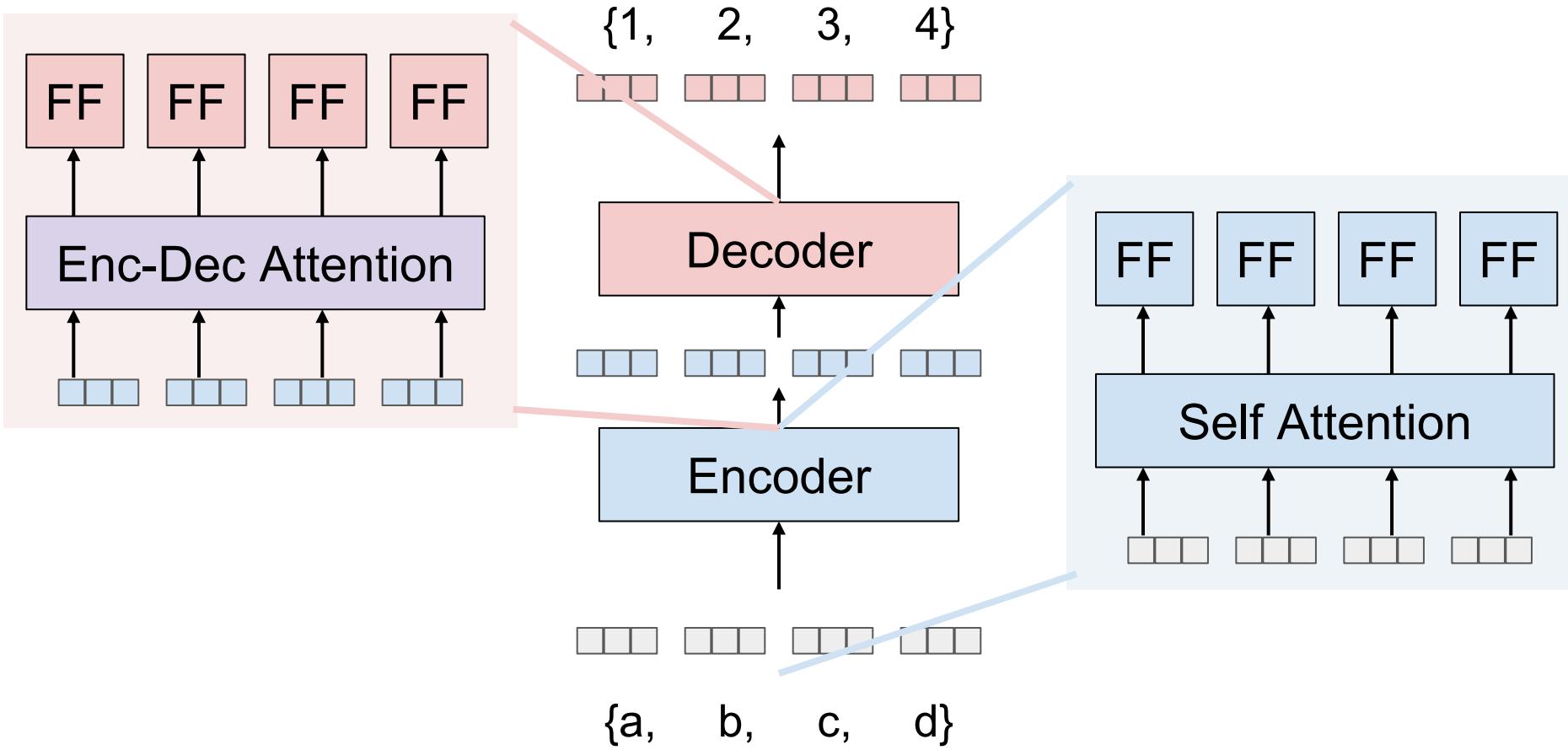
# Contextual Word Embeddings



# Transformers Lite

- Transformer  $T : S_a \rightarrow S_b$  is a function from a set  $a$  in  $S_a$  to a set  $b$  in  $S_b$
- Simple transformer consists of a single layer encoder  $E$  and a single layer decoder  $D$  with single-headed attention

# Transformers Lite

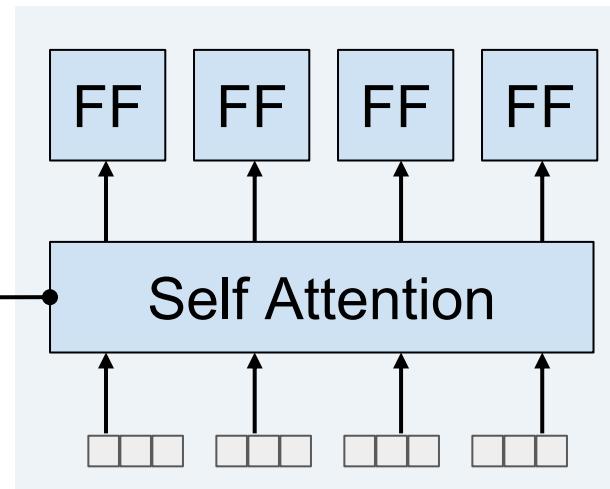


# Scaled dot-product attention

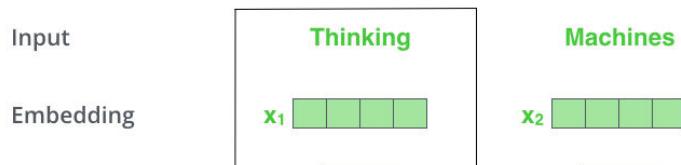
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

Our input for *self-attention* is the matrix **X** which is the embeddings of the input sequence

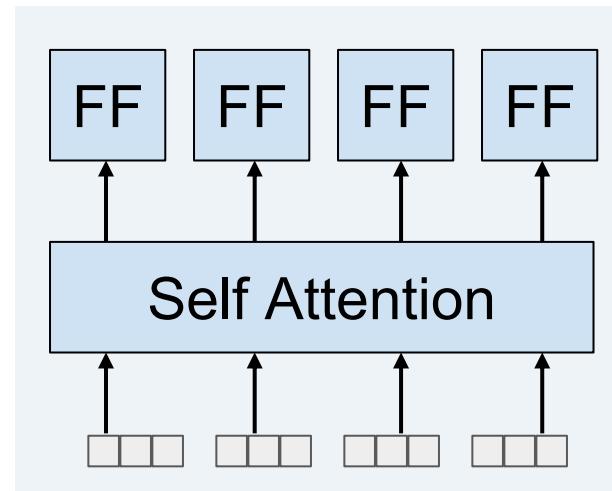
So we call  $\text{Attention}(XW_Q, XW_K, XW_V)$



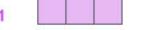
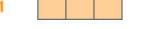
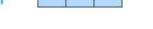
# Self-Attention

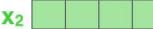
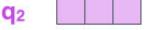
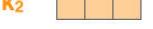


Attention( $\mathbf{XW}_Q, \mathbf{XW}_K, \mathbf{XW}_V$ )  
 $\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$   
 $q_1 = x_1 W_Q \dots$



# Self-Attention

Input	<b>Thinking</b>
Embedding	$x_1$ 
Queries	$q_1$ 
Keys	$k_1$ 
Values	$v_1$ 

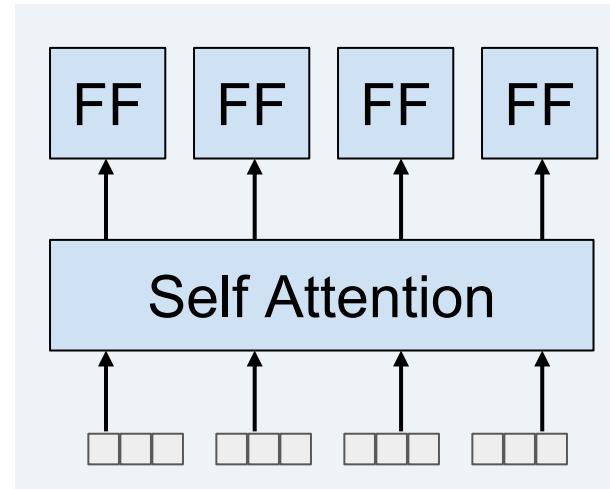
Machines	$x_2$ 
	$q_2$ 
	$k_2$ 
	$v_2$ 

Attention( $\mathbf{XW}_Q$ ,  $\mathbf{XW}_K$ ,  $\mathbf{XW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

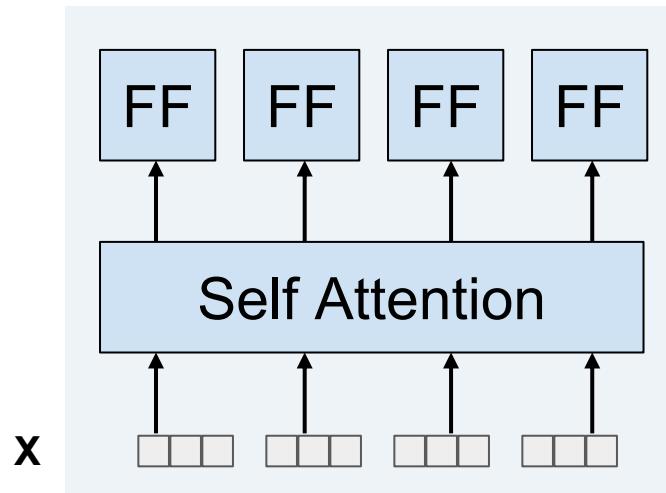
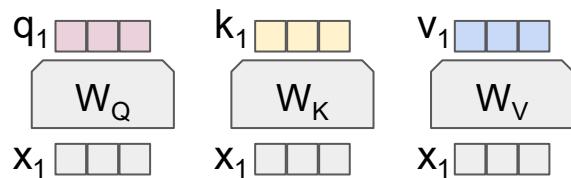
$$\left. \begin{array}{l} q_1 = x_1 W_Q \dots \\ k_1 = x_1 W_K \dots \\ v_1 = x_1 W_V \dots \end{array} \right\}$$

Learnable weights  $W_Q$ ,  $W_K$ ,  $W_V$  for each attention head.



# Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$



# Self-Attention

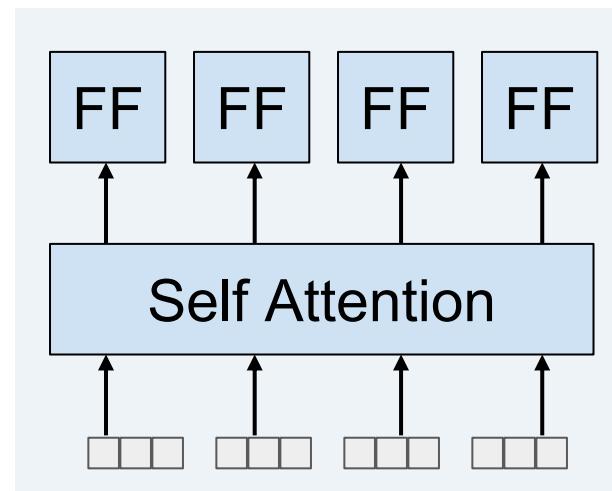
Input	Thinking		Machines	
Embedding	$x_1$	[4 green boxes]	$x_2$	[4 green boxes]
Queries	$q_1$	[3 purple boxes]	$q_2$	[3 purple boxes]
Keys	$k_1$	[3 orange boxes]	$k_2$	[3 orange boxes]
Values	$v_1$	[3 blue boxes]	$v_2$	[3 blue boxes]
Score	$q_1 \cdot k_1 = 112$		$q_1 \cdot k_2 = 96$	
Divide by 8 ( $\sqrt{d_k}$ )	14		12	
Softmax	0.88		0.12	
Softmax X Value	$v_1$	[3 blue boxes]	$v_2$	[3 blue boxes]
Sum	$z_1$	[3 pink boxes]	$z_2$	[3 pink boxes]

Attention( $\mathbf{XW}_Q, \mathbf{XW}_K, \mathbf{XW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

Learnable weights  $W_Q, W_K, W_V$

Attention of  $x_1$  to  $x_2$  is non-symmetric via Q, K matrices



# Self-Attention

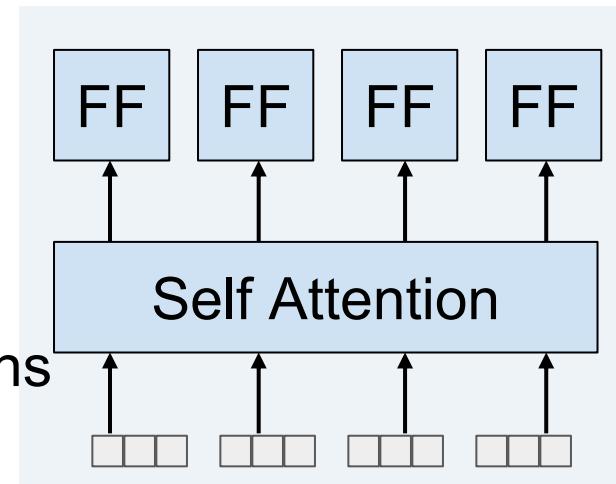
Input	<b>Thinking</b>	<b>Machines</b>
Embedding	$x_1$	$x_2$
Queries	$q_1$	$q_2$
Keys	$k_1$	$k_2$
Values	$v_1$	$v_2$
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ( $\sqrt{d_k}$ )	14	12
Softmax	0.88	0.12
Softmax X Value	$v_1$	$v_2$
Sum	$z_1$	$z_2$

Attention( $\mathbf{XW}_Q$ ,  $\mathbf{XW}_K$ ,  $\mathbf{XW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

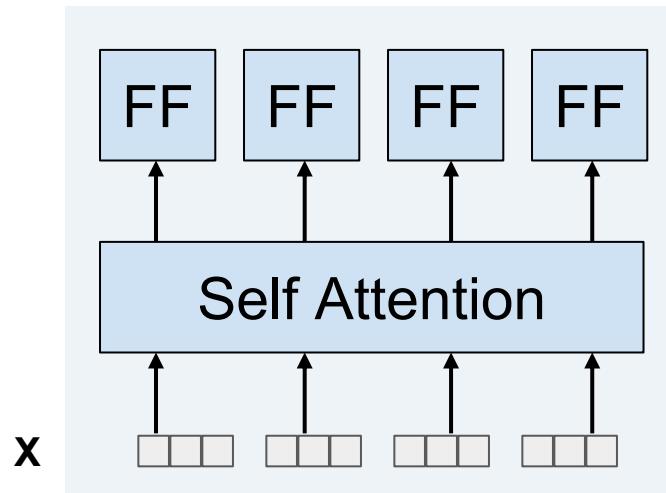
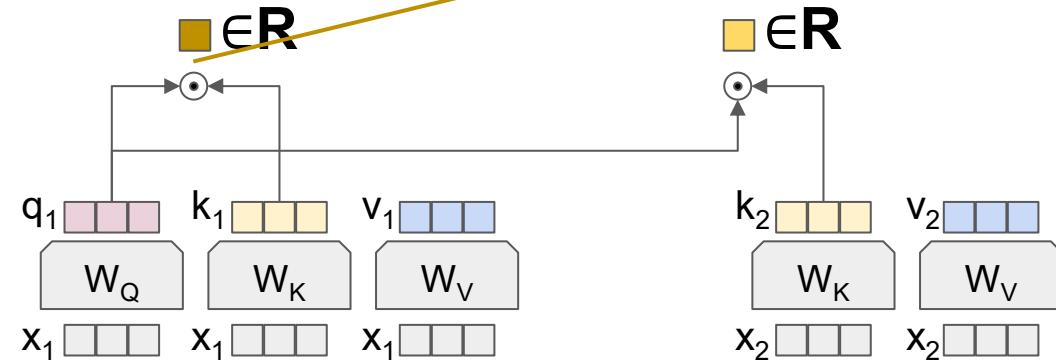
Attention of  $x_1$  to  $x_2$  is non-symmetric via Q, K matrices

$W_V$  matrix learns a “useful” output for each word’s context



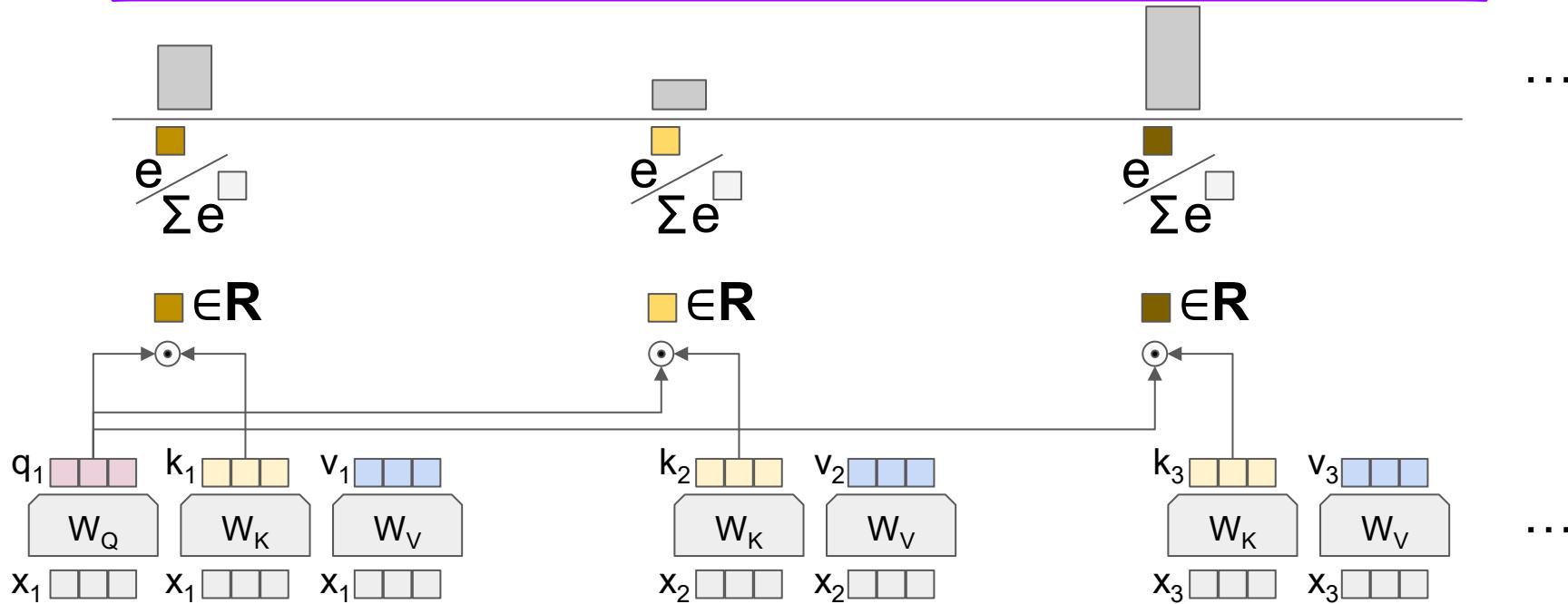
# Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

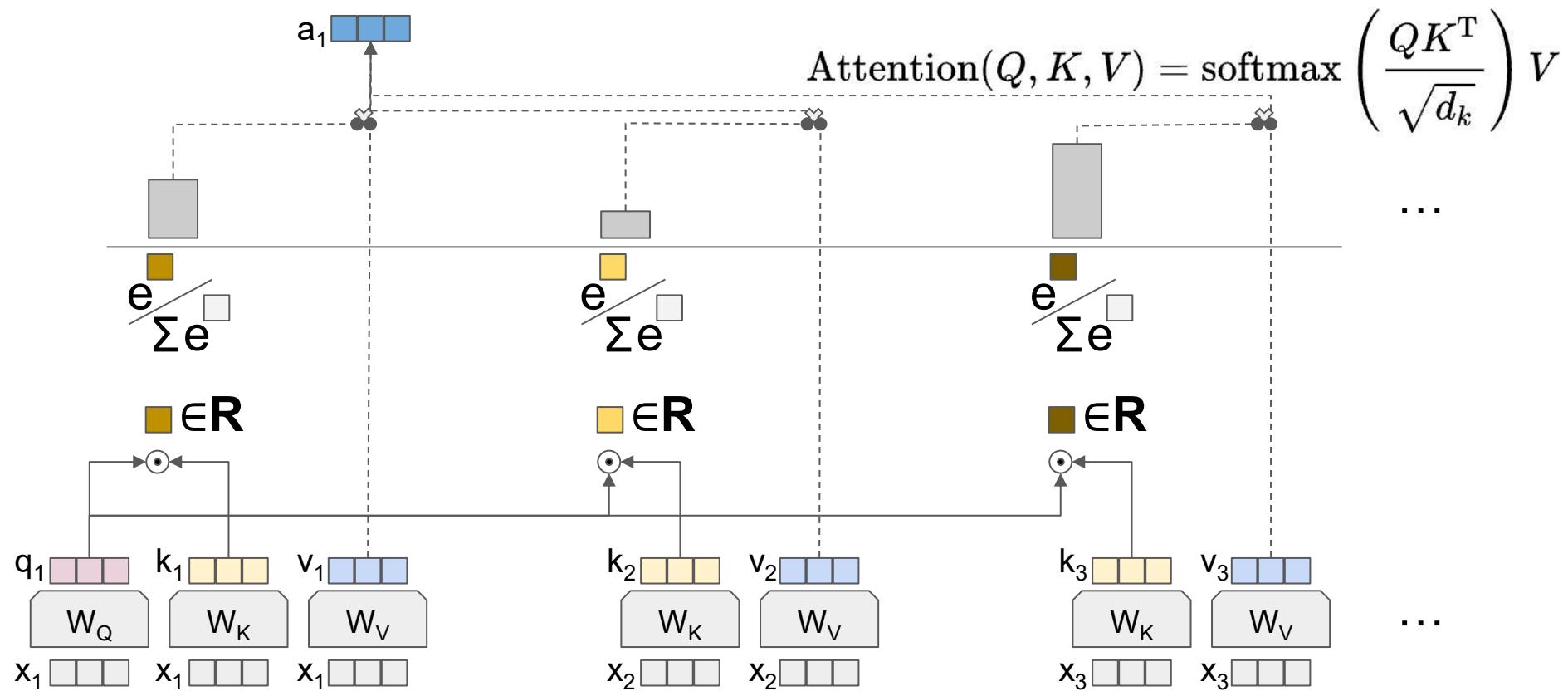


# Self-Attention

$$\text{Attention}(Q, K, V) = \underbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}_{\dots}$$

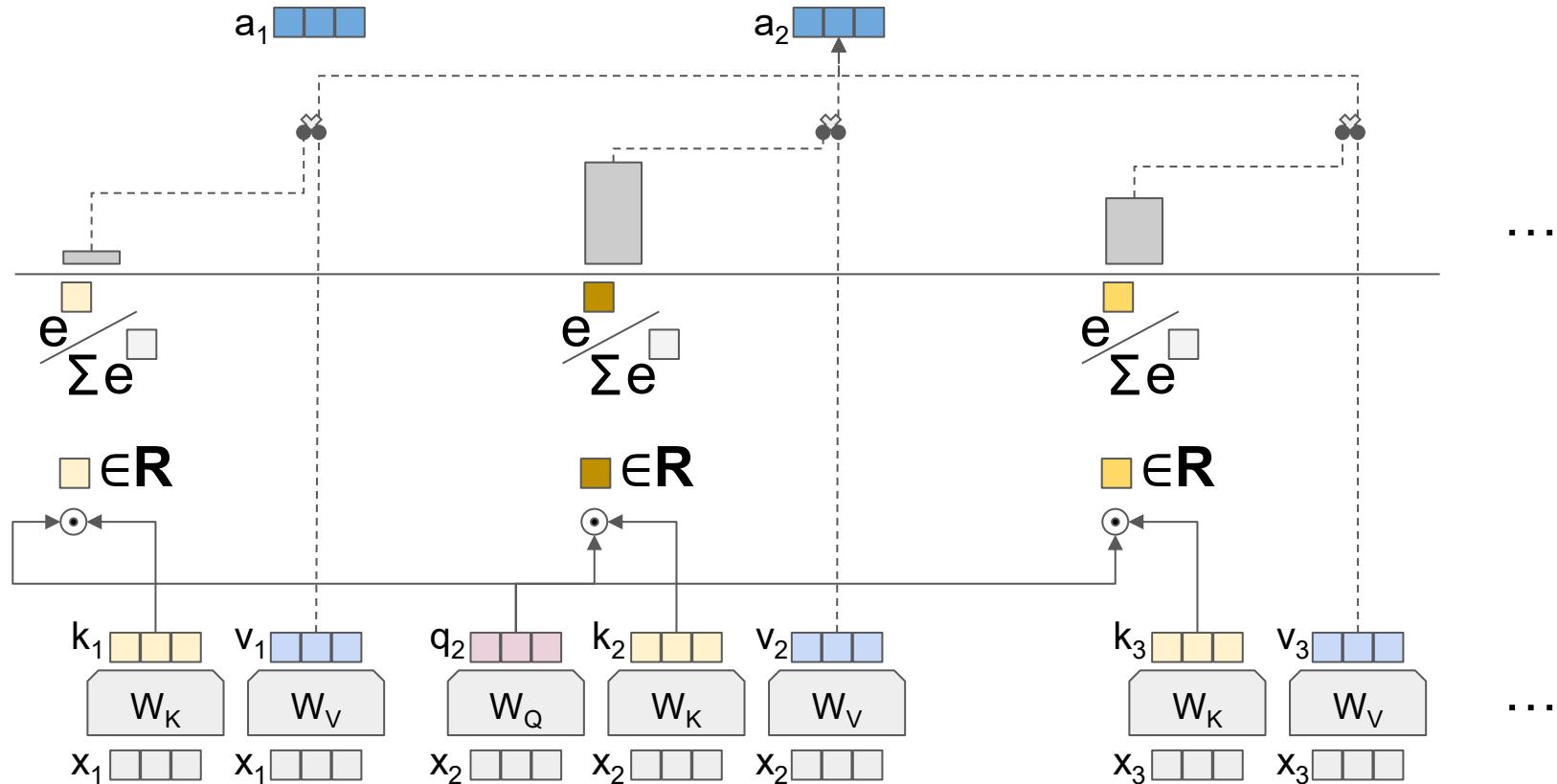


# Self-Attention



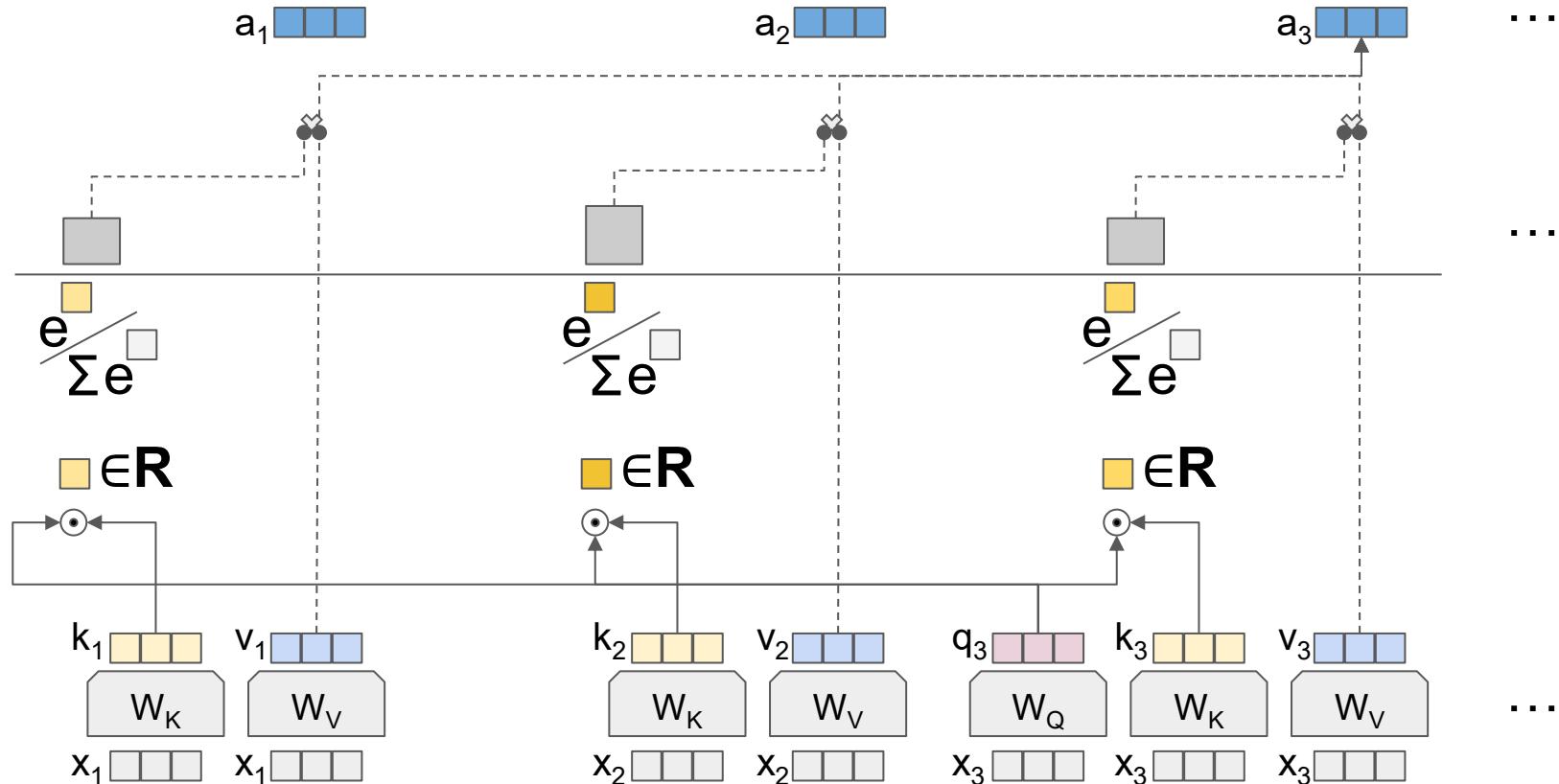
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

# Self-Attention



$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

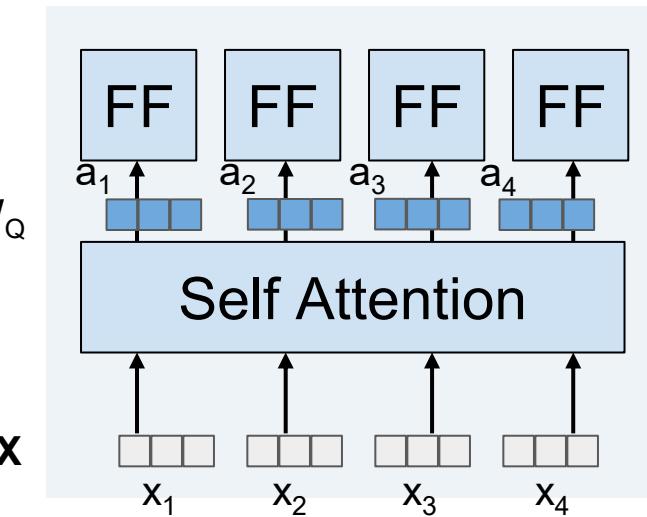
# Self-Attention



# Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$$A(XW_Q, XW_K, XW_V)$$



# Self-Attention

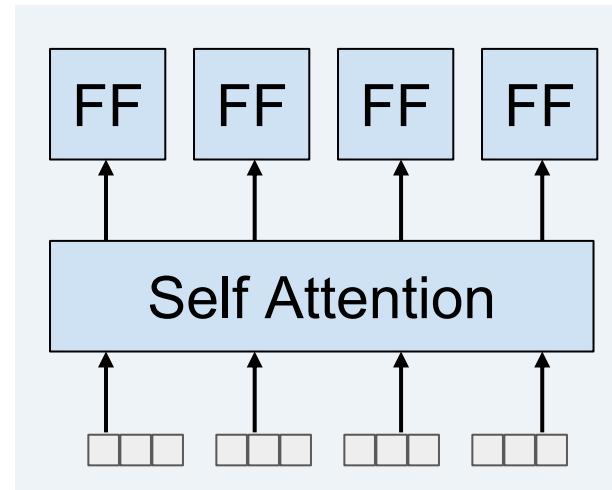
Input  
Embedding



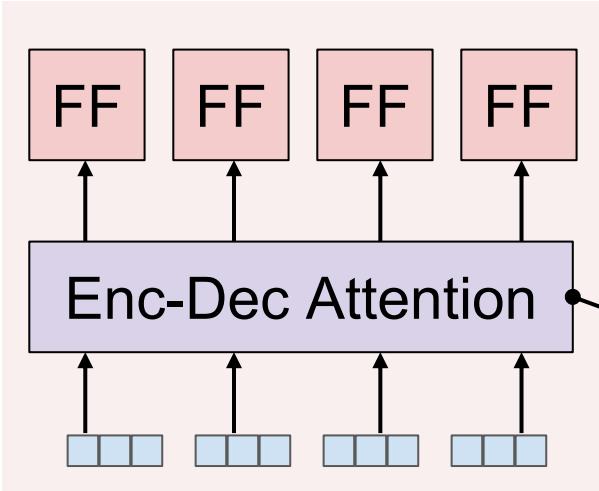
Attention( $\mathbf{XW}_Q, \mathbf{XW}_K, \mathbf{XW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$$q_1 = x_1 W_Q \dots$$



# Encoder-Decoder Attention



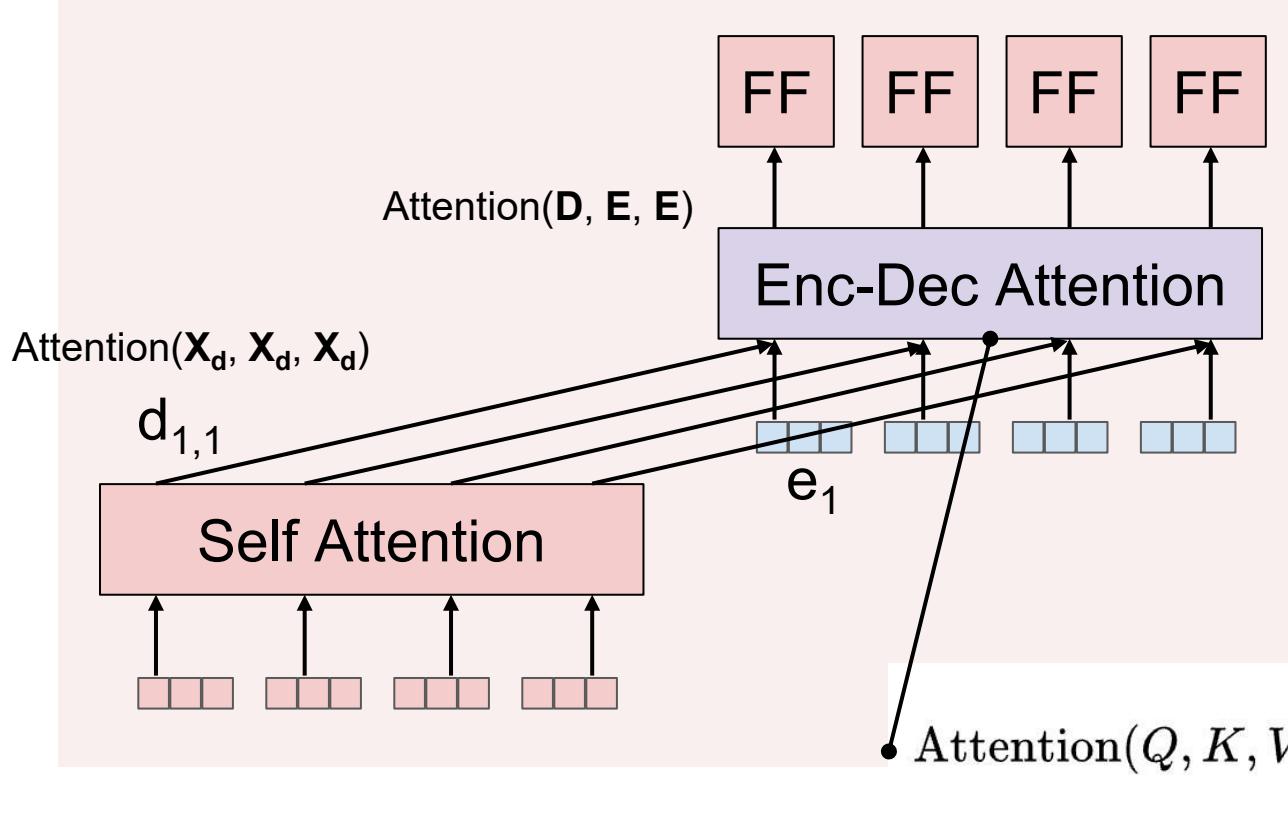
Call outputs from encoder self-attention  $\mathbf{E}$ ,  
decoder self-attention  $\mathbf{D}$

So we call Attention( $\mathbf{DW}_Q$ ,  $\mathbf{EW}_K$ ,  $\mathbf{EW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

- Learnable weights  $W_Q$ ,  $W_K$ ,  $W_V$  for every attention layer in both encoder and decoder.
- Multi-headed attention: multiple learnable  $\mathbf{W}$  matrices per encoder/decoder layer, all added/softmax before FF layers.

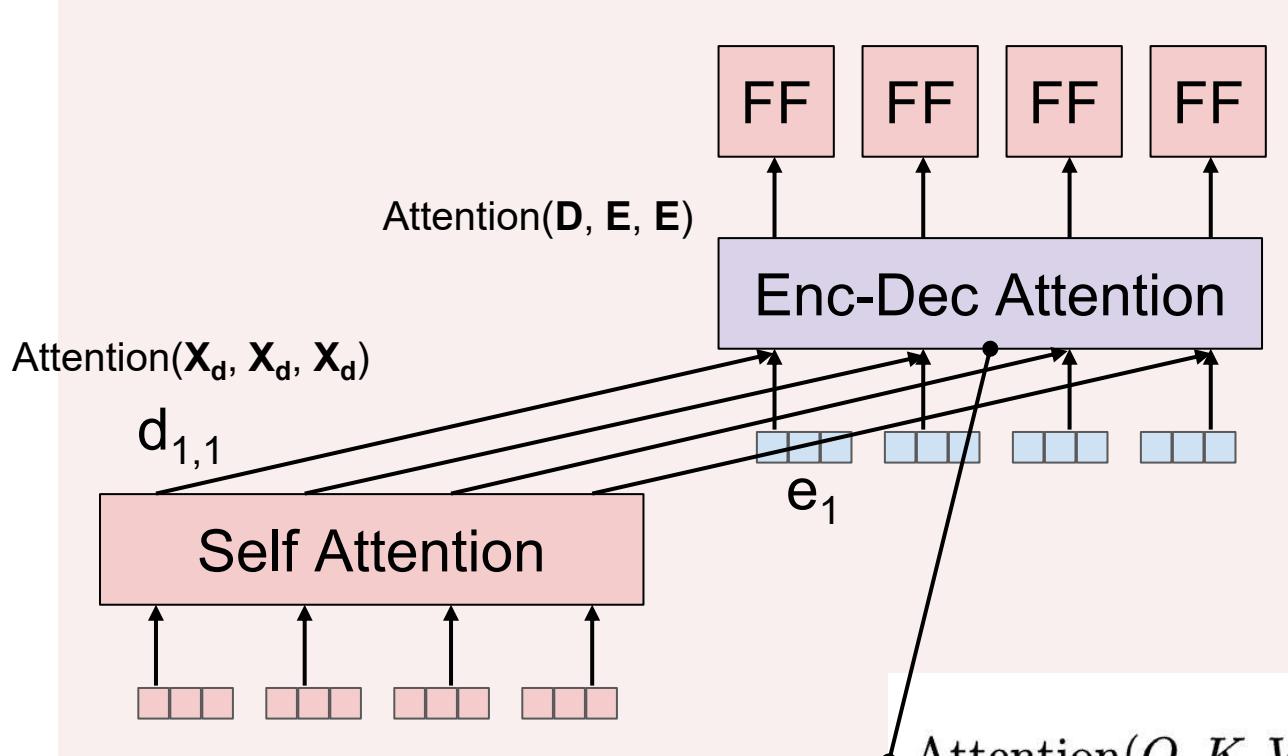
# Encoder-Decoder Attention



- Commonly: encoder's outputs form the keys and values, and decoder self-attn outputs are queries.
- e.g.,  $q_1=d_{1,1}W_Q$ ;  
 $k_1=e_1W_K; v_1=e_1W_V$

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

# Encoder-Decoder Attention

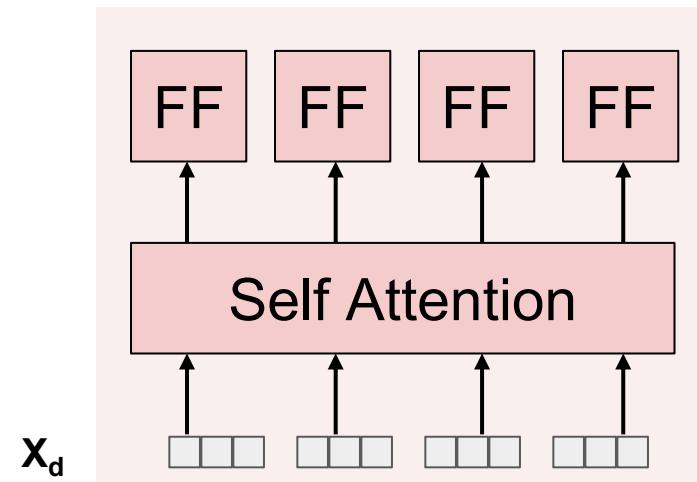
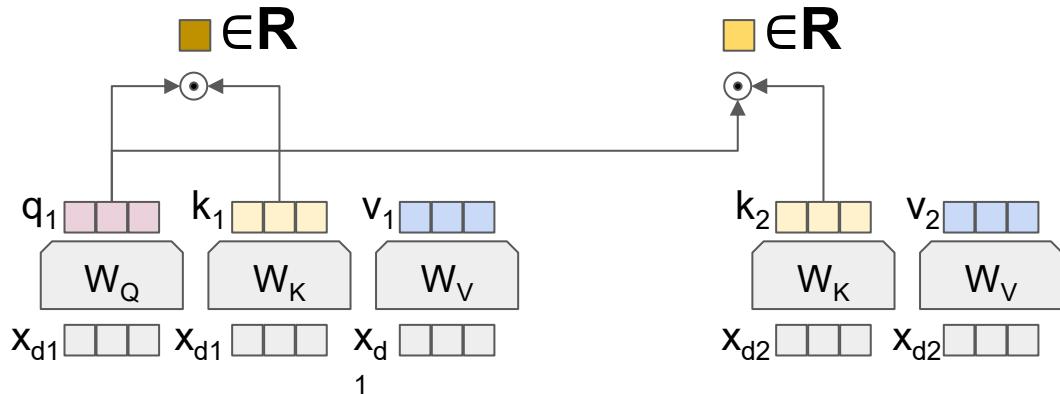


- So Q queries allow relations between dec target and enc rep
- Q from dec run against enc keys K and values V

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

# Encoder-Decoder Attention

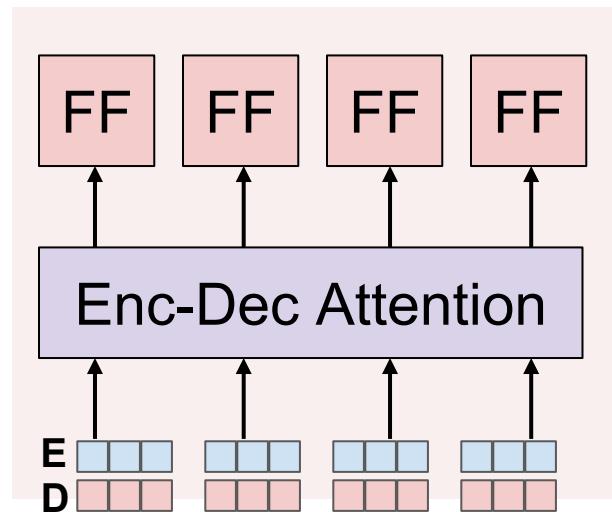
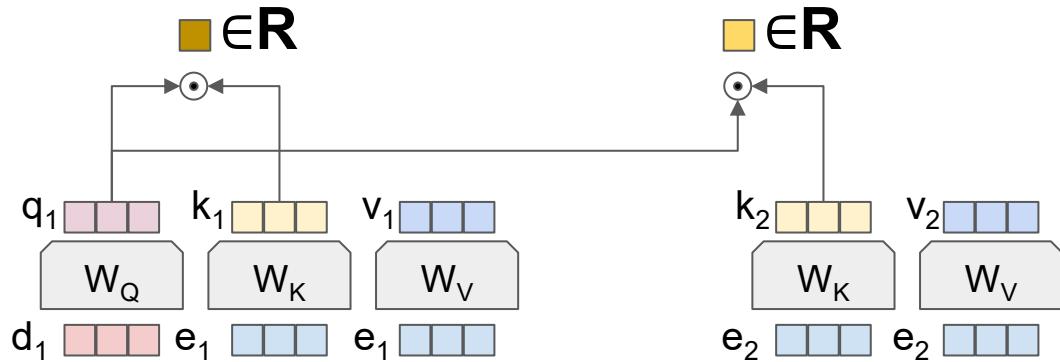
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$



# Encoder-Decoder Attention

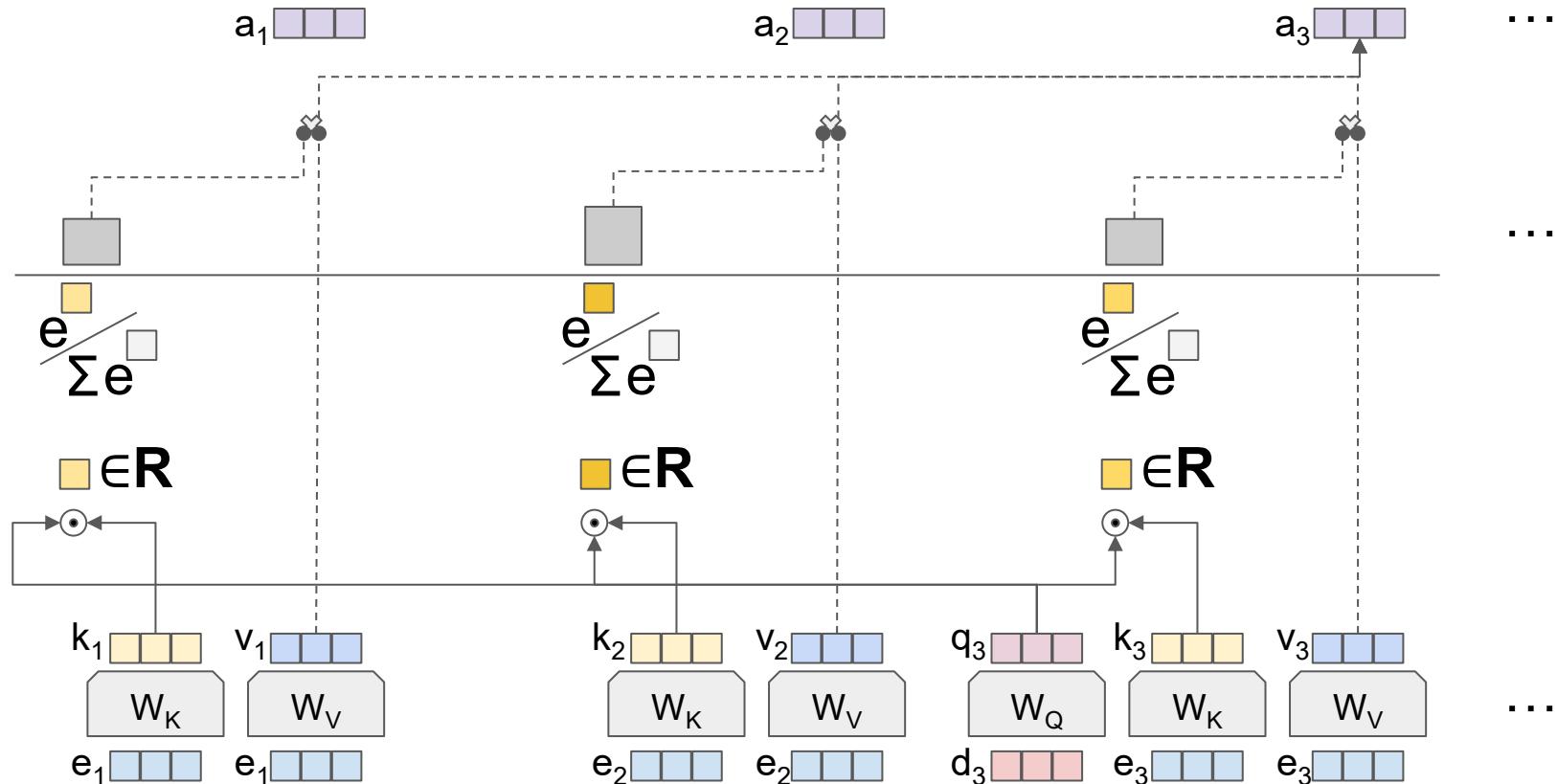
Attention( $\mathbf{DW}_Q, \mathbf{EW}_K, \mathbf{EW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

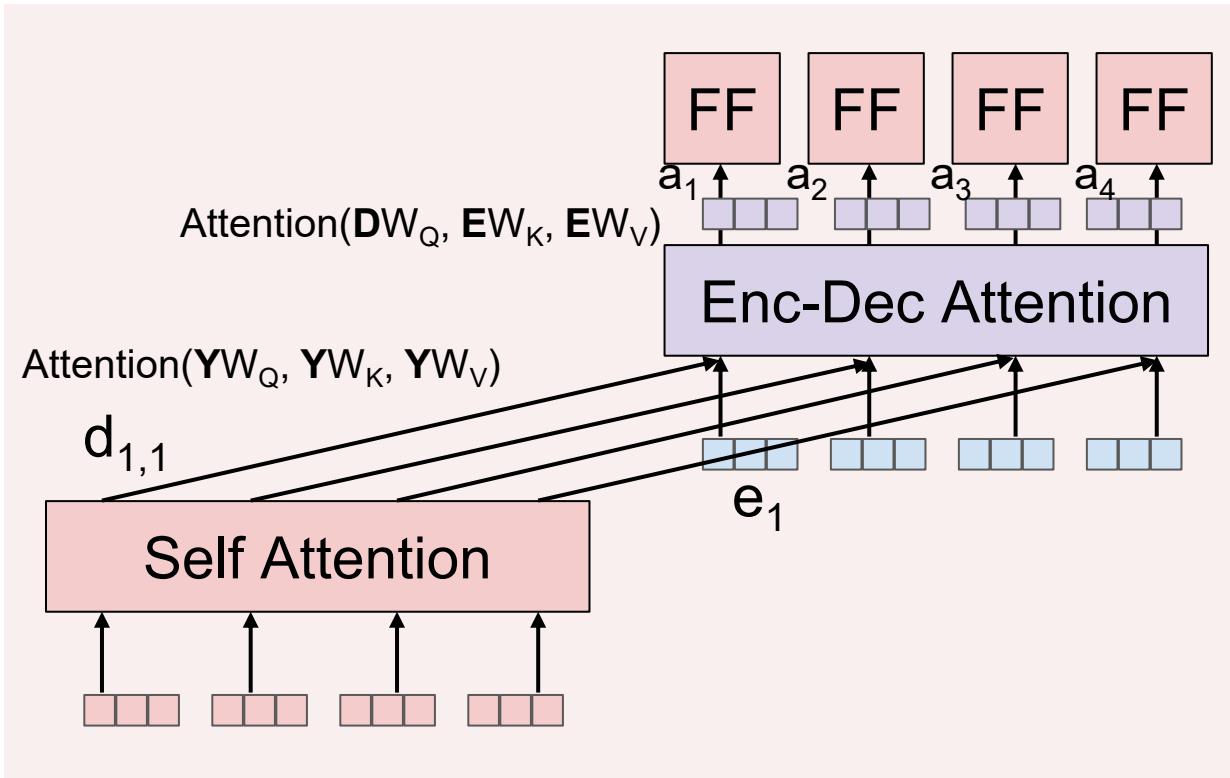


$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

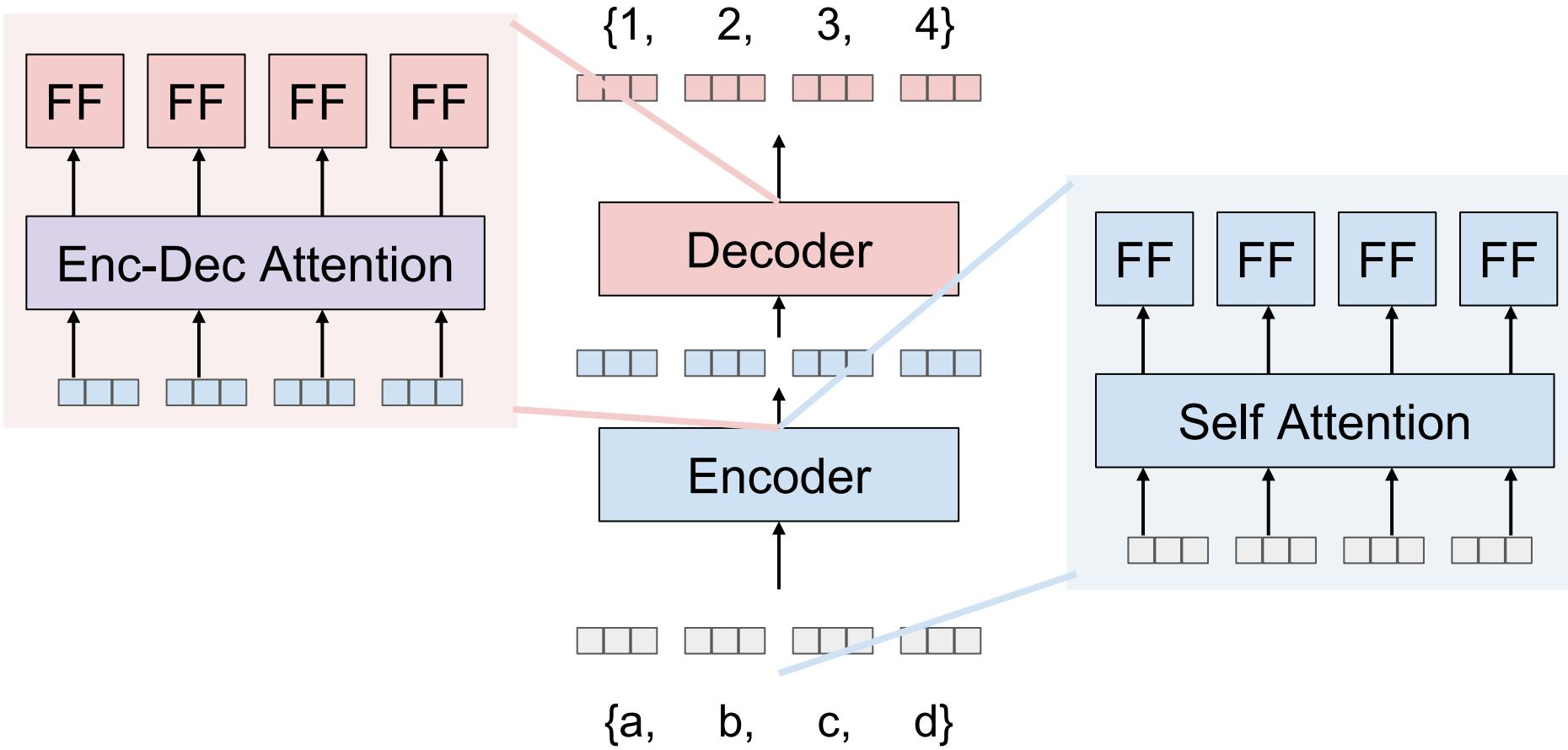
# Encoder-Decoder Attention



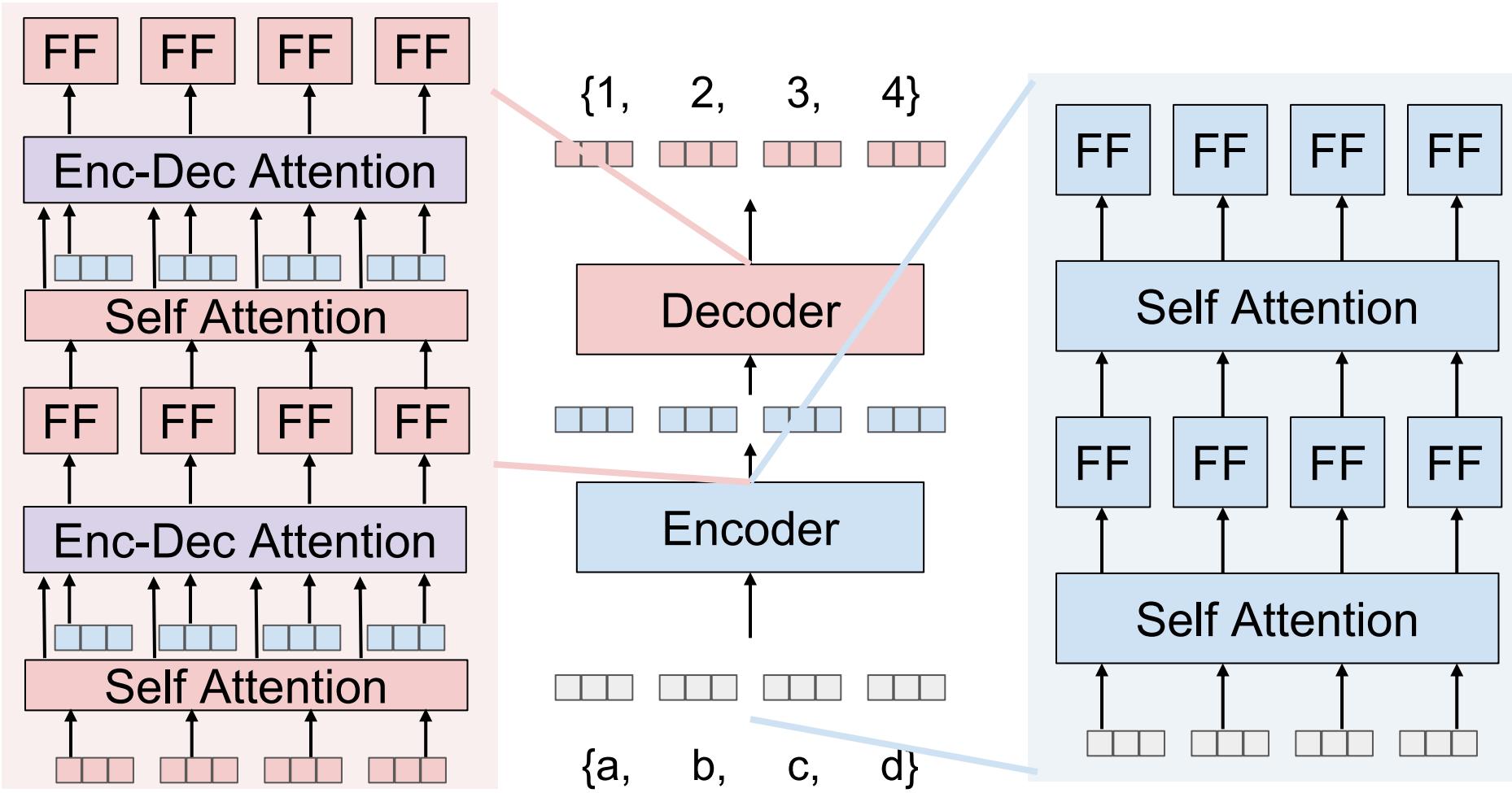
# Encoder-Decoder Attention



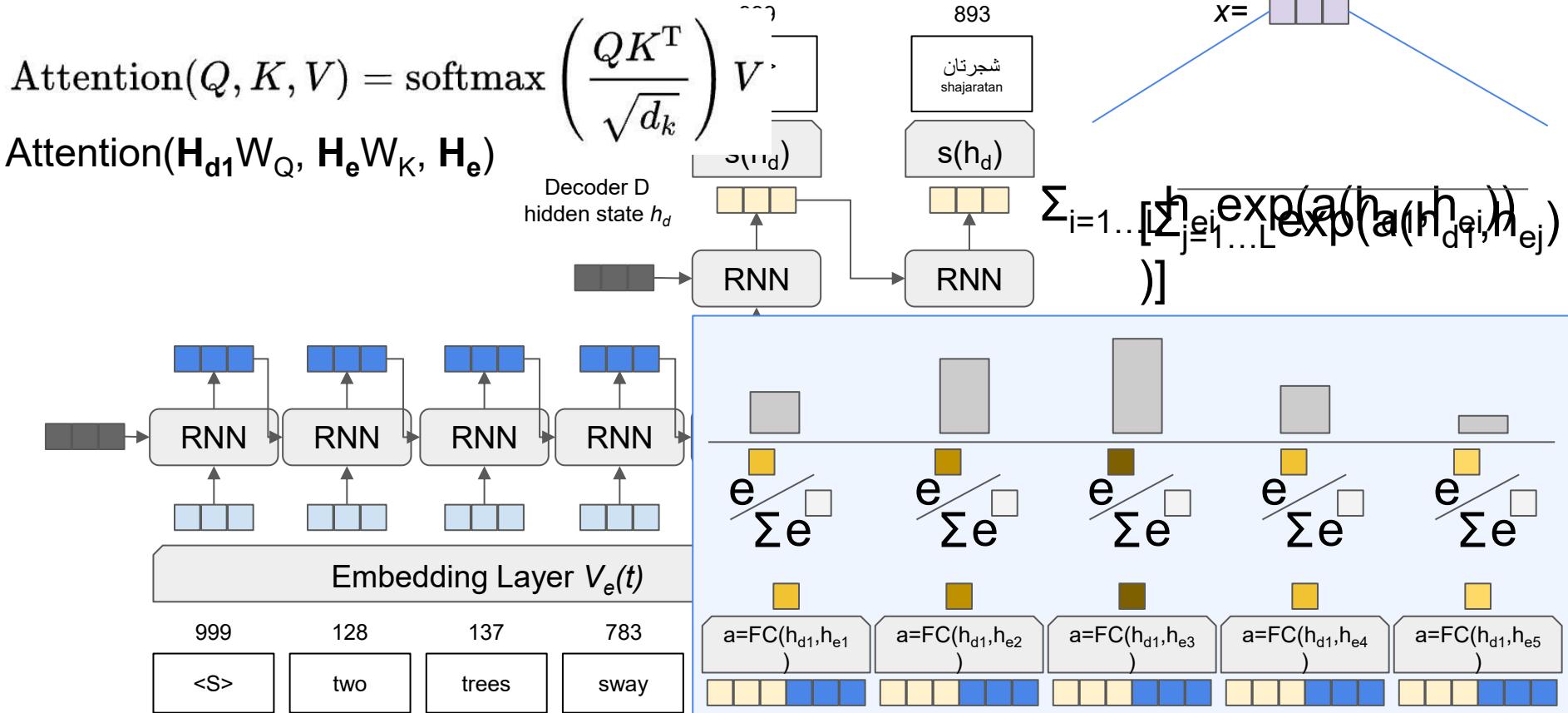
# Transformers Lite



# Deeper Encoders / Decoders



# Relationship to *Global Soft Attention*



# Self-Attention Intuitions

- Transformer Encoder Self Attention:  $\text{Attention}(\mathbf{XW}_Q, \mathbf{XW}_K, \mathbf{XW}_V)$ 
  - How the words in the input sequence relate to one another
  - Query how each word relates to the Keys of other words in the set (and itself), to get weighted Value of each words' representation with respect to others in the sequence
- Key takeaway: in self-attention, we encode the entire set of tokens with respect to one another, ending with a size L set of self-conditioned embeddings

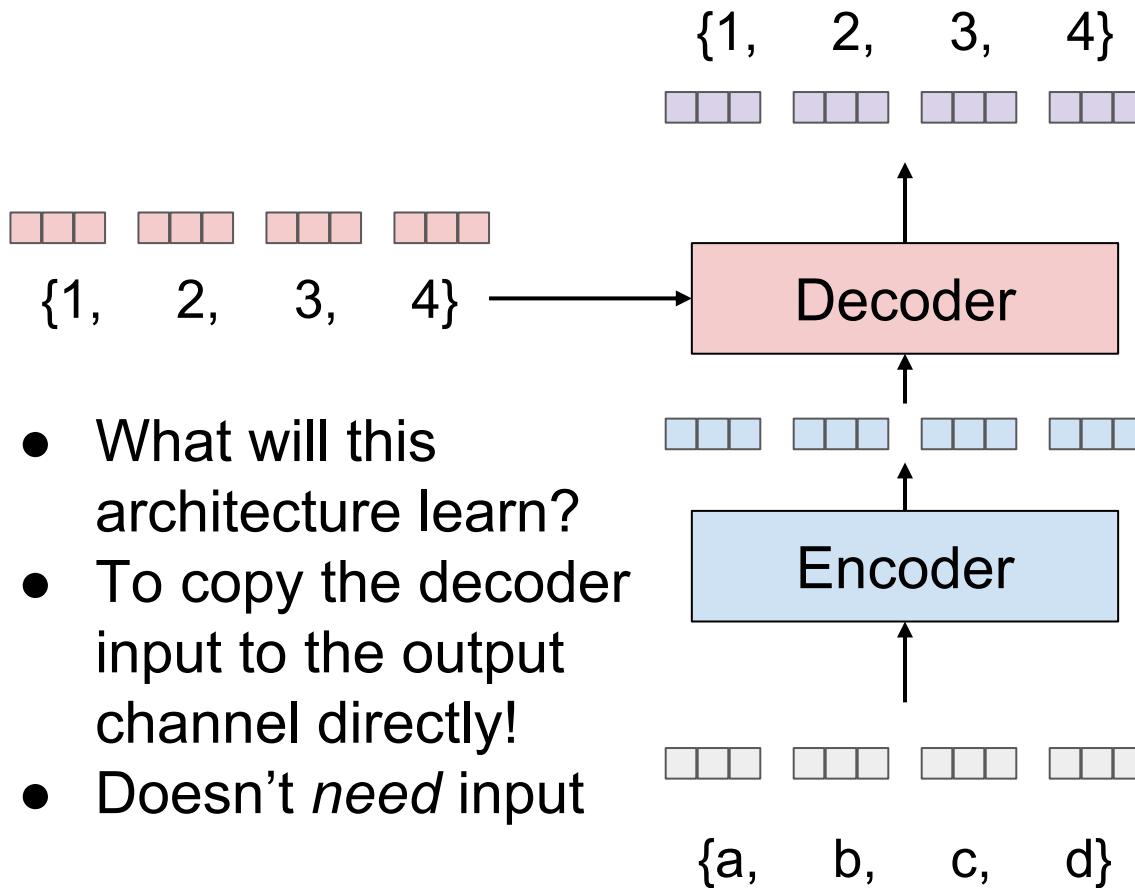
# Self-Attention Intuitions

- Transformer Decoder Self Attention:  $\text{Attention}(\mathbf{Y}\mathbf{W}_Q, \mathbf{Y}\mathbf{W}_K, \mathbf{Y}\mathbf{W}_V)$ 
  - How the words in the output sequence relate to one another
  - Query how each word relates to the Keys of other words in the set (and itself), to get weighted Value of each words' representation with respect to others in the sequence
- Key takeaway: In a Transformer, we have access to the input and output sets at encoding time, so the decoder self-attention can create a latent embedding too

# Lecture Outline

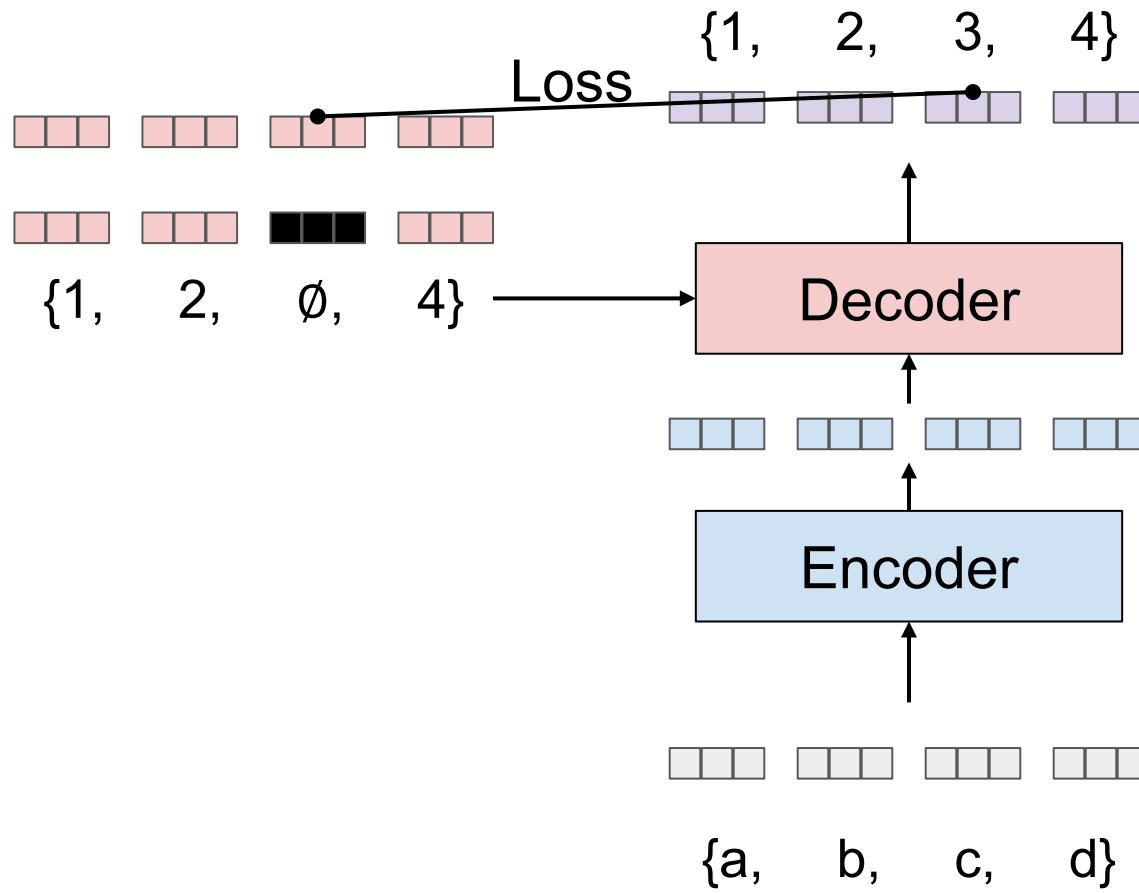
- Language Model Decoding
- Encoder-Decoder Attention Recap
- Transformer Networks and the Attention Mechanism
- **Using Transformers for Language Modeling**
- Zoom Out

# How Would We Train a Transformer?

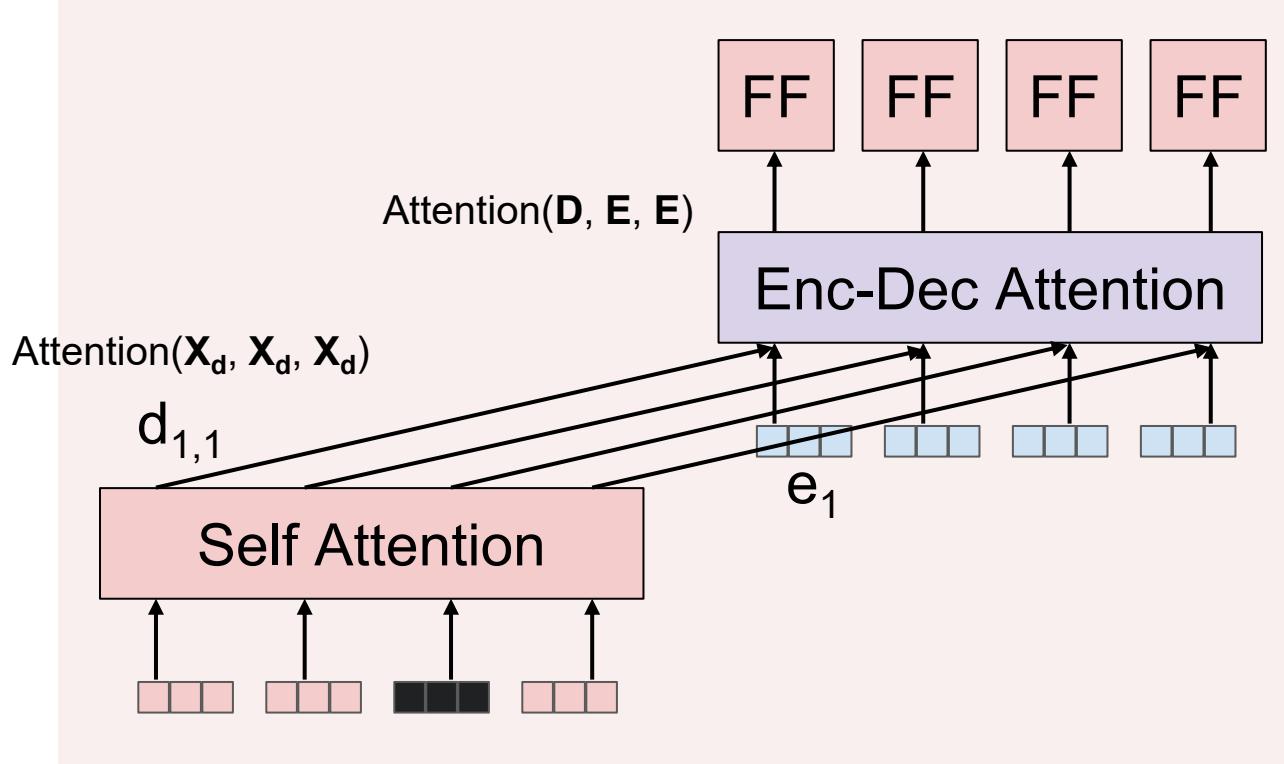


- Say we want to learn something really easy
- $a \rightarrow 1, b \rightarrow 2, \dots$
- Inputs: sets of letters
- Outputs: sets of numbers

# Supervision: Masking

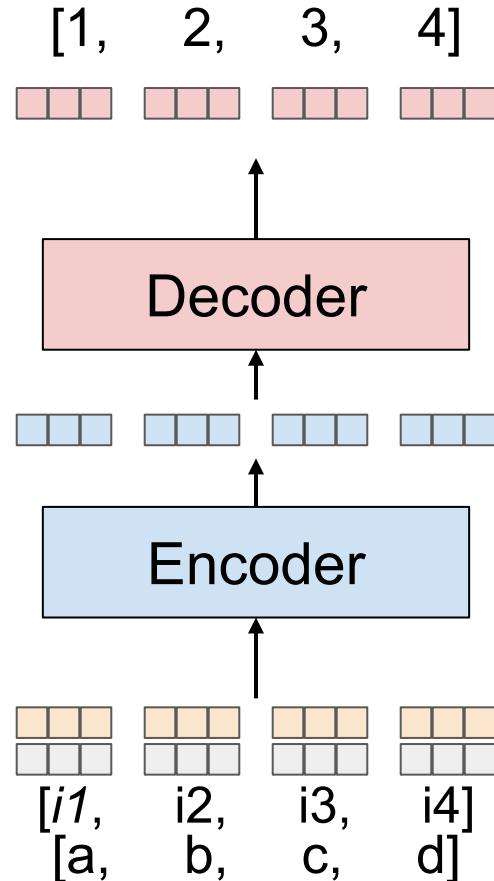


# Supervision: Masking



- Learn to *recover* missing elements of the target output sequence conditioned on information from the input.

# Sequences



- What if we want our input/output to be *sequences*?
- E.g., sentences!
- Prev:
  - $\{a, c, d\}$   
→  $\{4, 3, 1\}$ ,
- Want:
  - $[a, c, d]$   
→  $[1, 3, 4]$

# Sequences: What is a Positional Encoding?

[1, 2, 3, 4]

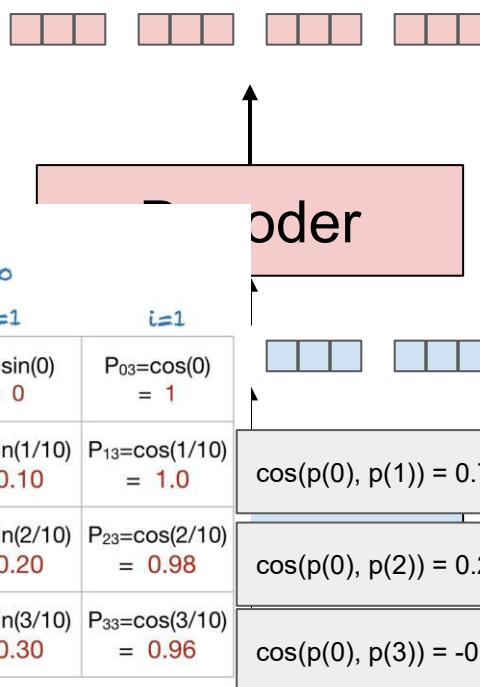
$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i + 1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Sequence	Index of token, k	Positional Encoding Matrix with d=4, n=100			
		i=0	i=0	i=1	i=1
I	0	P <sub>00</sub> =sin(0) = 0	P <sub>01</sub> =cos(0) = 1	P <sub>02</sub> =sin(0) = 0	P <sub>03</sub> =cos(0) = 1
am	1	P <sub>10</sub> =sin(1/1) = 0.84	P <sub>11</sub> =cos(1/1) = 0.54	P <sub>12</sub> =sin(1/10) = 0.10	P <sub>13</sub> =cos(1/10) = 1.0
a	2	P <sub>20</sub> =sin(2/1) = 0.91	P <sub>21</sub> =cos(2/1) = -0.42	P <sub>22</sub> =sin(2/10) = 0.20	P <sub>23</sub> =cos(2/10) = 0.98
Robot	3	P <sub>30</sub> =sin(3/1) = 0.14	P <sub>31</sub> =cos(3/1) = -0.99	P <sub>32</sub> =sin(3/10) = 0.30	P <sub>33</sub> =cos(3/10) = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

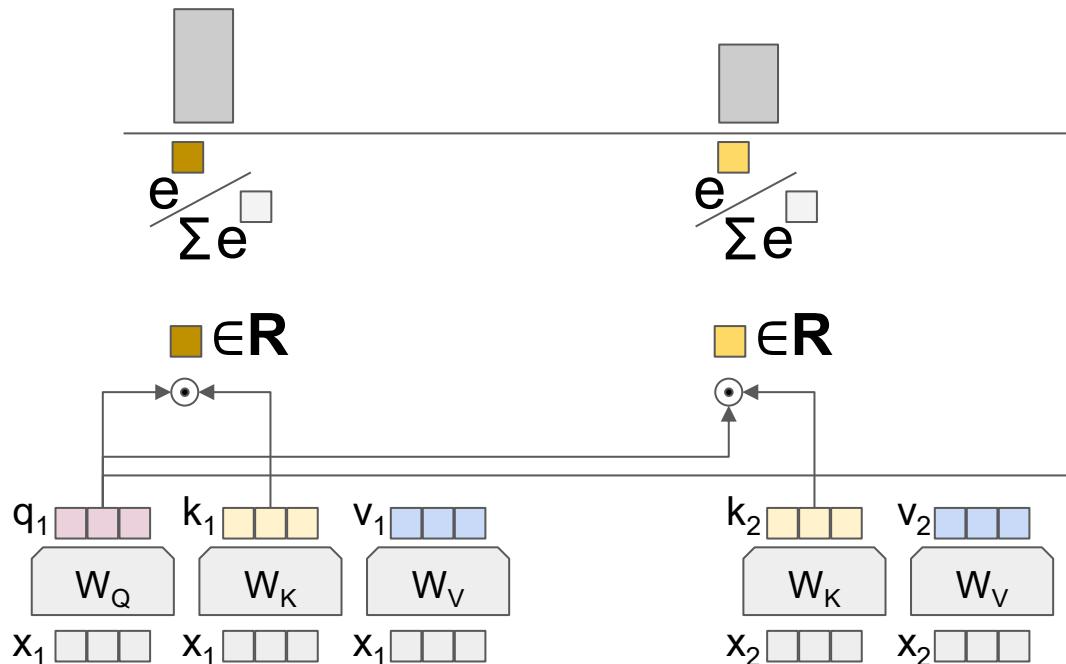
i3, i4]



- Creating a consistent “indicator vector” for each position
- “Nearness” between adjacent positions preserved by encoding functions.

# Positional Encodings

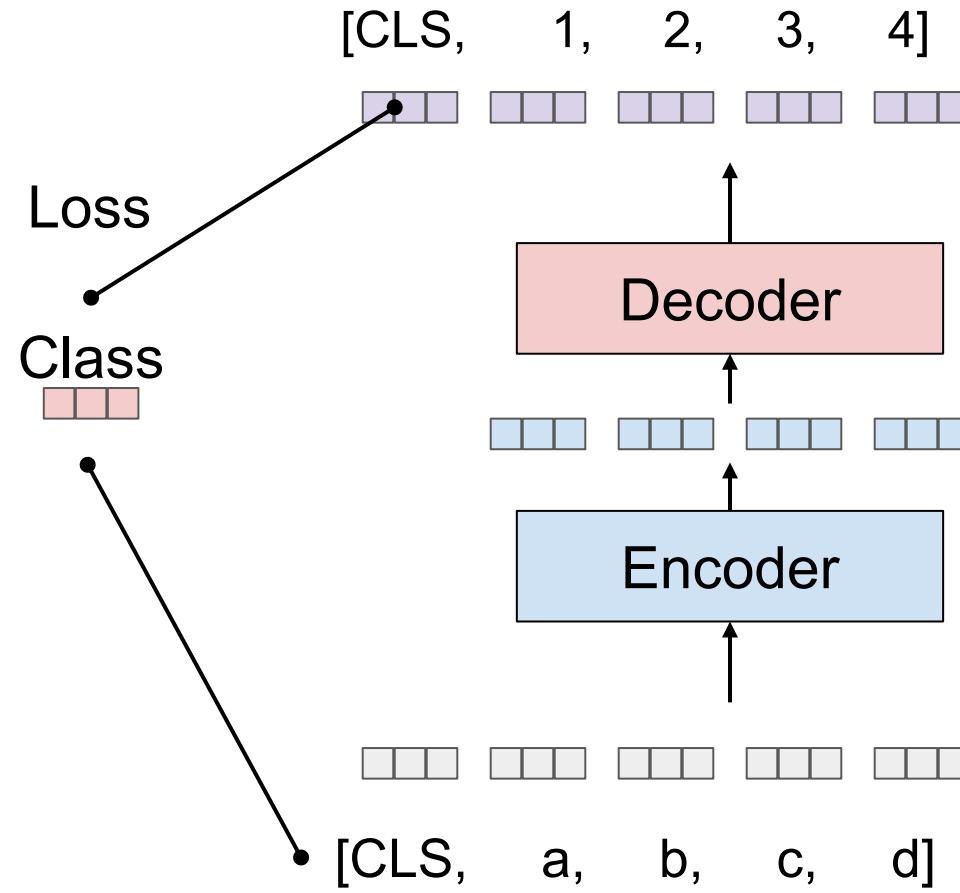
$$x_1 \quad \square \quad \square \quad = e_{x1} \quad \square \quad \square + p_1 \quad \square \quad \square$$



- **Assume:**  $W_Q$  and  $W_K$  matrix transformations preserve distance.
- Then in general:
  - $q_1 * k_2 > q_1 * k_3$
- Positional encodings *bias* the attention mechanism to give more weight to adjacent words without respect to the words themselves

$x_3 \quad \square \quad \square \quad \square$     $x_3 \quad \square \quad \square \quad \square$

# Supervision: Classification

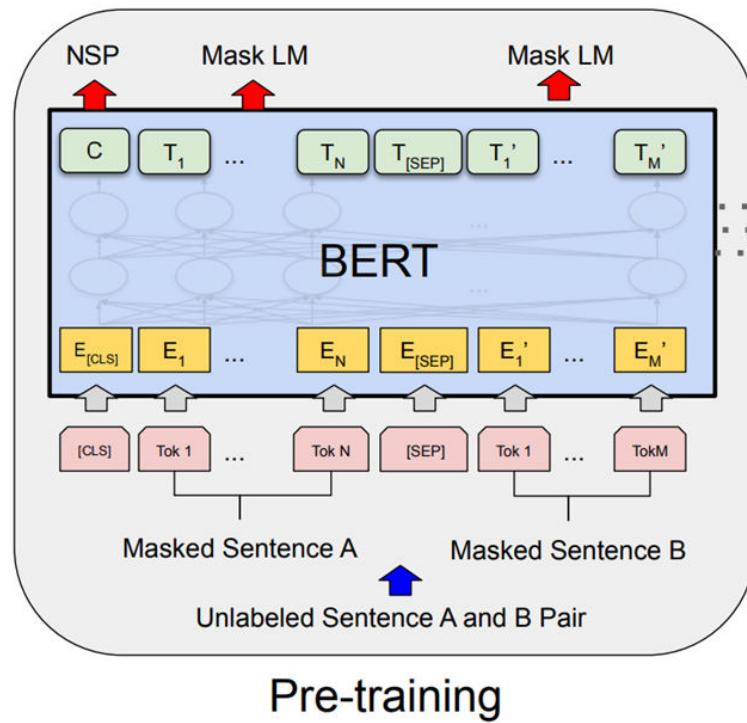


# Bidirectional Encoder Representations from Transformers

Input

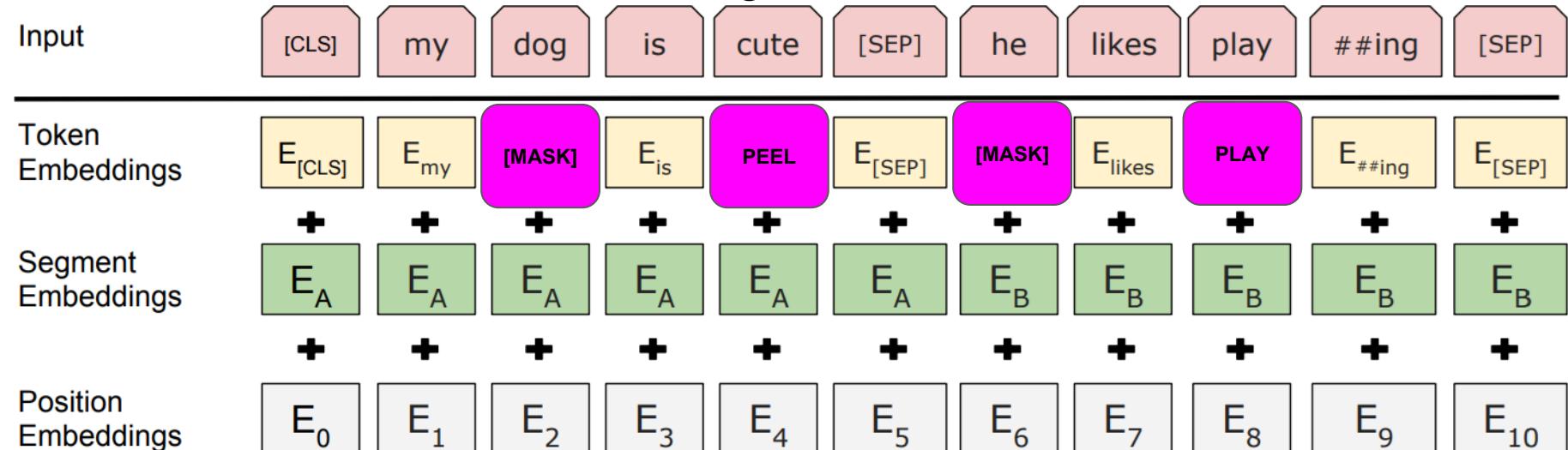


# Bidirectional Encoder Representations from Transformers



# Masked Language Modeling Pretraining

15% of words selected for training

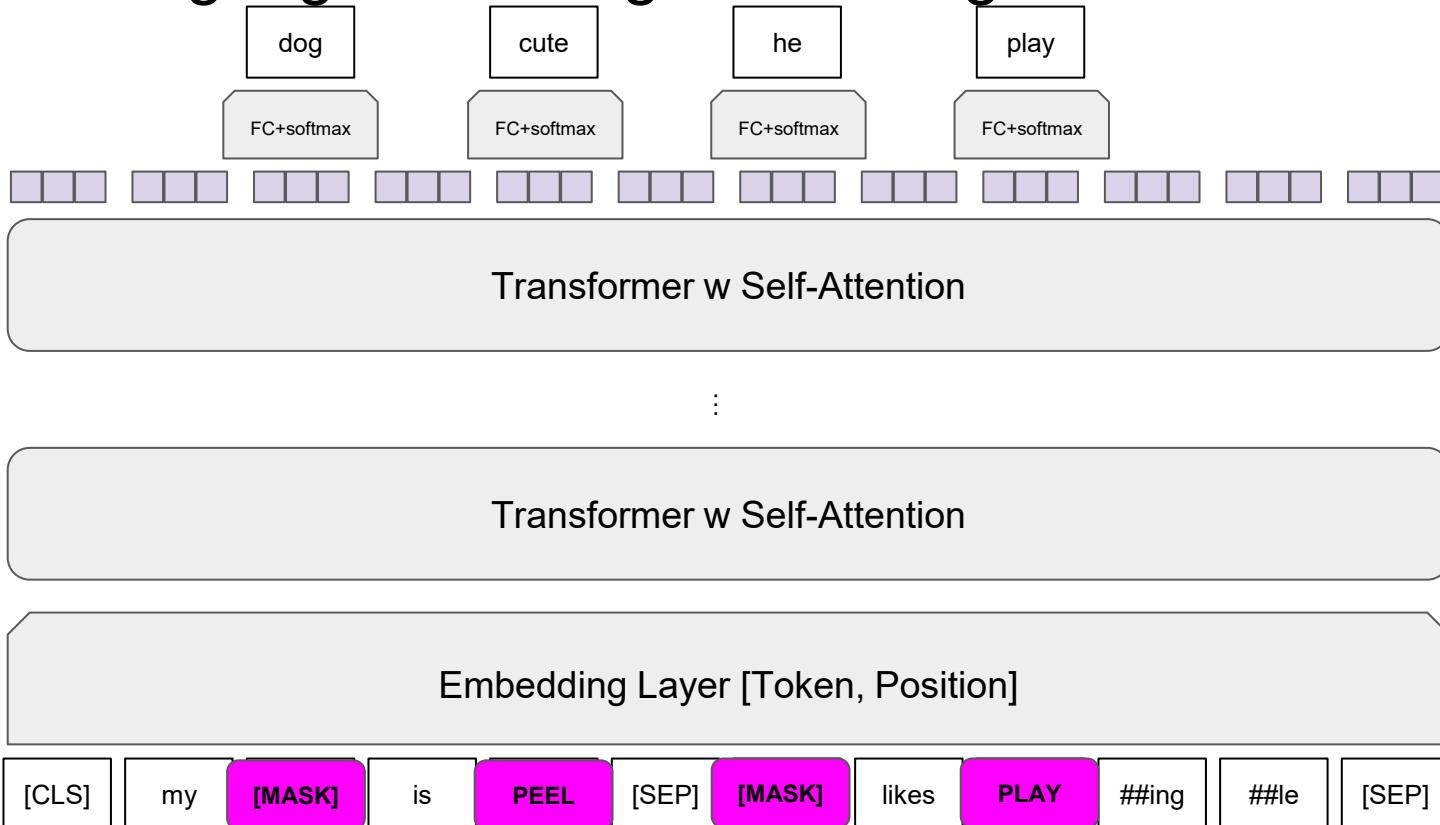


80% of those [MASK]

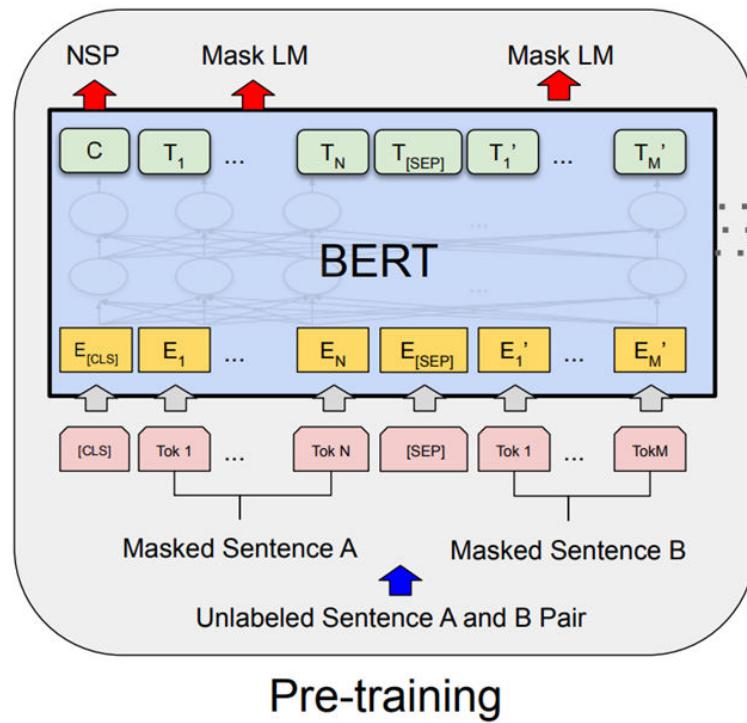
10% random token

10% original token

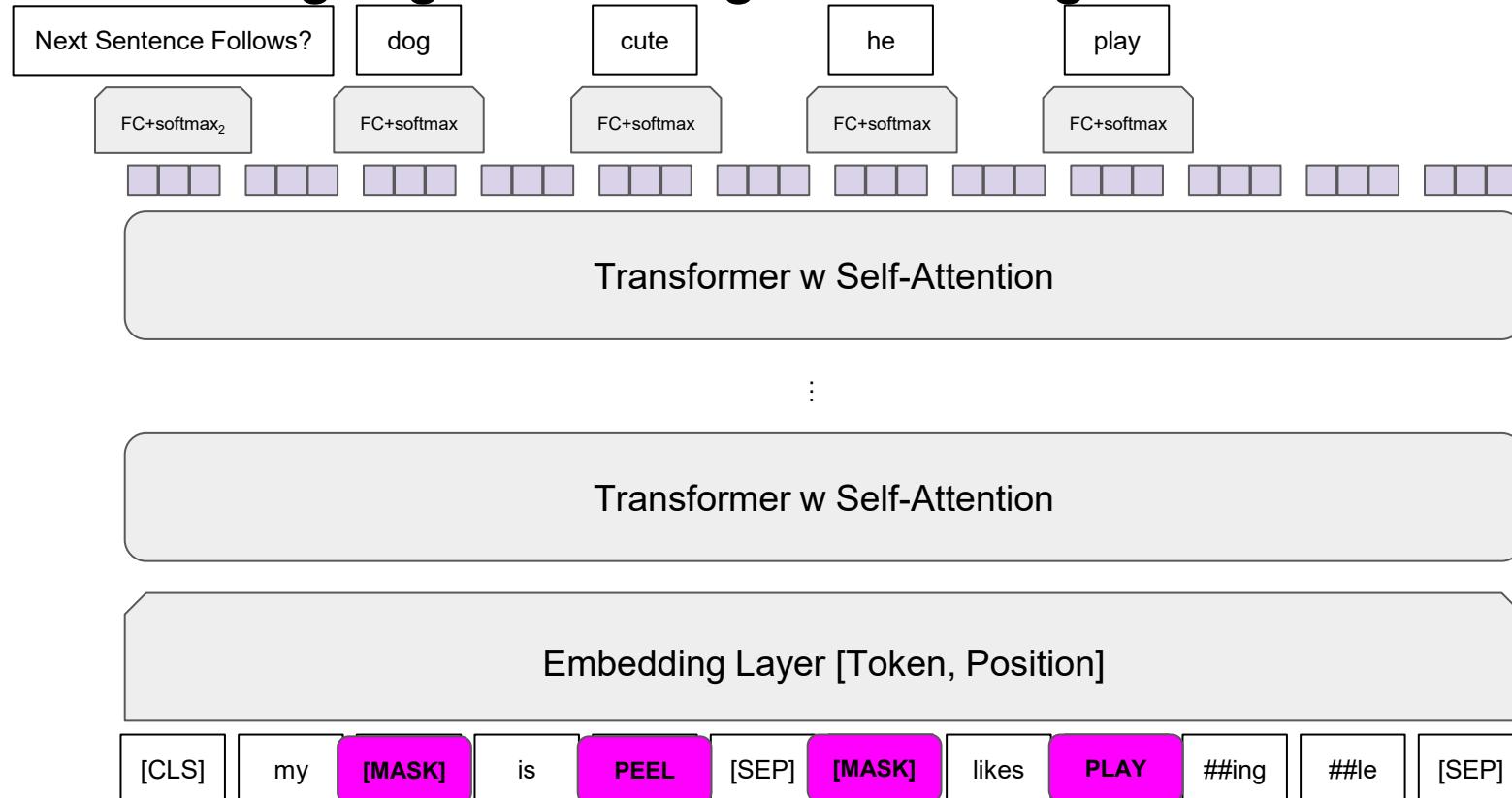
# Masked Language Modeling Pretraining



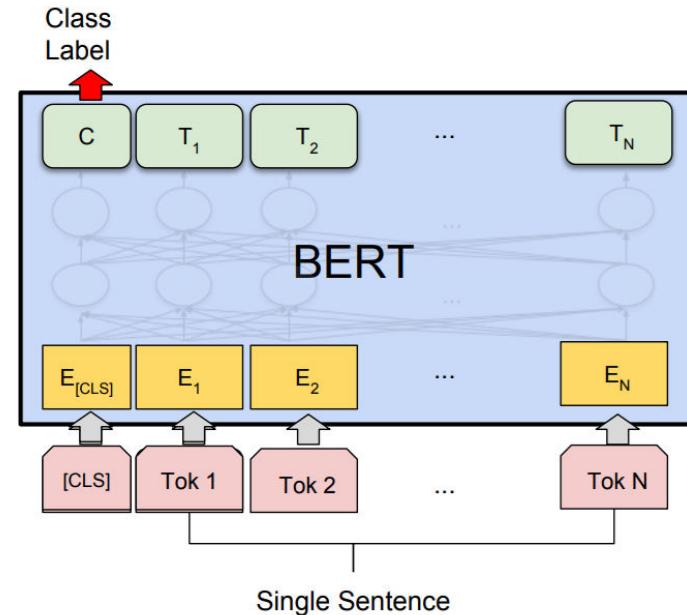
# Bidirectional Encoder Representations from Transformers



# Masked Language Modeling Pretraining

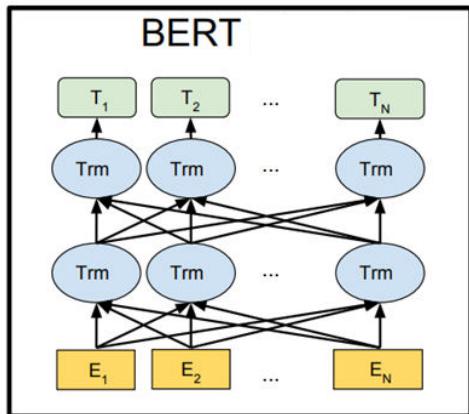


# Bidirectional Encoder Representations from Transformers

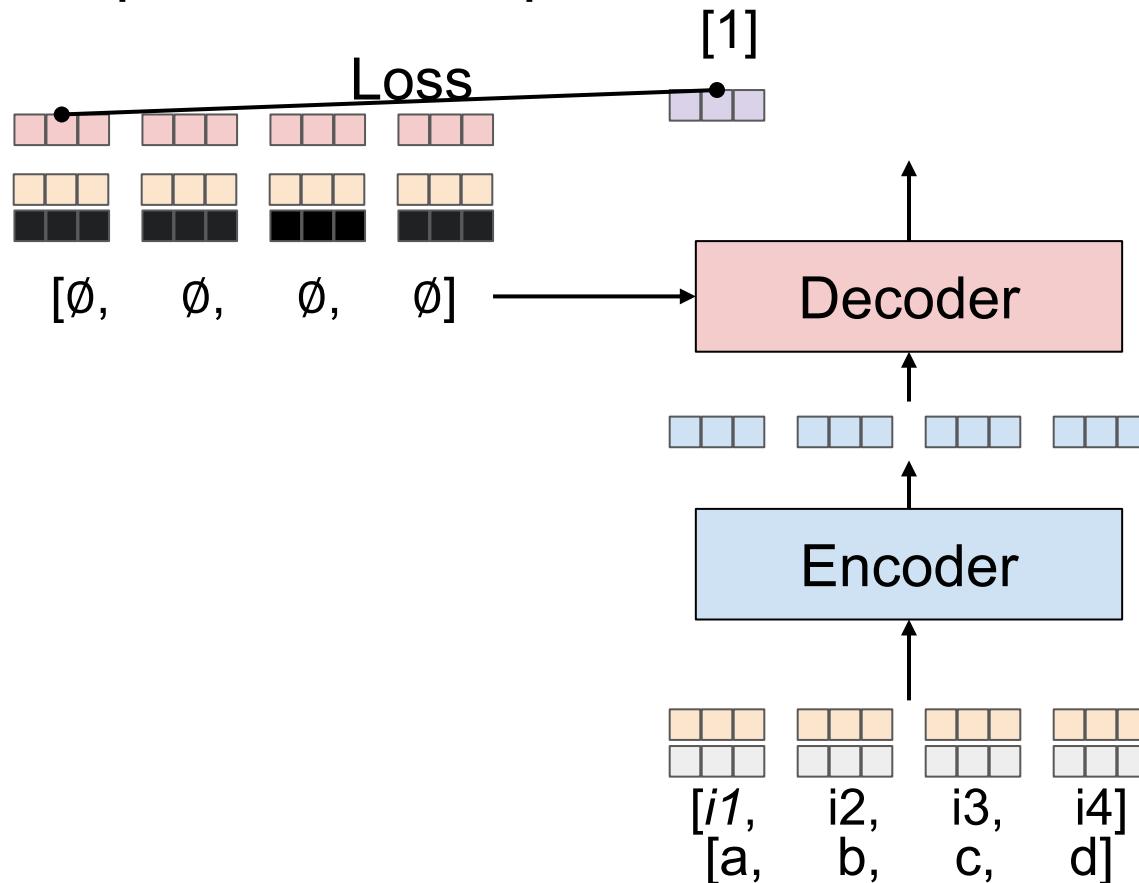


(b) Single Sentence Classification Tasks:  
SST-2, CoLA

# Bidirectional versus Autoregressive

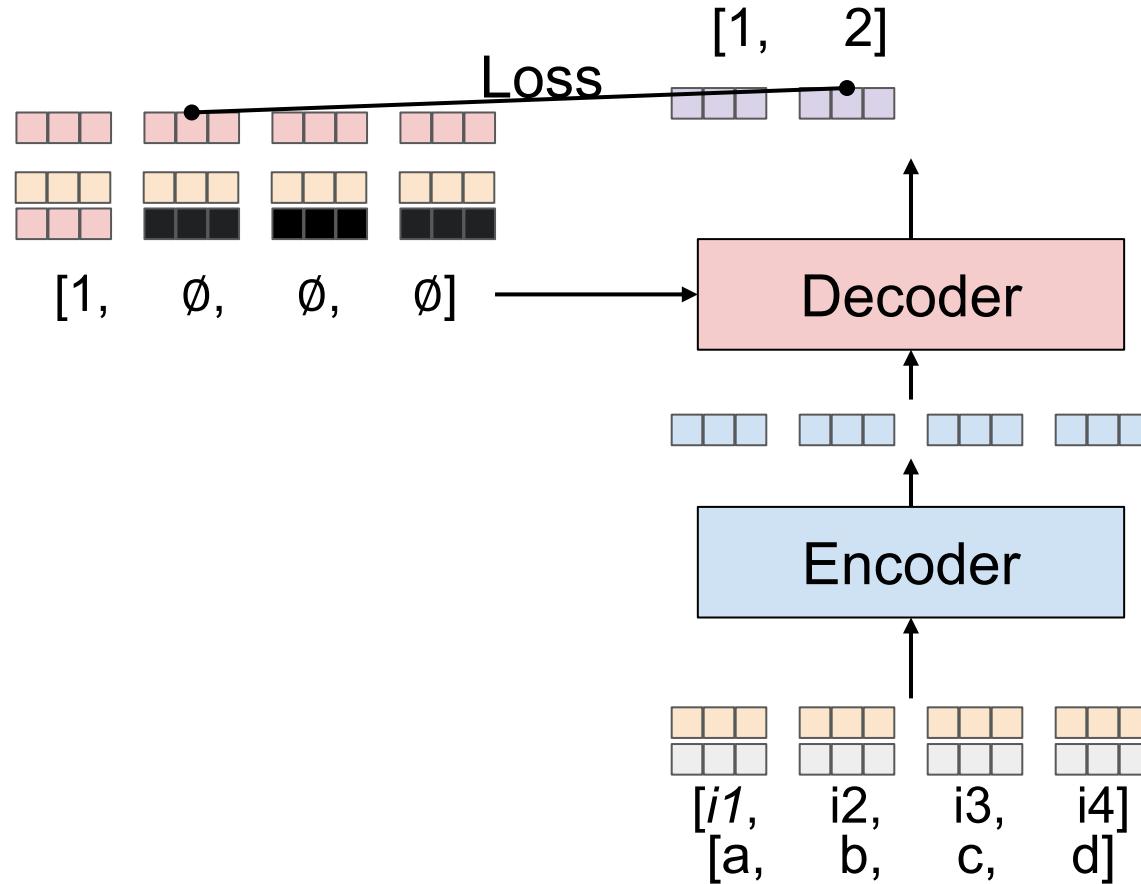


# Supervision: Sequences

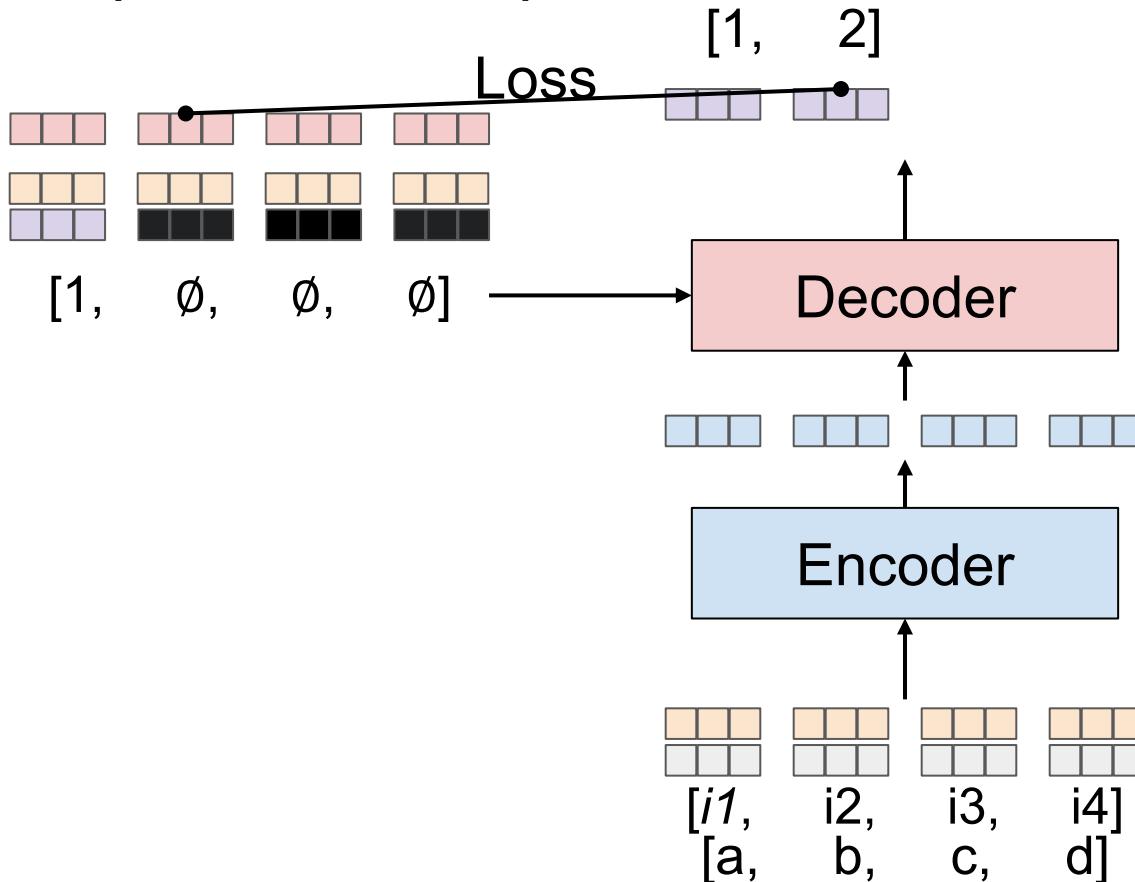


- Left-to-right masking of decoder input
- On each run, predict the next token.
- Need position information for decoder inputs as well.

# Supervision: Sequences

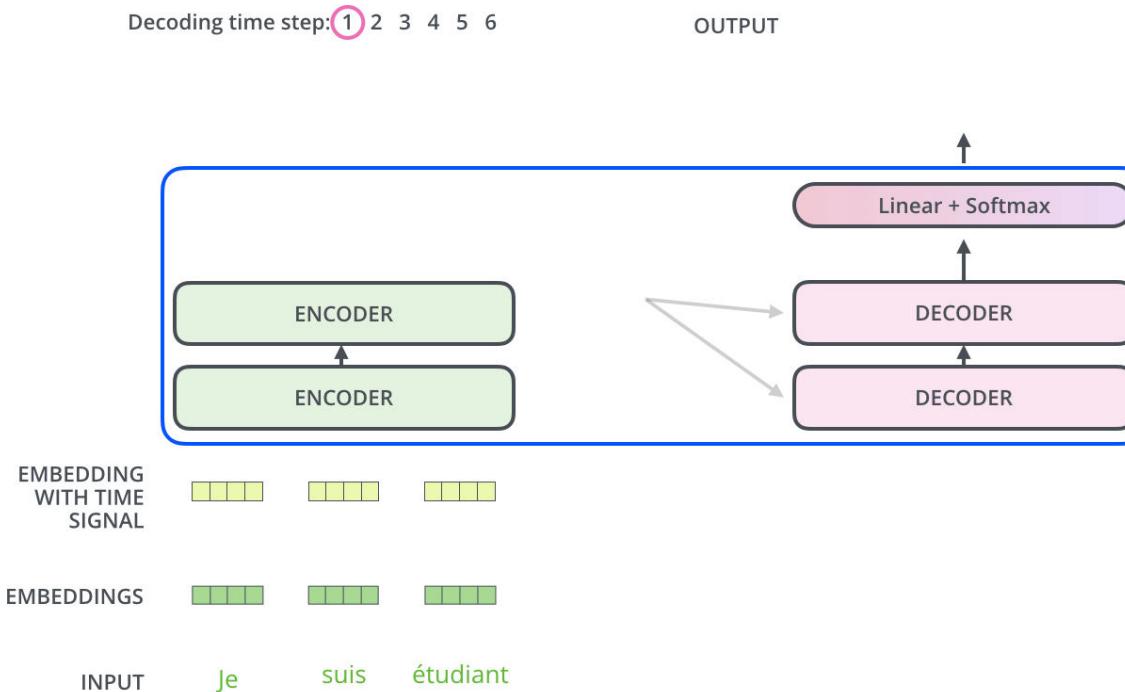


# Supervision: Sequences

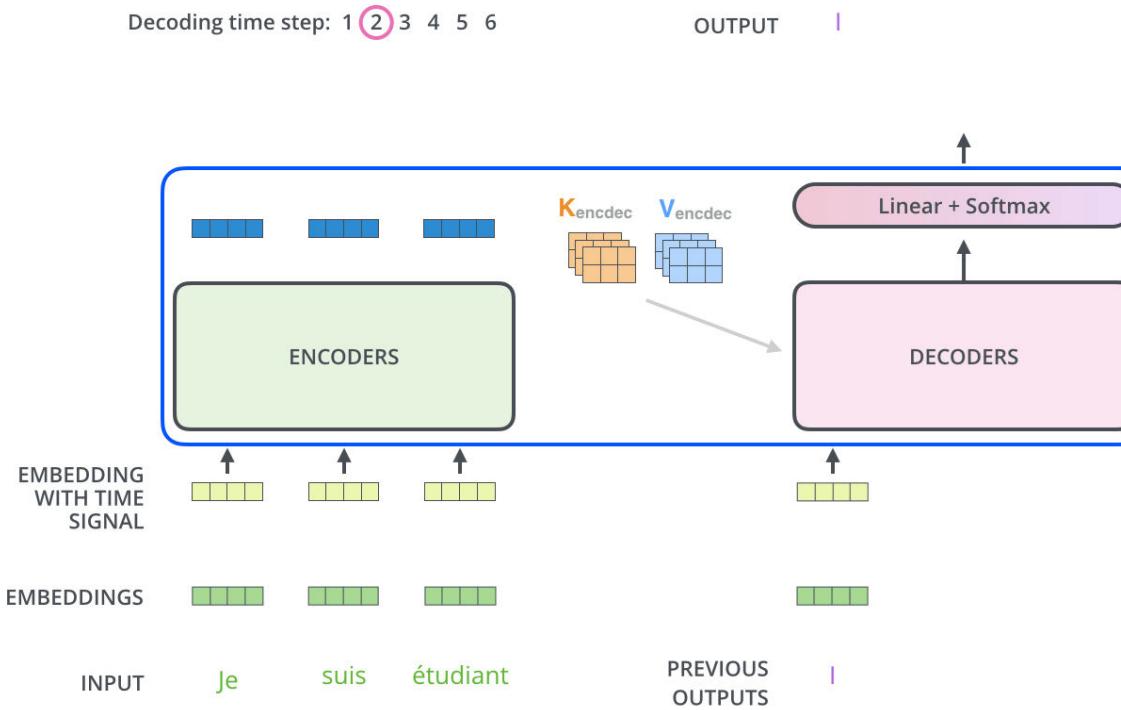


- In practice, decoder inputs will be previous timestep outputs.
- Related to general language modeling questions for decoding: sampling, beam search, etc.

# Supervision: Sequences

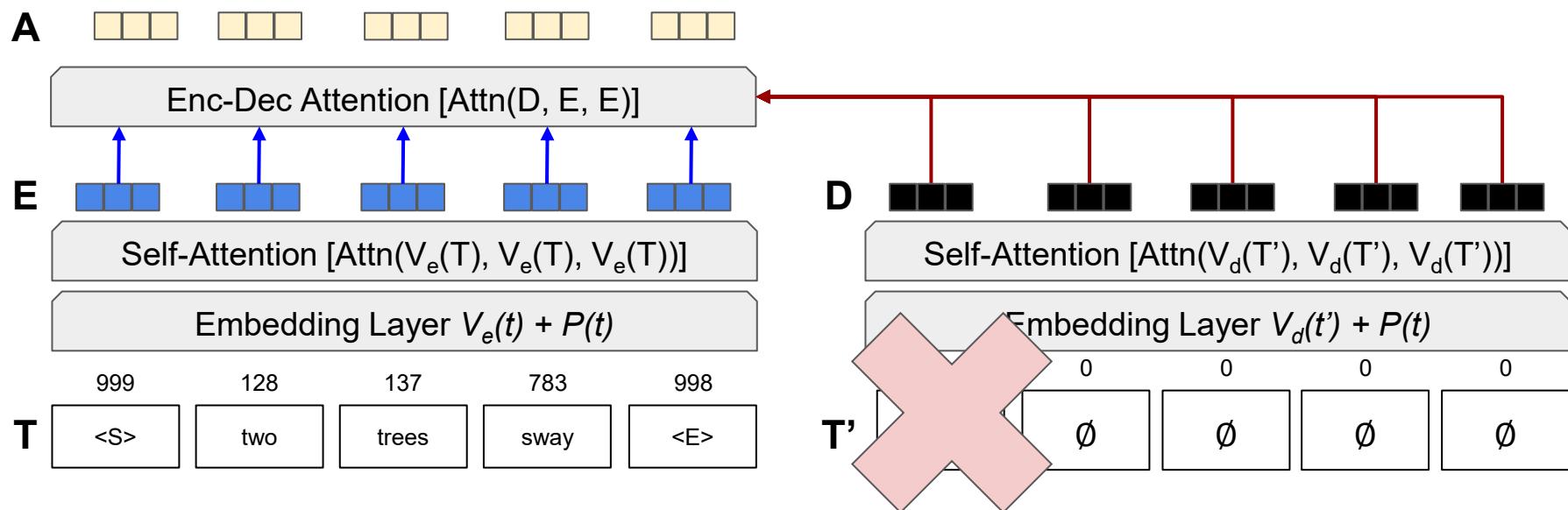


# Supervision: Sequences

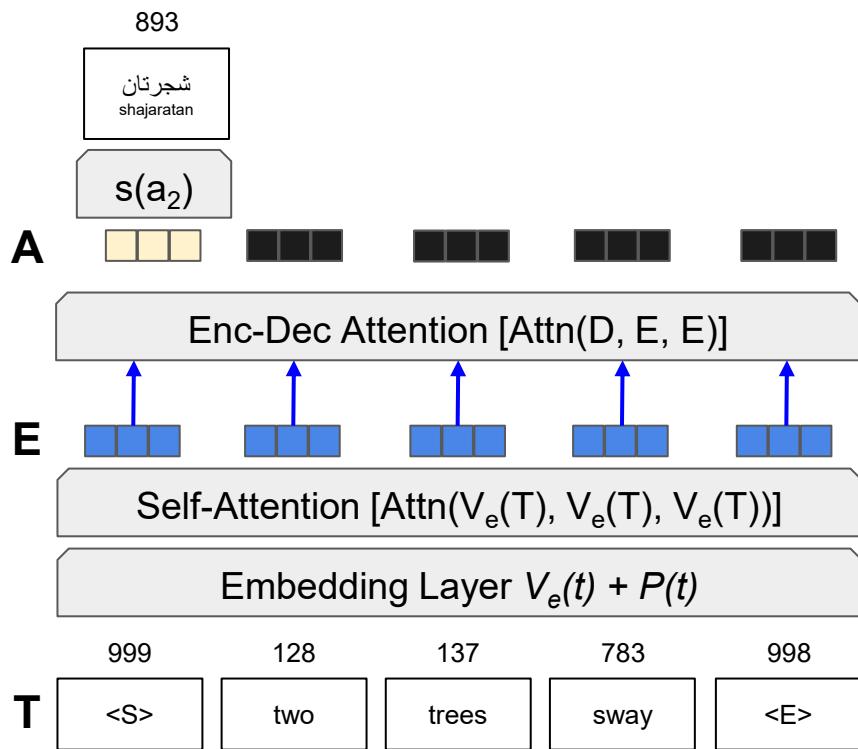


# Sequence-to-Sequence Transformer

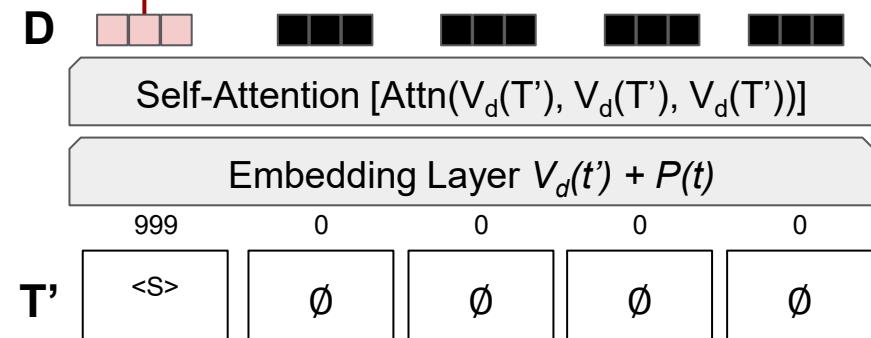
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$



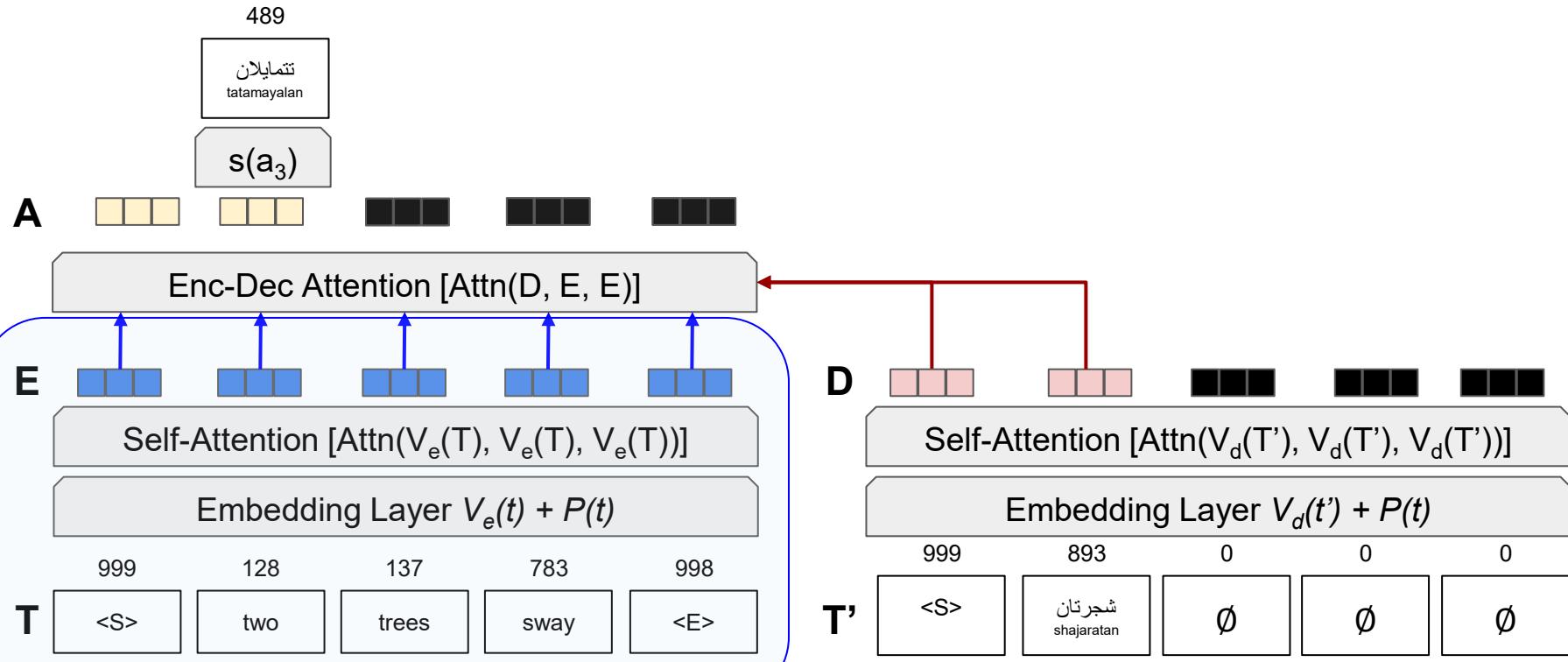
# Sequence-to-Sequence Transformer



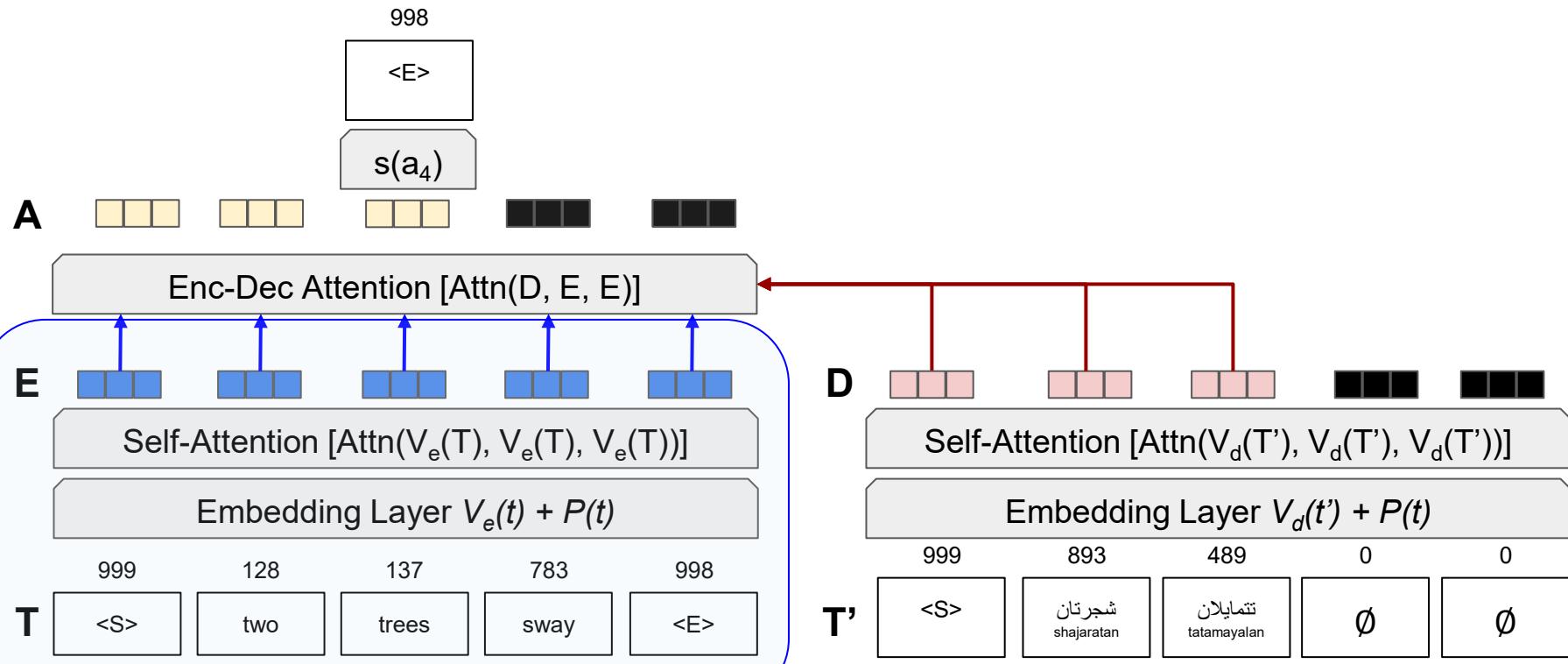
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$



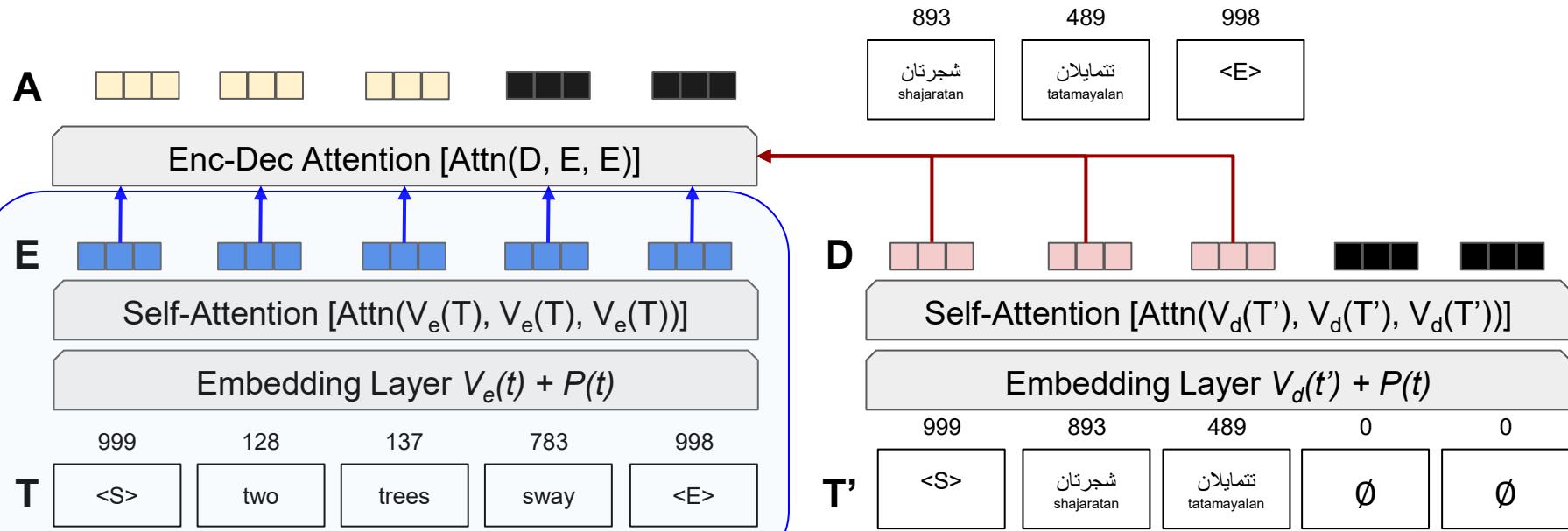
# Sequence-to-Sequence Transformer



# Sequence-to-Sequence Transformer



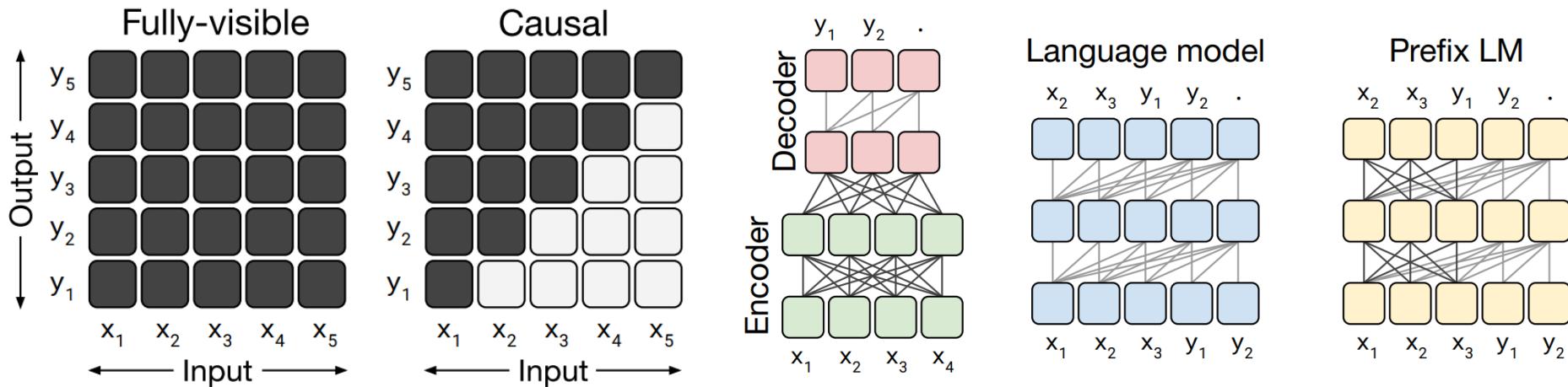
# Sequence-to-Sequence Transformer



# Lecture Outline

- Language Model Decoding
- Encoder-Decoder Attention Recap
- Transformer Networks and the Attention Mechanism
- Using Transformers for Language Modeling
- **Zoom Out**

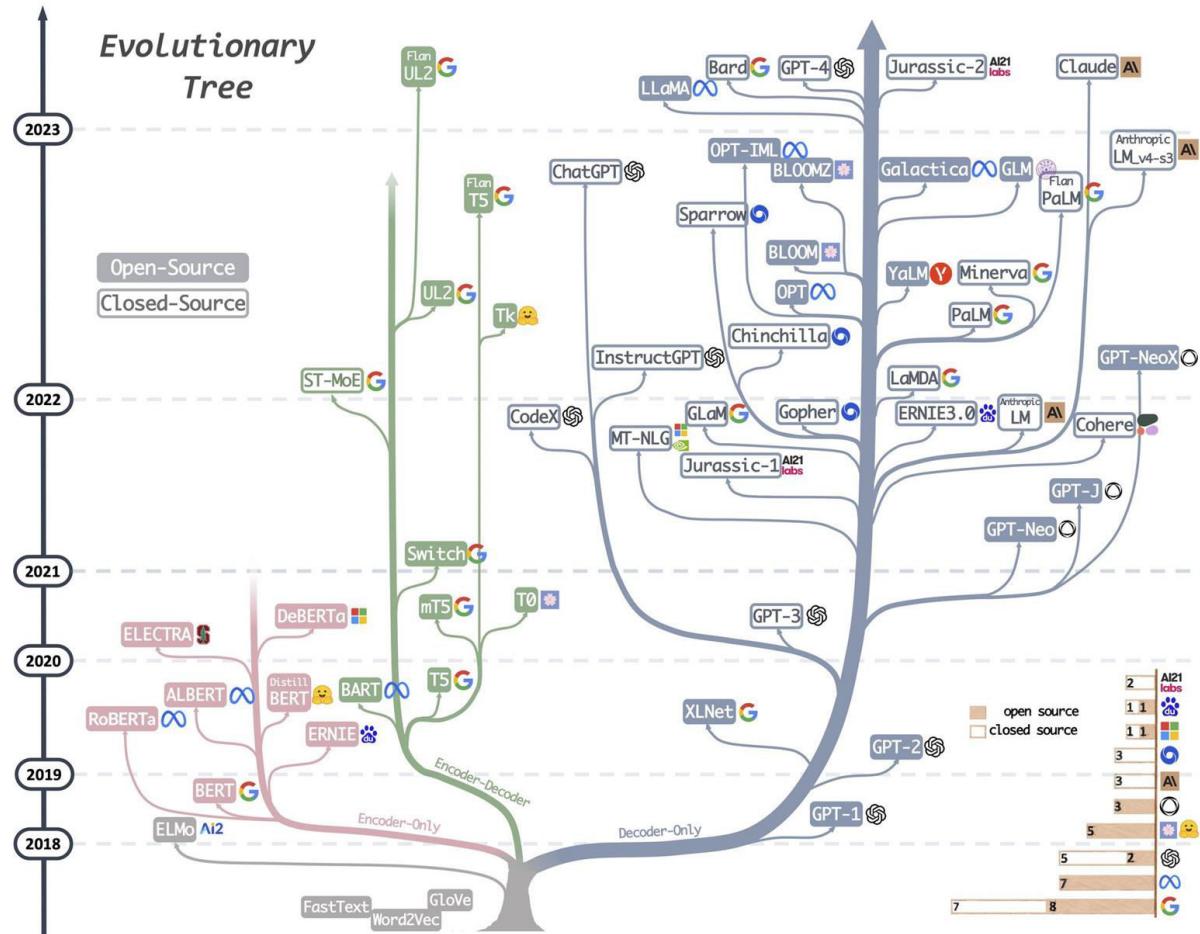
# Redundant Terminology



- Autoregressive → left-to-right; causal
- Bidirectional → fully-visible
- What differences do you see here for Enc/Dec vs LM vs Pref LM?

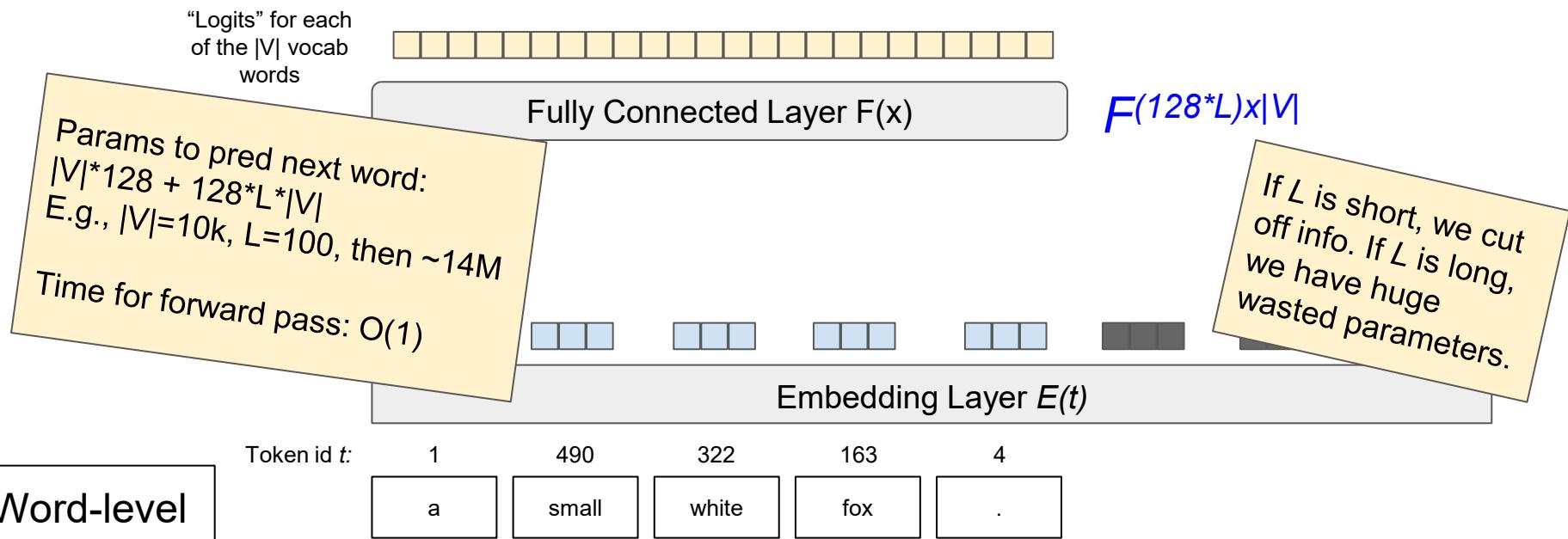
# Transformers

- Encoder-only
- Encoder-Decoder
- Decoder-only

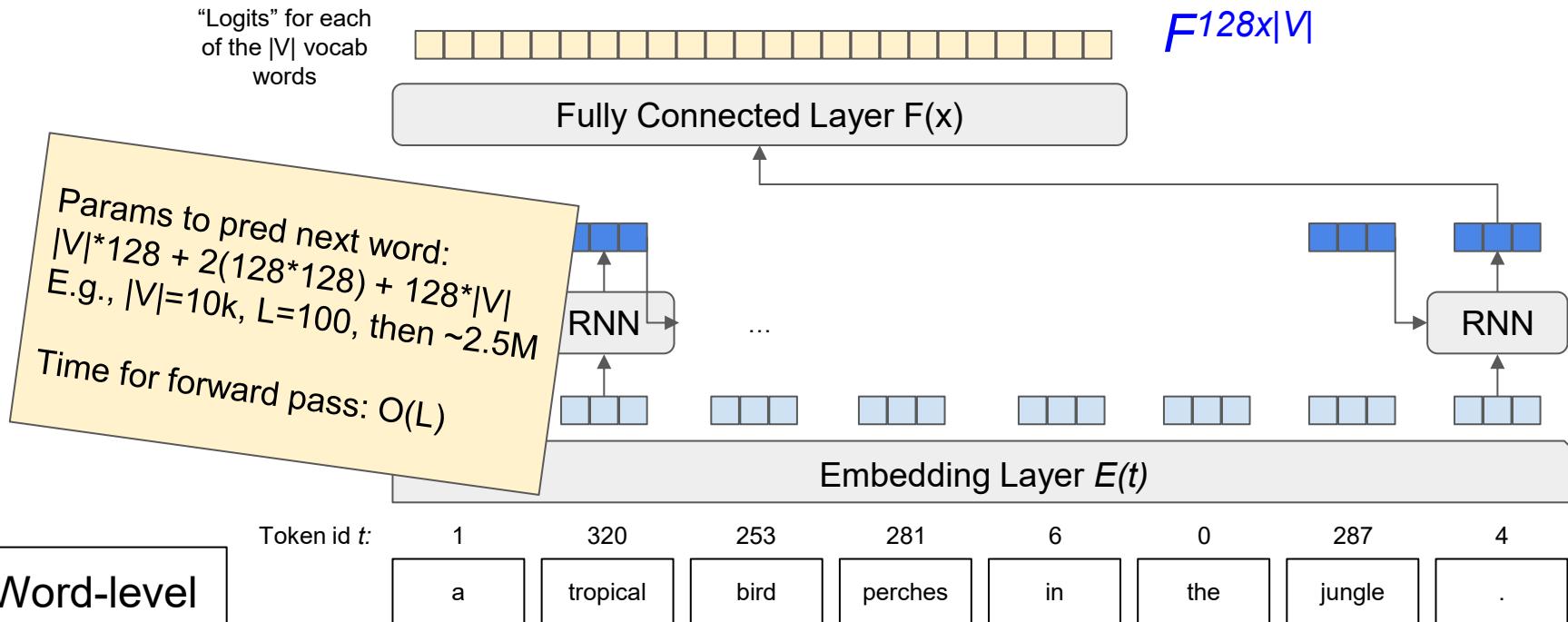


# Parameters and Efficiency

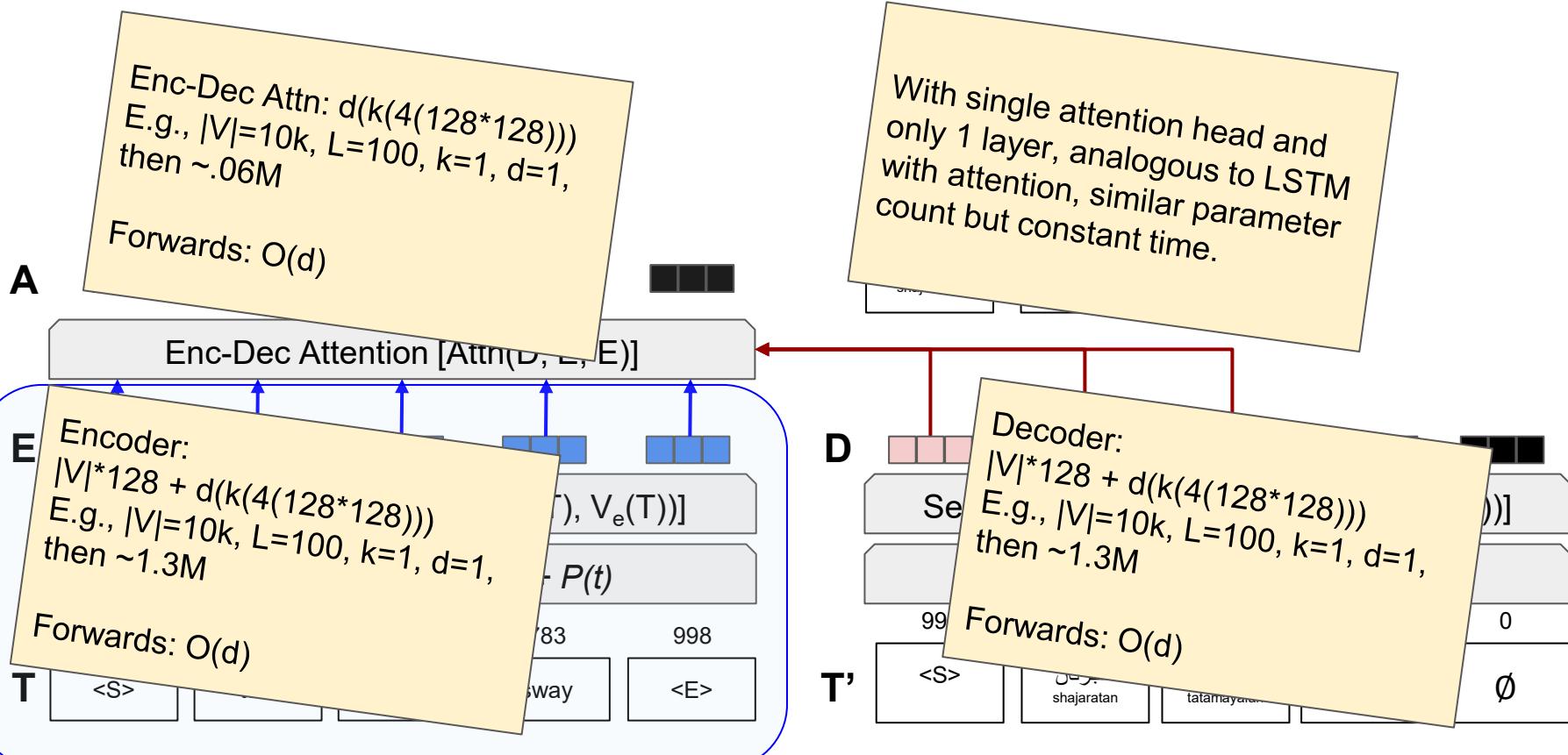
- Remember our Fully-Connected option for classification?



# Parameters and Efficiency



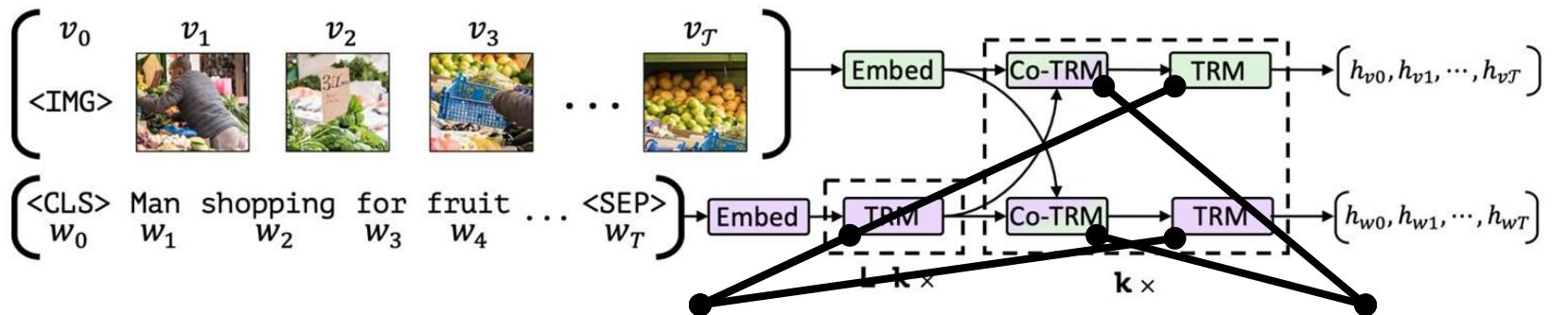
# Sequence-to-Sequence (Encoder-Decoder) Transformer



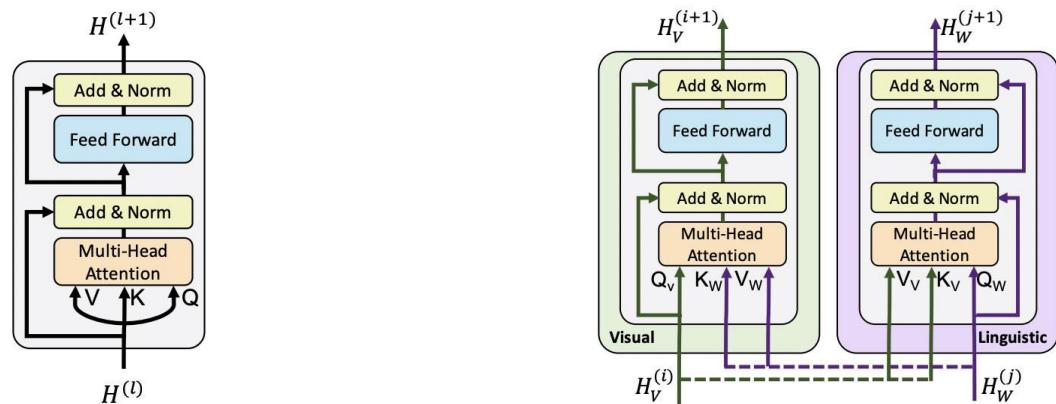
# Transformers: The Future of the Present

- Transformers *seem* like they might be the universal encoders we've been waiting for in multimodal space
- Language tokens, image regions, image patches, audio snippets, joint positions, transformers don't care!

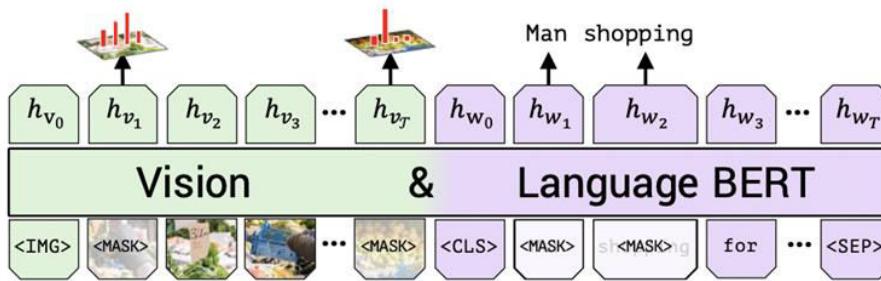
# Language and Vision: Regions



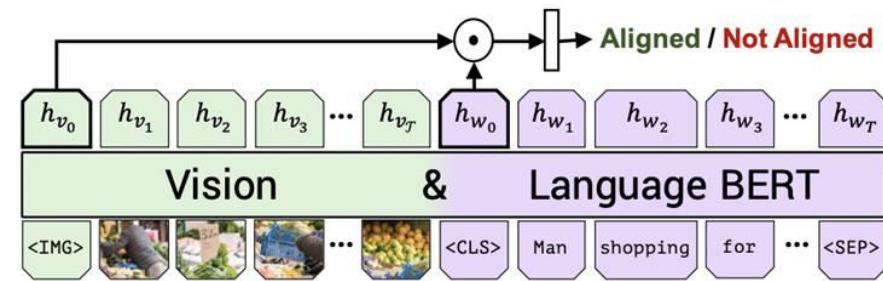
- Word attentions  $H_w$  as keys/values for image queries
- Region attentions  $H_v$  as keys/values for language queries



# Language and Vision: Regions

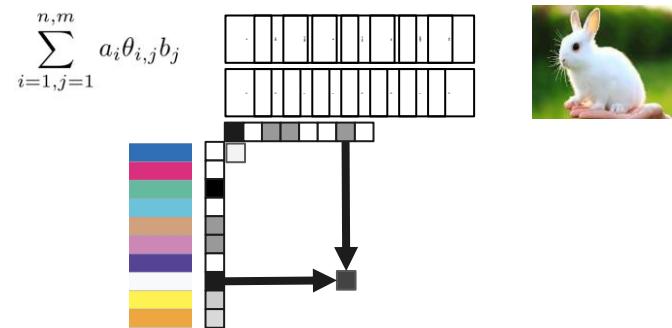


(a) Masked multi-modal learning

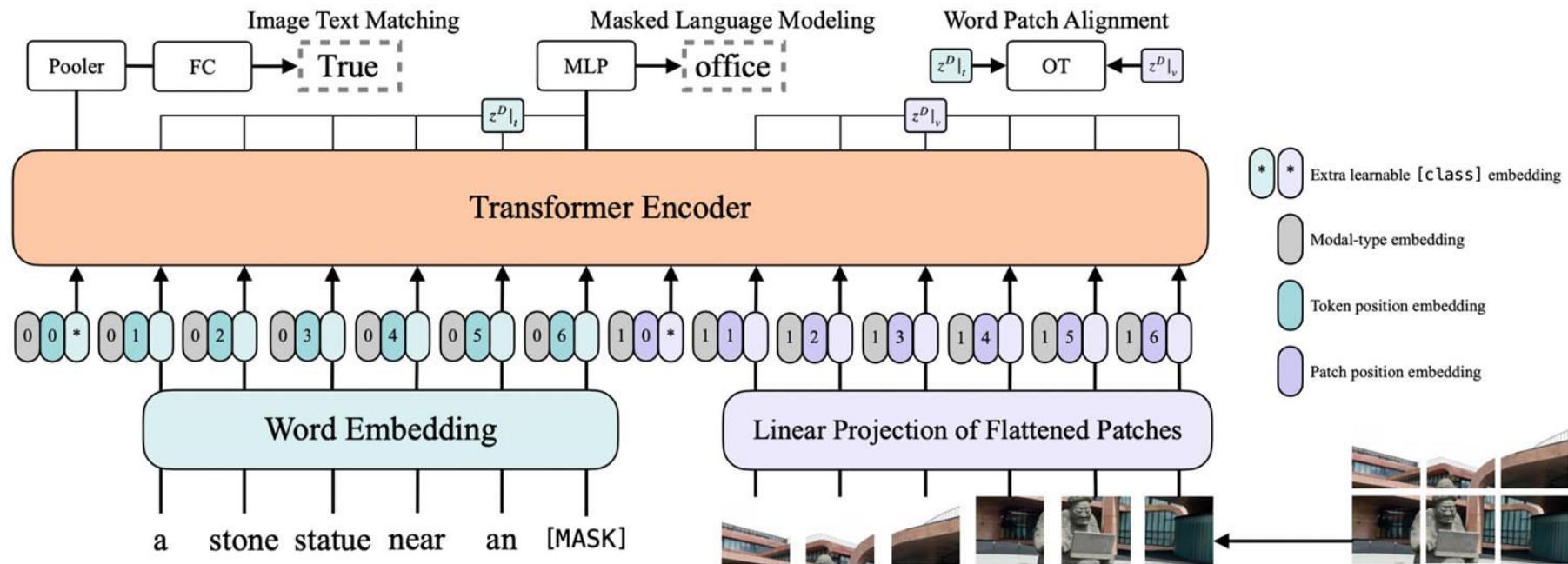


(b) Multi-modal alignment prediction

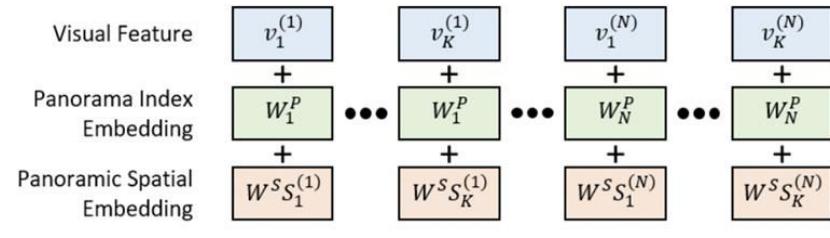
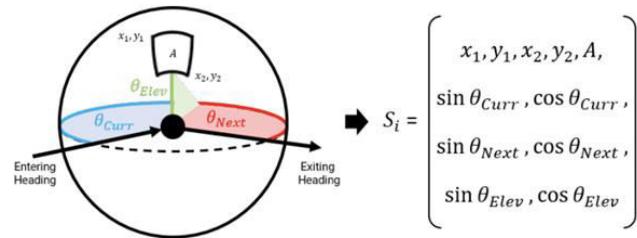
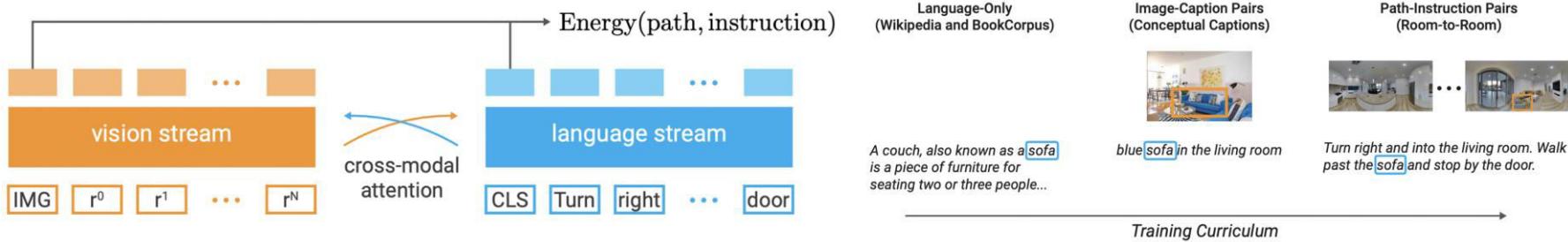
- Training outputs to have high cosine similarity is the same core idea as the grounding lecture linear model!



# Language and Vision: Patches



# Language and Vision and “Time”



# Transformers: The Future of the Present

- Transformers *seem* like they might be the universal encoders we've been waiting for in multimodal space
- Language tokens, image regions, image patches, audio snippets, joint positions, transformers don't care!
- BUT we've seen this paradigm shift before:
  - CNNs for images (vision) and characters (language)
  - RNNs for sequences of words (lang) or frames (vision)
  - SVMs for classifying features in vision and/or language
  - ...

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 2: Lecture 2  
LM Decoding and Transformer Networks

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 2: Lecture 3  
DL 102 for NLP & Pretraining and Finetuning



## Homework 2 Pt 1

Due Oct 11



## Homework 2 Pt 2

Due Nov 1



## Midterm Report

Due Nov 8



## Paper Review

Due Nov 15

SUN	MON	TUE	WED	THU	FRI	SAT
<b>Oct</b>		1	2	3	4	5
6	7	8	9	10	11	12
13	14 Columbus Day	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31 Halloween		
3	4	5	6 Election Day	7	8	9
10	11 Veterans Day	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28 Thanksgiving Day	29 Black Friday	30

# Lecture Outline

- Recap: Transformers for Language Modeling
- DL 102 for NLP: Activations, Normalization, and Regularization
- Large Language Models: Pretraining and Finetuning
- Zoom Out

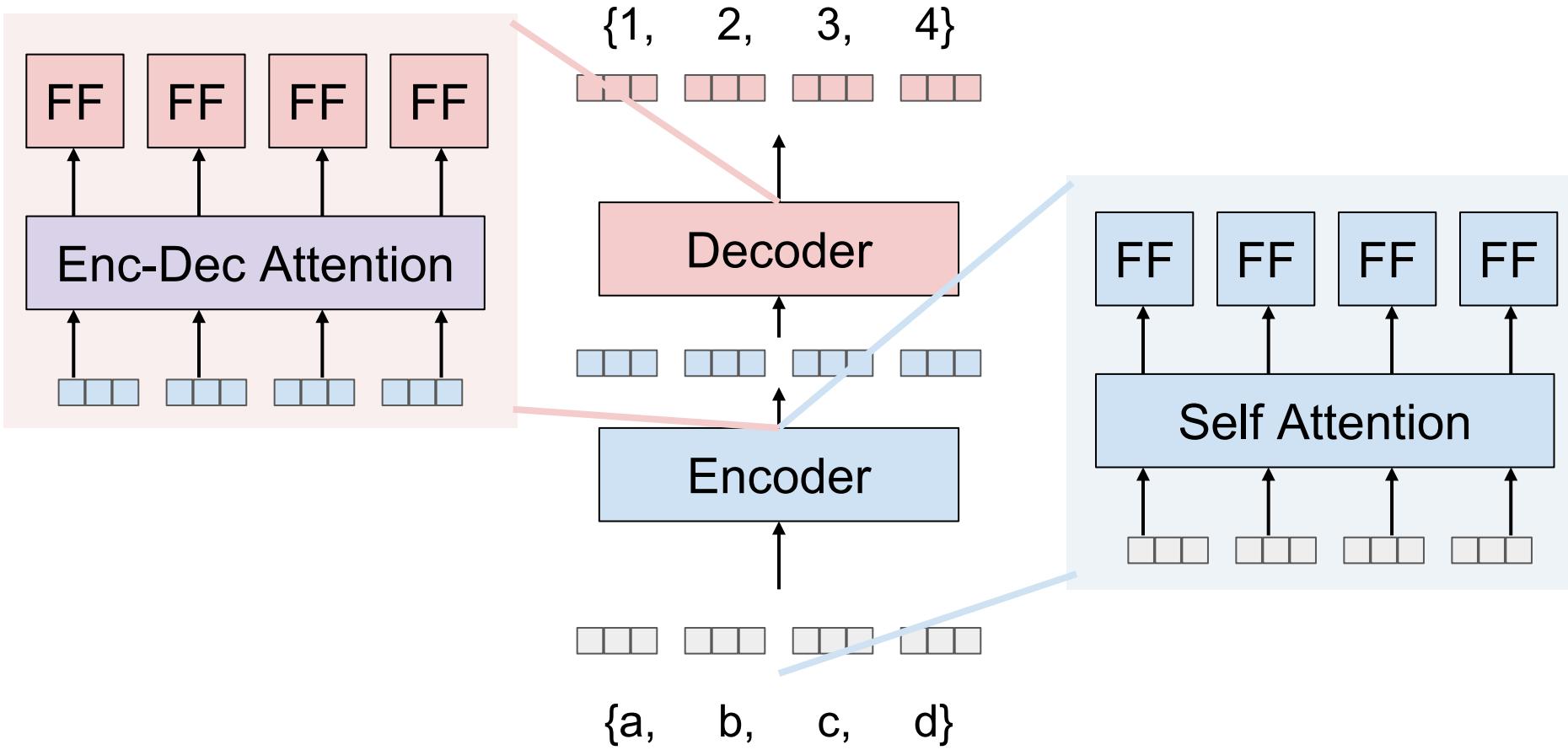
# Lecture Outline

- Recap: Transformers for Language Modeling
- DL 102 for NLP: Activations, Normalization, and Regularization
- Large Language Models: Pretraining and Finetuning
- Zoom Out

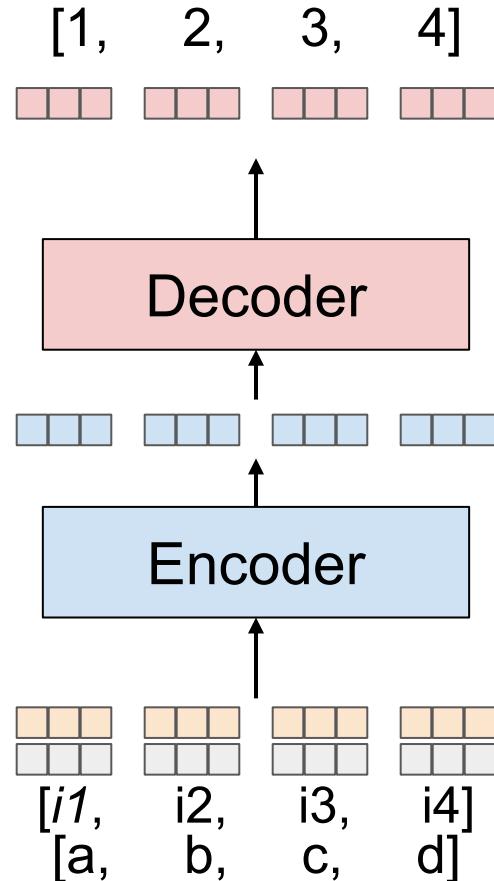
# Transformers Lite

- Transformer  $T : S_a \rightarrow S_b$  is a function from a set  $a$  in  $S_a$  to a set  $b$  in  $S_b$
- Simple transformer consists of a single layer encoder  $E$  and a single layer decoder  $D$  with single-headed attention

# Transformers Lite

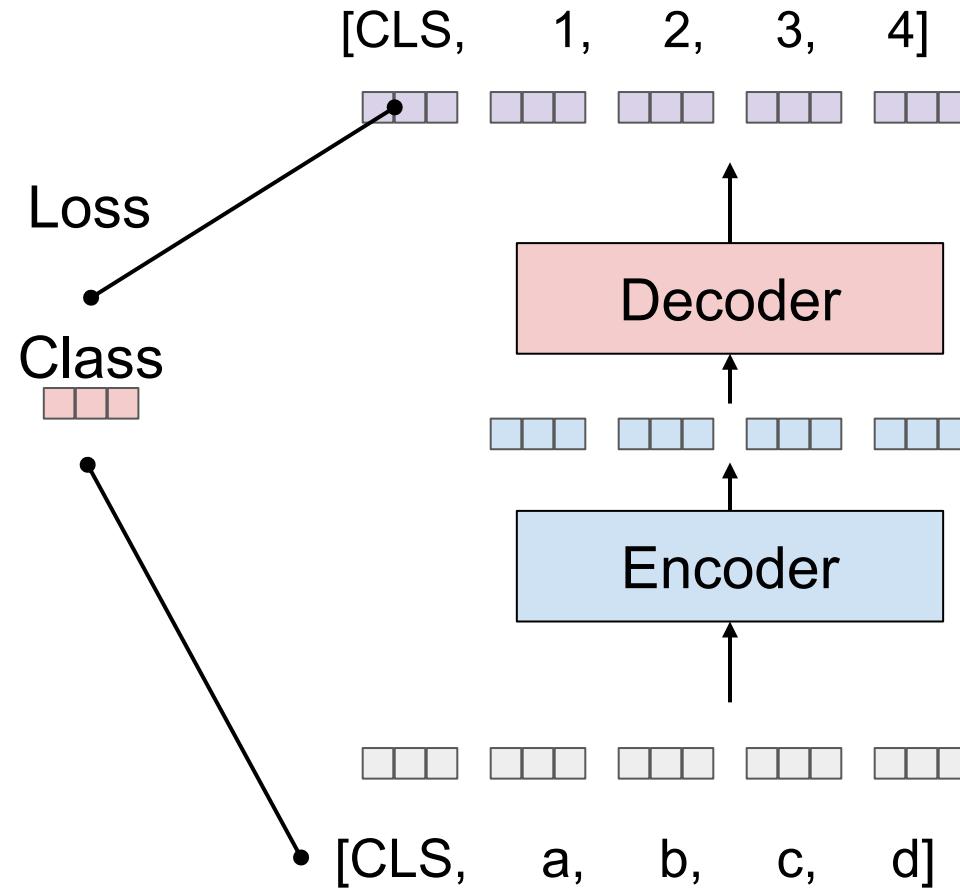


# Sequences



- What if we want our input/output to be *sequences*?
- E.g., sentences!
- Prev:
  - $\{a, c, d\}$   
→  $\{4, 3, 1\}$ ,
- Want:
  - $[a, c, d]$   
→  $[1, 3, 4]$

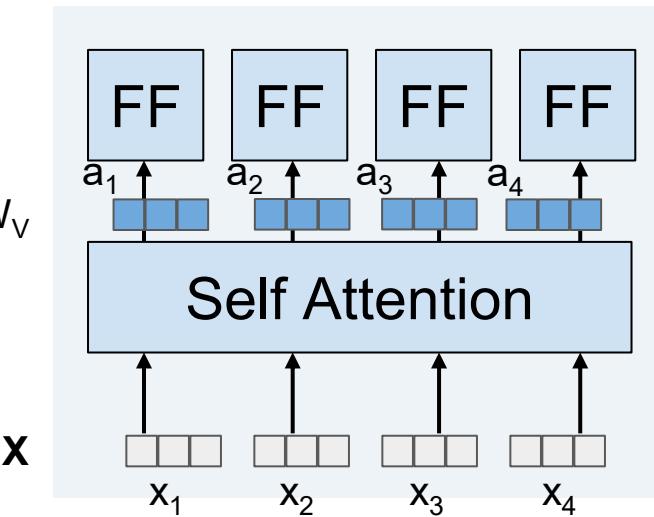
# Supervision: Classification



# Self-Attention

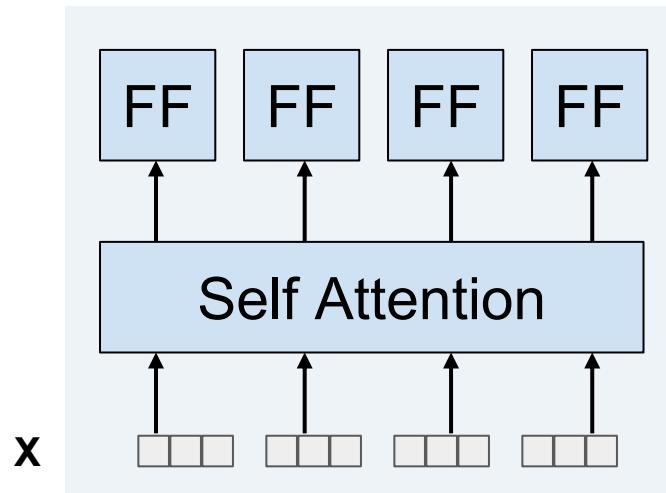
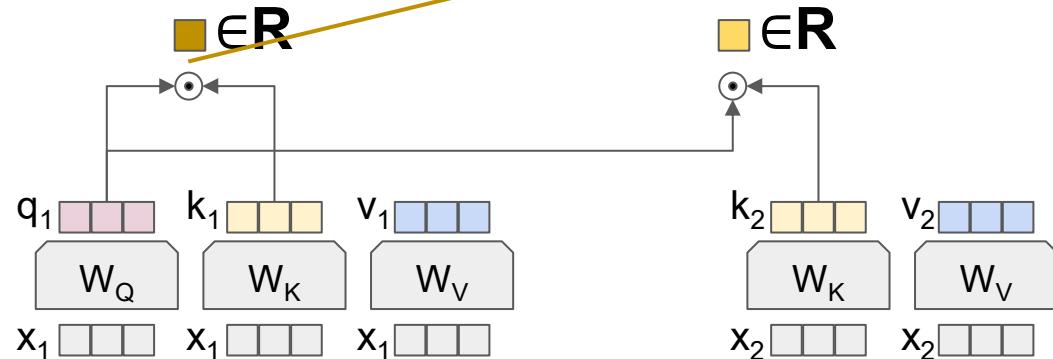
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$$A(XW_Q, XW_K, XW_V)$$



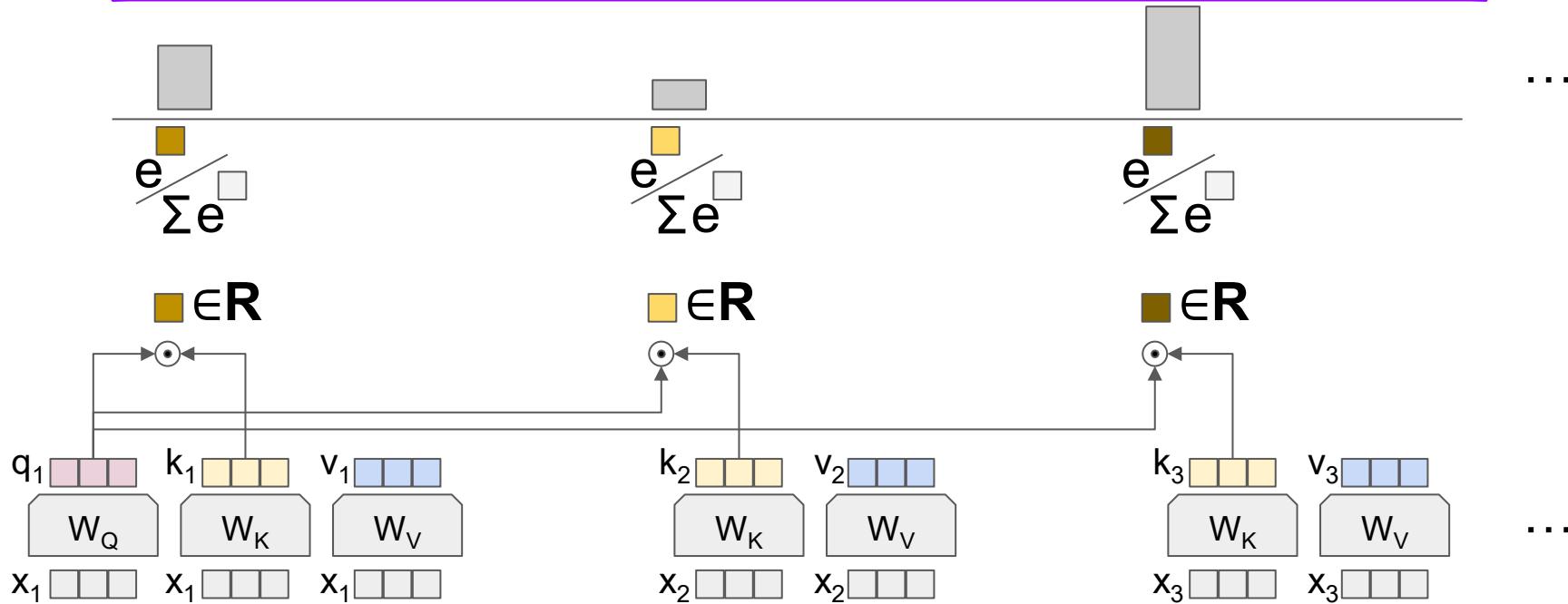
# Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

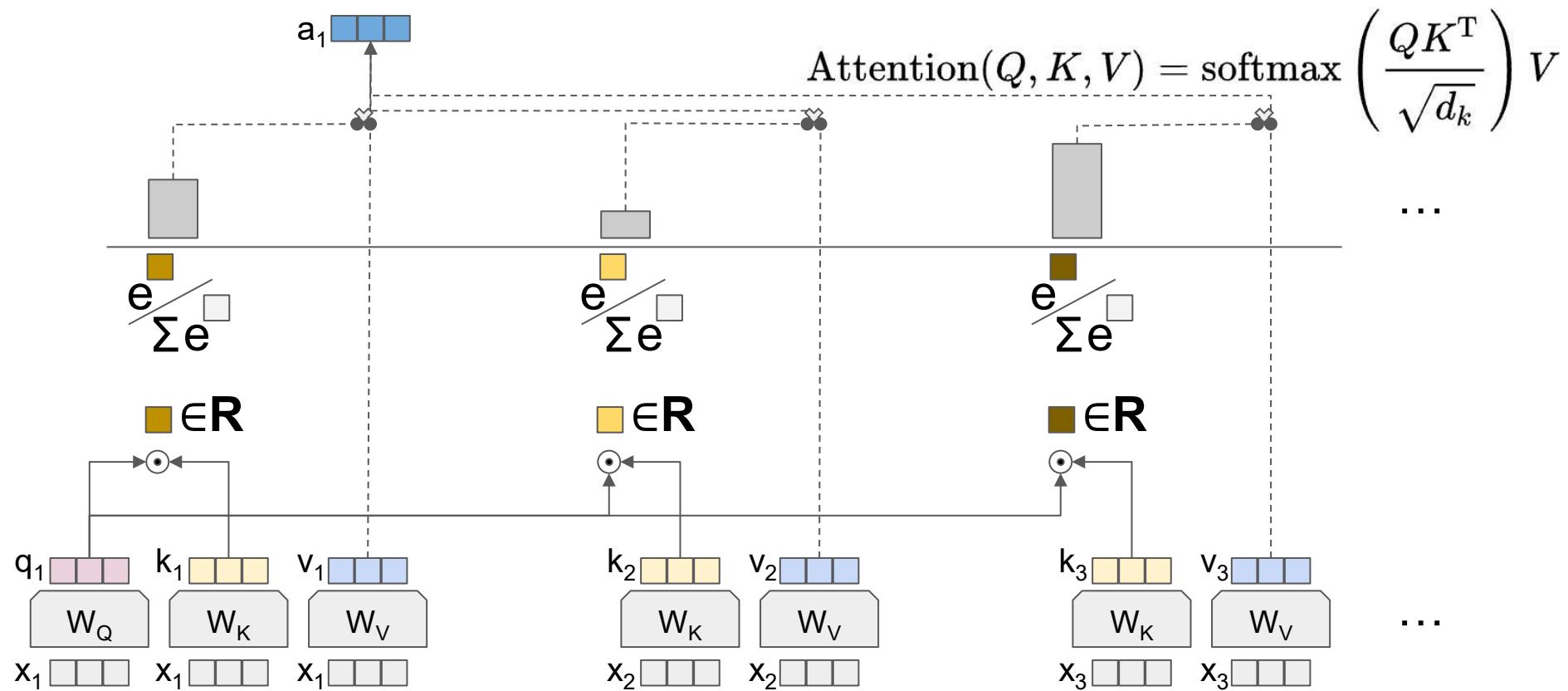


# Self-Attention

$$\text{Attention}(Q, K, V) = \underbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}_{\dots}$$

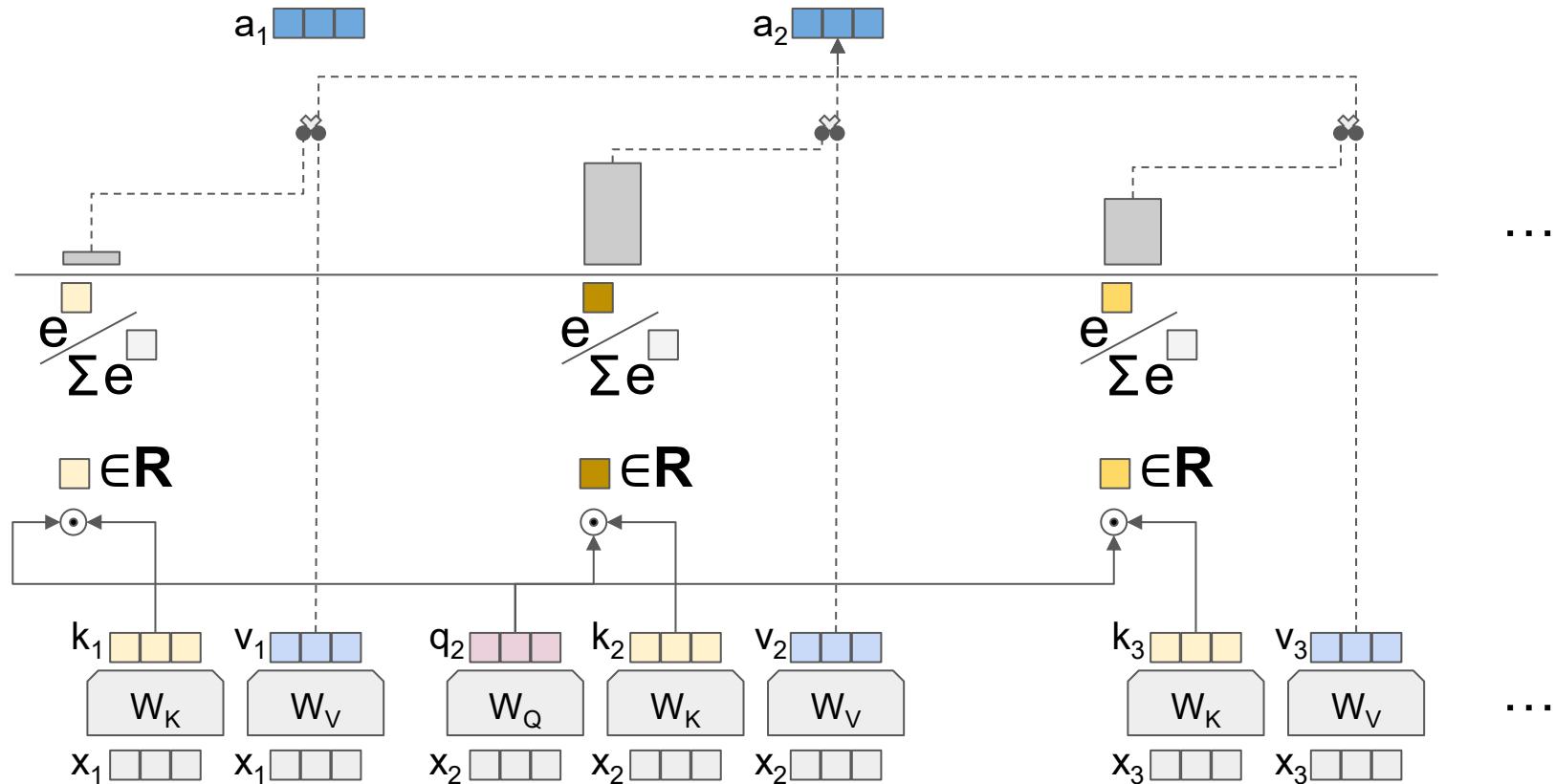


# Self-Attention



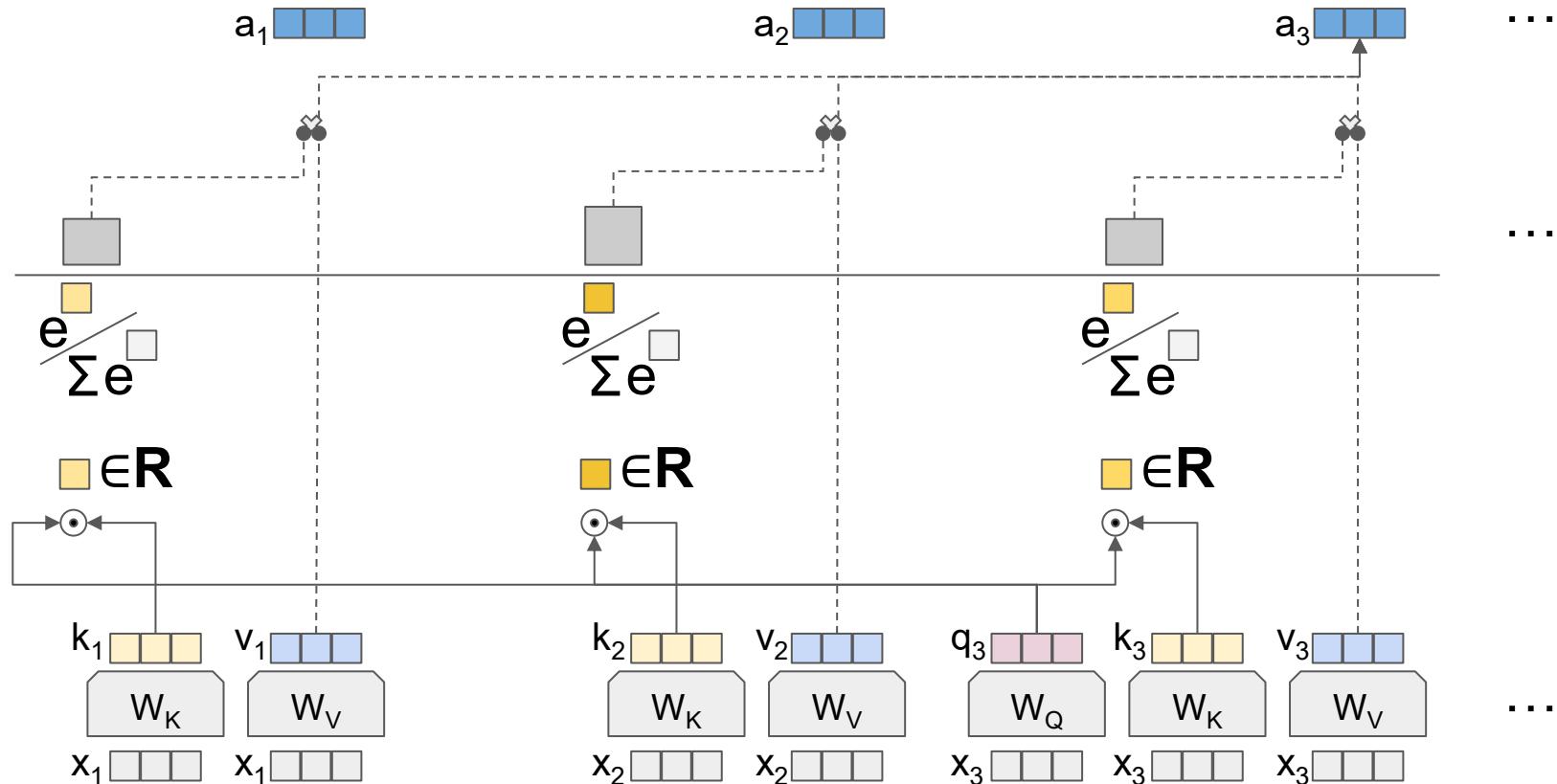
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

# Self-Attention



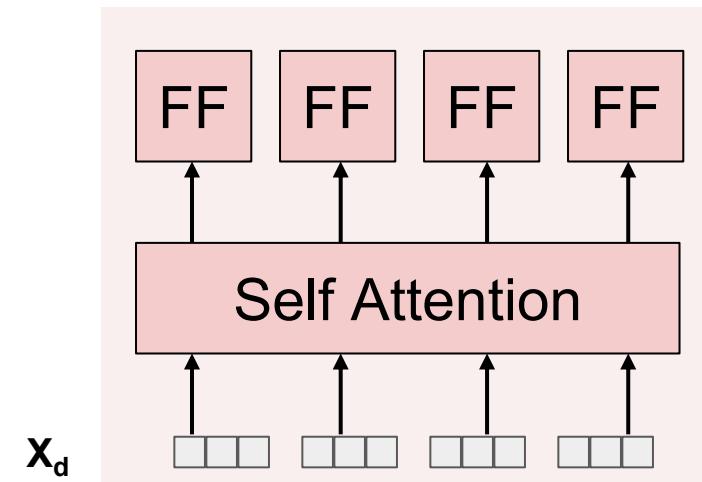
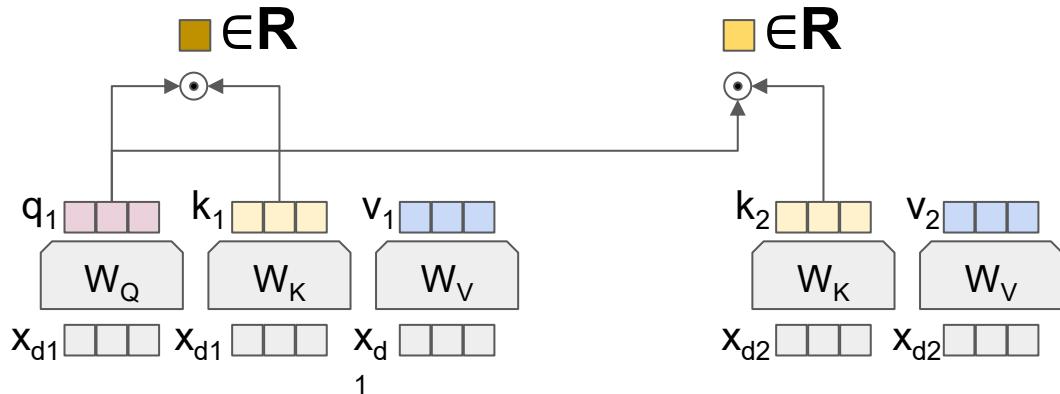
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

# Self-Attention



# Encoder-Decoder Attention

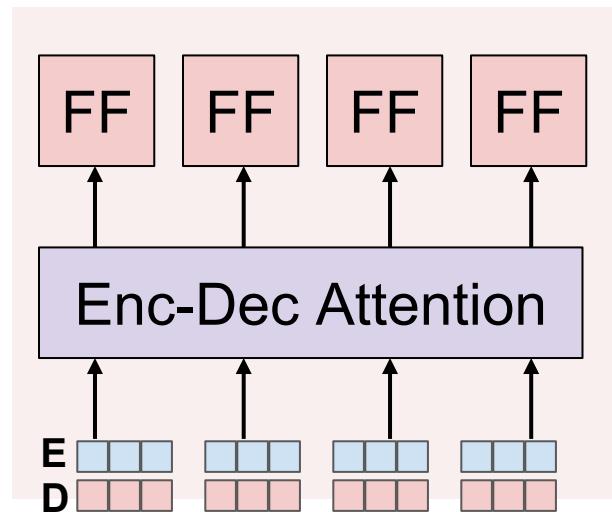
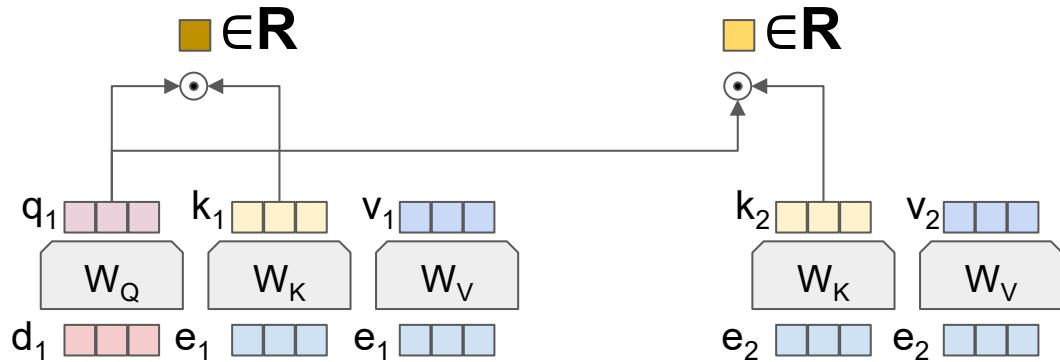
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$



# Encoder-Decoder Attention

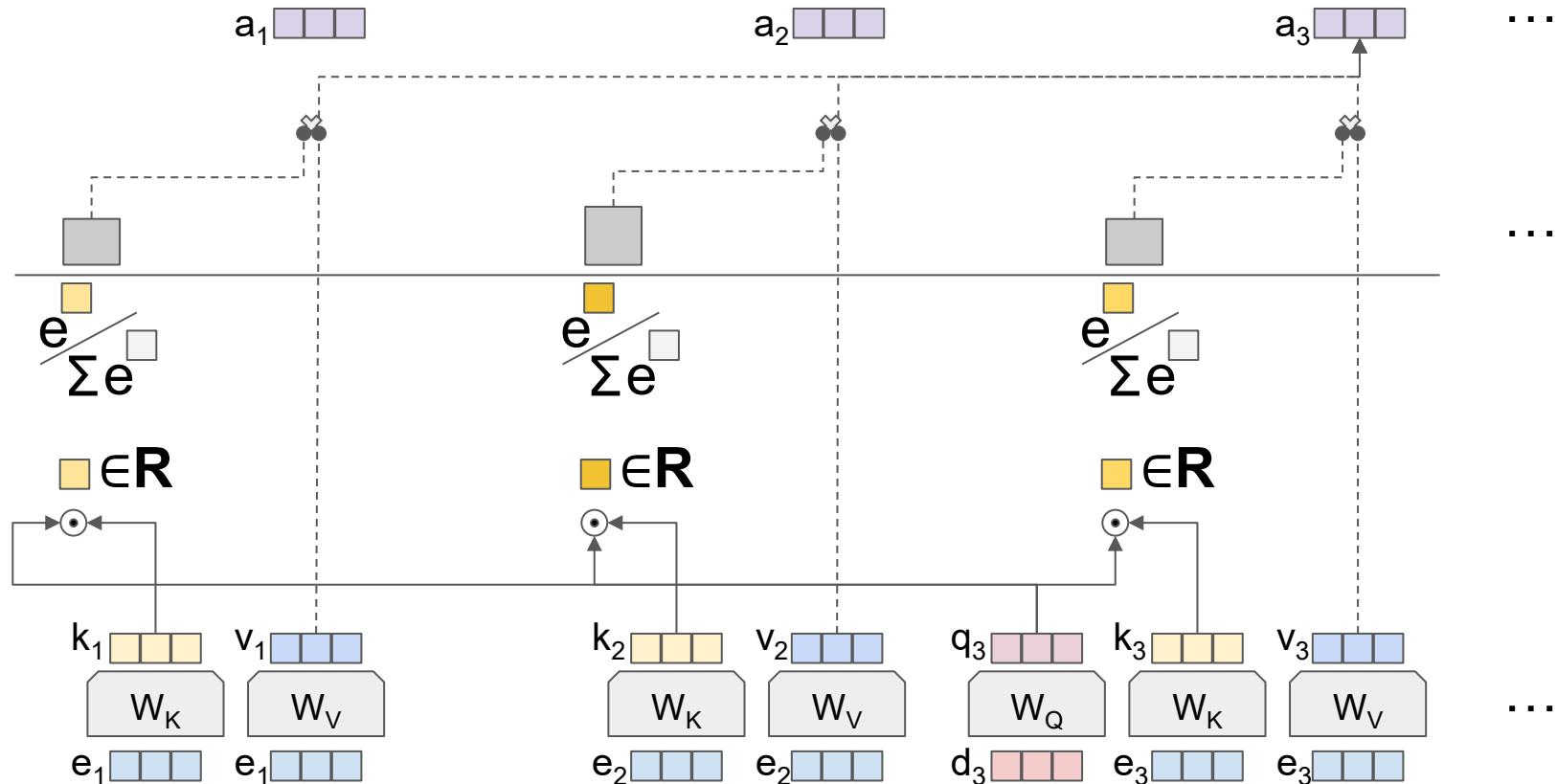
Attention( $\mathbf{DW}_Q, \mathbf{EW}_K, \mathbf{EW}_V$ )

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

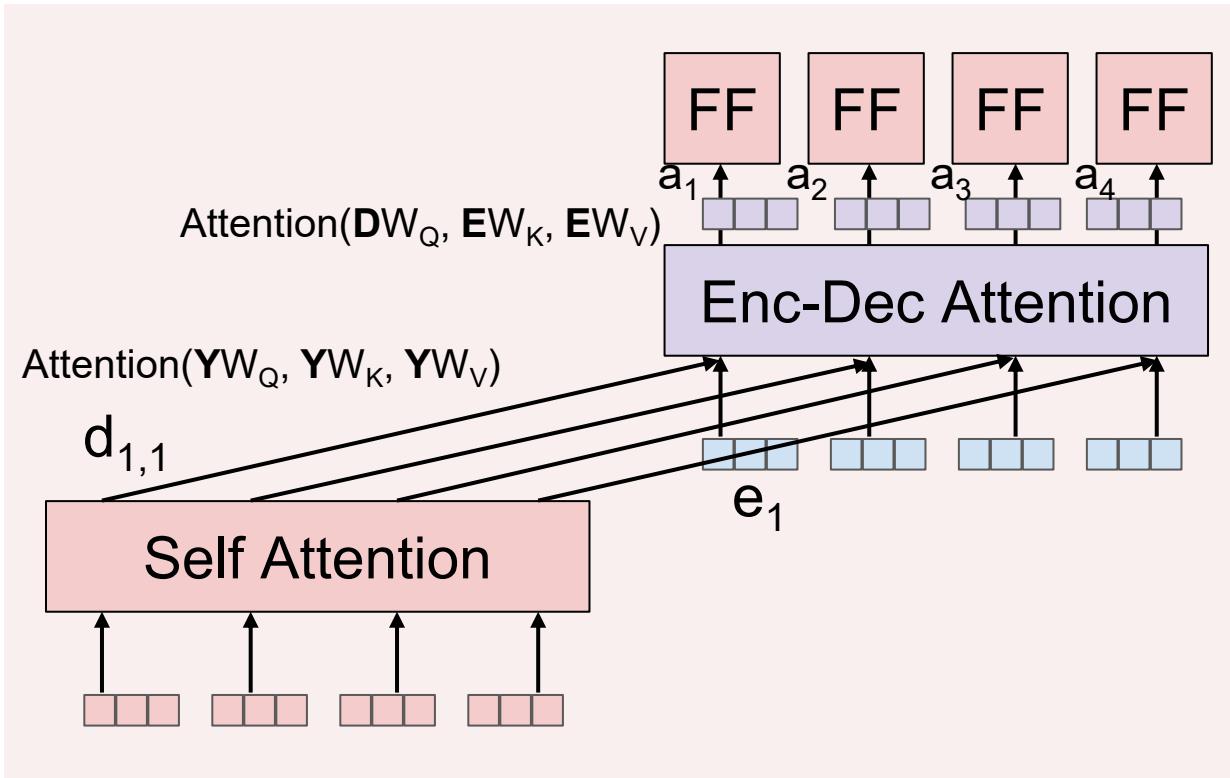


$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

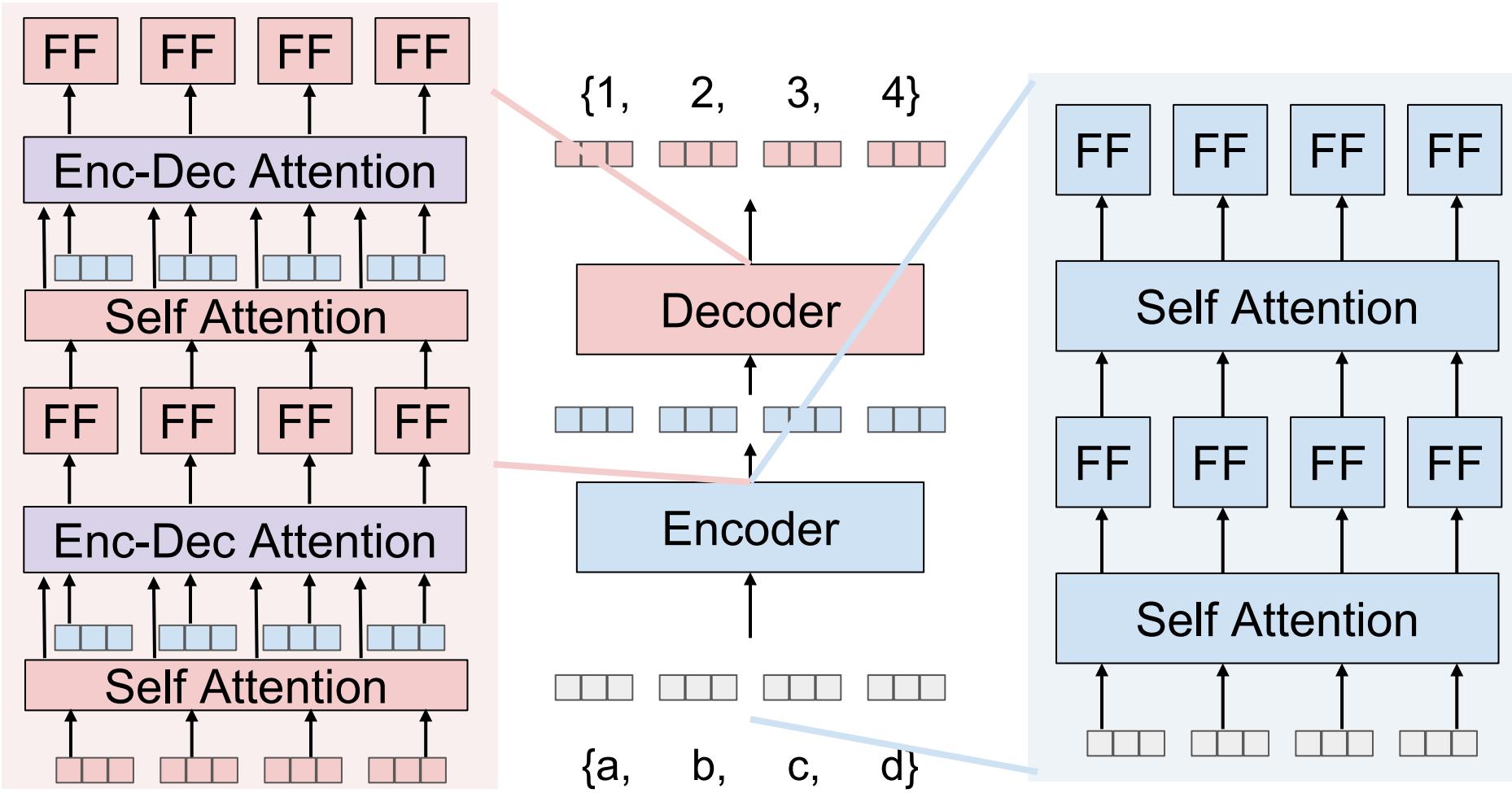
# Encoder-Decoder Attention



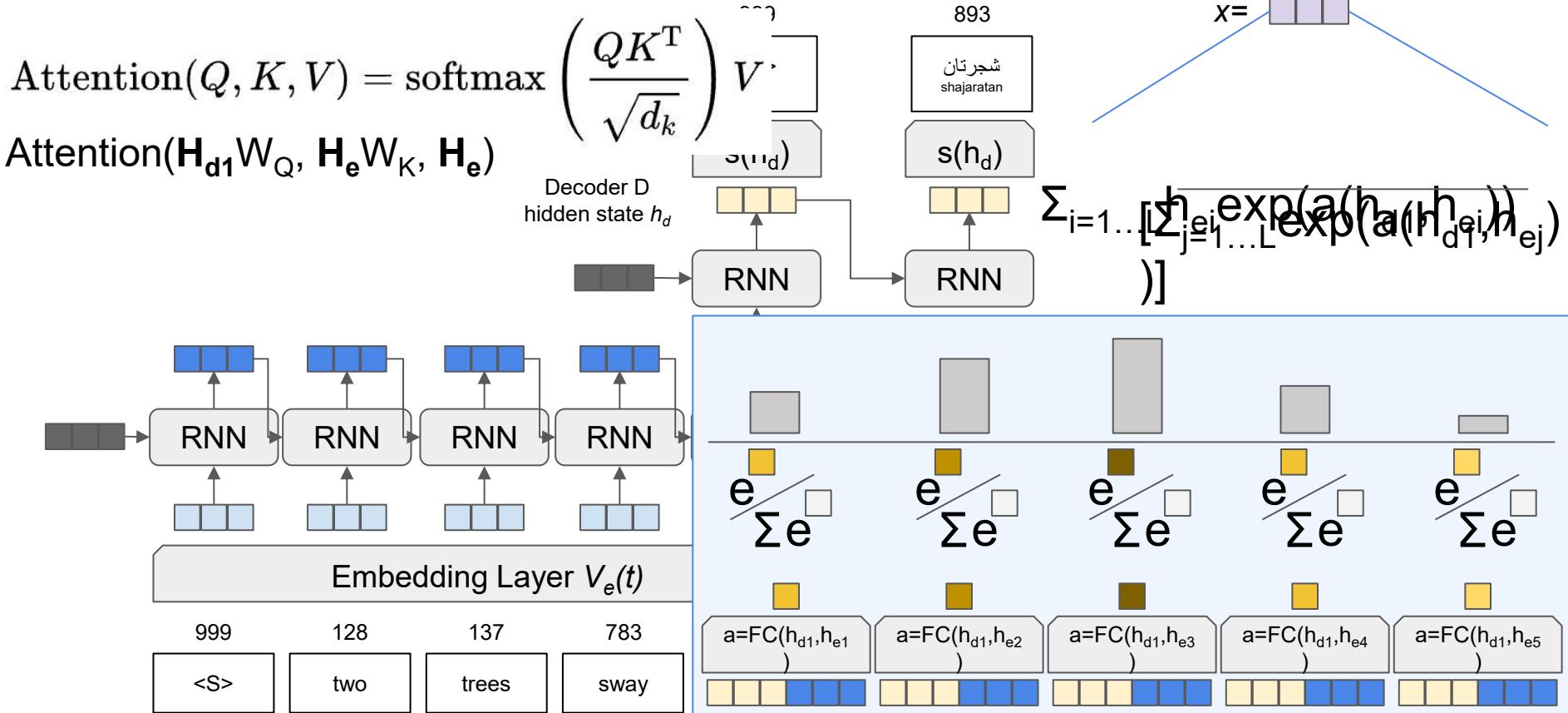
# Encoder-Decoder Attention



# Deeper Encoders / Decoders

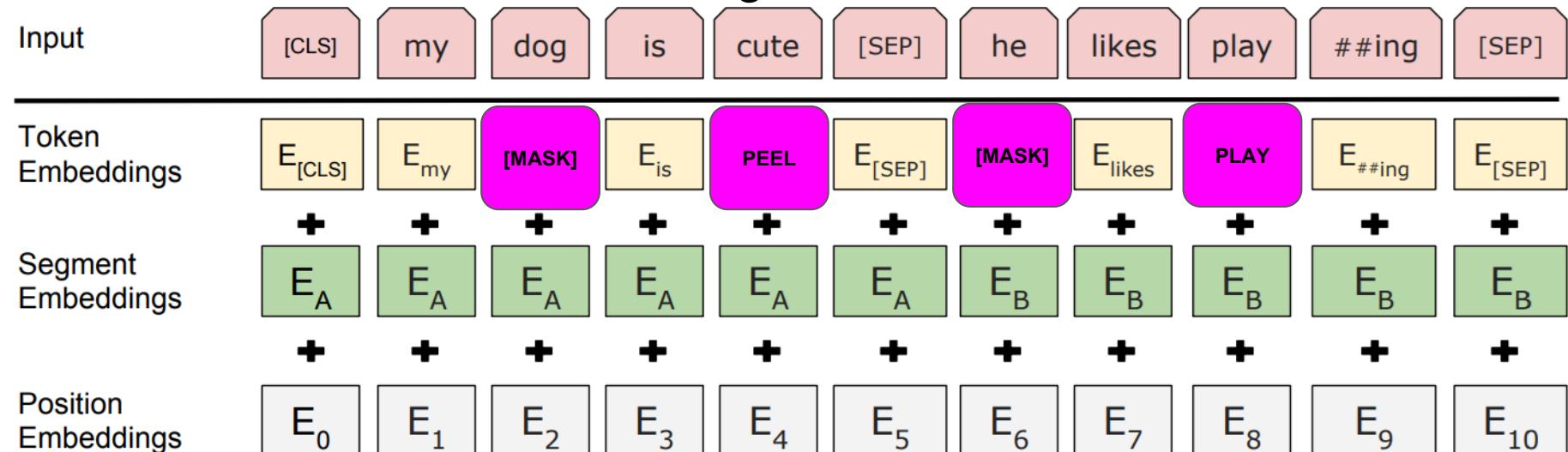


# Relationship to *Global Soft Attention*



# Masked Language Modeling Pretraining

15% of words selected for training

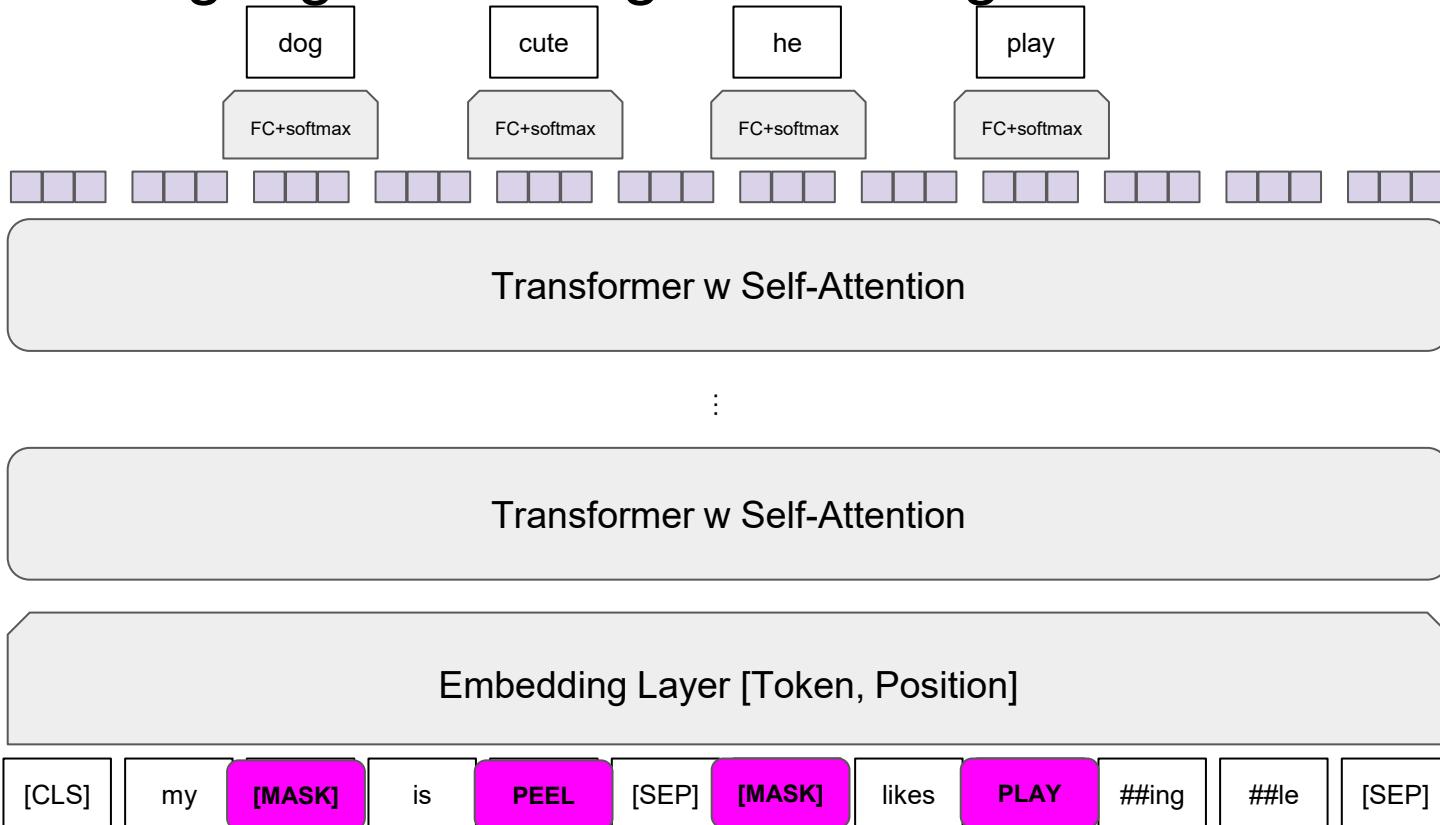


80% of those [MASK]

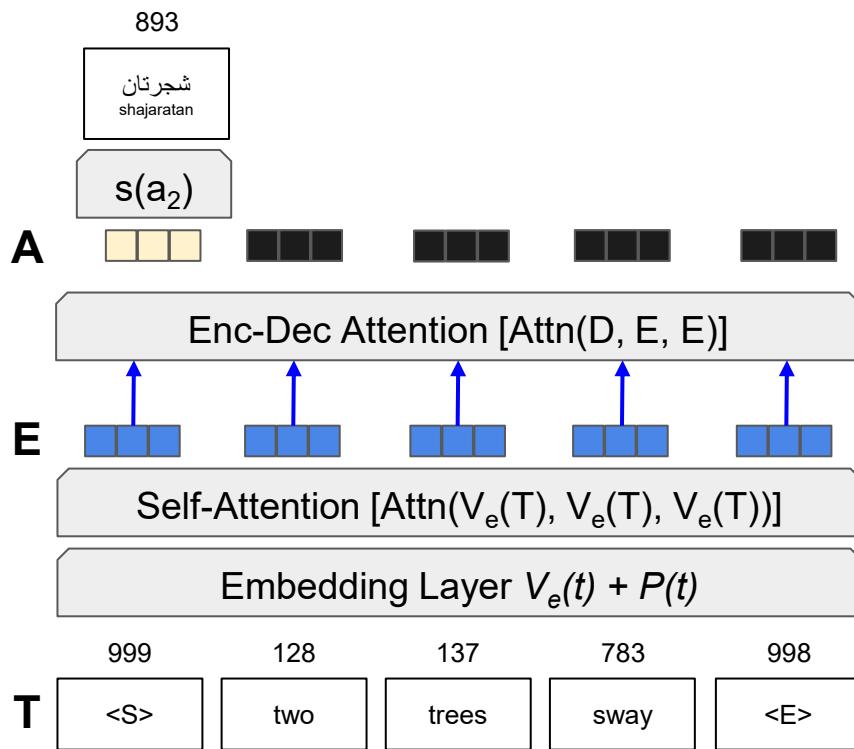
10% random token

10% original token

# Masked Language Modeling Pretraining

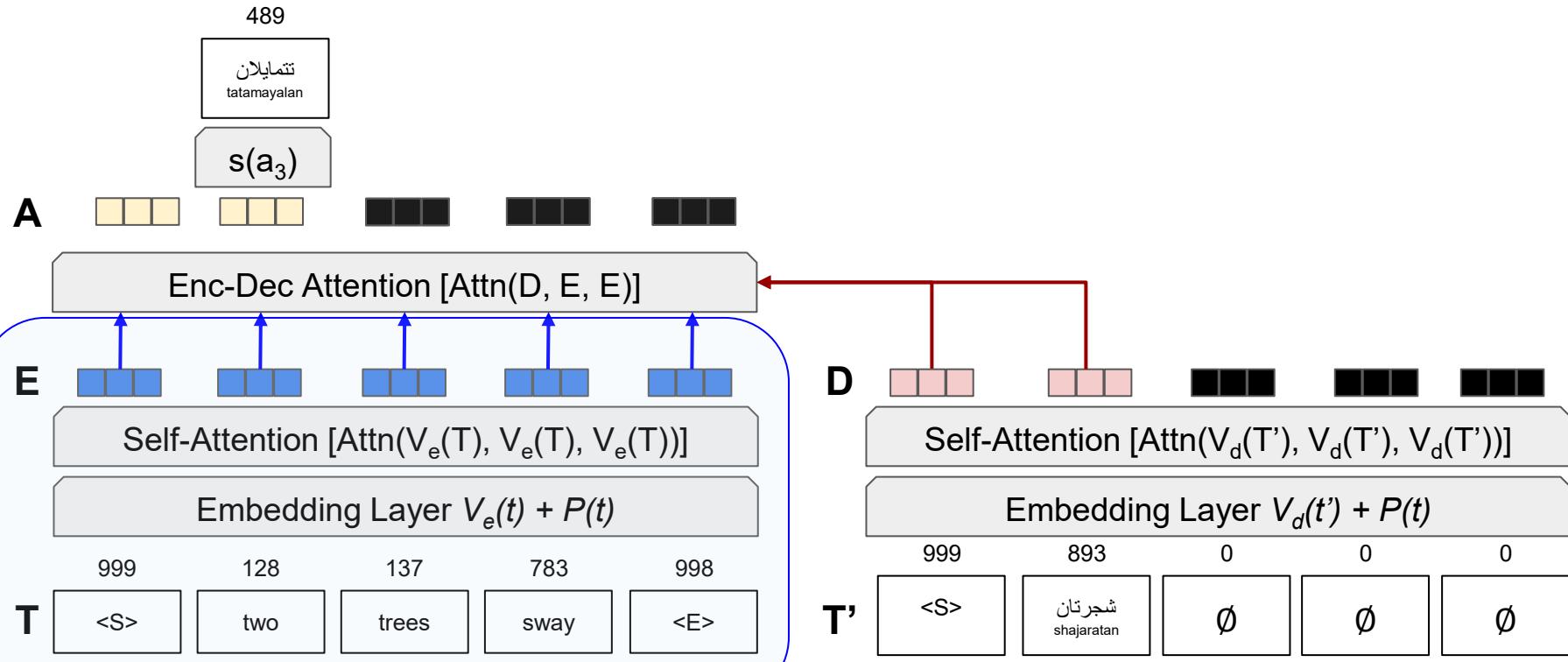


# Sequence-to-Sequence Transformer

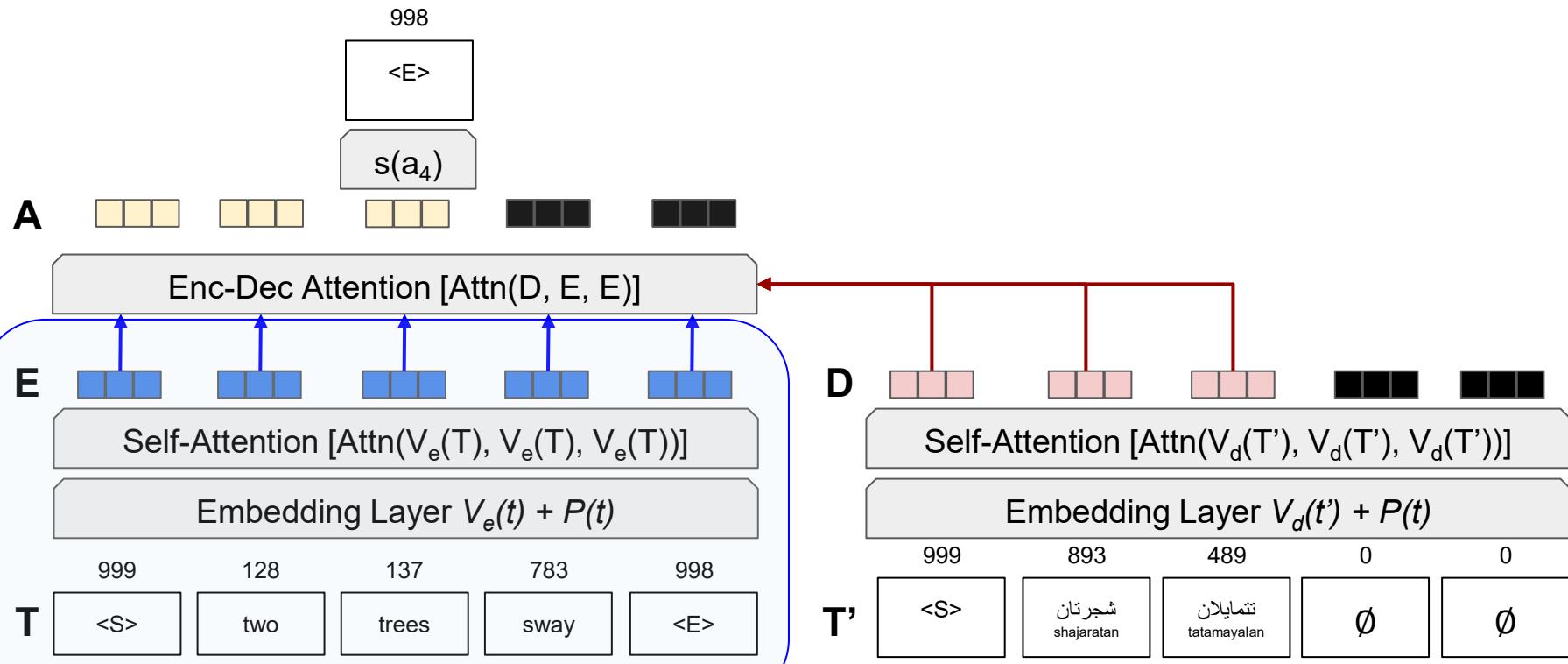


$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

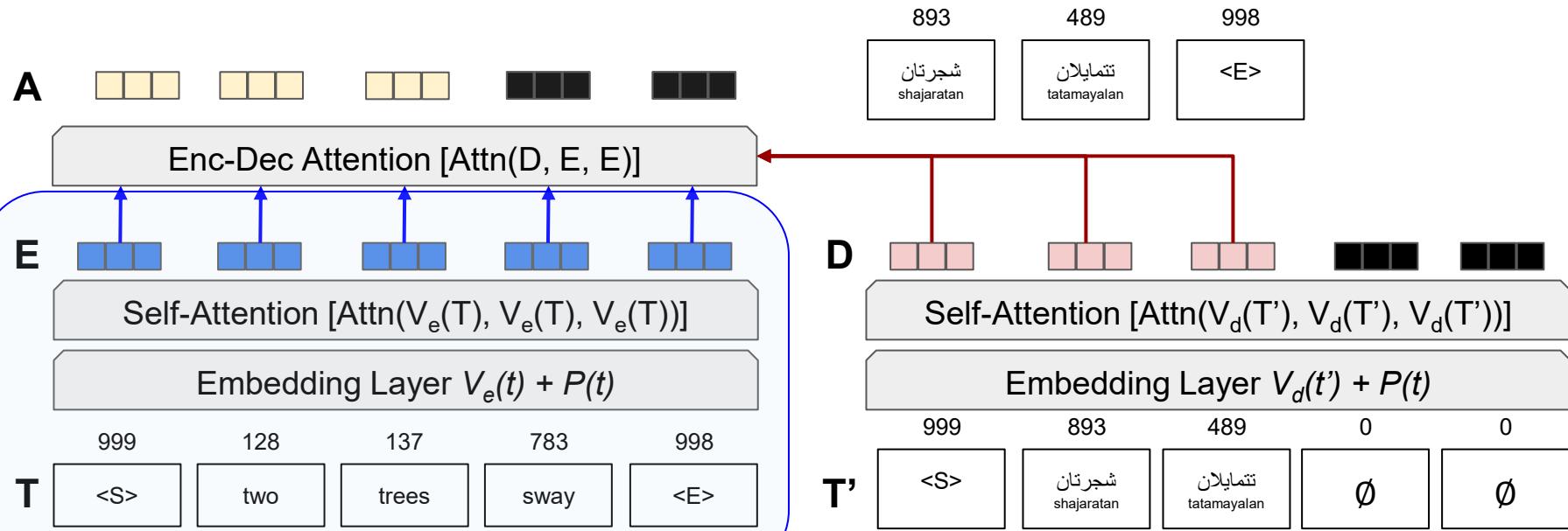
# Sequence-to-Sequence Transformer



# Sequence-to-Sequence Transformer



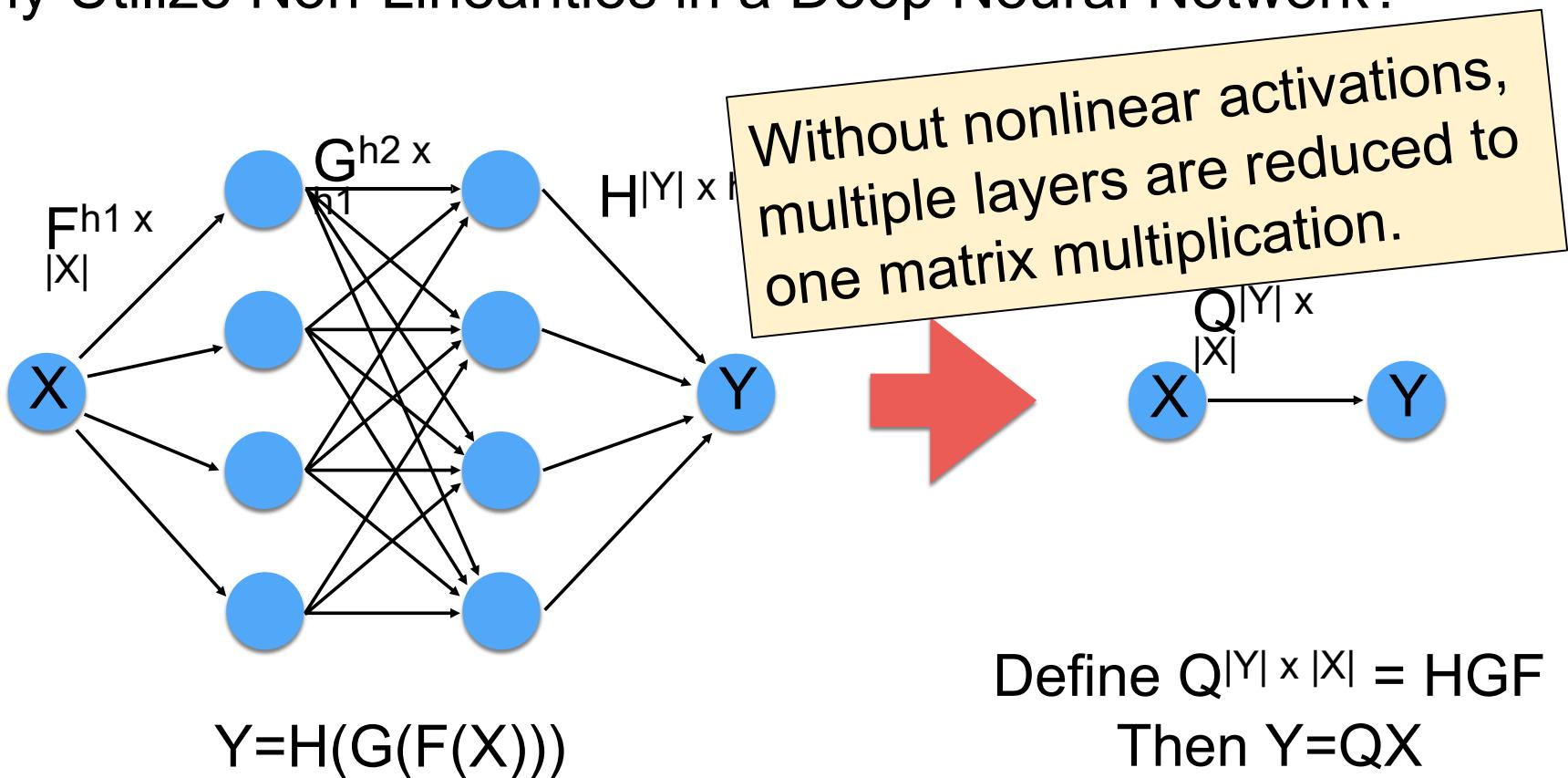
# Sequence-to-Sequence Transformer



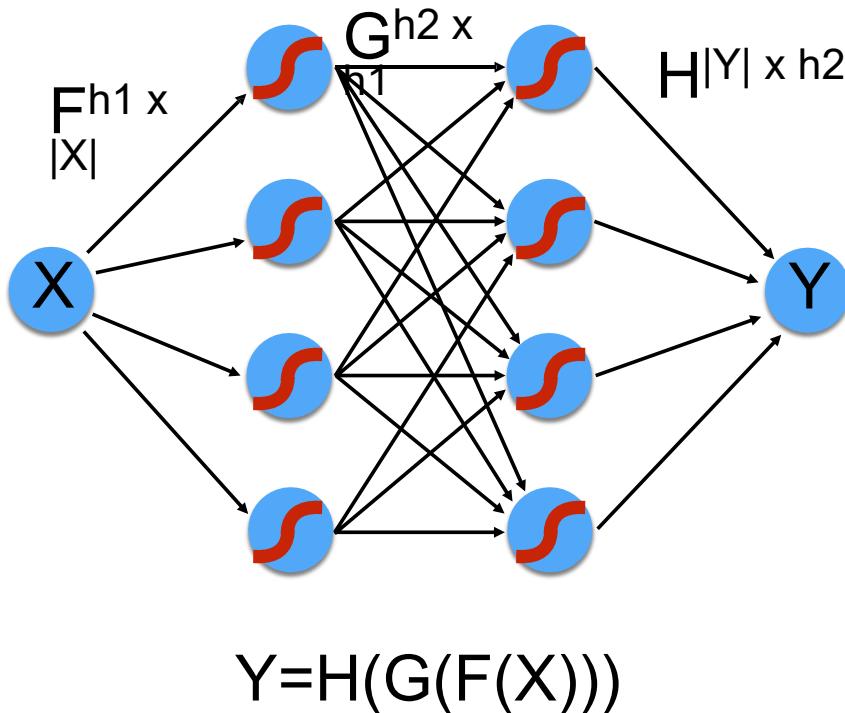
# Lecture Outline

- Recap: Transformers for Language Modeling
- DL 102 for NLP: Activations, Normalization, and Regularization
- Large Language Models: Pretraining and Finetuning
- Zoom Out

# Why Utilize Non-Linearities in a Deep Neural Network?



# Why Utilize Non-Linearities in a Deep Neural Network?

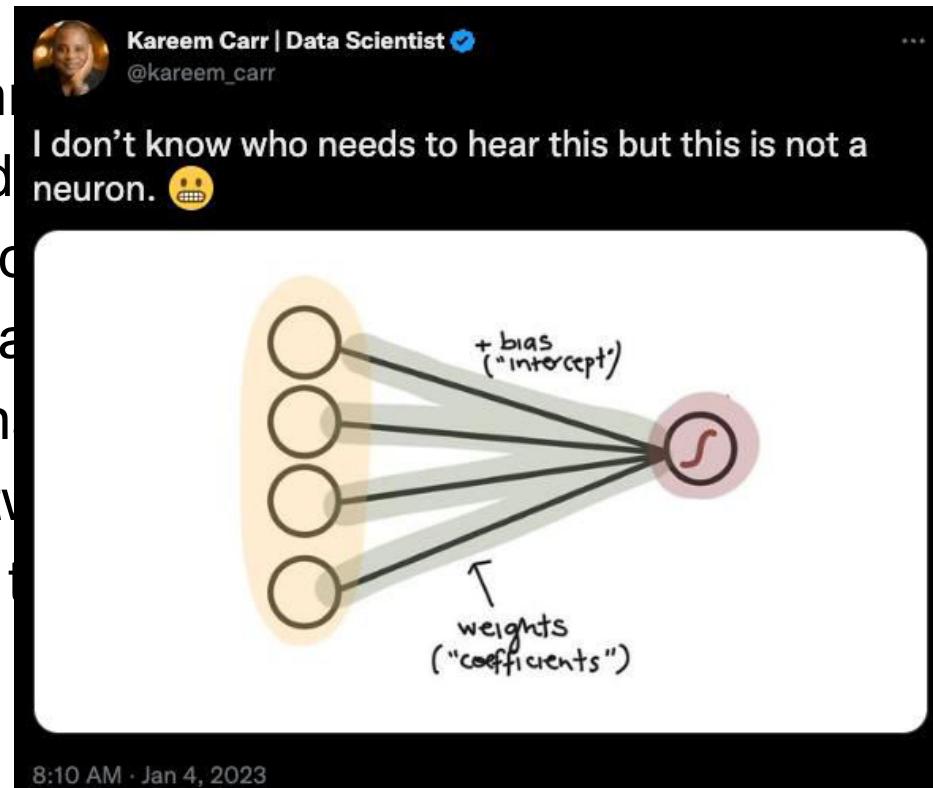


$$F(\mathbf{x}) = a(W_F(\mathbf{x}) + \mathbf{b}_F)$$
$$G(\mathbf{x}_{h_1}) = a(W_G(\mathbf{x}_{h_1}) + \mathbf{b}_G)$$

*Activation* function  $a$  is non-linear and not expressible as a matrix multiplication.

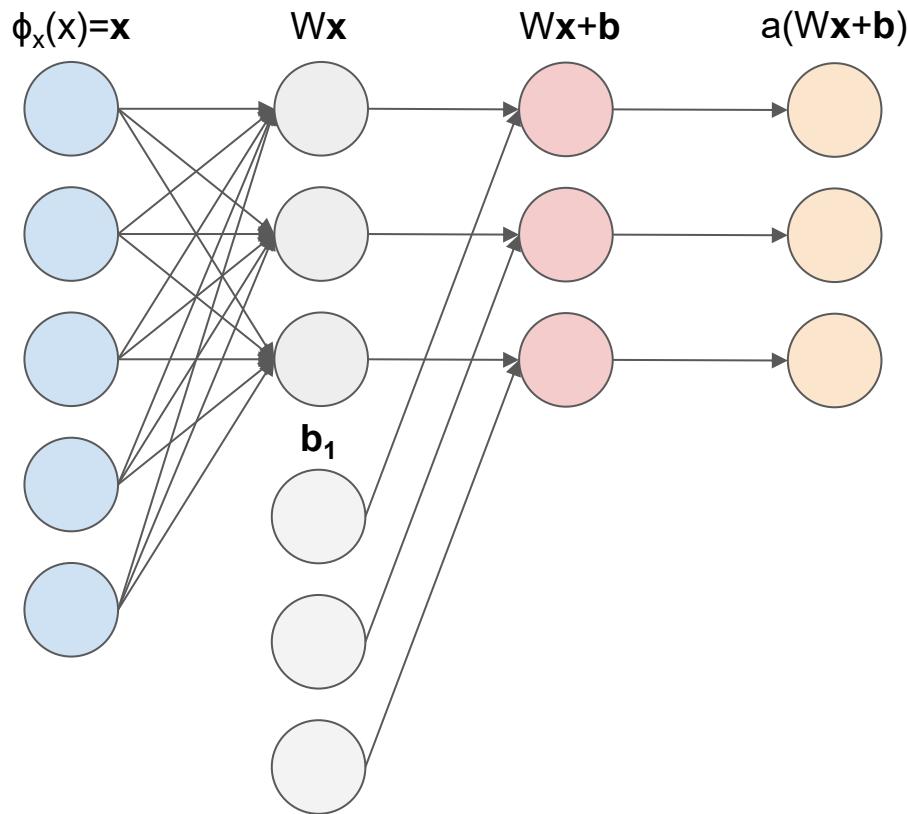
# Deep Learning Borrows\* Terminology from Neuroscience

- Brains are complex, interconnected systems
- Neurons are connected via dendrites, through chemical and electrical signals
- “Artificial neural networks” work by variables **“activated”** by signals
- In a “feed forward neural network”, layers are activated in order from input to output
- A “deep neural network” has many layers

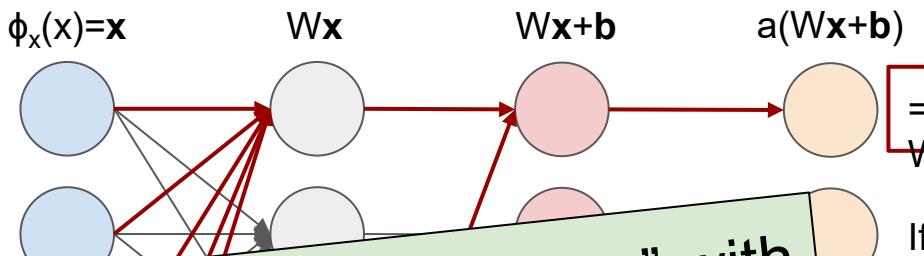


\*Abuses, one might say.

# Activation Functions



# Activation Functions



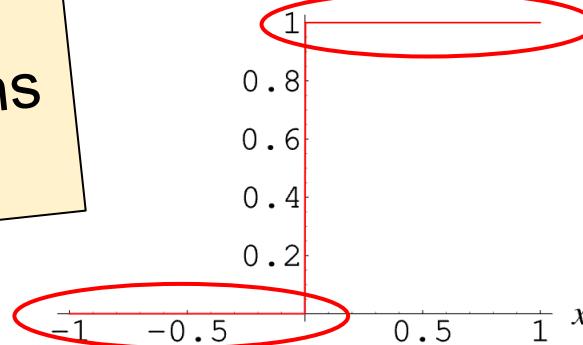
$$= a(W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + W_{1,4}x_4 + W_{1,5}x_5 + b_1)$$

What's the “problem” with defining  $a$  this way?

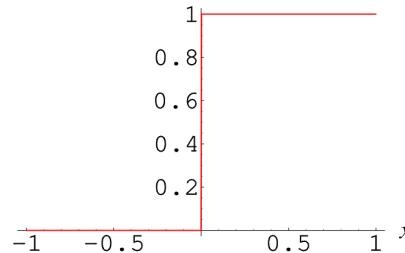
To perform backpropagation, we need differentiable functions with non-zero gradients.

If we think about a “neuron” as inspiration, the output after  $a$  should be 1 if  $W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + W_{1,4}x_4 + W_{1,5}x_5 + b_1$  exceeds a threshold and 0 else.

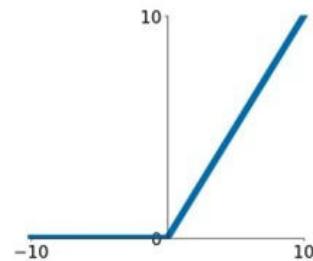
$$g., a(x) = \{1 \text{ if } x > 0 \text{ else } 0\}$$



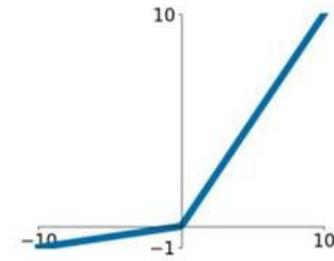
# Activation Functions



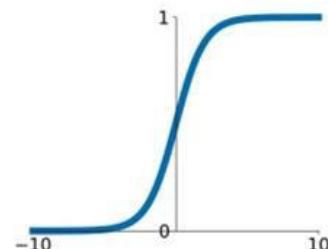
**ReLU**  
 $\max(0, x)$



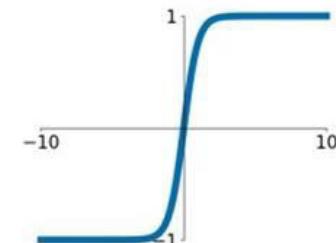
**Leaky ReLU**  
 $\max(0.1x, x)$



**Sigmoid**  
 $\sigma(x) = \frac{1}{1+e^{-x}}$



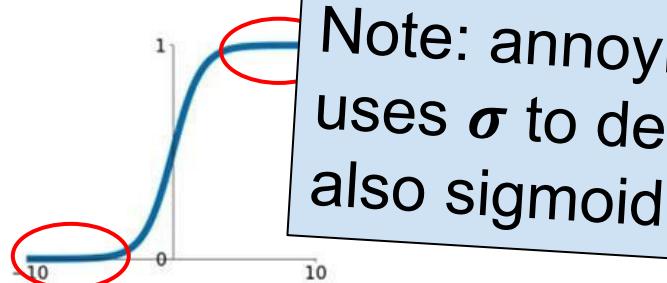
**tanh**  
 $\tanh(x)$



# Activation Functions: Sigmoid

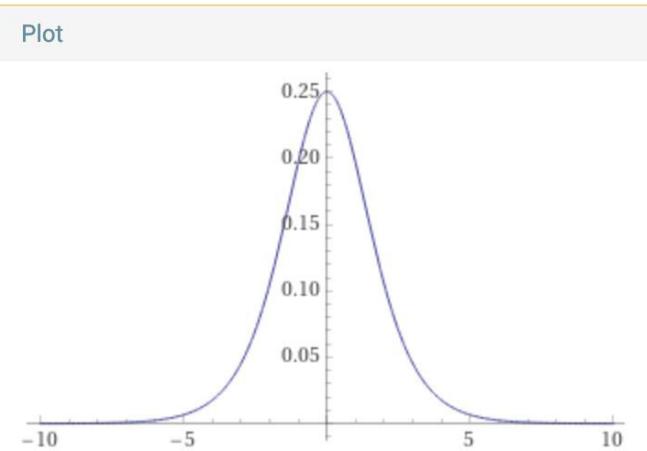
## Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



plot  $\frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right)$   $x = -10 \text{ to } 10$

- Advantages:
  - “Looks” a lot like our step function! Easier to understand.
- Disadvantages:
  - Learning can become prohibitively slow, especially if multiple sigmoids are involved across layers!

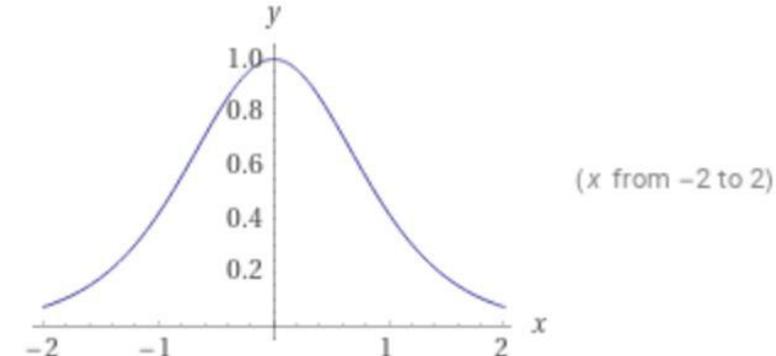
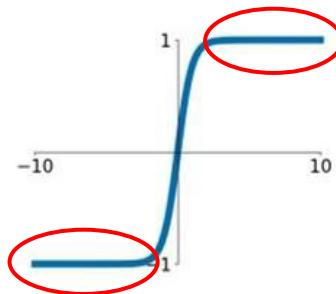


Learning can become prohibitively slow, especially if multiple sigmoids are involved across layers!  
 $\frac{d}{dx} \sigma(x) = \sigma(x)(1-\sigma(x))$  [derivation]

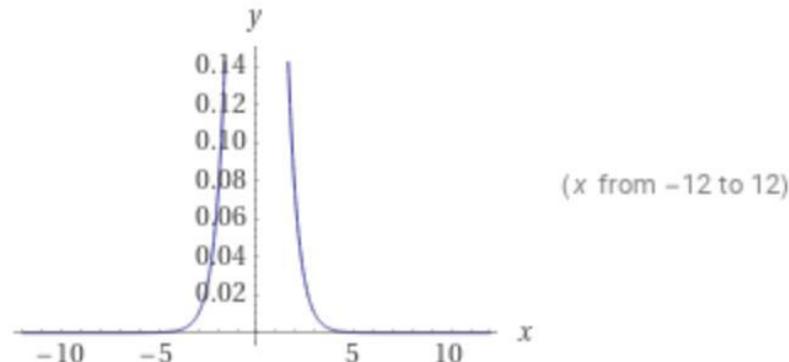
# Activation Functions: Hyperbolic Tangent

**tanh**

$\tanh(x)$



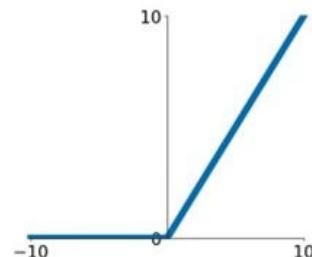
- Advantages:
  - “Looks” a lot like our step function
- Disadvantages:
  - Derivative is near 0 almost everywhere
  - What is  $d/dx \tanh(x)$ ?
    - $d/dx \tanh(x) = \text{sech}^2(x)$



# Activation Functions: Rectified Linear Unit (ReLU)

**ReLU**

$$\max(0, x)$$

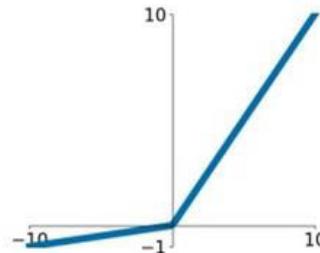


Learning can stall if activations are below zero because gradient stops flowing.

- Advantages:
  - For  $x > 0$ , reliable and cheap to calculate gradient
  - What is  $\frac{d}{dx} \max(0, x)$ ?
    - $\frac{d}{dx} \max(0, x) = \{1 \text{ if } x > 0; \text{ else } 0\}$
- Disadvantages:
  - Derivative is exactly 0 for  $x < 0$
  - Output in the positive direction is unbounded

# Activation Functions: Rectified Linear Unit (ReLU)

**Leaky ReLU**  
 $\max(0.1x, x)$

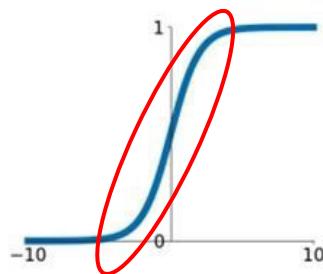


- Advantages:
  - Reliable and cheap to calculate gradient
  - What is  $\frac{d}{dx} \max(0.1x, x)$ ?
    - $\frac{d}{dx} \max(0.1x, x) = \{1 \text{ if } x > 0; \text{ else } 0.1\}$
- Disadvantages:
  - As with ReLU, outputs are still *unbounded* and could cause problems in later layers

# What Assumption Do Our Activation Functions Make?

## Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

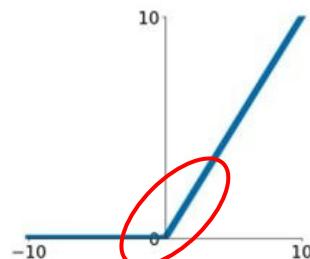


Our activation functions are all on their best behavior when the input  $x$  is close to zero.



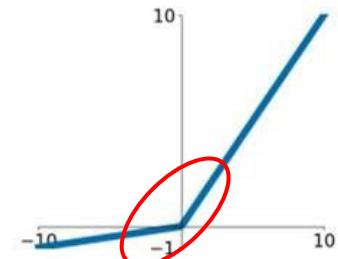
## ReLU

$$\max(0, x)$$



## Leaky ReLU

$$\max(0.1x, x)$$



# Weight Initialization: Zeros

- Since input to activation functions behaves well around zero, how about we start all our weights at zero?
- Initialize all weight matrix  $W$  and bias vector  $b$  parameters are initialized to zero for all layers (e.g., fully connected, CNN filters, etc.):
  - What is the effect on the final output vector?
    - Guaranteed to be all zeros for any input!
  - Gradients will vanish; model can't learn

# Weight Initialization

- We want well-behaved, random initial parameter values that are near zero but not uniformly zero
- What's the easiest way to accomplish that goal?
  - Draws from a Gaussian distribution!

$$W \sim \mathcal{N}(\mu, \sigma^2)$$

- What's a good value for  $\mu$ ?
  - $\mu=0$
- What's a good value for  $\sigma^2$ ?
  - $0 < \sigma^2 < 1$ , e.g.,  $\sigma^2=0.01$

# Weight Initialization: Gaussian Distribution

- Draws from a Gaussian distribution

$$W \sim \mathcal{N}(\mu, \sigma^2)$$

- Advantages:
  - Easy to implement
- Disadvantages:
  - Hyperparameter we need to set:  $\sigma^2$
  - If the weights are too small, the activation signals shrink quickly during training
  - If the weights are too large, the activation signals explode during training

# Weight Initialization: Controlling Activation Variance

- Glorot and Bengio argue that when the variance of the layer outputs (and hence the downstream layer inputs) is not  $\approx 1$ , depending on the activation function, models will converge more slowly, especially when the layer output variance is  $< 1$
- “Fixes” that initialize parameters to encourage layer outputs whose variance is  $\approx 1$  depend the activation function

# Weight Initialization: Controlling Activation Variance

- Activation functions with centered average (sigmoid; tanh):
  - Set the variance to inverse of the number of input units that contribute to each logit in the layer [*Glorot Initialization*]

$$W \sim \mathcal{N}(\mu, \sigma^2) \quad \text{---} \quad \text{Var}(W) = \frac{1}{n_{in}}$$

- Activation functions with unbounded average (ReLU):
  - Because average activation is a higher positive number, need to adjust up to  $\text{Var}(W)=2/n_{in}$  [*Kaiming initialization*]

[Xavier Glorot Initialization in Neural Networks — Math Proof](#) [blog post on Glorot]

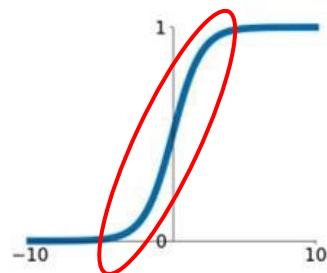
[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification](#) [Kaiming He paper]

[L11.6 Xavier Glorot and Kaiming He Initialization](#) [Brief Lecture on Differences Between These]

# What Assumption Do Our Activation Functions Make?

## Sigmoid

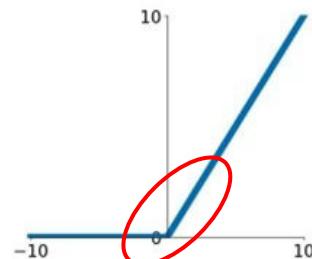
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Our activation functions are all on their best behavior when the input  $x$  is close to zero.

## ReLU

$$\max(0, x)$$



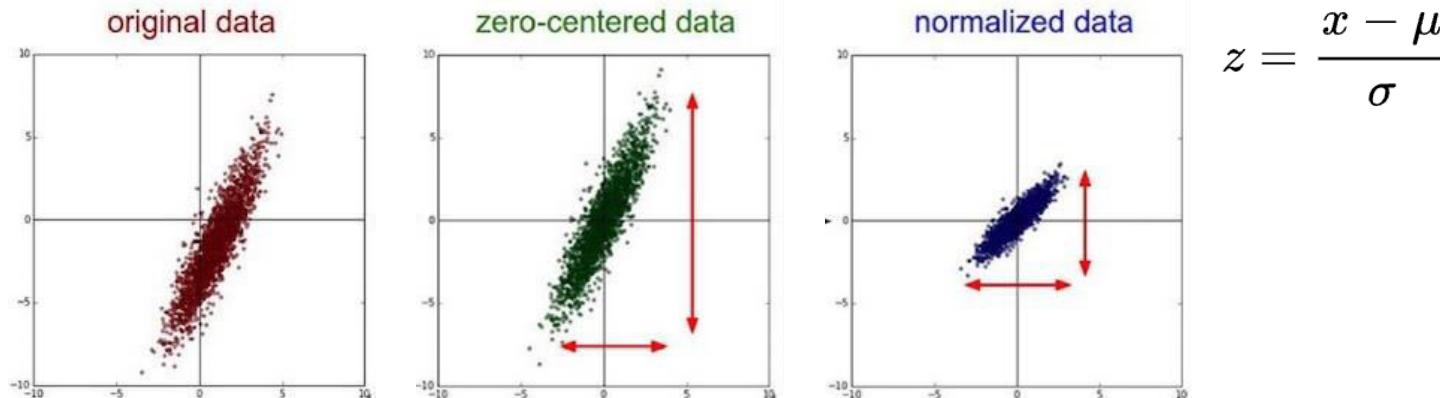
What aspect of the path from input to output won't weight initialization alone help with?

Max(...)



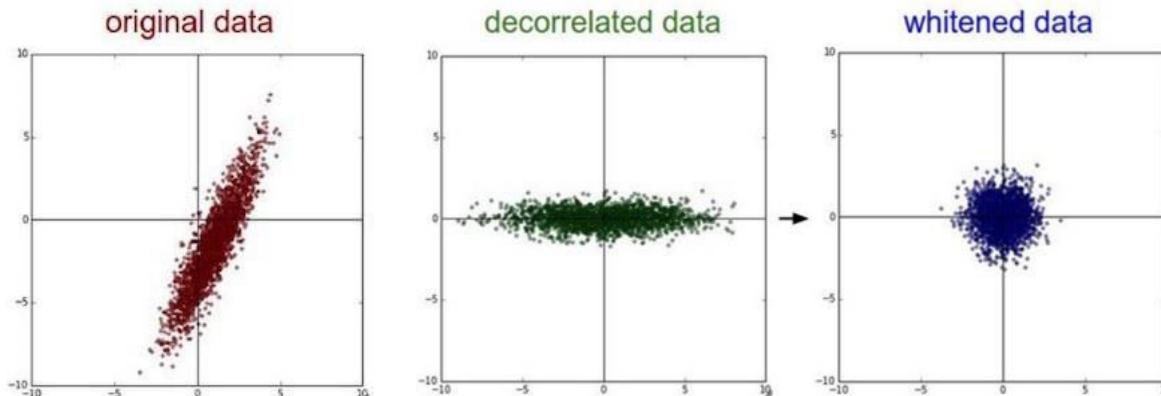
# Input Representation and Data Preprocessing

- We can make our input representation  $\phi$  a function that considers the entire training dataset to *standardize* it
- Zero-centering: shift the data to have features with mean 0
- Normalization: adjust features to have variance 1
  - (e.g., z-score normalization)



# Input Representation and Data Preprocessing

- We can make our input representation  $\phi$  a function that considers the entire training dataset to *standardize* it
- Remove correlations between input features via dimensionality reduction (e.g., PCA)

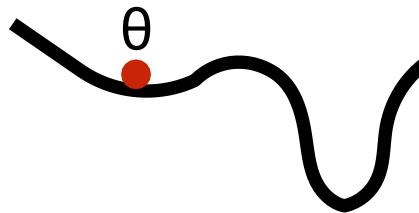


# Gradient Descent

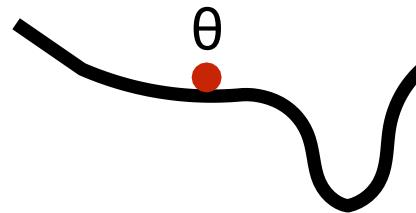
- Vanilla (true) Gradient Descent
    - Estimate  $\theta$  over the whole dataset
  - Stochastic Gradient Descent (SGD)
    - Estimate  $\theta$  at a random datapoint
  - Minibatch Gradient Descent
    - Estimate  $\theta$  as an average over a random batch of training data
    - Commonly we use “SGD” to refer to this setup
- $$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} f(\theta_t)$$
- $$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} f(\theta_t; x^{(i)})$$
- $$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} f(\theta_t; x^{(i:i+n)})$$

# Saddles and Local Minima

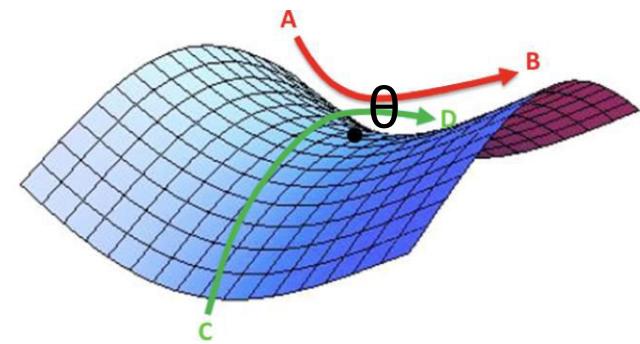
- Gradient descent drives the parameters  $\theta$  to towards the closest low point in the loss space
- Gets “stuck” if the loss is locally minimal (basin) or the gradient goes in orthogonal directions across dimensions (saddle)



local minimum



saddle points

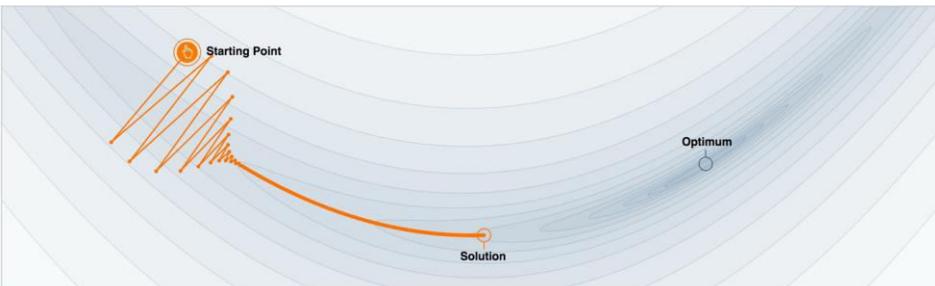


# Gradient Descent with Momentum

- Consider the history:
  - Add a fraction  $\gamma$  of the update vector of the past time step to the current update vector

$$\nu_t = \boxed{\gamma \nu_{t-1}} + \eta \nabla_{\theta_t} f(\theta_t) \quad \theta_{t+1} = \theta_t - \nu_t$$

GD *without* momentum



GD *with* momentum

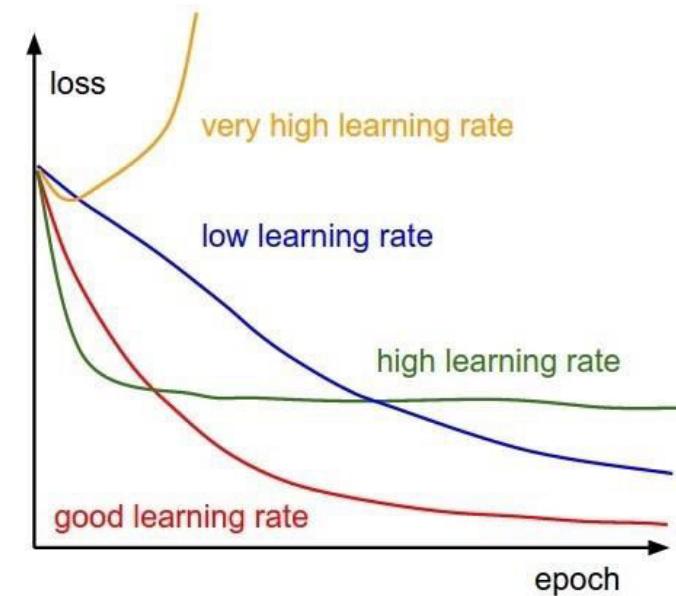


# Optimizer Parameters: Learning Rate

- How fast or slow we update the model parameters

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

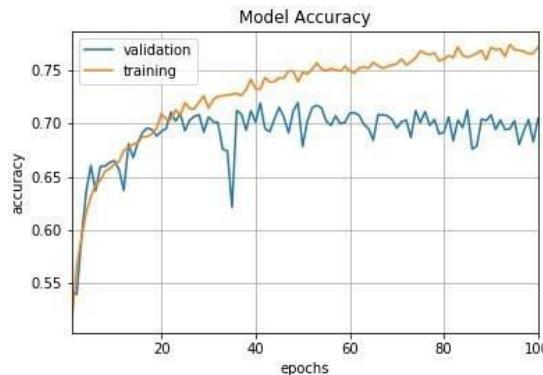
- If the learning rate is too large, the model parameters “explode” (move too fast; gain in magnitude)
- If the learning rate is too small, the model parameters move slowly and may be unable to escape local minima



# Optimizer Parameters: Learning Rate

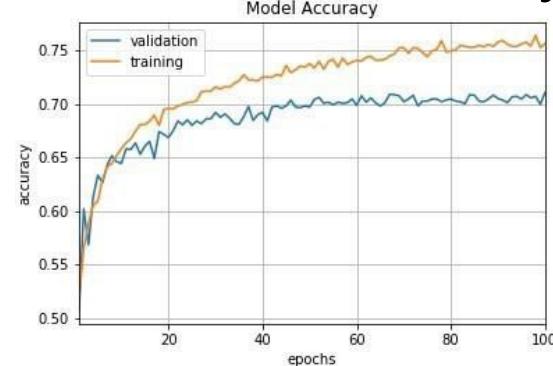
- Learning rate *schedulers* change the LR during training

Constant



$$lr = lr_0$$

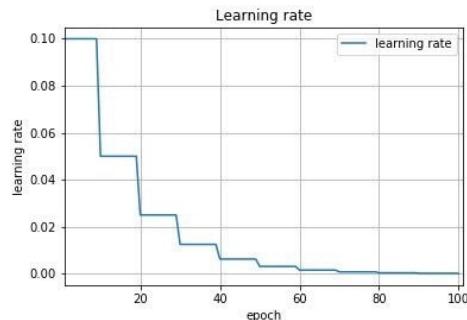
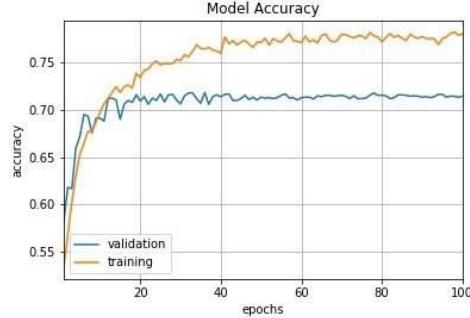
Time-based decay



$$lr = \frac{lr_0}{1 + k * iter}$$

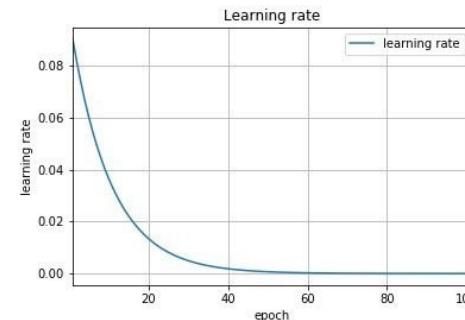
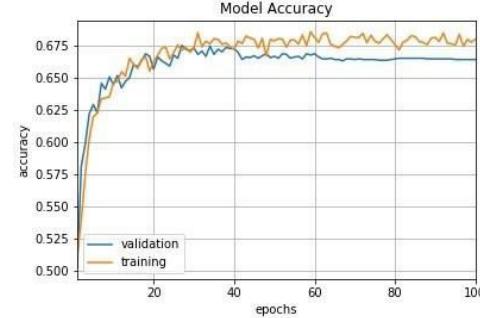
# Optimizer Parameters: Learning Rate

## Step decay



$$lr = lr_0 * \text{floor}\left(\frac{\text{epoch}}{\text{epoch\_drops}}\right)$$

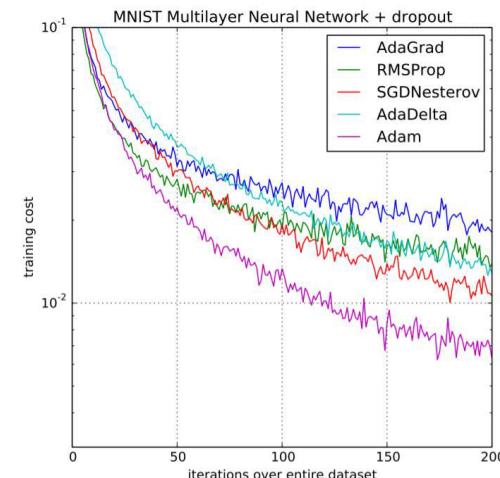
## Exponential decay



$$lr = \frac{lr_0}{\exp^{k*iter}}$$

# Optimizer Selection In Practice

- Aside from tutorial code, you probably haven't seen "SGD" as the selection for an optimizer
- SGD with momentum is the backbone of most optimizers, with adaptive learning rate schedulers cooked in
- So which optimizer is best?
  - **It depends!**
- In practice, the optimizer and its parameters are *hyperparameters* you'll jitter around when training a network



# Parameter Regularization During Training

- We can add a regularization “penalty” to our loss  $L$

$$L = L_{original} + \lambda R(\theta)$$

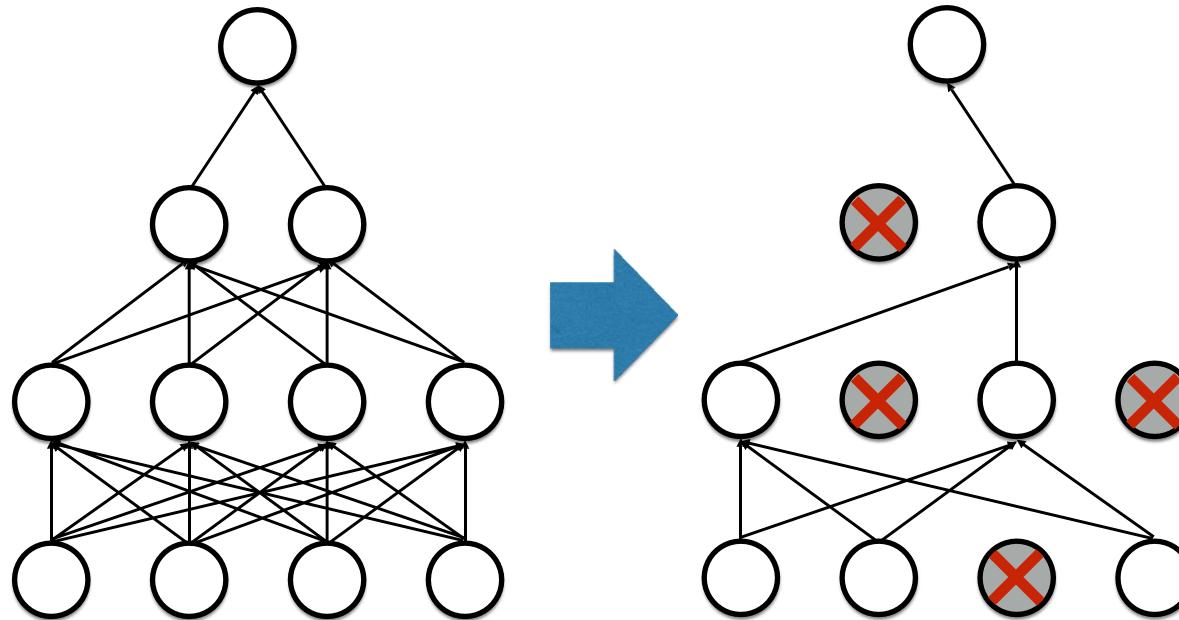
- L1 parameter regularization  $R(\theta) = \sum_i \sum_j |\theta_{i,j}|$

- L2 parameter regularization  $R(\theta) = \sum_i \sum_j \theta_{i,j}^2$

- L1+L2 parameter regularization  $R(\theta) = \sum_i \sum_j |\theta_{i,j}| + \alpha \theta_{i,j}^2$

# Implicit Ensemble Models: Dropout

- At each iteration, randomly shutdown some neurons with a certain probability (usually 0.5)



# Implicit Ensemble Models: Dropout

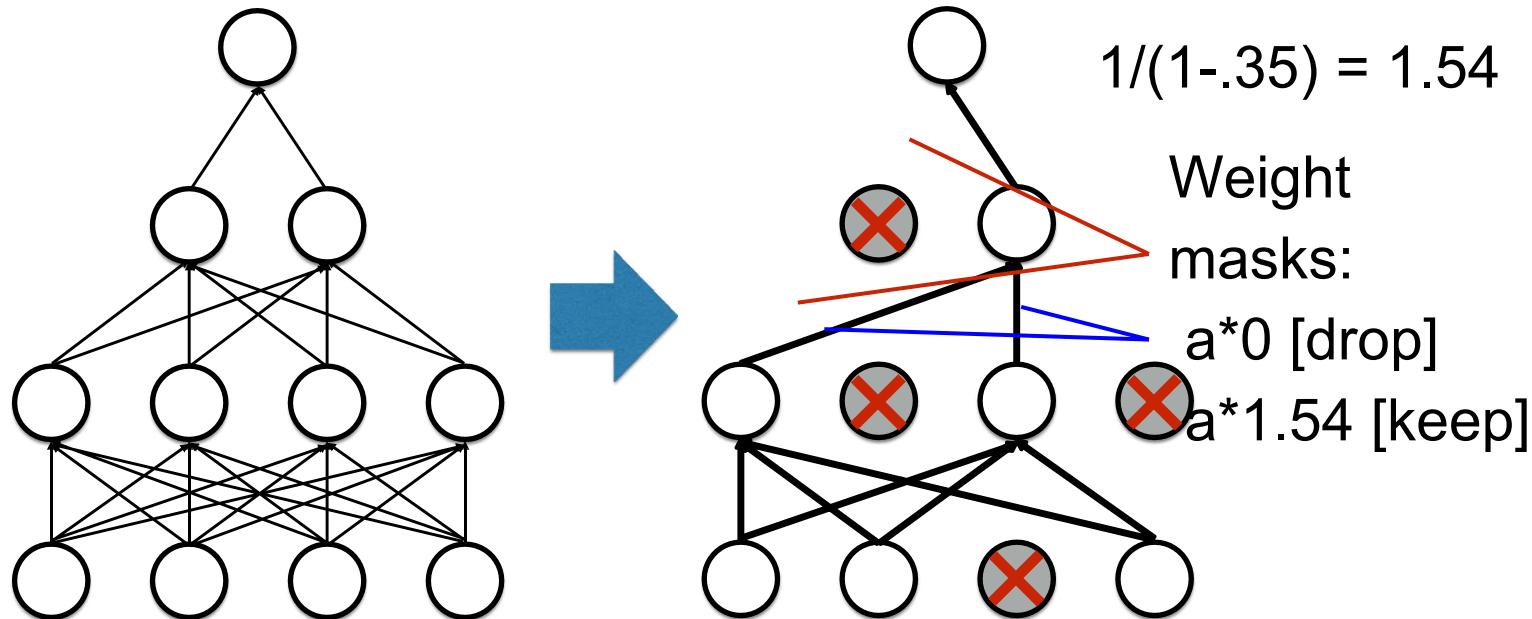
- At each iteration, randomly shutdown some neurons with a certain probability (usually 0.5)
- Intuition:
  - Forces the network to have a redundant representation
    - Same pieces of information are “stored” in multiple different neurons
  - Training a bunch of small models that share parameters
  - Each binary mask is one such “small model”
- At inference time, these models are “ensembled”

# Training Versus Inference Considerations for Dropout

- If we randomly drop 0.5 nodes during training time but use all during test time, what might go wrong?
  - Test time inputs at each layer will be, on average, x2 what was seen at training time!
- To compensate, during training time we apply an *inverted dropout* correction
  - At training time, for every node not dropped out, increase its output by a magnifying factor of  $1/(1-d)$  for  $d$  probability of dropping each unit; e.g., x10 for dropout 0.9

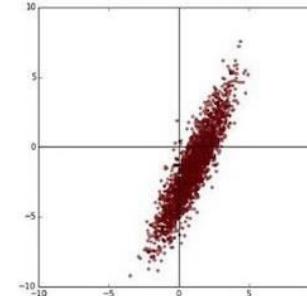
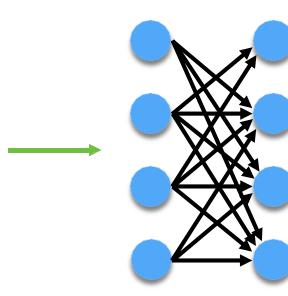
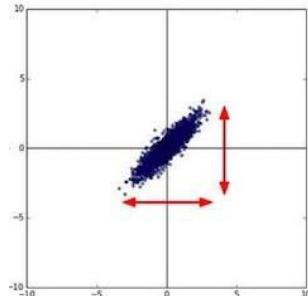
# Training Versus Inference Considerations for Dropout

- To compensate, during training time we apply an *inverted dropout* correction; e.g., for dropout value 0.35:

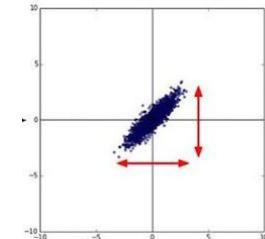


# Normalization Between Layers

- After each layer, the distribution of activation signals changes (Internal covariate shift)
- As a network becomes deeper, distribution shifts more
- Just as we preprocess the input data with our representation function to make features well-behaved, we can consider re-normalizing outputs per layer



$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$



# BatchNorm Layer

- At training time, compute the estimated mean and variance of the activations of a layer across the current batch

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)2} + \epsilon}} \quad \mu_B = \frac{1}{m} \sum_{i=1}^m x_i$$
$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

- How could we make this transformation a little more flexible?
- Introduce learnable parameters to the BatchNorm Layer that enable *scaling* and *shifting* individual input indices

# BatchNorm Layer

Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ; calculate mean/var of the activations.  
Should we do it over the batch?

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i$$

No! Might be out of expected distribution. Use empirical averages from training data!

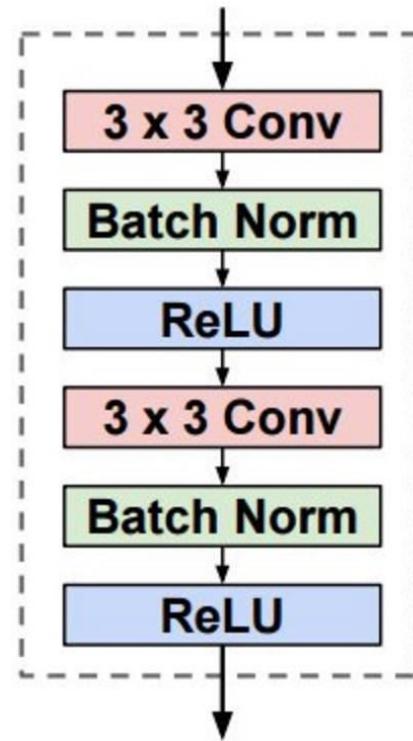
$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

- Gamma and Beta are learnable vectors
- These params allow the transformed input to have flexible, learned mean & var
- BatchNorm layer is differentiable

# BatchNorm Layer

- Usually added after convolutional layers or fully connected layers but before non-linearities
- Why?
  - Point of BatchNorm is to help data get into a space where the activation function behaves well



# INDUSTRY PRACTICE: Layer Normalization

- Like BatchNorm, except compute the estimated mean and variance of the activations of a *neurons* in a *single layer*

$$\mu^l = \frac{1}{H} \sum_{i=1}^H a_i^l \quad \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2}$$
$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

- Learnable  $\gamma$ ,  $\beta$  vectors; more stability and faster convergence
- Same operation at training and inference time
- Has become *industry standard* to include LayerNorm between stacked Transformer layers

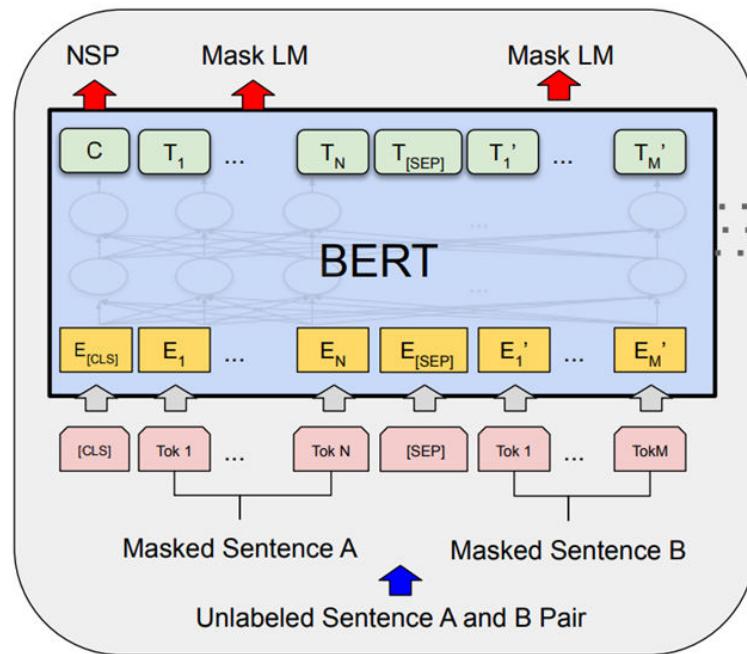
# Lecture Outline

- Recap: Transformers for Language Modeling
- DL 102 for NLP: Activations, Normalization, and Regularization
- **Large Language Models: Pretraining and Finetuning**
- Zoom Out

# Pretraining and Finetuning

- These terms have become woefully overloaded, but
- Pretraining:
  - *Task-independent training*, often self-supervised
  - n-gram models (e.g., Word2Vec CBOW), cloze models (e.g., BERT training), next word prediction (e.g., GPT)
- Finetuning:
  - *Task-specific training*, often supervised
  - Pretraining provides *weight initialization* for task model
  - Often used with the addition of new layers and losses

# Pretraining and Finetuning (BERT)

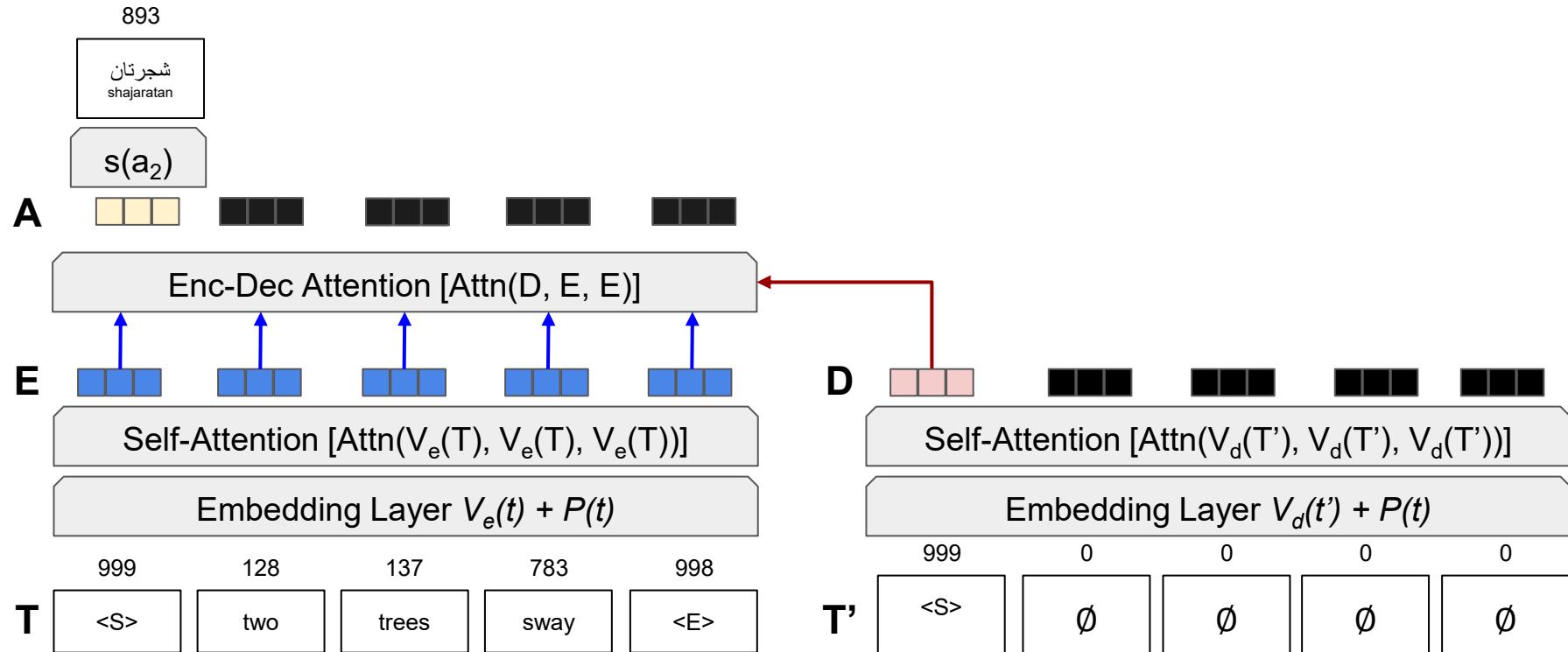


Pre-training

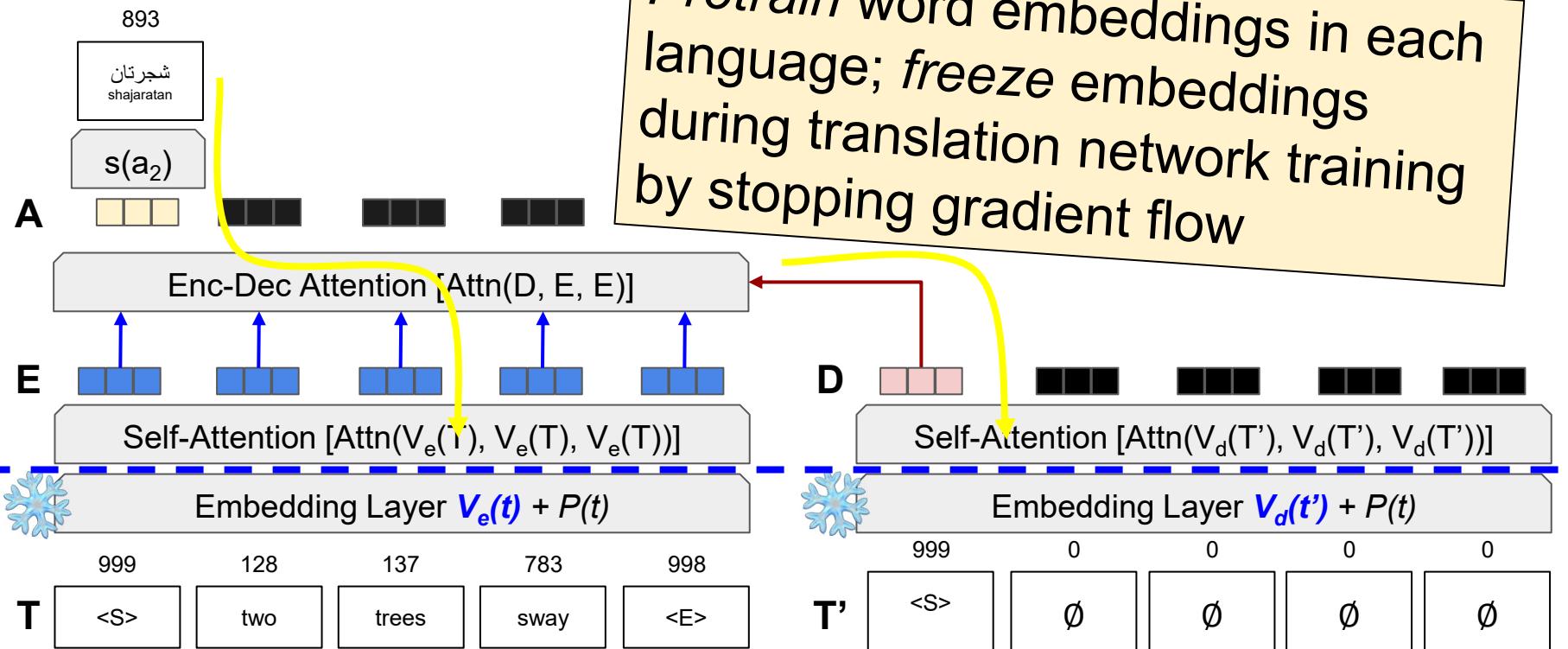
# Parameter Efficient Fine-tuning

- Pretraining can happen *at scale*, e.g., from a company, and typically has orders of magnitude more *data* than each task
- We need to reconcile the small data and compute of learning weights for our *task* versus the pretrained weights
- Most methods revolve around the intuition that the pretrained weights are more important or better calibrated than the gradients for the task

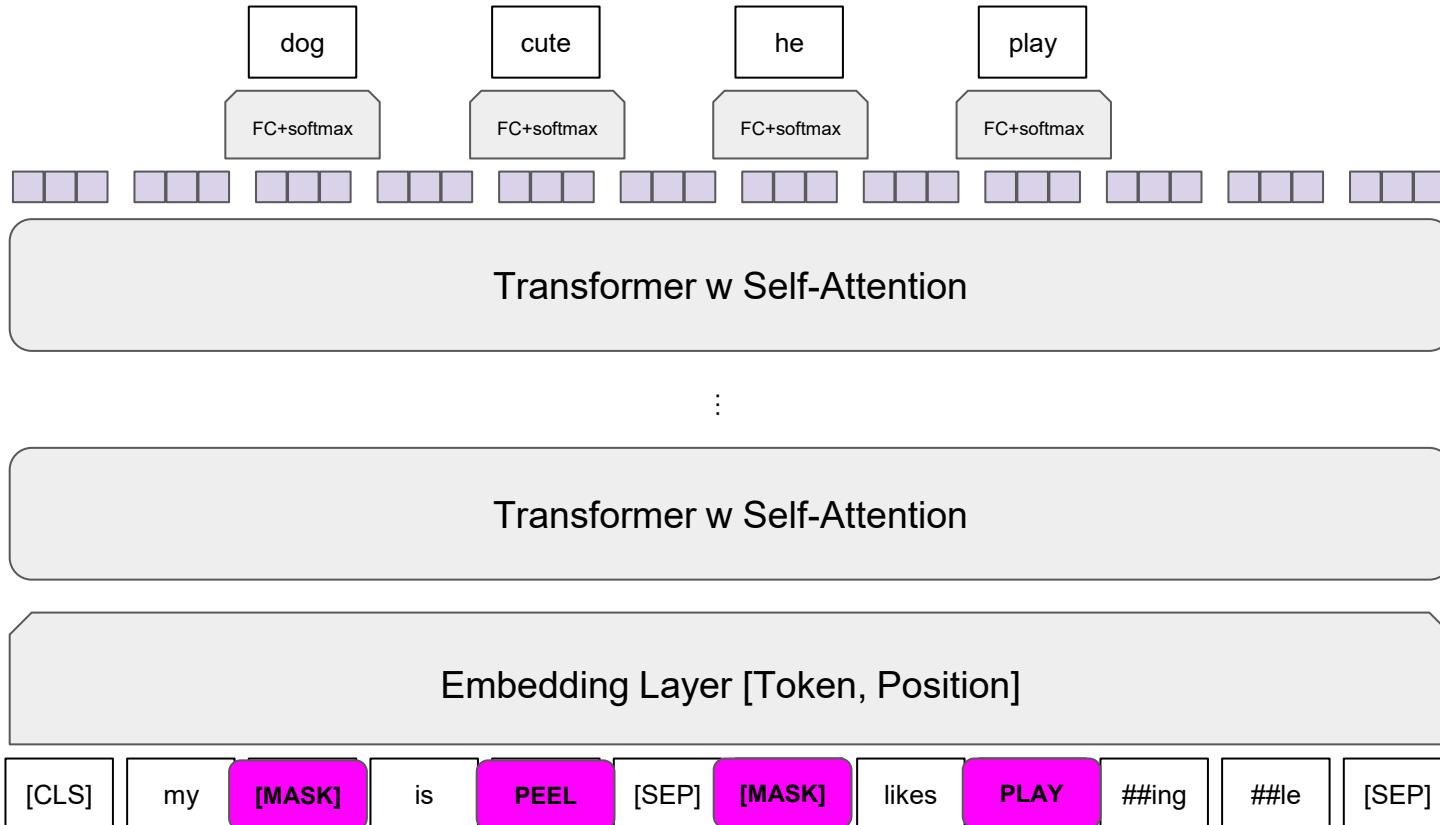
# Parameter Efficient Fine-tuning: Weight Freezing



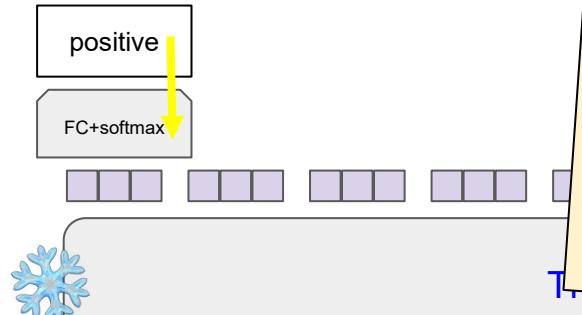
# Parameter Efficient Fine-tuning: Weight Freezing



# Parameter Efficient Fine-tuning: Weight Freezing



# Parameter Efficient Fine-tuning: Weight Freezing



*Pretrain entire BERT network and freeze it; learning only a new classification “head” over the CLS token during fine-tuning*

Transformer w Self-Attention

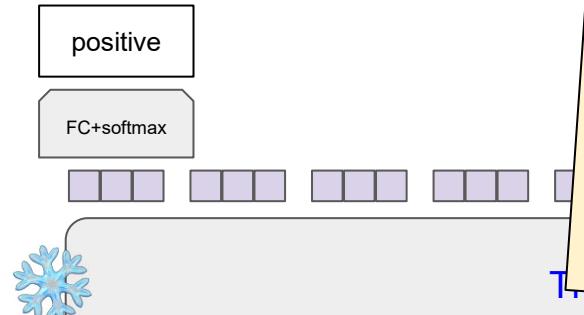
Embedding Layer [Token, Position]

[CLS] my dog is cute [SEP] he likes play ##ing ##le [SEP]

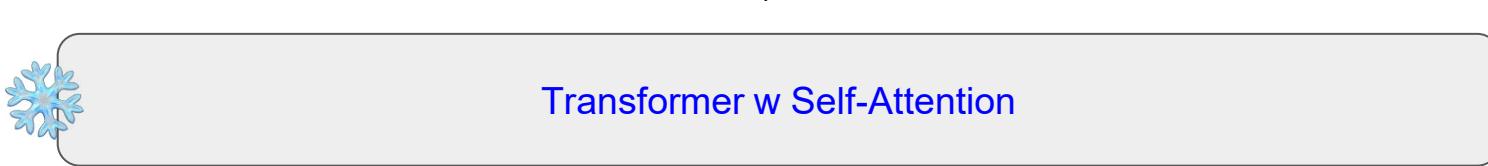
# Parameter Efficient Fine-tuning: Weight Dampening

- Rather than kill the gradient on “frozen” layers, we could just dampen it (e.g.,  $\text{grad} * 0.1$ ) to learn “slower” on pretrained parameters that we sort of trust
- Some algorithms perform scheduled dampening too, letting gradient updates to pretrained parameters return to full size as training runs longer
- Similarly, layers can be on an “unfreezing schedule” to allow gradient flow only after newly initialized network parts have trained for some epochs

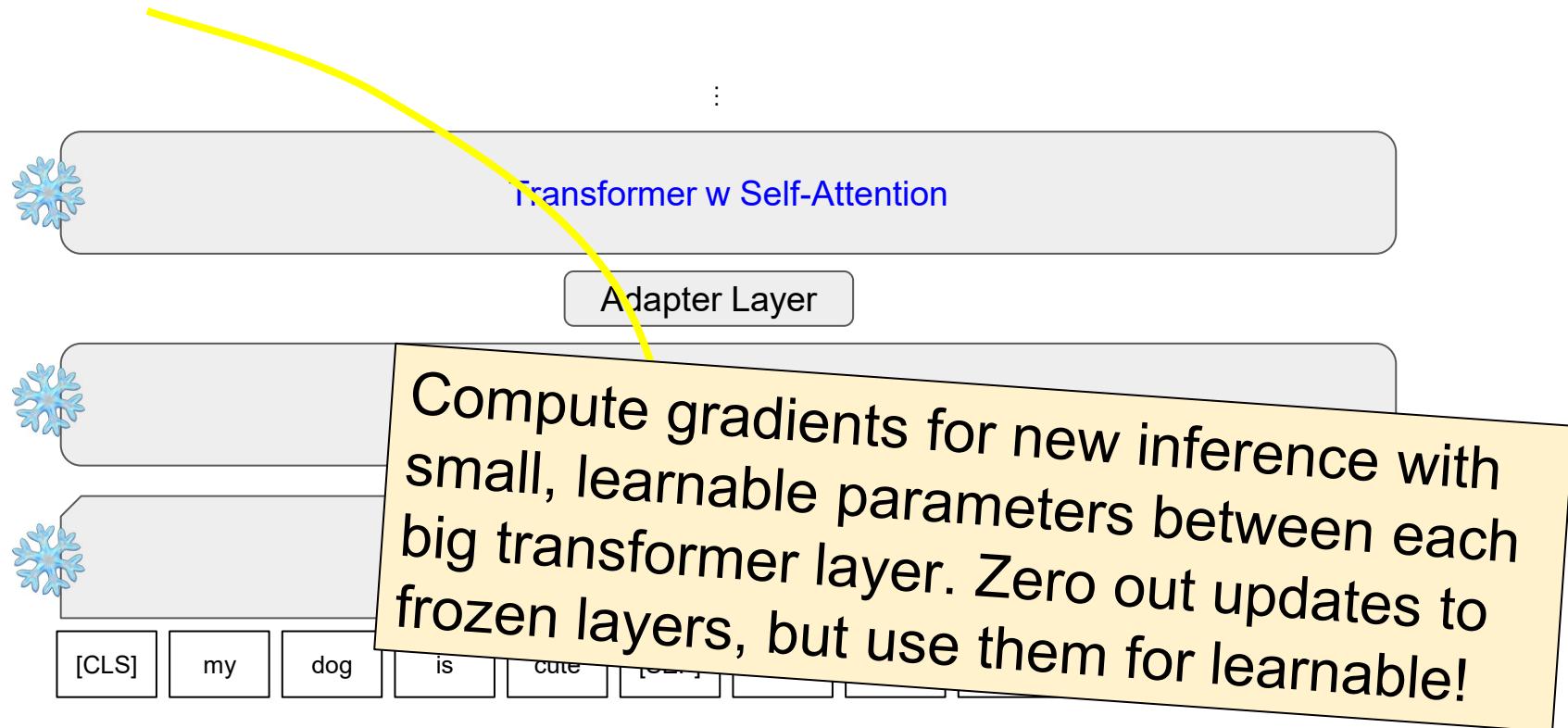
# Parameter Efficient Fine-tuning: Adapter Layers



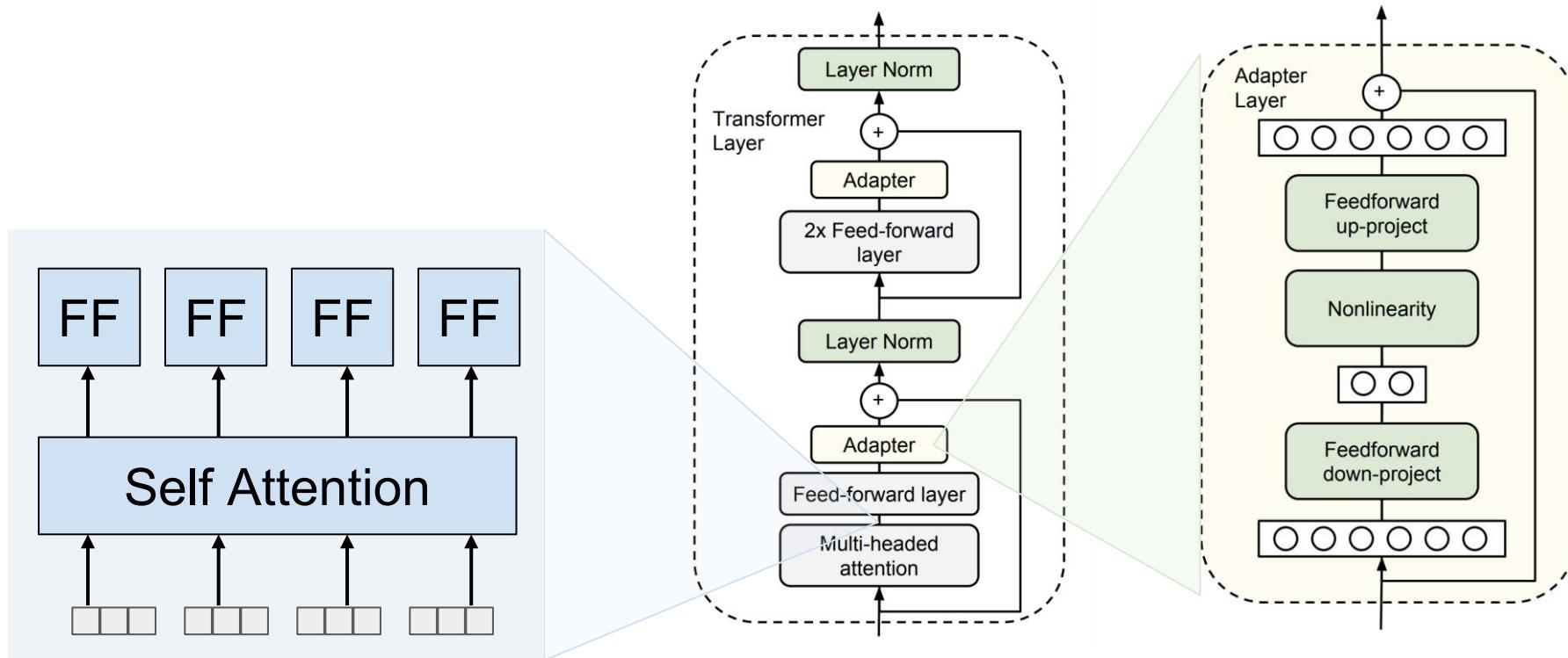
What if we need to learn new,  
lower-level language processing  
for the task than simply a final  
classification decision?



# Parameter Efficient Fine-tuning: Adapter Layers



# Parameter Efficient Fine-tuning: Adapter Layers

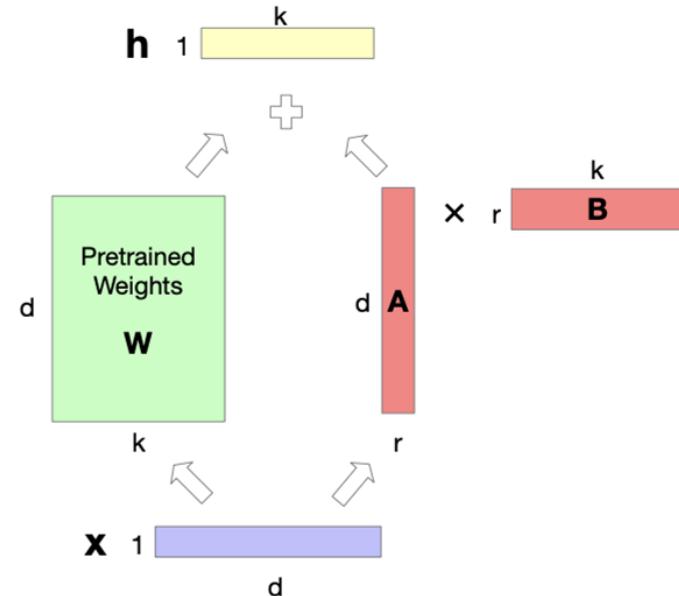


# Low Rank Adaptation (LoRA)

- Adapters address the *computational* constraints of updating a large pretrained model by avoiding touching those huge matrices
- Weight freezing avoids both computation and the possibility of *catastrophic forgetting*—overriding good pretrained weights with garbage as we learn a task
- Low Rank Adaptation also hits both these advantages: don't update the giant weight matrices but do let the model change

# Low Rank Adaptation (LoRA)

- Instead of updating each  $W_k$ ,  $W_Q$ ,  $W_V$ ,  $W_O$  matrix, update a low rank pair of matrices whose sizes unpack to the expected
  - $h = xW$  becomes  $h = xW + AB$
- Choose  $r$  to be pretty small
- Then  $AB$  is size  $d*k$ , same as  $W$
- But  $A$  and  $B$  have only  $2*r*k$  params, versus  $W$ 's  $d*k$
- E.g., for  $d=k=128$  hidden dim, if we set  $r=8$ , we'd get  $|AB|=2048$ ,  $|W|=16384$



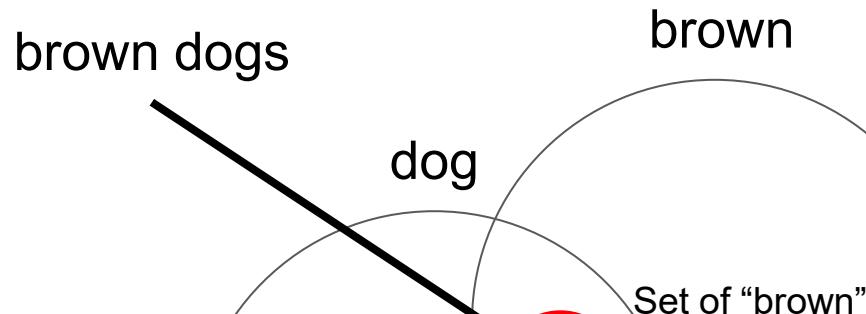
# Lecture Outline

- Recap: Transformers for Language Modeling
- DL 102 for NLP: Activations, Normalization, and Regularization
- Large Language Models: Pretraining and Finetuning
- **Zoom Out**

# Transformers As a Generic Umbrella

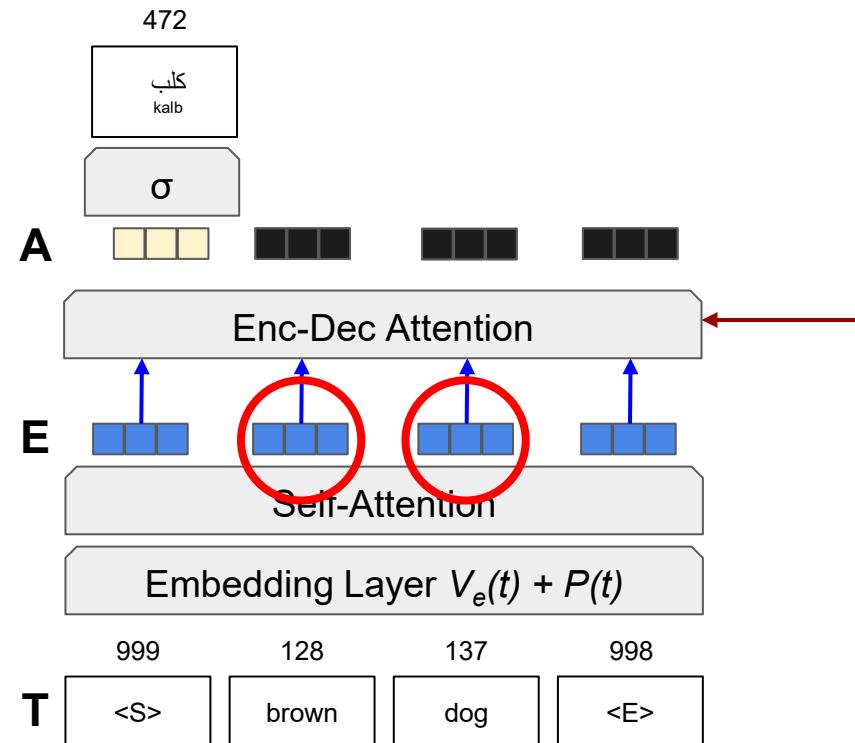
- Transformers implement, at the cost of higher computation, a chunk of intuitions we've had for sequence processing for language for a long time!
- It's cool!

# Categorical Semantics Lite

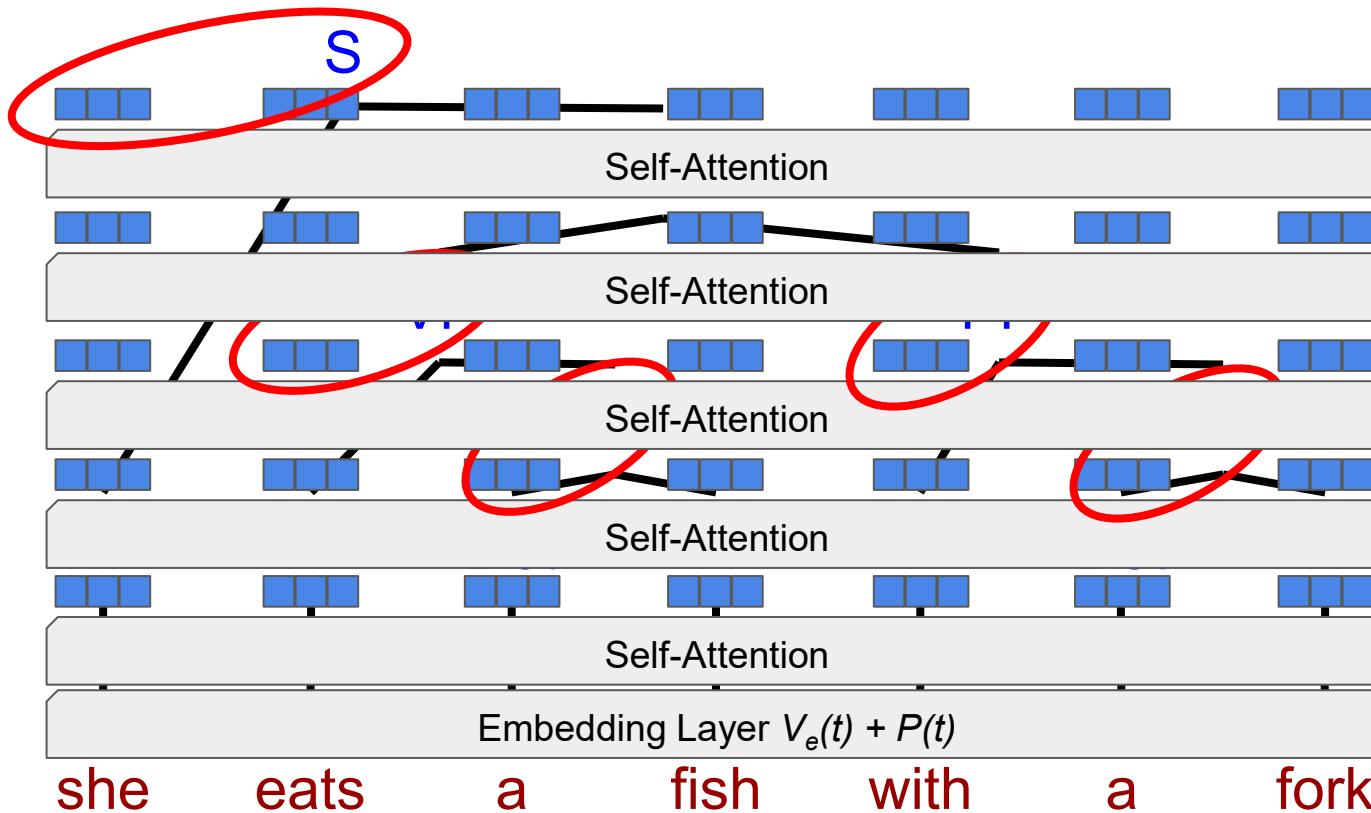


**Nouns are vectors, adjectives are matrices:**  
**Representing adjective-noun constructions in semantic space**

**Marco Baroni and Roberto Zamparelli**  
Center for Mind/Brain Sciences, University of Trento  
Rovereto (TN), Italy  
[{marco.baroni,roberto.zamparelli}@unitn.it](mailto:{marco.baroni,roberto.zamparelli}@unitn.it)



# Symbol Systems from Syntax and Semantics



# Predicate Logic and Montague Grammar

the( $\lambda x.(\text{office}(x) \wedge \text{owns}(\text{alice}, x))$ )

$\lambda y.(\text{the}(\lambda x.(\text{office}(x) \wedge \text{owns}(y, x))))$

alice

$\lambda P. \lambda y. (\text{the}(\lambda x. (P(x) \wedge \text{owns}(y, x))))$

# office

Alice

'S

## office

# Encoder-only Models Maybe Actually Do This???

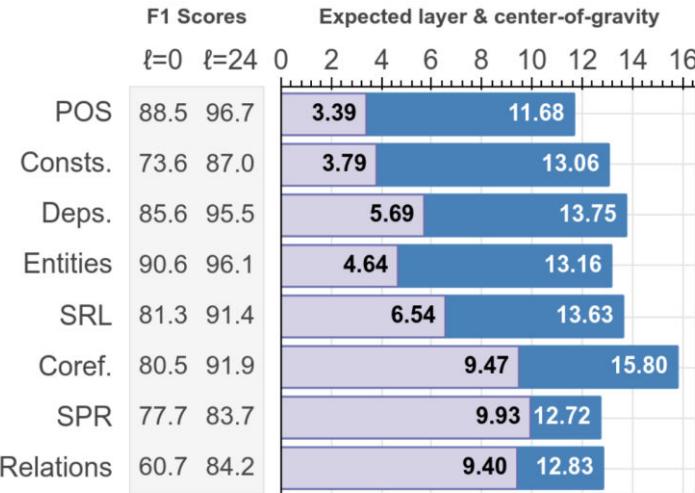
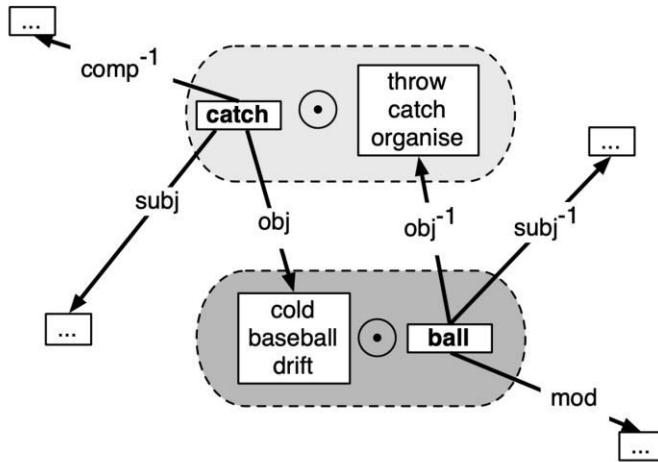


Figure 1: Summary statistics on BERT-large. Columns on left show F1 dev-set scores for the baseline ( $P_{\tau}^{(0)}$ ) and full-model ( $P_{\tau}^{(L)}$ ) probes. Dark (blue) are the mixing weight center of gravity (Eq. 2); light (purple) are the expected layer from the cumulative scores (Eq. 4).

# The Field Wanted Something Like BERT For Decades!



$(v_a, R_a, R_a^{-1})$  and  $(v_b, R_b, R_b^{-1})$  be the representations of the two words, and let  $r \in \mathcal{R}$  be the relation linking  $a$  to  $b$ . Then, we define the meaning of  $a$  and  $b$  in this context as a pair  $(a', b')$  of vectors, where  $a'$  is the meaning of  $a$  in the context of  $b$ , and  $b'$  the meaning of  $b$  in the context of  $a$ :

In the *structured vector space (SVS)* model, we represent the meaning of a lemma  $w$  as a triple

$$w = (v, R, R^{-1})$$

where  $v \in D$  is a lexical vector describing the word  $w$  itself,  $R : \mathcal{R} \rightarrow D$  maps each relation label onto a vector that describes  $w$ 's selectional preferences, and  $R^{-1} : \mathcal{R} \rightarrow D$  maps from role labels to vectors describing inverse selectional preferences of  $w$ .

$$\begin{aligned} a' &= (v_a \odot R_b^{-1}(r), R_a - \{r\}, R_a^{-1}) \\ b' &= (v_b \odot R_a(r), R_b, R_b^{-1} - \{r\}) \end{aligned}$$

# The Field Wanted Something Like BERT For Decades!

In the *structured vector space* (SVS) model, we represent the meaning of a lemma  $w$  as a triple

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$$w = (v, R, R^{-1})$$

$X = \text{Embed(tokens)}$

Attn( $XW_Q$ ,  $XW_K$ ,  $XW_V$ )

where  $v \in D$  is a lexical vector describing the word  $w$  itself,  $R : \mathcal{R} \rightarrow D$  maps each relation label onto a vector that describes  $w$ 's selectional preferences and  $R^{-1} : \mathcal{R} \rightarrow D$  maps from role labels to vectors describing inverse selectional preferences of  $w$ .

Reminder:  $QK^T \neq KQ^T$  because relations are not symmetric!

$$\begin{aligned} a' &= (v_a \odot R_b^{-1}(r), R_a - \{r\}, R_a^{-1}) \\ b' &= (v_b \odot R_a(r), R_b, R_b^{-1} - \{r\}) \end{aligned}$$

# Lecture Outline

- Recap: Transformers for Language Modeling
- DL 102 for NLP: Activations, Normalization, and Regularization
- Large Language Models: Pretraining and Finetuning
- Zoom Out

# Recent High-Profile NLP Technology?



Prove that  $1 = 2$  implies  $2 = 1$ .

< 3 / 3 >



It is impossible to prove that  $1 = 2$  implies  $2 = 1$ , as the premise of the statement is false. In mathematics, 1 is always equal to 1 and never equal to 2, and therefore the premise of the statement is not true. Additionally, the statement contradicts the fundamental principle of equality, which states that if  $a = b$ , then  $b = a$ . Therefore, the statement is not logically or mathematically sound.



- Pretrain a huge decoder-only transformer on a bunch of Internet text
- Apply our decoding strategies learned in the previous lecture (e.g., top-p sampling)
- Will we get this output?
- Why not?

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 2: Lecture 2  
DL 102 for NLP & Pretraining and Finetuning

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 3: Lecture 1  
RL 101 for NLP

# Lecture Outline

- Recap: Module 2 Transformers as LMs
- RL 101 for NLP

# Lecture Outline

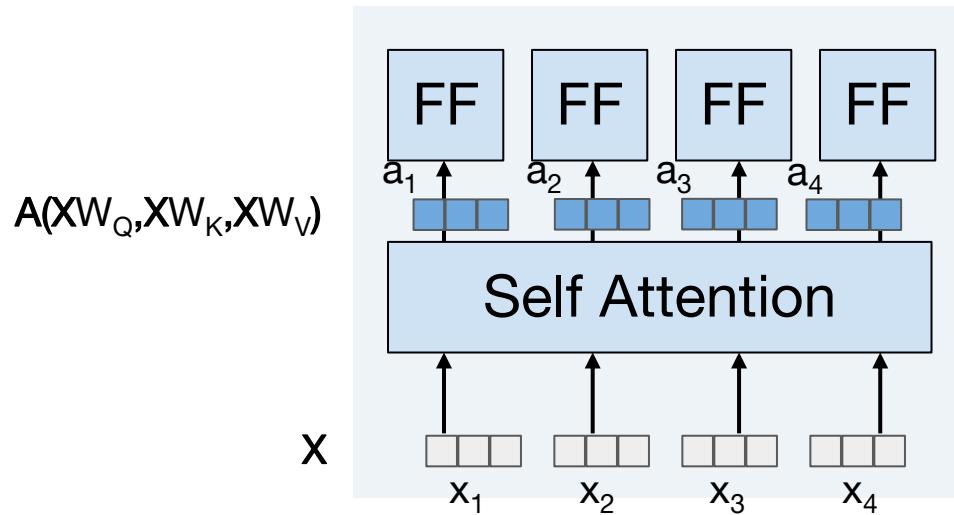
- Recap: Module 2 Transformers as LMs
- RL 101 for NLP

# Transformers Lite

- Transformer  $T : S_a \rightarrow S_b$  is a function from a set  $a$  in  $S_a$  to a set  $b$  in  $S_b$
- Simple transformer consists of a single layer encoder  $E$  and a single layer decoder  $D$  with single-headed attention

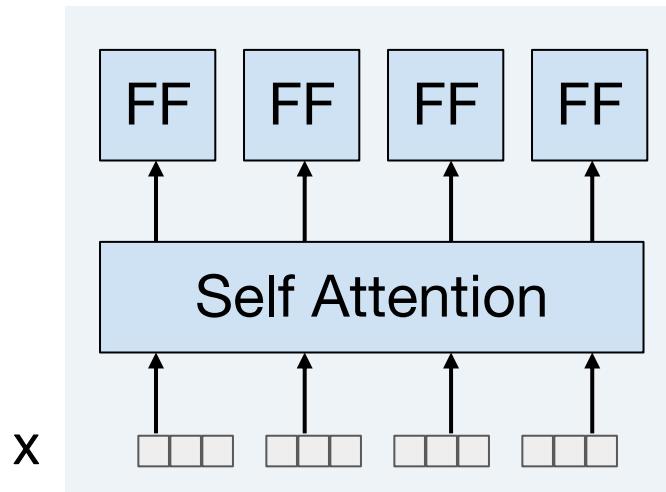
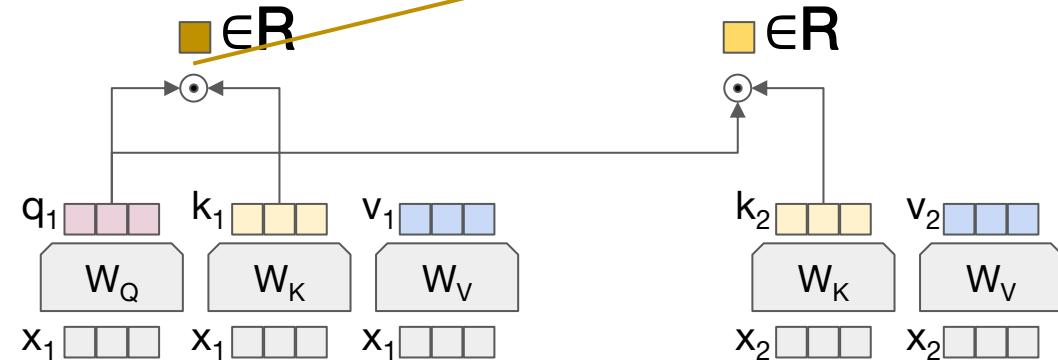
# Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$



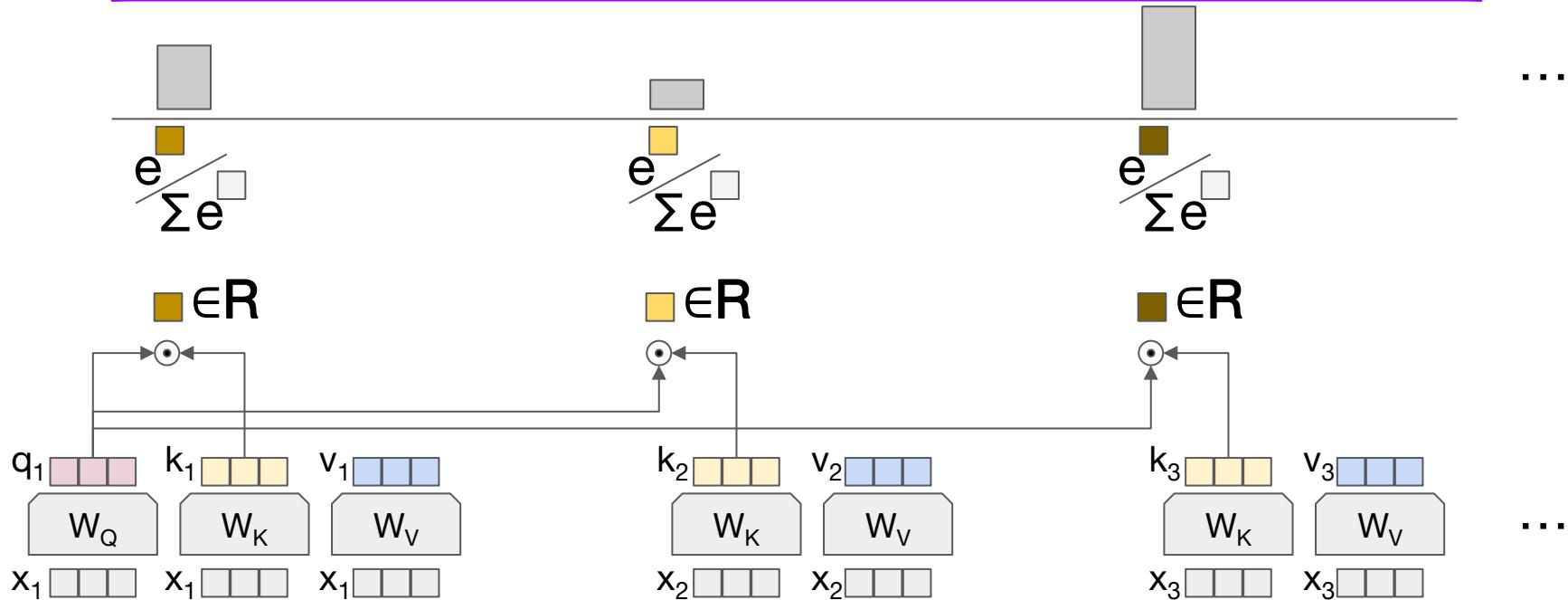
# Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

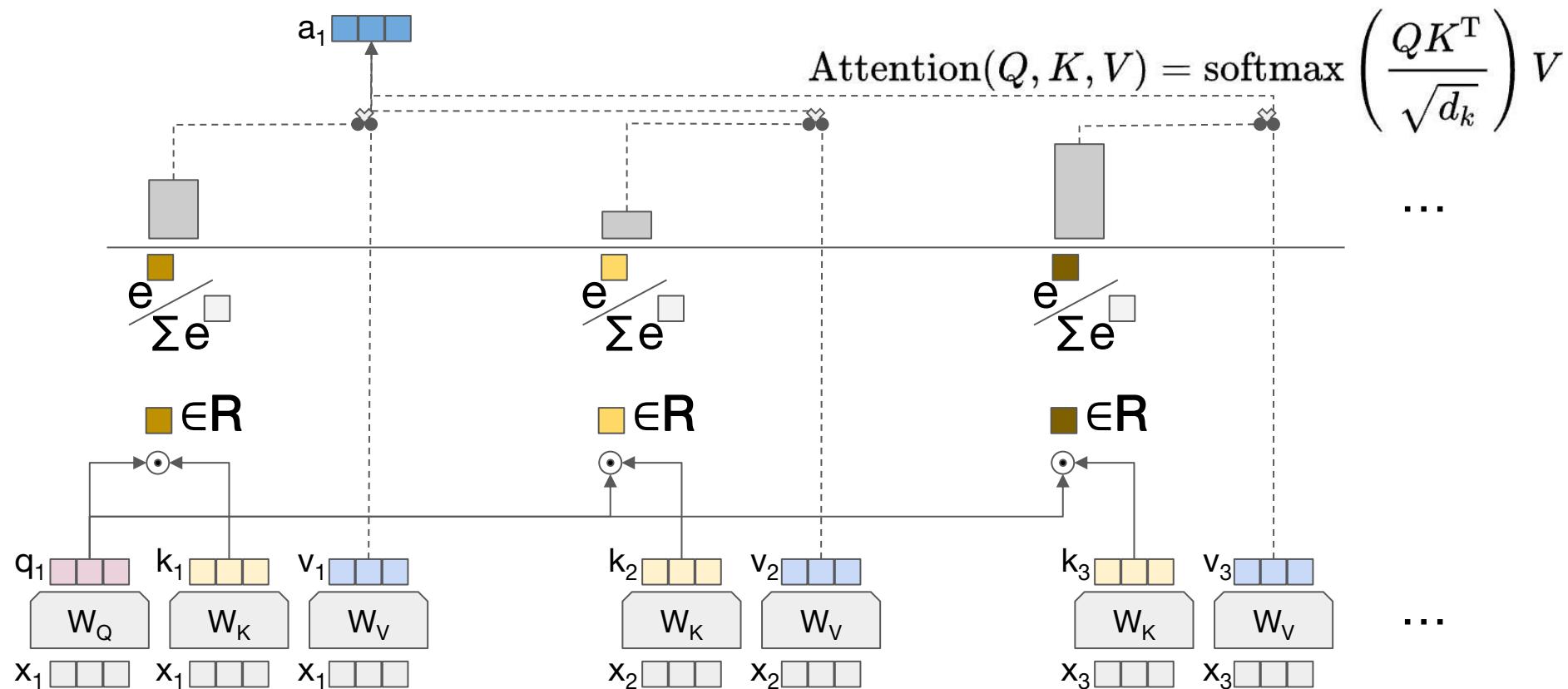


# Self-Attention

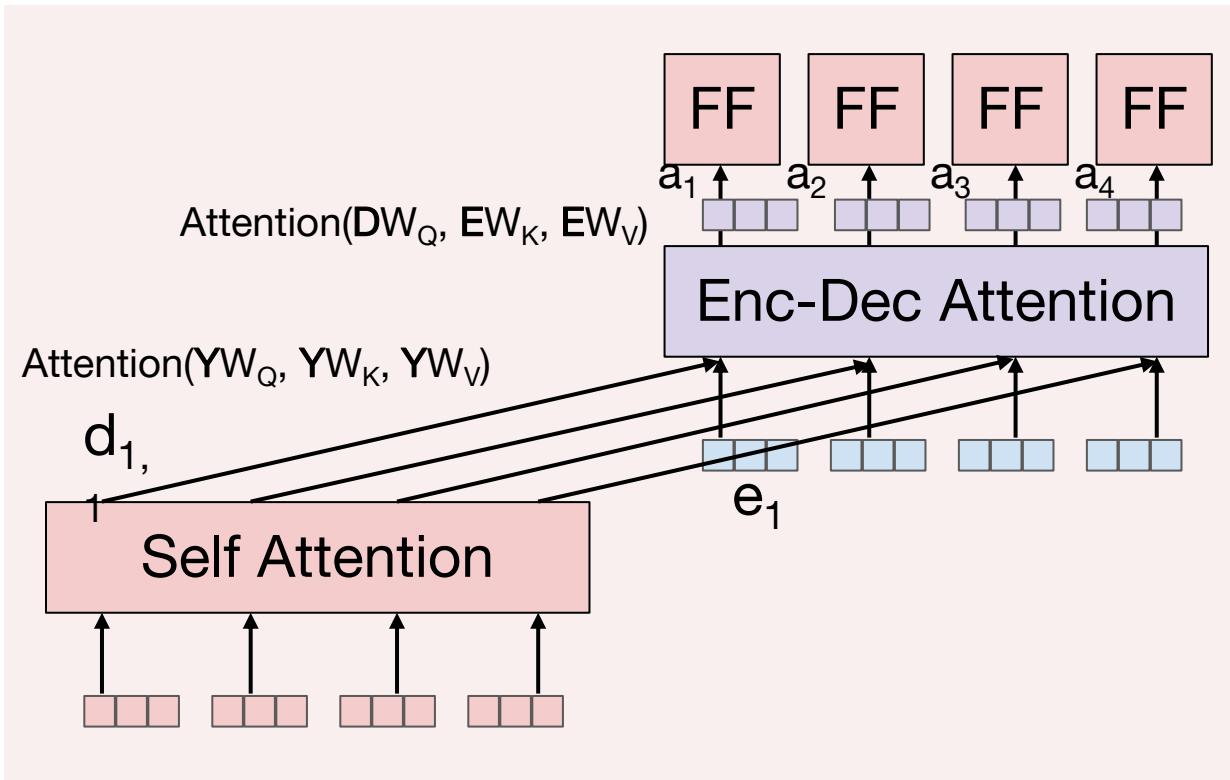
$$\text{Attention}(Q, K, V) = \underbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V}_{\dots}$$



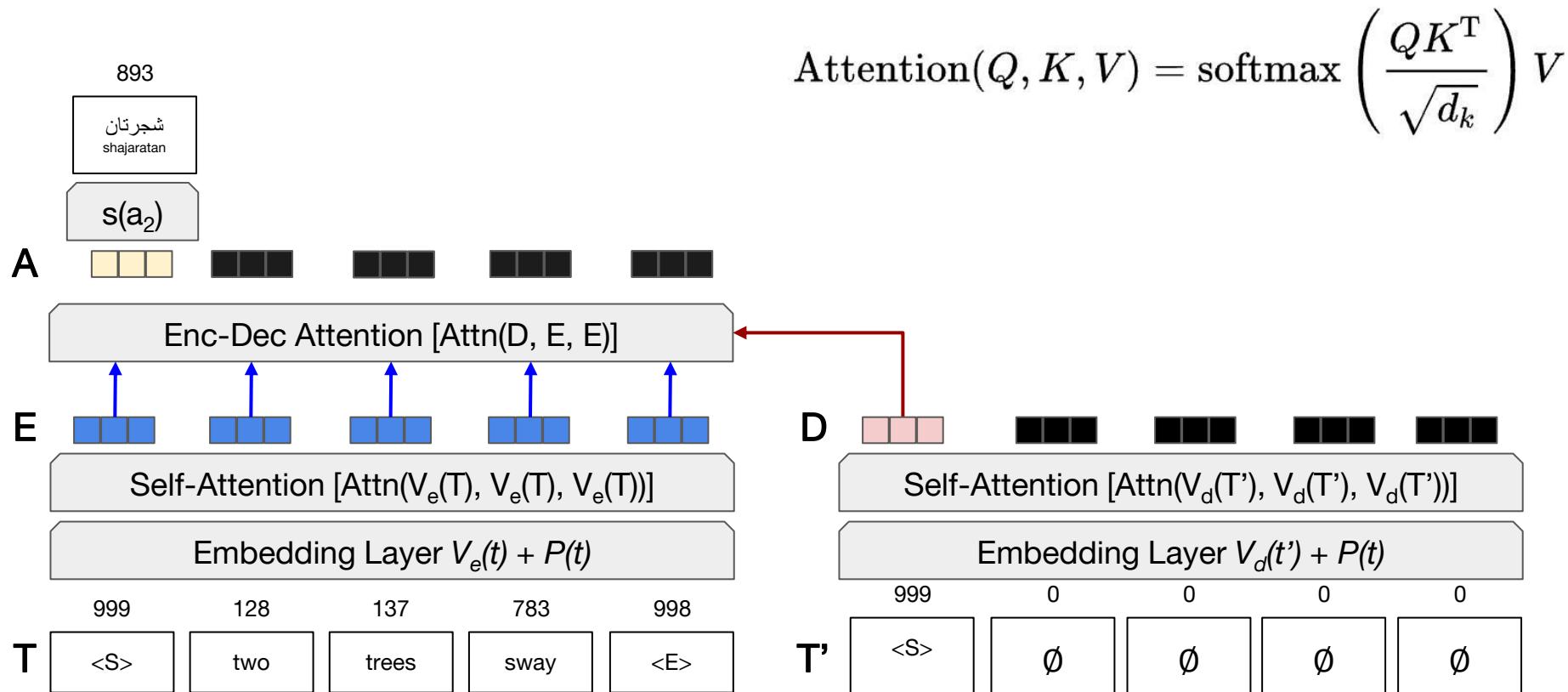
# Self-Attention



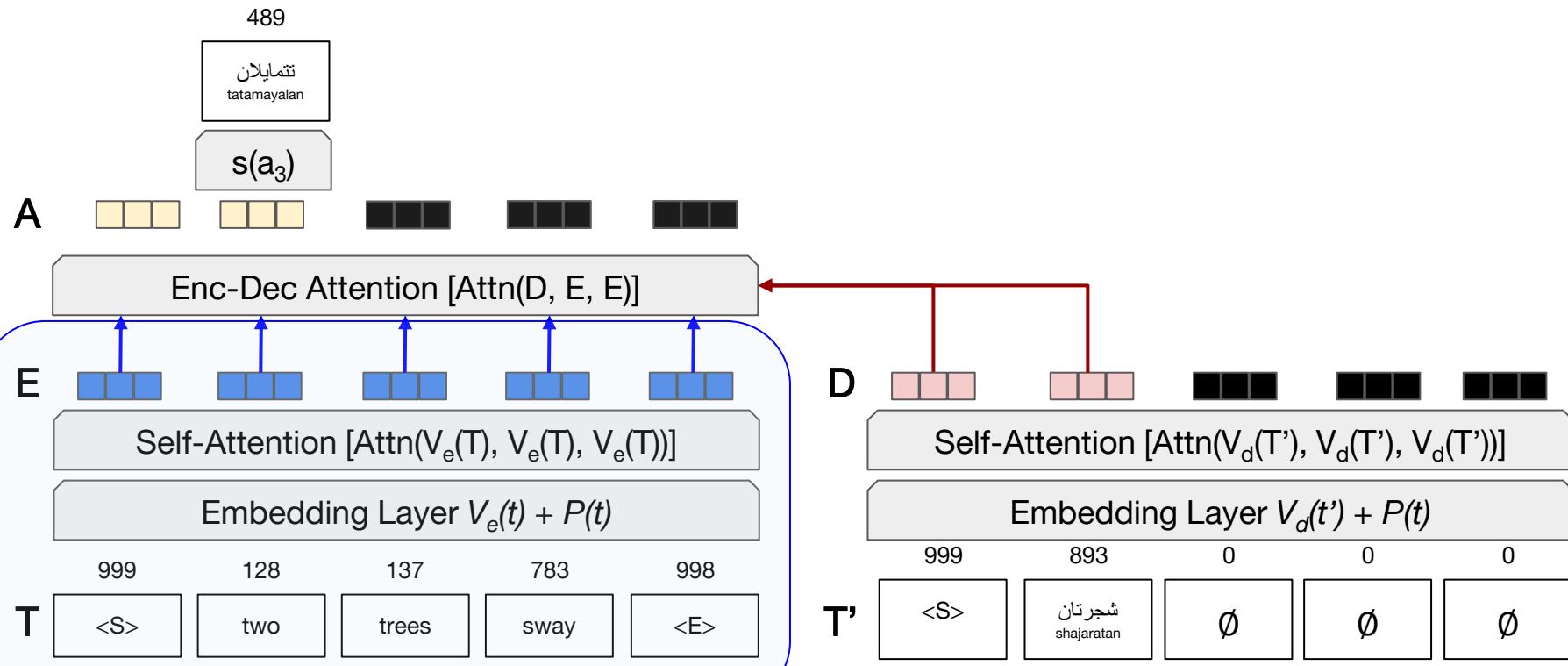
# Encoder-Decoder Attention



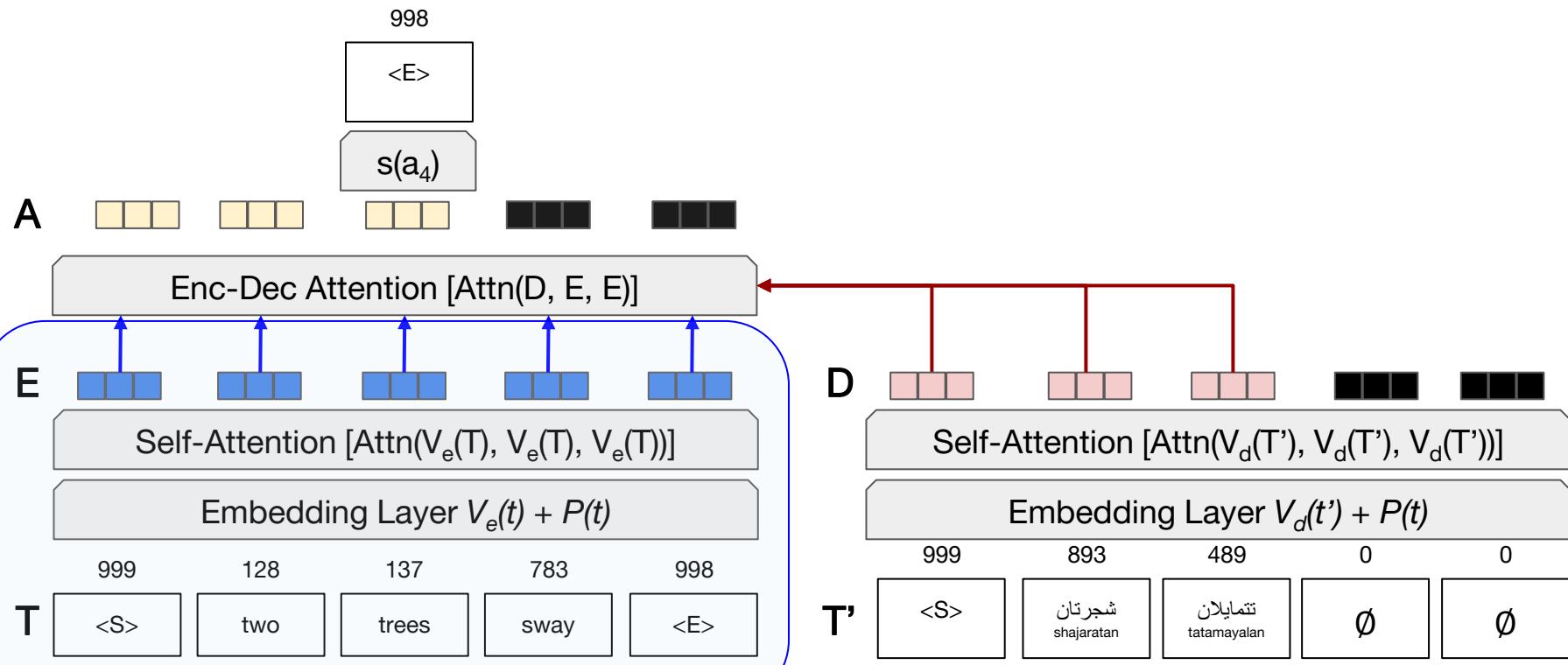
# Sequence-to-Sequence Transformer



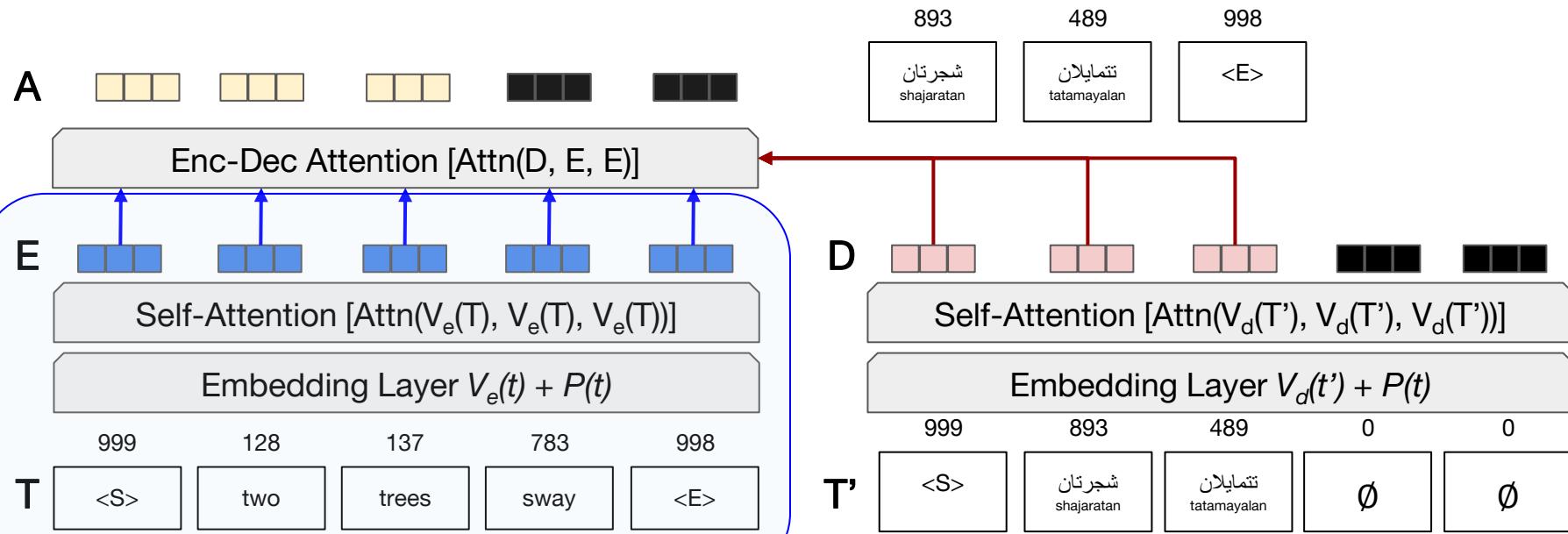
# Sequence-to-Sequence Transformer



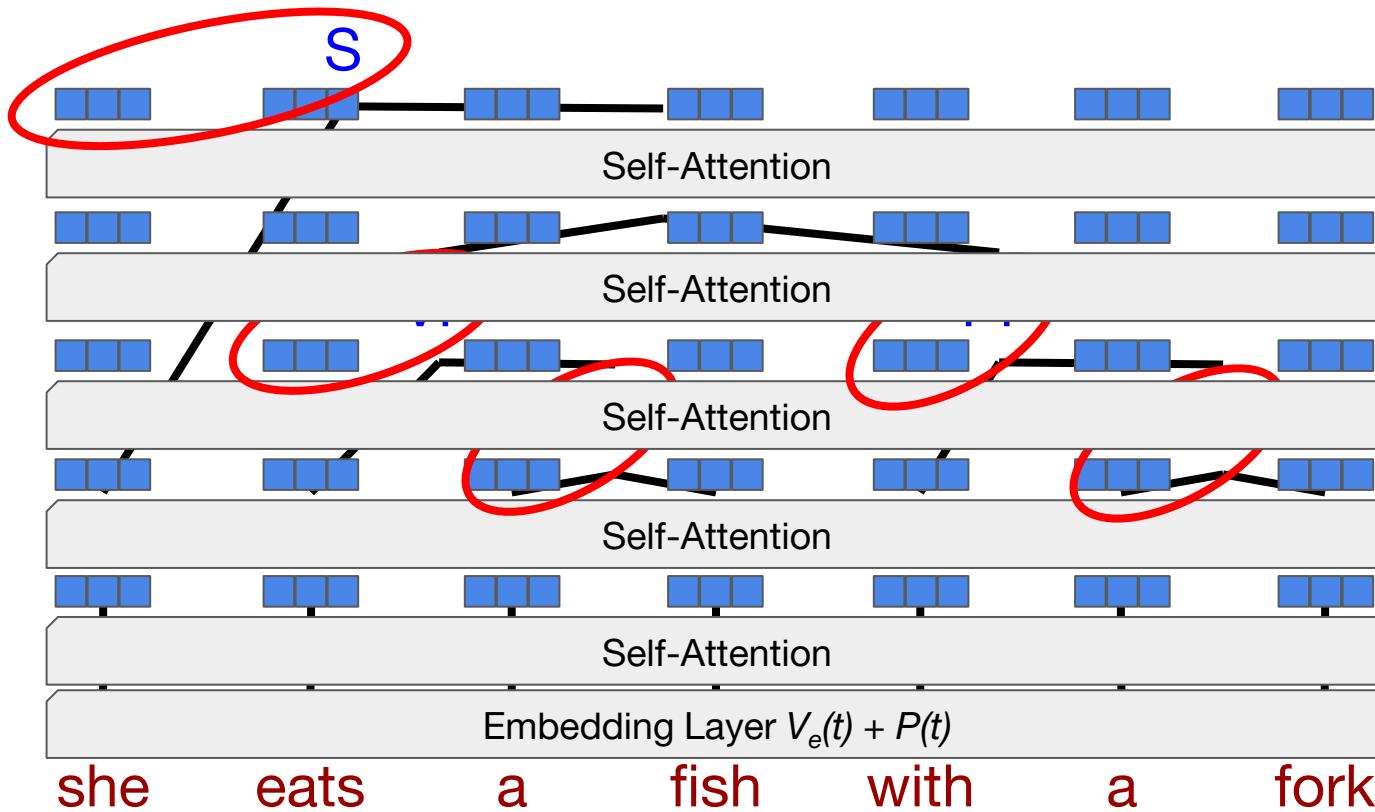
# Sequence-to-Sequence Transformer



# Sequence-to-Sequence Transformer

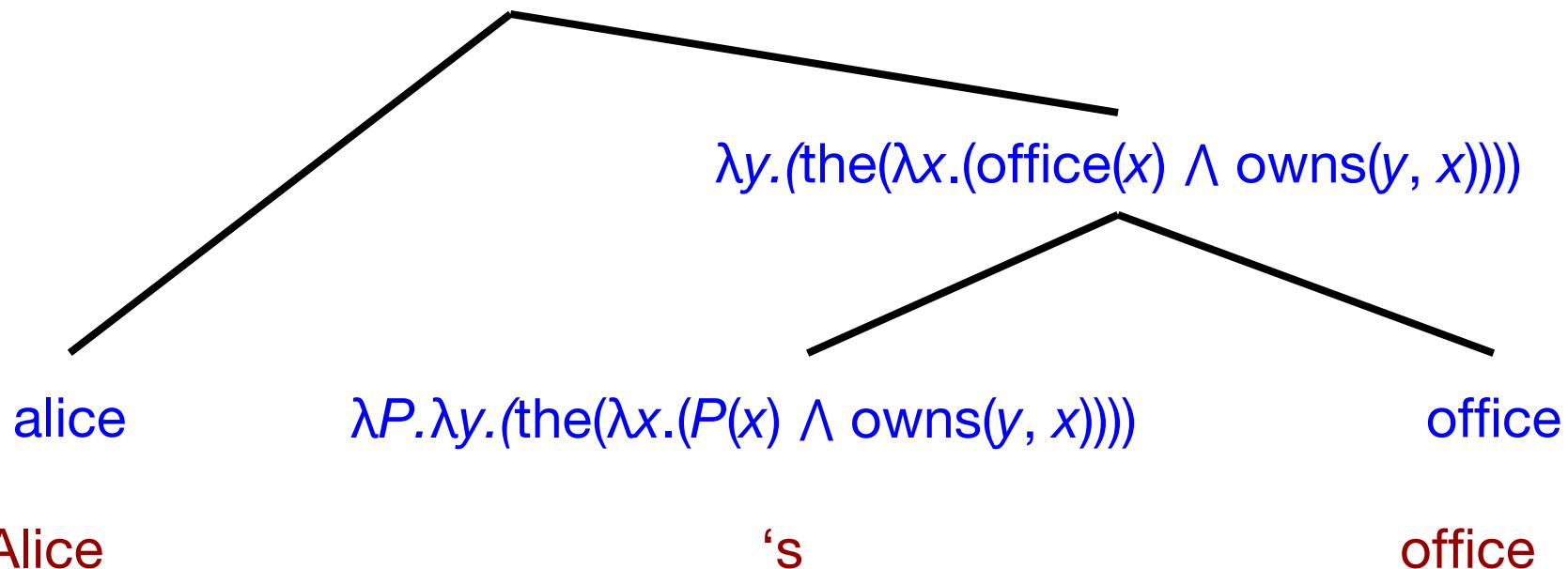


# Symbol Systems from Syntax and Semantics



# Predicate Logic and Montague Grammar

$\text{the}(\lambda x.(\text{office}(x) \wedge \text{owns(alice, } x)))$



# Encoder-only Models Maybe Actually Do This???

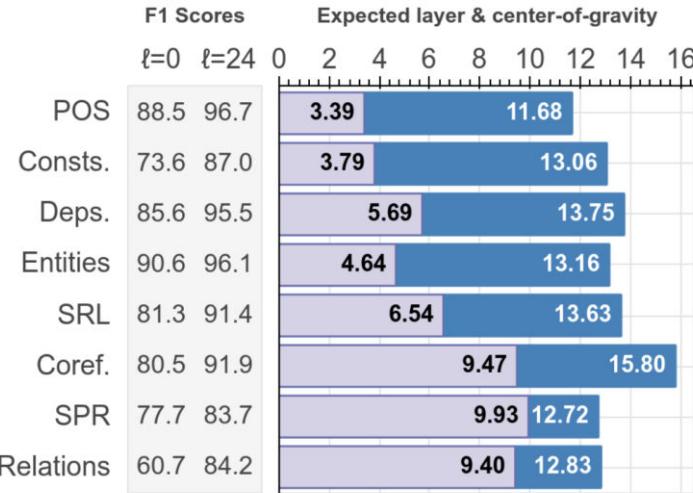
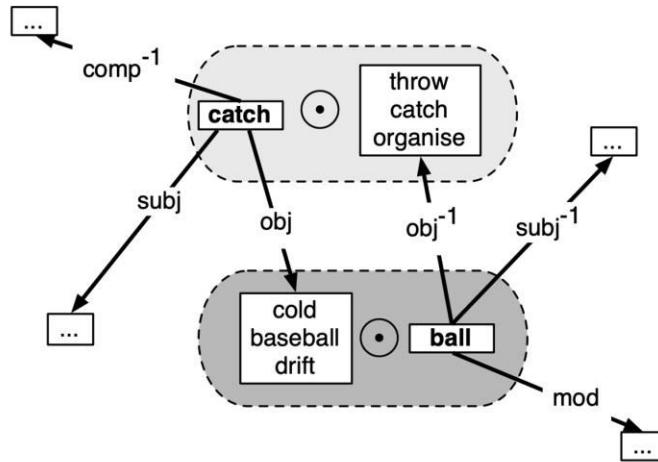


Figure 1: Summary statistics on BERT-large. Columns on left show F1 dev-set scores for the baseline ( $P_{\tau}^{(0)}$ ) and full-model ( $P_{\tau}^{(L)}$ ) probes. Dark (blue) are the mixing weight center of gravity (Eq. 2); light (purple) are the expected layer from the cumulative scores (Eq. 4).

# The Field Wanted Something Like BERT For Decades!



$(v_a, R_a, R_a^{-1})$  and  $(v_b, R_b, R_b^{-1})$  be the representations of the two words, and let  $r \in \mathcal{R}$  be the relation linking  $a$  to  $b$ . Then, we define the meaning of  $a$  and  $b$  in this context as a pair  $(a', b')$  of vectors, where  $a'$  is the meaning of  $a$  in the context of  $b$ , and  $b'$  the meaning of  $b$  in the context of  $a$ :

In the *structured vector space* (SVS) model, we represent the meaning of a lemma  $w$  as a triple

$$w = (v, R, R^{-1})$$

where  $v \in D$  is a lexical vector describing the word  $w$  itself,  $R : \mathcal{R} \rightarrow D$  maps each relation label onto a vector that describes  $w$ 's selectional preferences, and  $R^{-1} : \mathcal{R} \rightarrow D$  maps from role labels to vectors describing inverse selectional preferences of  $w$ .

$$\begin{aligned} a' &= (v_a \odot R_b^{-1}(r), R_a - \{r\}, R_a^{-1}) \\ b' &= (v_b \odot R_a(r), R_b, R_b^{-1} - \{r\}) \end{aligned}$$

# The Field Wanted Something Like BERT For Decades!

In the *structured vector space* (SVS) model, we represent the meaning of a lemma  $w$  as a triple

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$$w = (v, R, R^{-1})$$

$X = \text{Embed}(\text{tokens})$

Attn  $XW_Q, XW_K, XW_V$ )

where  $v \in D$  is a lexical vector describing the word  $w$  itself,  $R : \mathcal{R} \rightarrow D$  maps each relation label onto a vector that describes  $w$ 's selectional preferences and  $R^{-1} : \mathcal{R} \rightarrow D$  maps from role labels to vectors describing inverse selectional preferences of  $w$ .

Reminder:  $QK^T \neq KQ^T$  because relations are not symmetric!

$$\begin{aligned} a' &= (v_a \odot R_b^{-1}(r), R_a - \{r\}, R_a^{-1}) \\ b' &= (v_b \odot R_a(r), R_b, R_b^{-1} - \{r\}) \end{aligned}$$

# Recent High-Profile NLP Technology?



Prove that  $1 = 2$  implies  $2 = 1$ .

< 3 / 3 >



It is impossible to prove that  $1 = 2$  implies  $2 = 1$ , as the premise of the statement is false. In mathematics, 1 is always equal to 1 and never equal to 2, and therefore the premise of the statement is not true. Additionally, the statement contradicts the fundamental principle of equality, which states that if  $a = b$ , then  $b = a$ . Therefore, the statement is not logically or mathematically sound.



- Pretrain a huge decoder-only transformer on a bunch of Internet text
- Apply our decoding strategies learned in the previous lecture (e.g., top-p sampling)
- Will we get this output?
- Why not?

# Lecture Outline

- Recap: Module 2 Transformers as LMs
- RL 101 for NLP

# Markov Decision Process with Rewards

$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor

- MPD assumption: the *state* captures everything we need to know about what the optimal *action* should be
- Note *rewards* could be defined for every (*state, action*)
  - In practice, we'll need to estimate the quality of states, state-action pairs, etc., especially when reward is sparse
- Note that we allow for a probabilistic *transition function*

# Markov Decision Process with Rewards

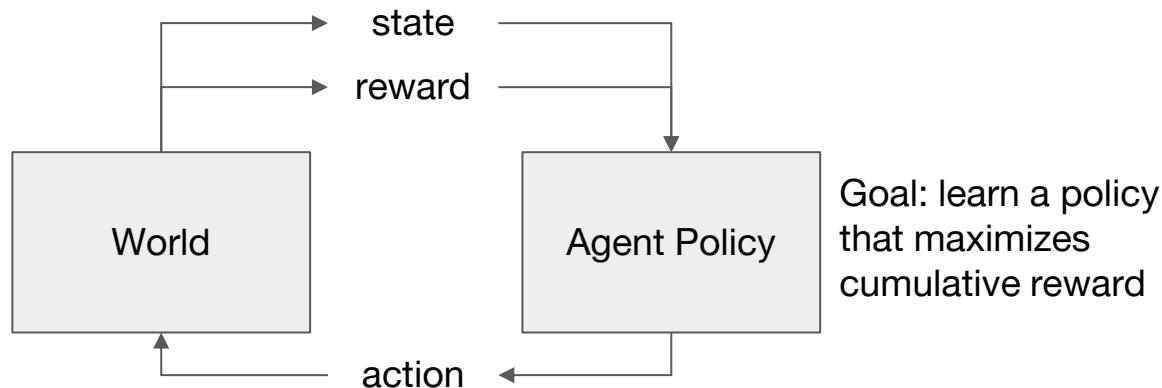
$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor

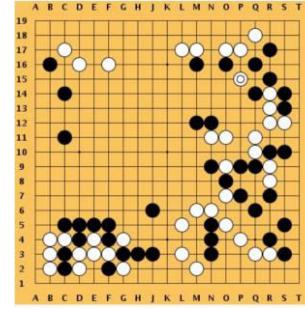
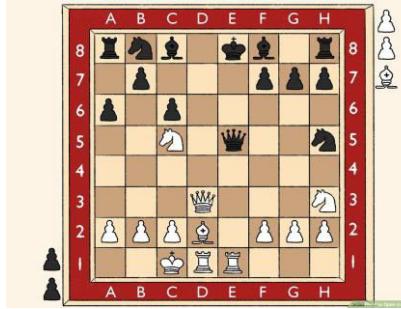


# Example - Atari Games



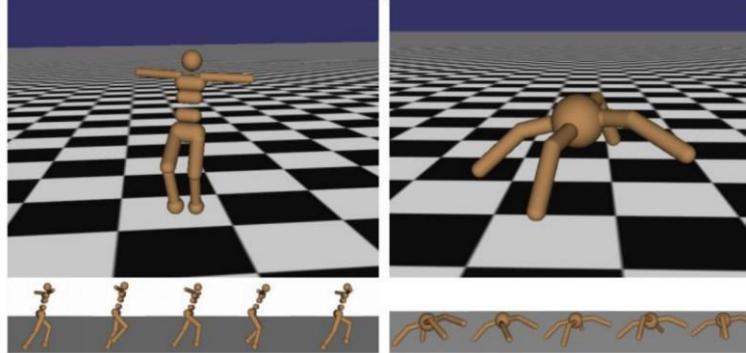
- Objective: Win the game (with scores as high as possible)
- State?
  - Pixels on the screen completely determine current state
- Actions?
  - Game controls (e.g. U, D, L, R, Shoot)
- Reward?
  - Score delta between  $t-1$  and  $t$  for action taken at time  $t-1$

# Example - Chess / Go



- Objective: Win the game
- State?
  - Positions of game pieces both on and off the board
- Actions?
  - Admissible game moves
- Reward?
  - Sparse: 1 if win, -1 if lose. Dense: heuristics (e.g., capture)

## Example - Control



- Objective: Cover ground by walking
- State?
  - Position and angles of the joints
- Actions?
  - Torques applied on joints
- Reward?
  - Increases in distance from origin at each timestep

# Markov Decision Process with Rewards

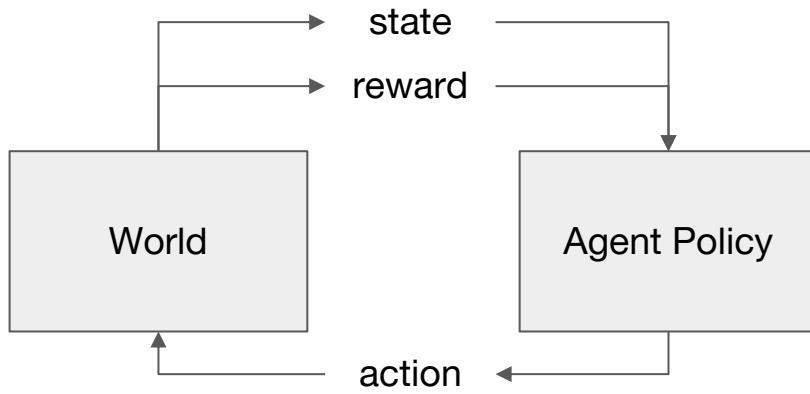
$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor



- At  $t=0$ , get initial state  $s_0$
- Get action  $a_t$  from policy  $\pi(s_t; \theta) = a_t$
- World samples next state  $s_{t+1}$  from transition function  $P(s_t, a_t) = s_{t+1}$
- Agent receives reward  $r_t = R(s_t, a_t)$ 
  - Can also be defined as  $r_t = R(s_{t+1})$

# Markov Decision Process with Rewards

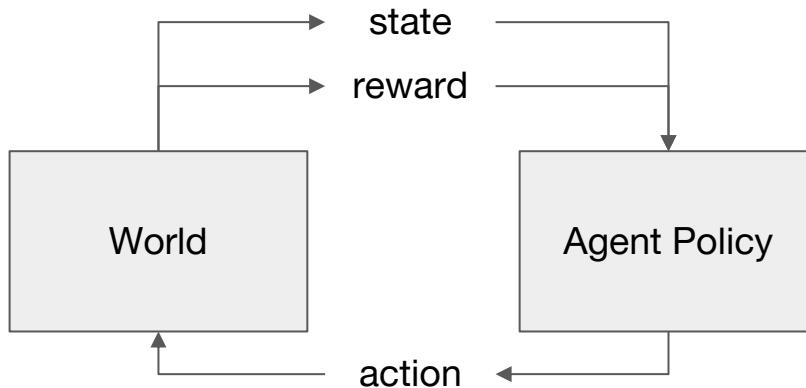
$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor

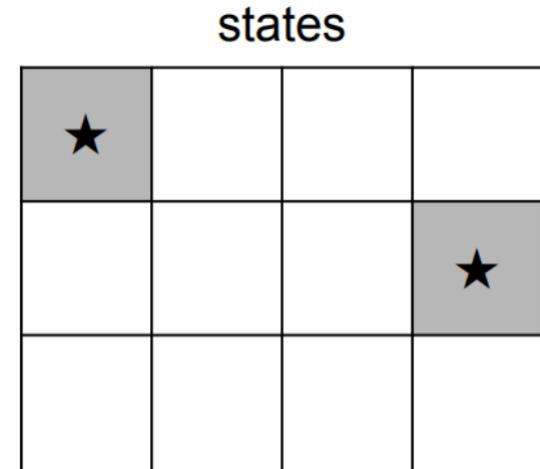


- Optimization Goal: learn policy parameters  $\theta$  to maximize *cumulative discounted reward*
  - For a given *trajectory* with sequence of rewards  $r$ :  $\sum_t \gamma^t r_t$

# Simple MPD Example And Solution

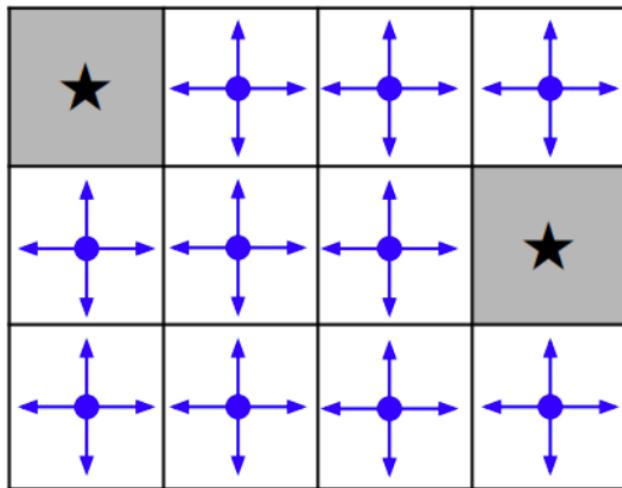
- Transition?
  - Deterministic
- Rewards?
  - +10 for reaching star
  - -1 for each action

actions = {  
    1. right →  
    2. left ←  
    3. up ↑  
    4. down ↓  
}

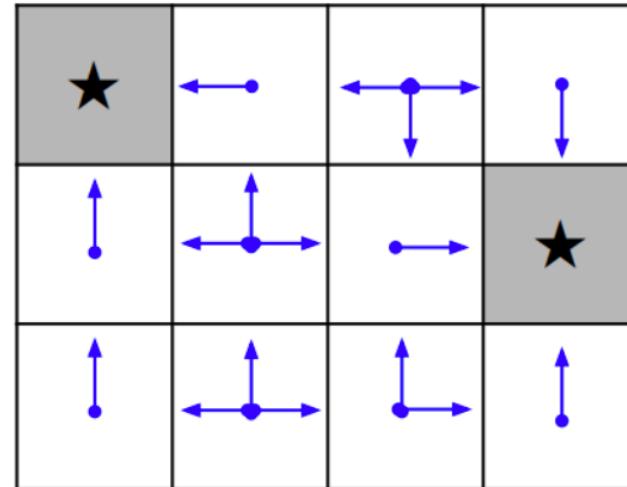


**Objective:** Reach stars (either one) with least number of actions

# Simple MPD Example And Solution



Random Policy



Optimal Policy

# Markov Decision Process with Rewards

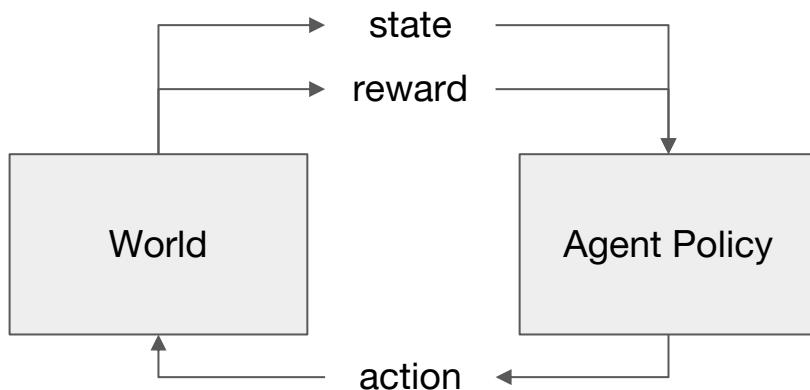
$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor



$$\pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right]$$

# Seeking An Optimal Policy

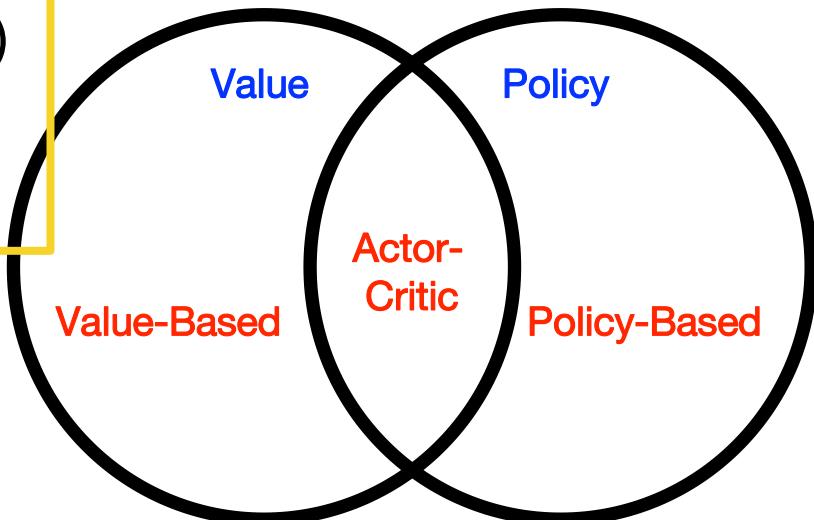
- Suppose we have a set of sampled trajectories
- A trajectory is defined by the tuple  $(s, a, r)$  through time
  - E.g.,  $\langle(s_0, a_0, r_0), (s_1, a_1, r_1), (s_2, a_2, r_2), \dots\rangle$
- One approach we can take is to estimate the *quality* of (state, action) pairs in terms of the eventual expected reward
- *Q-value* function:

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

- The expected cumulative reward if following the policy at state  $s$ , taking the action  $a$

# Seeking An Optimal Policy

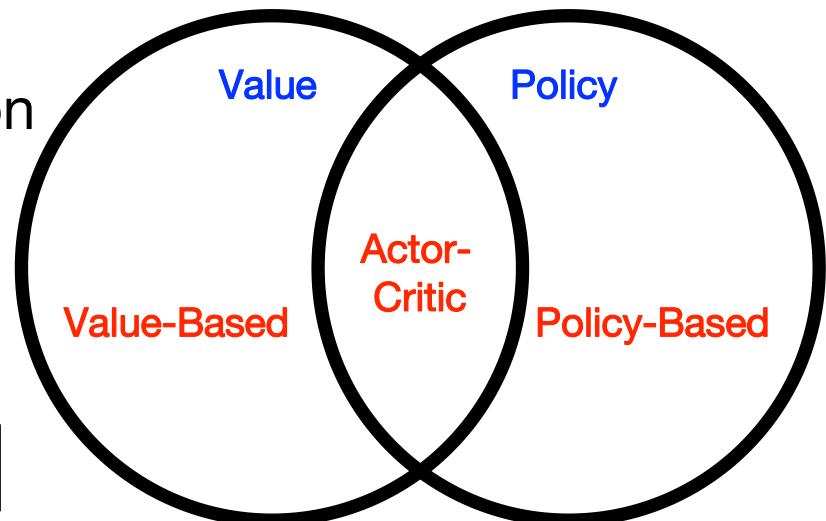
- Value Based (e.g., Deep Q-Network)
  - Learn Value Function
  - Implicit Policy
- Policy Based (e.g., guided policy)
  - No Value Function
  - Directly Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy



# Seeking An Optimal Policy

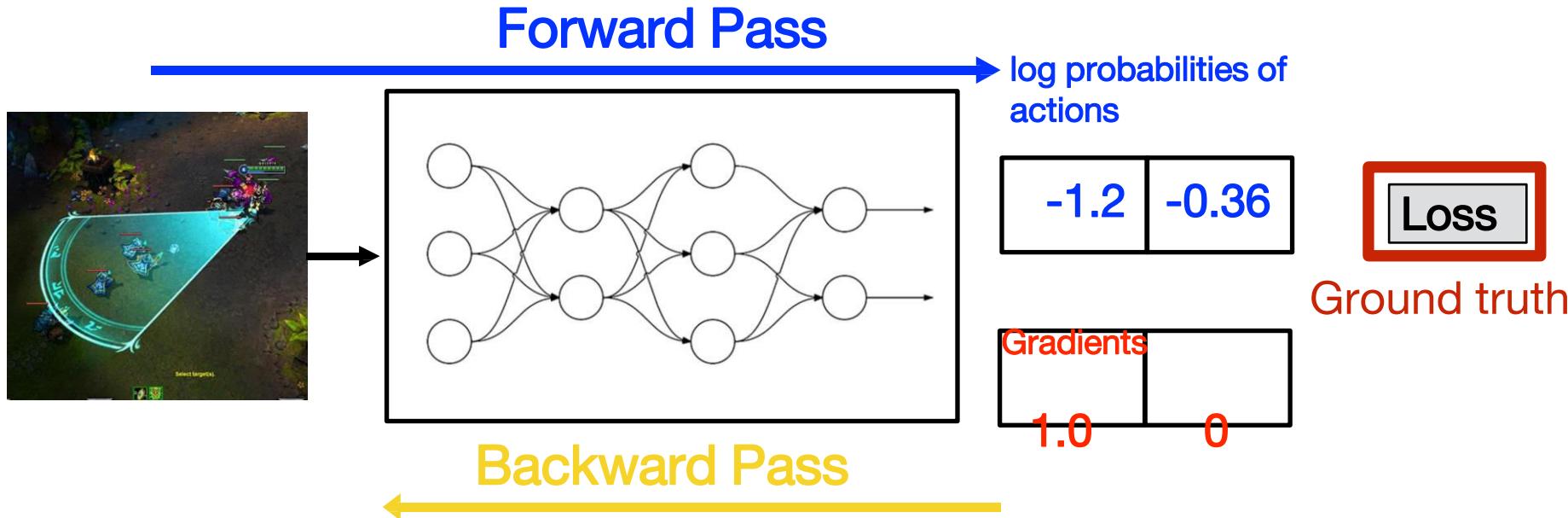
- Valued-based: a function that measures how valuable action  $a$  is at state  $s$ 
  - Indirectly yields a policy
- Policy-based: a function that measures the probability of action  $a$  leading to the best final outcome from a given state  $s$ 
  - Directly yields a policy

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$



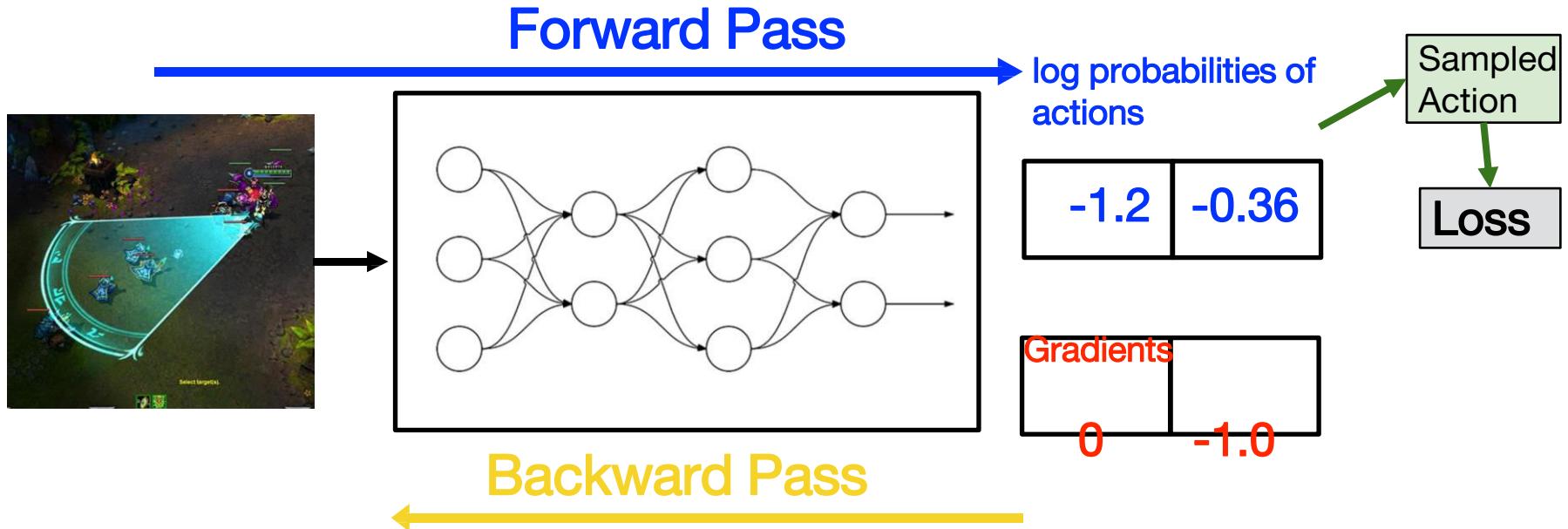
$$\theta^* = \operatorname{argmax}_\theta J(\theta) \quad J(\theta) = \mathbb{E}_{\pi(\theta)} \left[ \sum \gamma^t r_t \right]$$

# Supervised Learning versus Policy Gradient



Learn:  $\pi: S \rightarrow A$  for  $\pi$  parameterized by  $\theta$

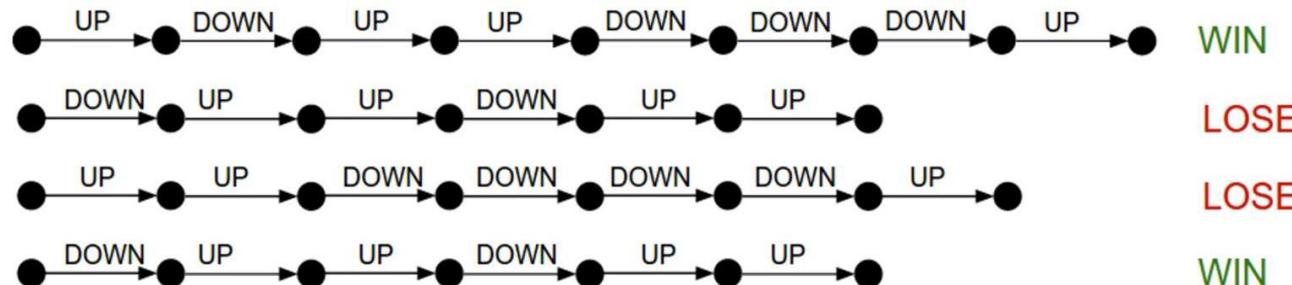
# Supervised Learning versus Policy Gradient



Eventual Reward = -1.0

# Policy Gradient

- Given a (possibly random) policy, run it for several iterations
- See what sticks!
- Upweight (*state, action*) transitions that lead to high eventual reward, downweight others
  - Policy Gradients

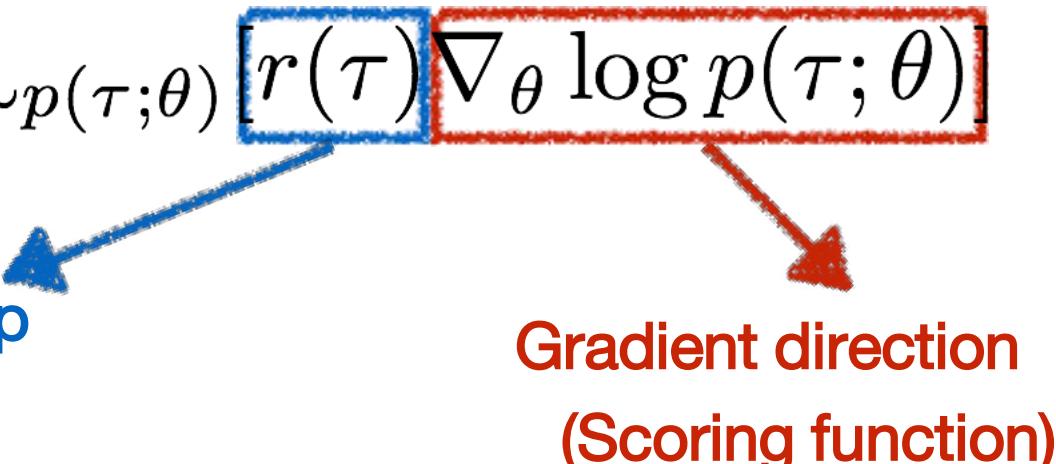


# Policy Gradient

- Objective to optimize:  $J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$
- Trajectory:  $\tau = (s_0, a_0, r_0, s_1, \dots)$
- Policy Gradient Theorem:
  - The derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy  $\pi_\theta$ .

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum p(\tau; \theta) r(\tau) \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)]\end{aligned}$$

# Policy Gradient

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)]$$


Learning step

Gradient direction  
(Scoring function)

# Training a Network with Policy Gradient

- Monte-Carlo Policy Gradient (REINFORCE algorithm)
- Upshot:
  - 1) Initialize a (possibly random) policy
  - 2) Generate sample trajectories  $\langle(s, a, r)\rangle$
  - 3) Update reward estimates at each timestep based on discounted reward from eventual result
  - 4) Improve the policy by updating the parameters based on discounted rewards

# Training a Network with Policy Gradient (REINFORCE)

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)]\end{aligned}$$

$$\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_{\theta}(a_t | s_t)$$

Write out the probability of the observed trajectory under the current policy

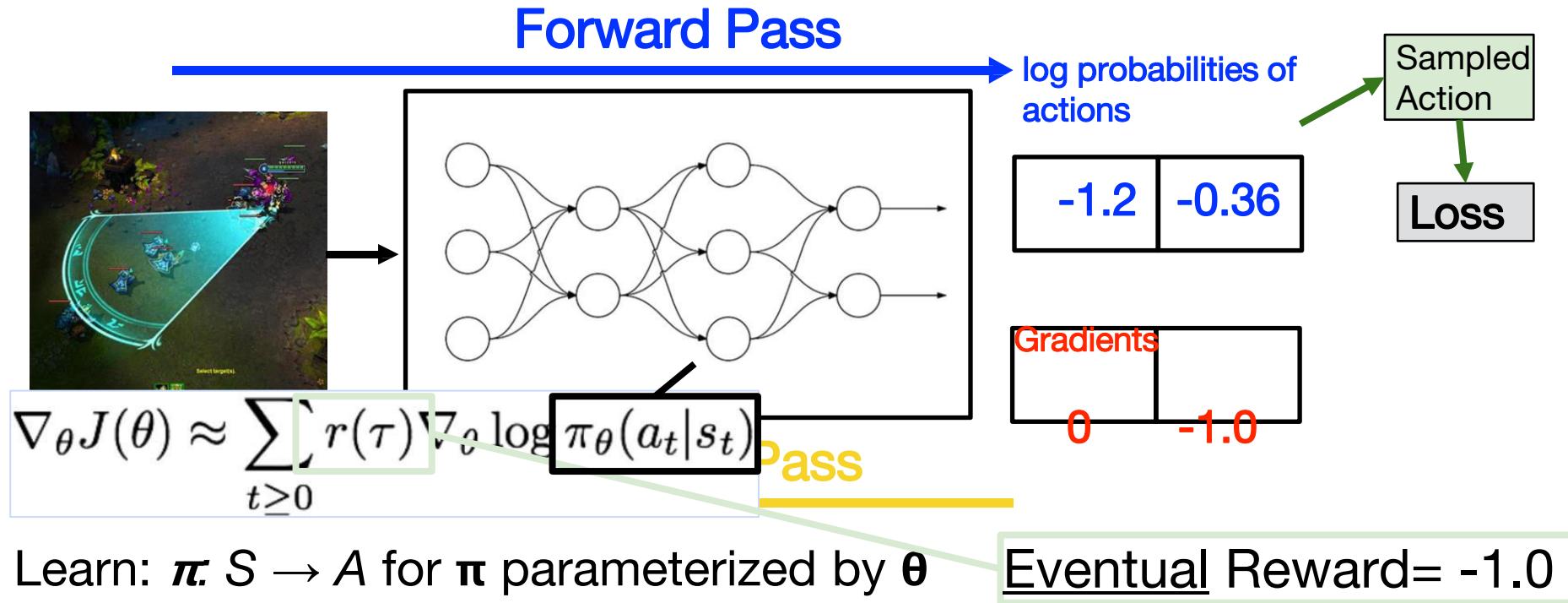
$$\nabla_{\theta} \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Differentiate with respect to policy parameters to get gradient

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Note that our estimate is now independent of the true state transition function that we can't observe directly!

# Training a Network with Policy Gradient (REINFORCE)



# Training a Network with Policy Gradient

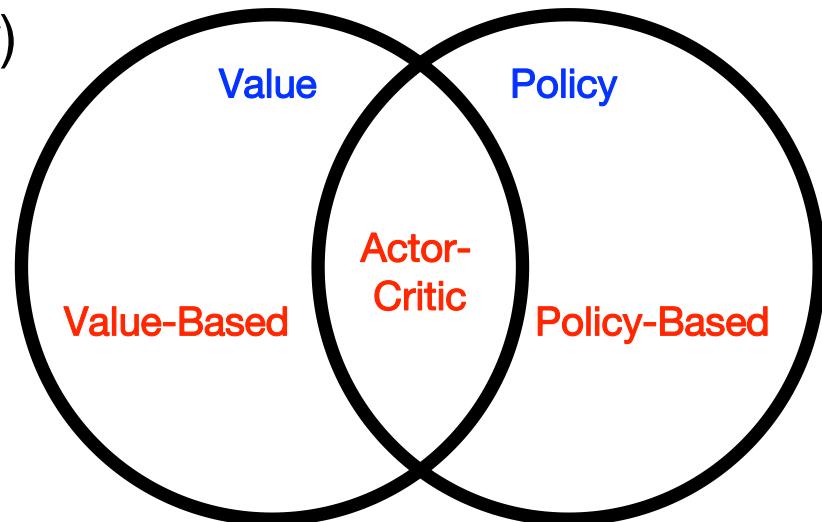
- Very general method, but policy gradients suffer from high variance and so can require a lot of samples (trajectories per policy) and iterations (policy refinements)
- Policy gradient methods converge to a local minima that is often “good enough” for a task, but that convergence can be costly in terms of time and compute

# Quiz 2

- Slides for:
  - Define a task with  $(x, y)$  strings
  - Going from string  $x$  to vector  $\mathbf{x}$  and string  $y$  to vector  $\mathbf{y}$
  - Define network  $M(\mathbf{x})=\mathbf{y}$  as composition of functions
  - For each function, define in terms of parameters and initialization
  - Define a loss function  $L(M(\mathbf{x}), \mathbf{y})$
- Teams:
  - Team 1: Vanilla RNN and LSTMs
  - Team 2: Transformer Encoder
  - Team 3: Transformer Encoder-Decoder

# Seeking An Optimal Policy

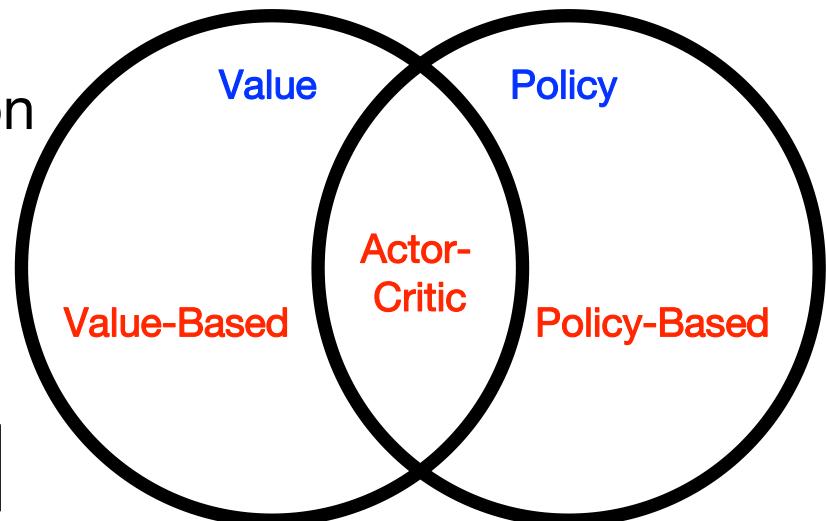
- Value Based (e.g., Deep Q-Network)
  - Learn Value Function
  - Implicit Policy
- Policy Based (e.g., guided policy)
  - No Value Function
  - Directly Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy



# Seeking An Optimal Policy

- Valued-based: a function that measures how valuable action  $a$  is at state  $s$ 
  - Indirectly yields a policy
- Policy-based: a function that measures the probability of action  $a$  leading to the best final outcome from a given state  $s$ 
  - Directly yields a policy

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$



$$\theta^* = \operatorname{argmax}_\theta J(\theta) \quad J(\theta) = \mathbb{E}_{\pi(\theta)} \left[ \sum \gamma^t r_t \right]$$

# Q-value Function

- Want a function that measures how valuable action  $a$  is at given state  $s$

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

- Optimal Q-value function:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

- The best expected cumulative reward at state  $s$  when taking the action  $a$  next

# Bellman Equation

- Optimal Q-value function gives maximum expected return

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

- Such an optimal function satisfies the optimality described by the *Bellman Equation*:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

- which in a nutshell says: the highest value of a (state, action) pair is the immediate reward plus the discounted best future reward from here

# Value Iteration

- **Intuition:** Update the Q-value function iteratively

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

- Note similarity to *Bellman Equation*:

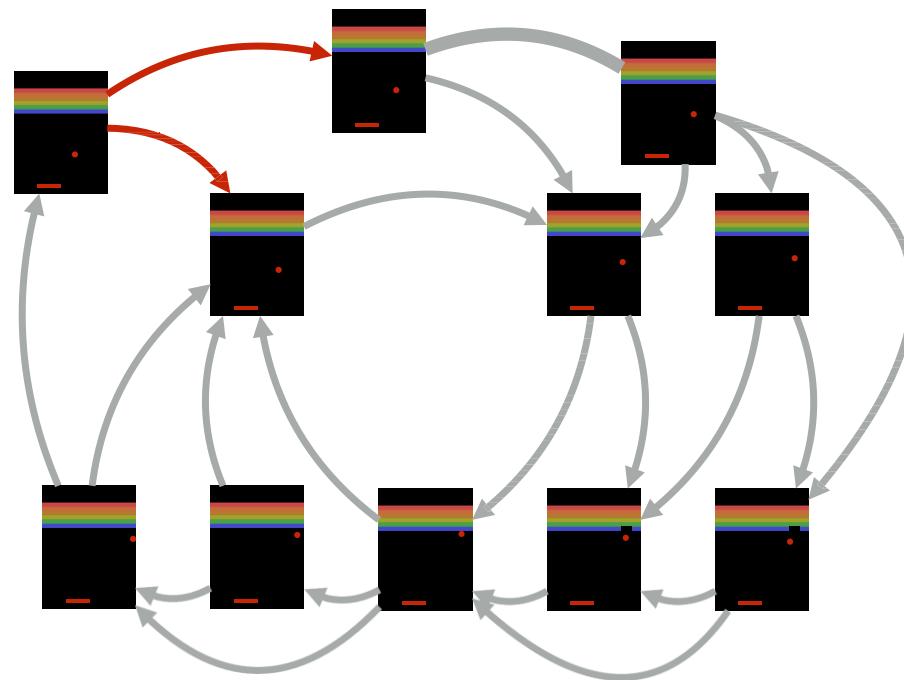
$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

- In expectation, such an iterative update *will converge* to the optimal estimate  $Q^*$  if trajectories show us all  $(s, a)$  pairs enough times

# Value Iteration

- Update:  $Q_0$

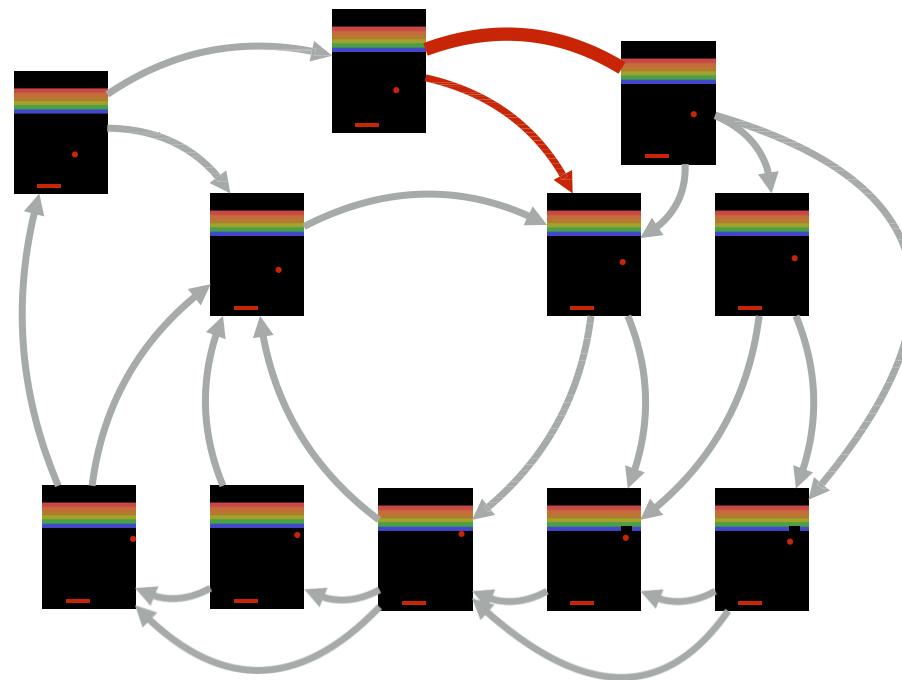
$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$



# Value Iteration

- Update:  $Q_1$

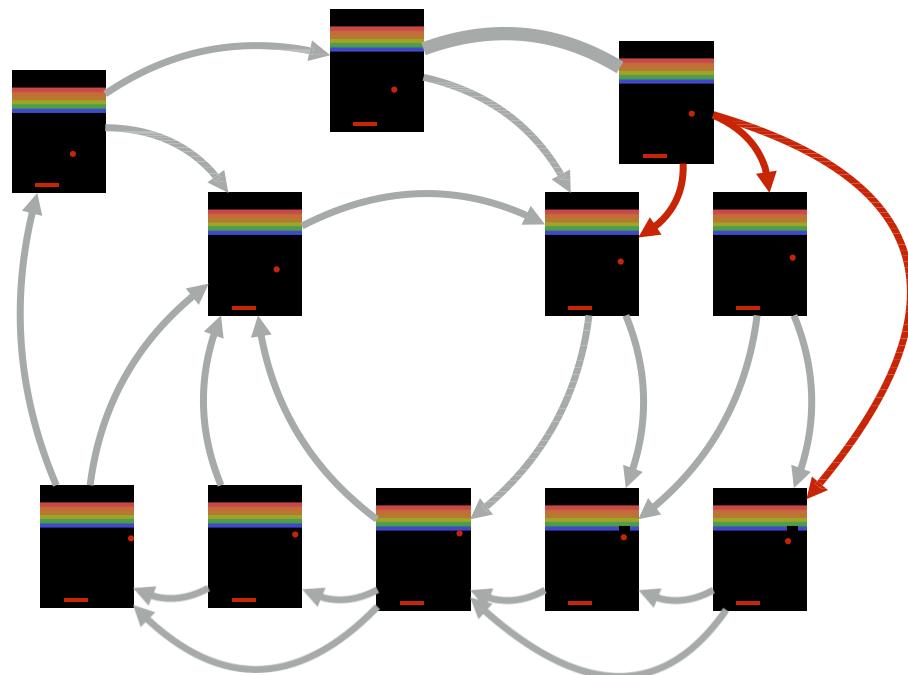
$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$



# Value Iteration

- Update:  $Q_2$

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$



# Value Iteration

- **Intuition:** Update the Q-value function iteratively

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

- Getting an accurate estimate  $Q^*(s, a)$  for every state-action pair is intractable for continuous state/action spaces and large discrete spaces too (e.g., all possible space invaders screens)
- What if we could *approximate* the Q function?

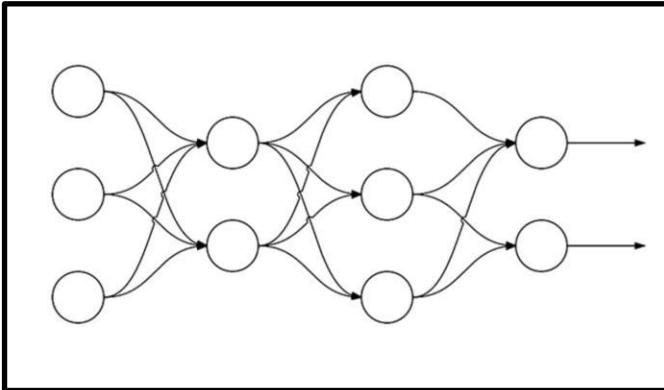
$$Q(s, a; \theta) \approx Q^*(s, a)$$

- What's a good way to do function approximation?

# Training a Network with Value Iteration

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Forward Pass



Action Q-value estimates

.94	.87
-----	-----

Note: our network here is from states to actions still; why can it model  $Q(s, a; \theta)$ ?

Backward Pass

Sharing params among action Q value estimate networks; what we're estimating depends on the loss function

# Training a Network with Value Iteration

$$Q(s, a; \theta) \approx Q^*(s, a)$$

**Forward Pass**



The Loss:  $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} [(y_i - Q(s, a; \theta_i))^2]$

“Ground truth” Q\*

Approximation

$$\mathbb{E}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a]$$

Estimate via value iteration intuition

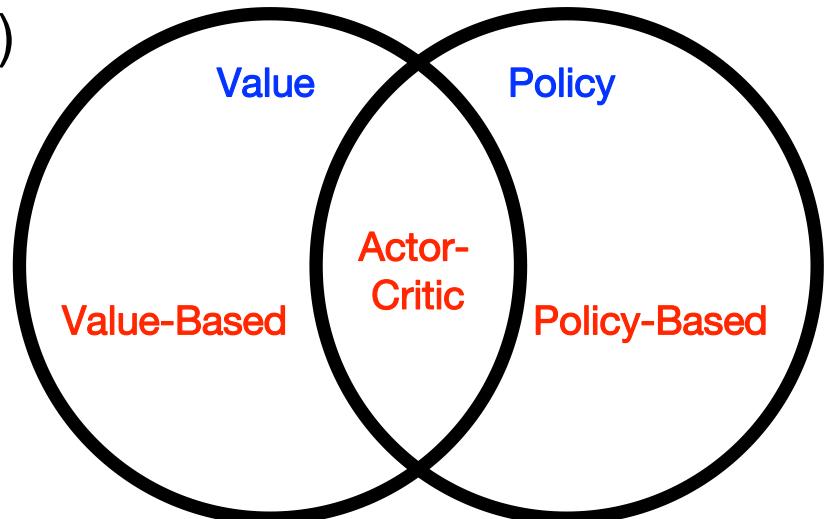
**Backward Pass**



Update  $\theta$ :  $\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)] \nabla_{\theta_i} Q(s, a; \theta_i)$

# Seeking An Optimal Policy

- Value Based (e.g., Deep Q-Network)
  - Learn Value Function
  - Implicit Policy
- Policy Based (e.g., guided policy)
  - No Value Function
  - Directly Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy



# Seeking An Optimal Policy

- Policy gradient intuition: sample trajectories with suboptimal policy, then for high reward trajectories push up action probabilities taken, for low reward trajectories push down action probabilities taken
  - Problem? Rewards across sometimes literally random trajectories are going to have high variance
- Value iteration intuition: for every state-action pair, estimate the best future reward we could get if we chose that action from this state
  - Problem? Needs a lot of state-action data

# Advantage Actor-Critic Algorithm

- Take advantage of the strengths of policy gradients and value iteration while removing some of their weaknesses
- In particular:
  - Goal: Reduce variance of policy gradient updates
  - Method: change the parameter updates to upweight an action proportional to how much better it *turned out* to have been when taken versus our expectation

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Discounted reward achieved from this state

Estimate of reward from this state

# Advantage Actor-Critic Algorithm

- How to estimate the reward we can achieve from a given state?
- Use Q-function and value function!
  - Value function: integrate over quality estimates from all possible actions based on policy probability to take them

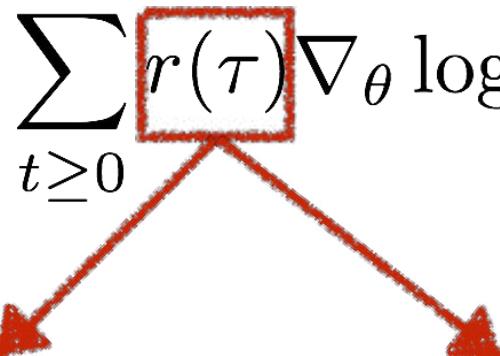
$$V^\pi(s) = \sum_{a \in A} \pi(a|s) * Q^\pi(s, a)$$

- Then we call the “advantage” an action  $a$  has at state  $s$ :

$$Q^\pi(s_t, a_t) - V^\pi(s_t)$$

# Advantage Actor-Critic Algorithm

- Goal: Reduce variance of policy gradient updates

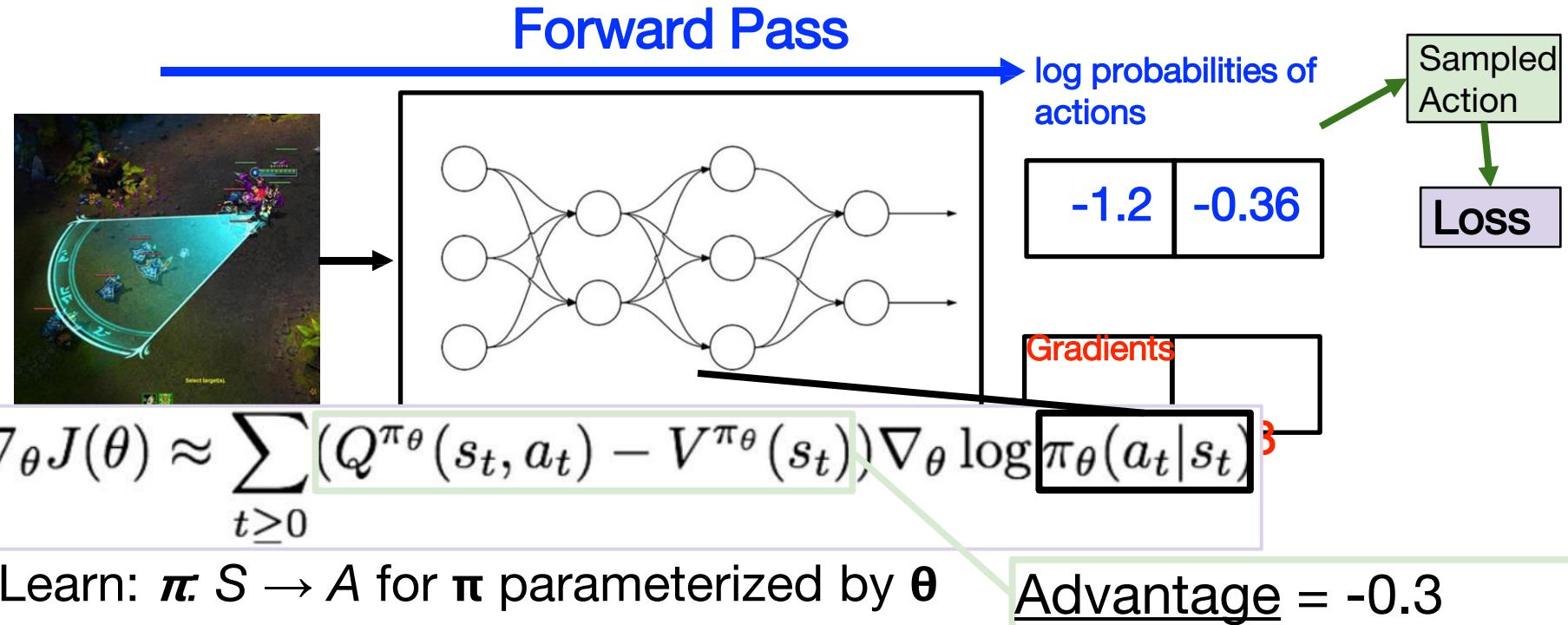
$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

$$Q(s_t, a_t)$$
$$Q(s_t, a_t) - V(s_t)$$

- Method: change the parameter updates to upweight an action proportional to how much better it turned out to have been to take it versus our expectation

# Advantage Actor-Critic Algorithm

- Can we learn Q and Value functions?
  - Q-Learning + Policy Gradient methods!
- Train both an **Actor** (policy) and a **Critic** (Q-function)
- Actor: Which action to take?
- Critic: How good the action was and how to adjust parameters?
- Advantage Function:  $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ 
  - How much better is  $a$  than expected?
  - Reduces the variance of policy gradient updates

# Training a Network with Advantage Actor-Critic (A2C)



# CSCI 444: Natural Language Processing

Jesse Thomason

Module 3: Lecture 1  
RL 101 for NLP

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 3: Lecture 2  
RLHF, ICL, RAG

# Lecture Outline

- Reinforcement Learning from Human Feedback (RLHF)
- Prompting and In-context Learning (ICL)
- Retrieval Augmented Generation (RAG)

# Lecture Outline

- Reinforcement Learning from Human Feedback (RLHF)
- Prompting and In-context Learning (ICL)
- Retrieval Augmented Generation (RAG)

# Markov Decision Process with Rewards

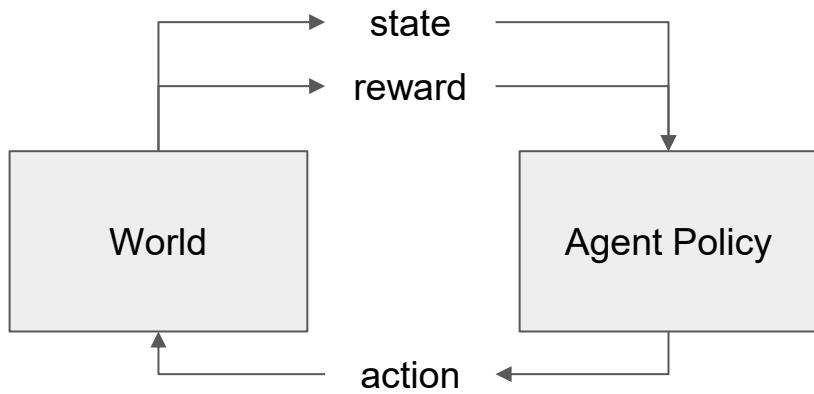
$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor



- At  $t=0$ , get initial state  $s_0$
- Get action  $a_t$  from policy  $\pi(s_t; \theta) = a_t$
- World samples next state  $s_{t+1}$  from transition function  $P(s_t, a_t) = s_{t+1}$
- Agent receives reward  $r_t = R(s_t, a_t)$ 
  - Can also be defined as  $r_t = R(s_{t+1})$

# Markov Decision Process with Rewards

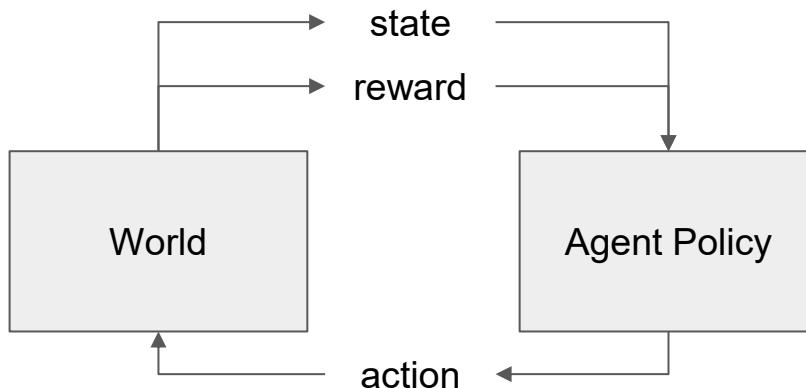
$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

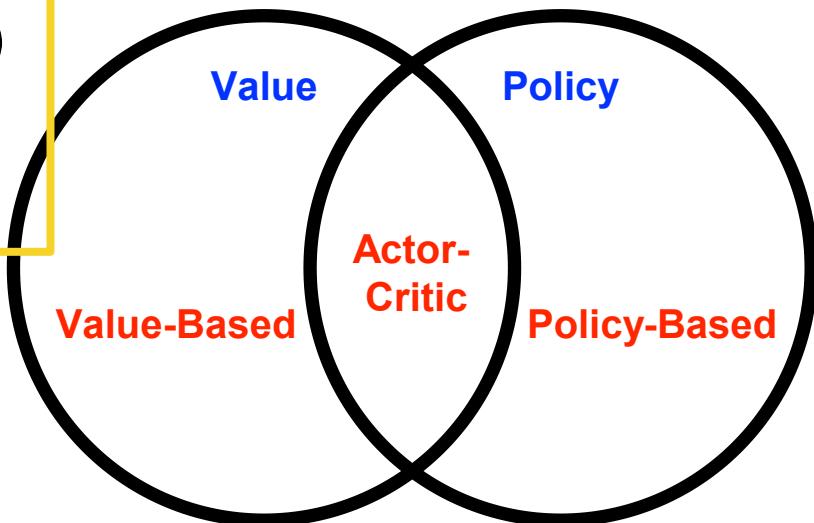
$\gamma$  : discount factor



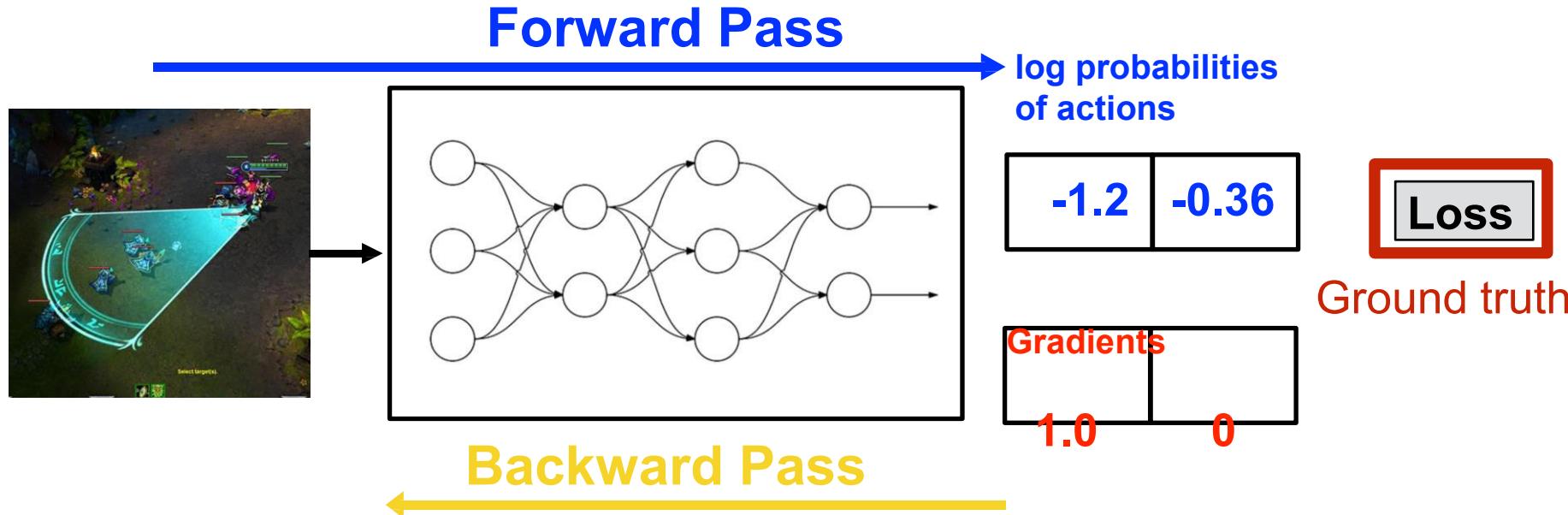
- Optimization Goal: learn policy parameters  $\theta$  to maximize *cumulative discounted reward*
  - For a given *trajectory* with sequence of rewards  $r$ :  $\sum_t \gamma^t r_t$

# Seeking An Optimal Policy

- Value Based (e.g., Deep Q-Network)
  - Learn Value Function
  - Implicit Policy
- Policy Based (e.g., guided policy)
  - No Value Function
  - Directly Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy

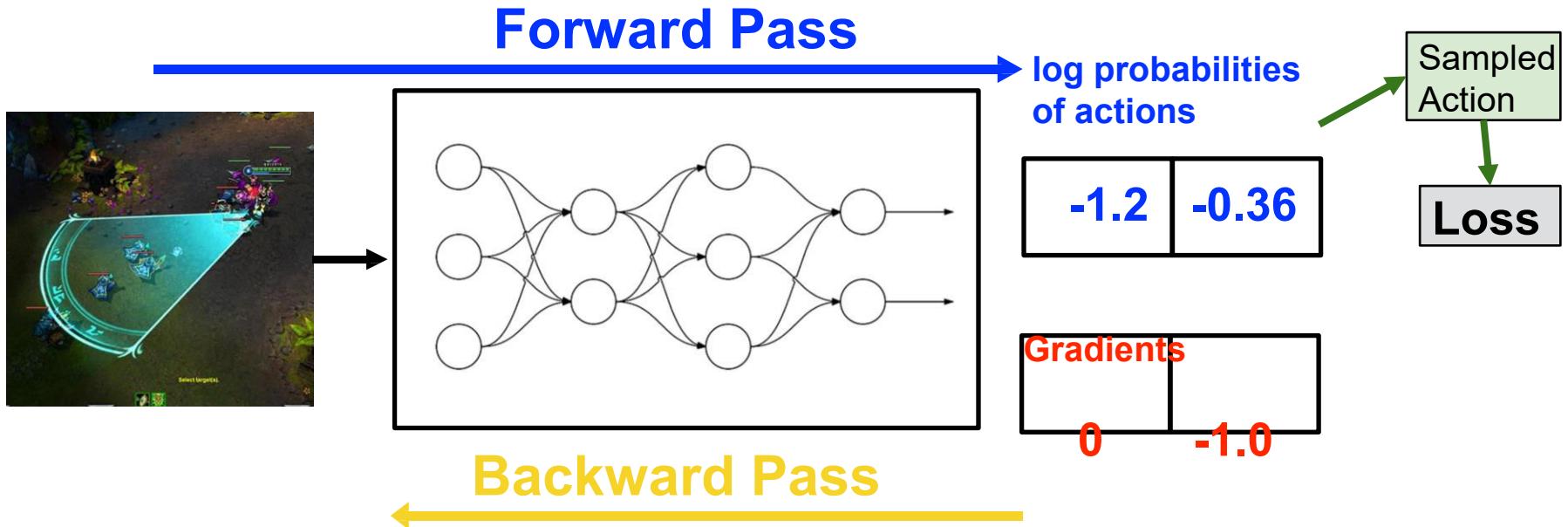


# Supervised Learning versus Policy Gradient



Learn:  $\pi: S \rightarrow A$  for  $\pi$  parameterized by  $\theta$

# Supervised Learning versus Policy Gradient



Learn:  $\pi: S \rightarrow A$  for  $\pi$  parameterized by  $\theta$

Eventual Reward = -1.0

# High-profile Recent Example of Deep RL?

- ChatGPT!
  - Reinforcement Learning from Human Feedback (RLHF)
    - Prithviraj Ammanabrolu was doing it before it was cool
- Upshot:
  - Remember the core function of a language model?
  - Autoregressive language model?
  - $p(w_i|w_{i-1}w_{i-2}\dots w_1)$
  - ...  $\pi(a|s)$  for next word *actions* and context *states*
- ChatGPT training uses a policy gradient fine-tuning approach

# High-profile Recent Example of Deep RL

LE

Can you explain to me how RL with human feedback works for language models like I am five?



Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

First, Rufus will say something using his language model. For example, he might say "I am a robot."

Then, a human will listen to what Rufus said and give him feedback. The human might say something like a natural sentence a human would say. The human might say "That's not how a human would say that." Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I'm a machine.'"

Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

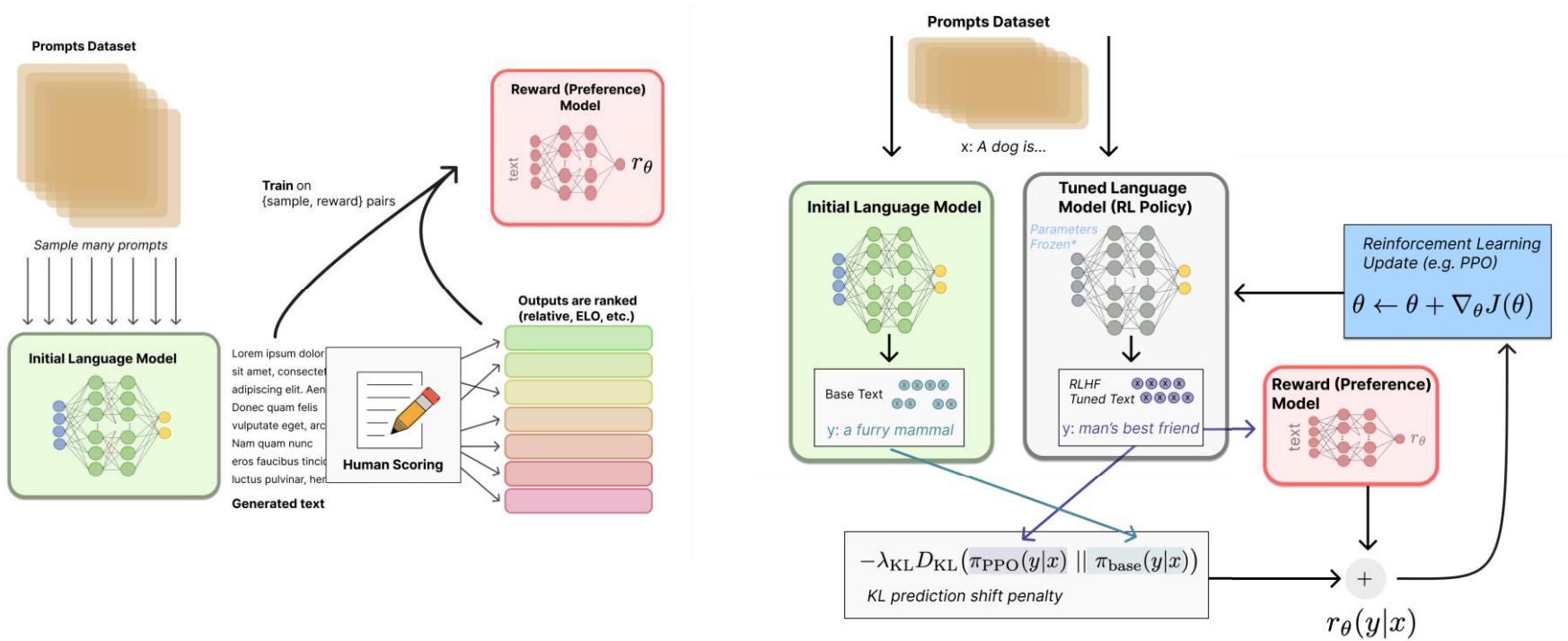
The human will listen again and give Rufus more feedback. This process will continue until Rufus can say sentences that sound natural to a human.

Over time, Rufus will learn how to talk like a human thanks to the feedback he receives from humans. This is how language models can be improved using RL with human feedback.

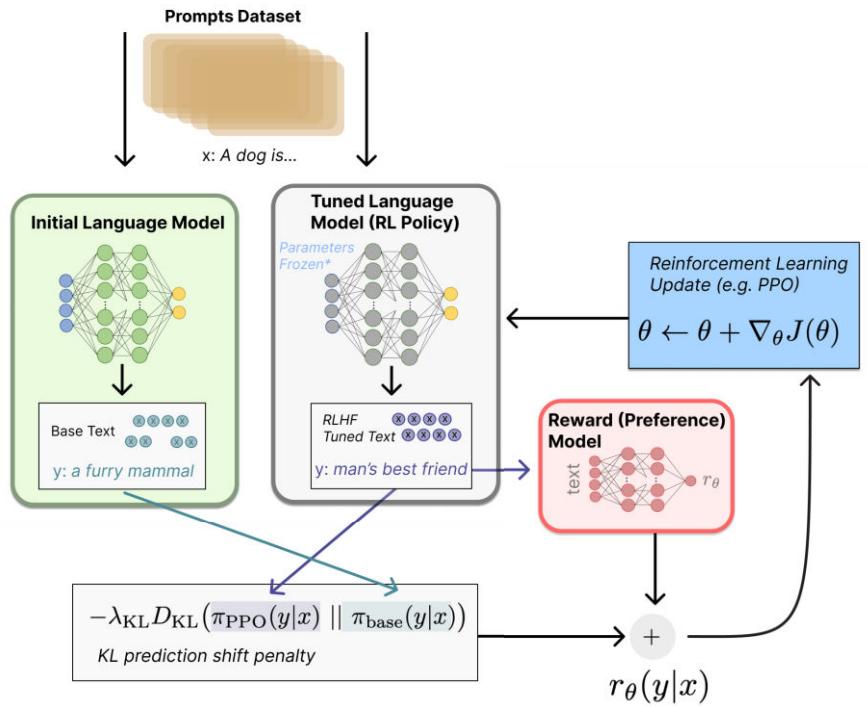
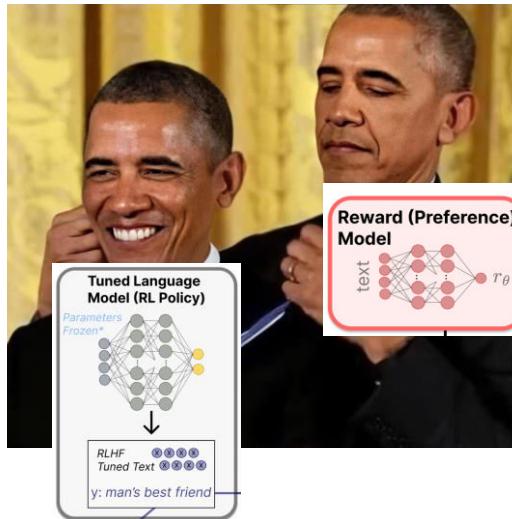
# Proximal Policy Optimization (PPO)

- Policy gradient:  $J(\theta) = E_{\tau \sim \pi_\theta} R(\tau) = \sum_{\tau} P(\tau; \theta)R(\tau)$        $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$
- Selecting a learning rate is a challenge; hard to pick a good learning rate given high variance in rewards
  - + Policy gradient has us toss trajectories after iterations
- PPO seeks to address these drawbacks.
- The popular implementation right now is called “Clipped PPO”
  - Upshot: dynamically adjust learning updates and keep past samples around as long as they seem ‘trustworthy’ even as the policy changes so we can learn more efficiently

# Reinforcement Learning from Human Feedback (RLHF)



# Reinforcement Learning from Human Feedback (RLHF)



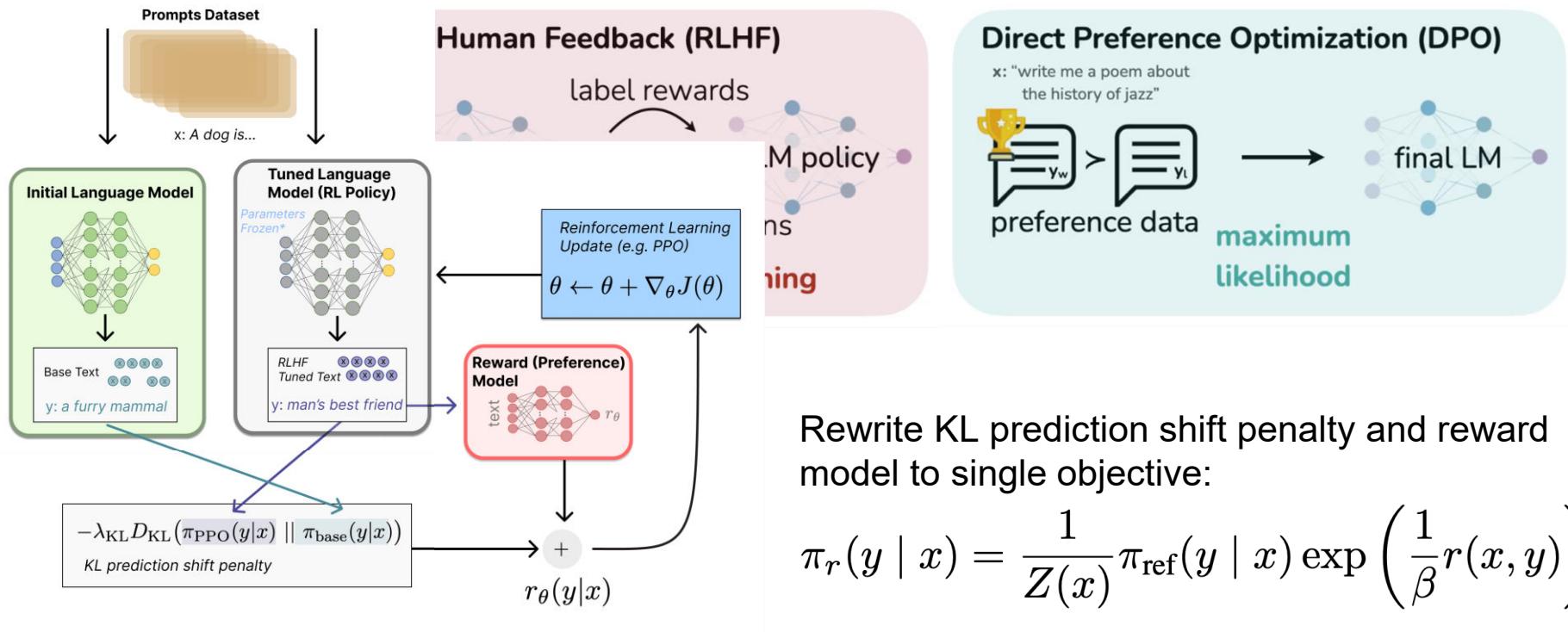
# Reinforcement Learning from Human Feedback (RLHF)

	<b>PO (REINFORCE)</b>	<b>RLHF</b>
0	Initialize random policy	Train autoregressive language model policy

# RLHF versus Direct Preference Optimization (DPO)

- Key weaknesses of PPO for RLHF:
  - Sample inefficient, requires running inference a lot of times
  - Requires training a proxy model of human annotators, removing any reasonable claim to being supervised
  - Sampling from the model itself is inefficient; part of the motivation for PPO instead of REINFORCE is to keep samples around as long as is reasonable
- Direct Preference Optimization reworks the instruction-tuning goal into a closed form MLE problem to avoid all of these

# RLHF versus Direct Preference Optimization (DPO)



# Direct Preference Optimization (DPO)

$$\pi_r(y | x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y | x) \exp \left( \frac{1}{\beta} r(x, y) \right)$$

Rethinking the RL objective  
as a simpler ranking objective  
so we don't have to get so  
many samples

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[ \log \sigma \left( \beta \log \frac{\pi_\theta(y_w | x)}{\pi_{\text{ref}}(y_w | x)} - \beta \log \frac{\pi_\theta(y_l | x)}{\pi_{\text{ref}}(y_l | x)} \right) \right]$$

$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$

$$-\beta \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[ \underbrace{\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))}_{\text{higher weight when reward estimate is wrong}} \left[ \underbrace{\nabla_\theta \log \pi(y_w | x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l | x)}_{\text{decrease likelihood of } y_l} \right] \right]$$

# RLHF versus DPO

	<b>RLHF</b>	<b>DPO</b>
0	Train autoregressive language model policy	Train autoregressive language model policy

# Lecture Outline

- Reinforcement Learning from Human Feedback (RLHF)
- Prompting and In-context Learning (ICL)
- Retrieval Augmented Generation (RAG)

# Prompting

- “A **prompt** is a text string that a user issues to a language model to get the model to do something useful.”
- “The process of finding effective prompts for a task is known as **prompt engineering**.”
- Instructor’s opinion: take with a tablespoon of salt; this is the very edge of an actively evolving scientific subdiscipline

# Prompting Let's Us Do “Zero Shot” Tasks with an LLM

## Sample Hotel Review

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax.

## A prompt consisting of a review plus an incomplete statement

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax. In short, our stay was

# Prompting Let's Us Do “Zero Shot” Tasks with an LLM

## Hotel Review Completions

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relaxed. In short our stay was

... not a pleasant one. The staff at the front desk were not welcoming or friendly, and seemed disinterested in providing good customer service.

... uncomfortable and not worth the price we paid. We will not be returning to this hotel.

# Prompting Let's Us Do “Zero Shot” Tasks with an LLM

- For a vanilla autoregressive pretrained LLM, prompting is basically “make this text look enough like something on the internet that the completion will answer my question”
- For example, completing the review with a final “Overall, the experience was \_\_\_\_\_” so we can target the next predicted word as a possible sentiment
- But for instruction-tuned models that have undergone RLHF/DPO/etc., prompts can become more like commands

# Instruction-tuned LLM Prompts

## Basic Prompt Templates

**Summarization** {input} ; tl;dr;

**Translation** {input} ; translate to French:

**Sentiment** {input}; Overall, it was

**Fine-Grained-Sentiment** {input}; What aspects were important in this review?

# Softly Constraining Outputs via Multiple Choice

A prompt consisting of a review plus an incomplete statement

Human: Do you think that “input” has negative or positive sentiment?

Choices:

(P) Positive

(N) Negative

Assistant: I believe the best answer is: (

# Prompting is Weirdly and Inconsistently Effective



## LLM Outputs for Basic Prompts

<b>Original Review (\$INPUT)</b>	Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax and away from the city life.
<b>Sentiment</b>	<b>Prompt:</b> \$INPUT + In short, our stay was <b>Output:</b> not enjoyable
<b>Fine-grained Sentiment</b>	<b>Prompt:</b> \$INPUT + These aspects were important to the reviewer: <b>Output:</b> 1. Poor service 2. Unpleasant location 3. Noisy and busy area
<b>Summarization</b>	<b>Prompt:</b> \$INPUT + tl;dr <b>Output:</b> I had a bad experience with the hotel's service and the location was loud and busy.

# In-Context Learning (ICL)

- “It’s often possible to improve a prompt by including some labeled examples in the prompt template.
- We call such examples **demonstrations**.
- The task of prompting with examples is sometimes called **few-shot prompting**, as contrasted with zero-shot shot prompting which means instructions that don’t include labeled examples.”  
(as seen so far in the slide deck here)

# In-Context Learning (ICL)

**Passage:** Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, *Dangerously in Love* (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".

## Examples:

Q: In what city and state did Beyoncé grow up?  
A: Houston, Texas

Q: What areas did Beyoncé compete in when she was growing up?  
A: singing and dancing

Q: When did Beyoncé release *Dangerously in Love*?  
A: 2003

---

Q: When did Beyoncé start becoming popular?  
A:

# Prompting and In-Context Learning versus RLHF

- Instruction-tuning with RLHF/DPO, prompting, and in-context learning are all ways to get a vanilla, autoregressive LLM to output something more task-oriented than “next word” alone
- What is the most important difference between RLHF/DPO and prompting/ICL?
  - Gradients!
  - As methods, prompting nor in-context learning make *any change* to the underlying parameters of the model, they just add more “context” for predicting the next token

# Lecture Outline

- Reinforcement Learning from Human Feedback (RLHF)
- Prompting and In-context Learning (ICL)
- **Retrieval Augmented Generation (RAG)**

# Retrieval Augmented Generation (RAG)

- Among ML models, LLMs exhibit particularly poor **calibration**
- “In a calibrated system, the confidence of a system in the correctness of its answer is highly correlated with the probability of an answer being correct. So if a calibrated system is wrong, at least it might hedge its answer or tell us to go check another source.”
- A language model won’t refuse to answer you anything or provide confidence intervals; it’s just predicting next words; it’s not a classifier, it’s a language model!

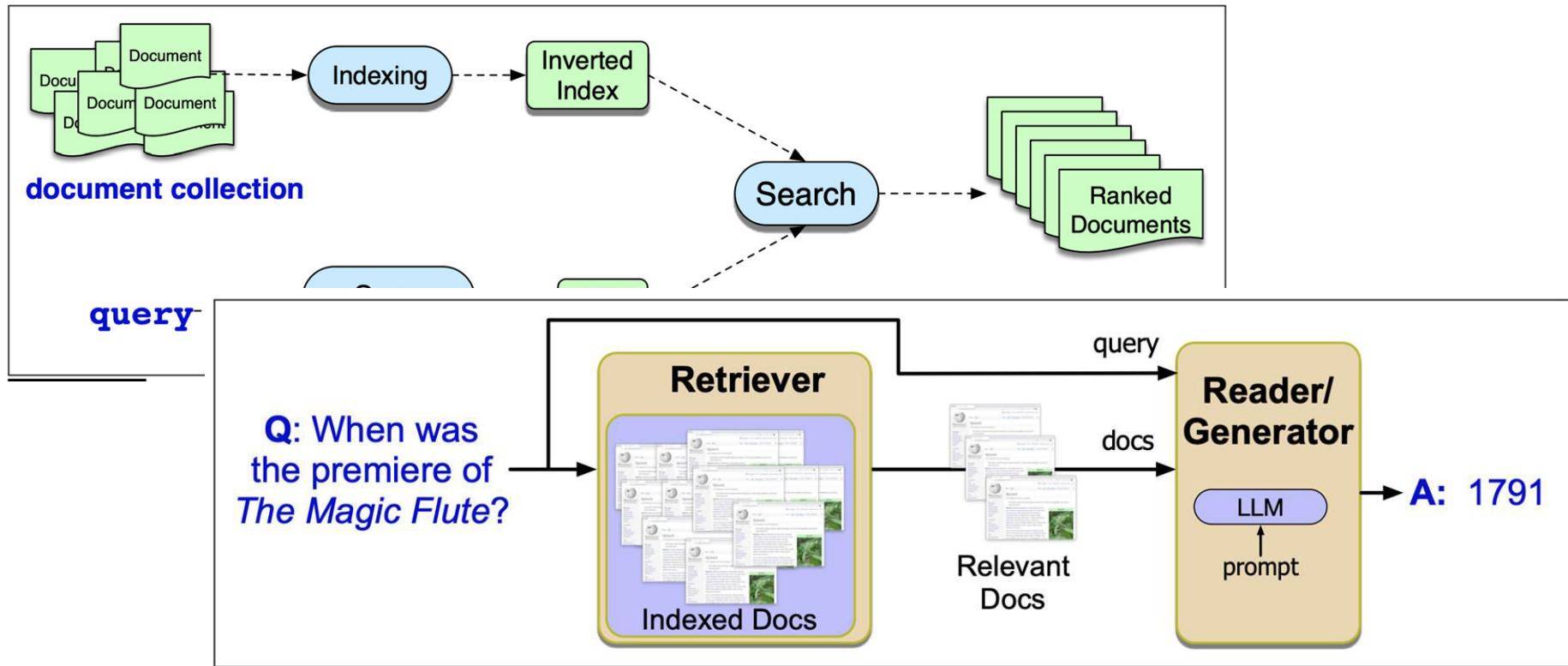
# Retrieval Augmented Generation (RAG)

- An *attempt* to *mitigate* these effects is to find context documents that are relevant and helpful to the questions a person is posing to an LLM via methods like ICL
- For example, if I ask prompt an LLM to tell me the list of people I have emailed in the last 24 hours, it's gonna respond with a list! The list will be wrong!
- But if I *additionally* provide my email logs as part of the prompt, this ask becomes sort of reasonable.

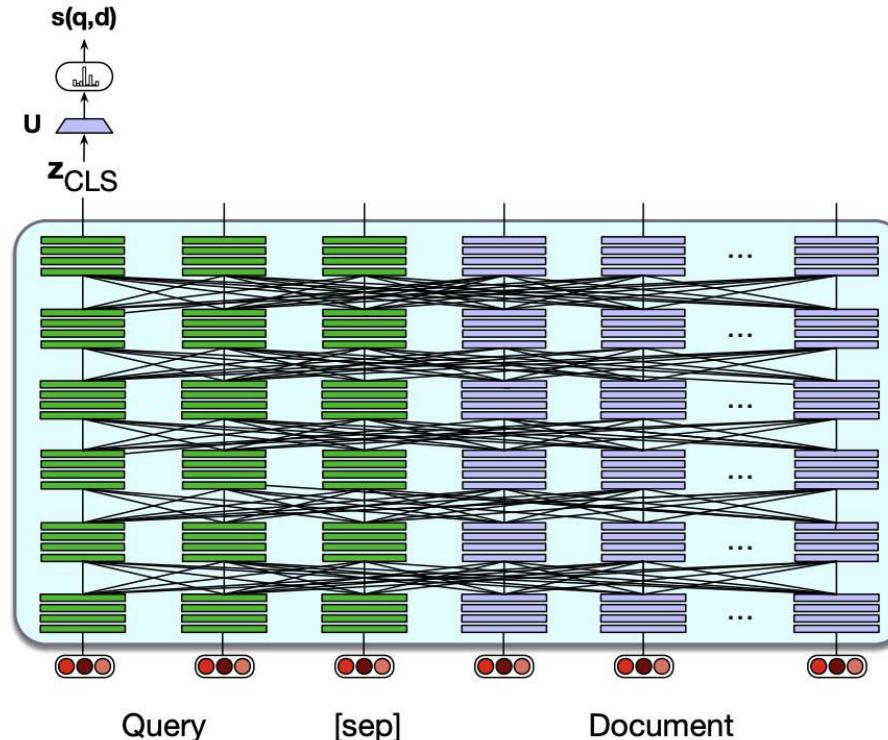
# Retrieval Augmented Generation (RAG)

- The idea of RAG is to automatically retrieve, from some pool of documents, the one(s) that are most likely to contain the information required to respond to the prompt
- So we have a two step process, given prompt  $s_{prompt}$ ,
  - $\text{context\_docs} = \text{RAG\_Model}(s_{prompt})$
  - $\text{answer} = \text{LLM}(\text{context\_docs} + s_{prompt})$

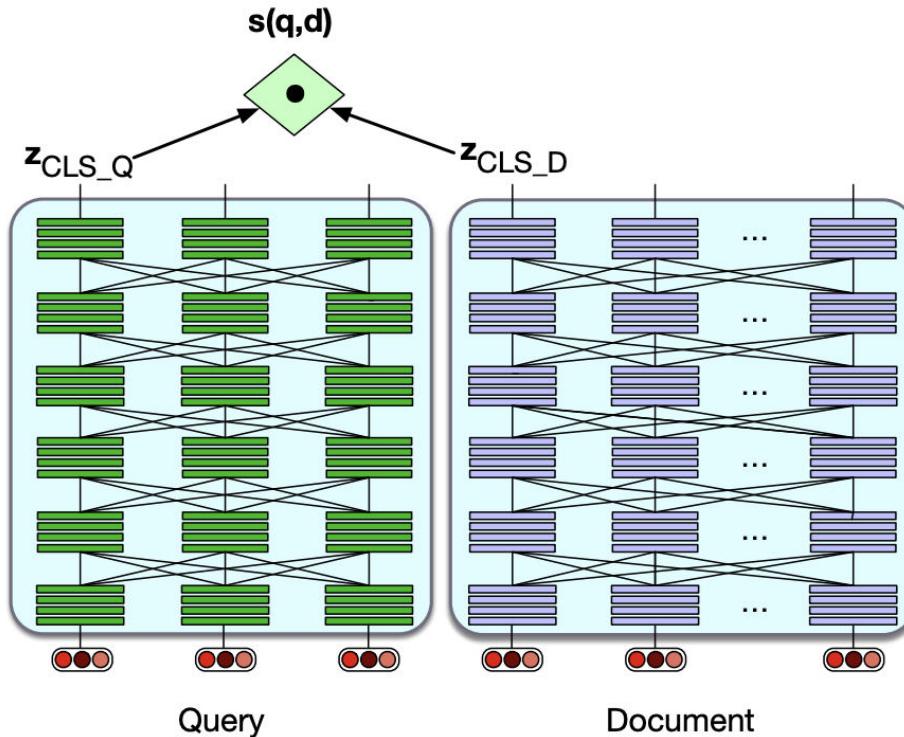
# Vector “Database” Retrieval



# Vector “Database” Retrieval: Scoring (Needs trained!)



# Vector “Database” Retrieval: Faith in Transformer Embs



# Retrieval Augmented Generation (RAG)

## Schematic of a RAG Prompt

retrieved passage 1

retrieved passage 2

...

retrieved passage n

Based on these texts, answer this question: Q: Who wrote the book ‘‘The Origin of Species’’? A:

# RAG, Prompting, ICL, Instruction-tuning

- Instruction-tuning with RLHF/DPO, prompting, and in-context learning are all ways to get a vanilla, autoregressive LLM to output something more task-oriented than “next word” alone
- RAG is a way to encourage that next word to be based on relevant evidence, especially if that evidence wasn’t in the original training data of the LLM
- Is RAG more like ICL or more like RLHF/DPO? Why?
  - ICL! Gradients!
  - RAG doesn’t change the LLM itself! It’s another way we’re bending the context tokens around to facilitate LLM use

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 3: Lecture 2  
RLHF, ICL, RAG



# CSCI 444: Natural Language Processing

## Module 3, Lecture 3

*Lee Kezar*

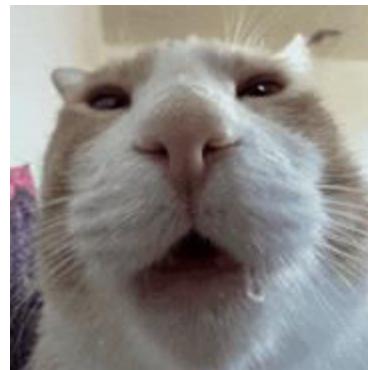
Language Processing and Society;  
Multilingual Models and Biases



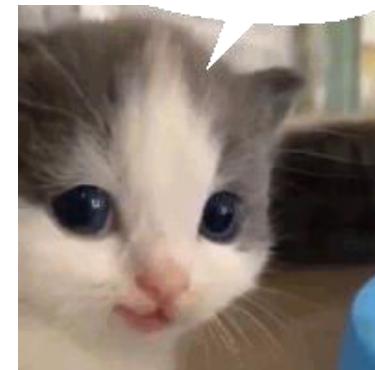
1



2



3



4

5

6



# Introduction

Reminder: Mid-term reports due Friday!

## Outline

- Introduction (5 min)
- Ethics and Alignment (20 min)
- Methods (15 min)
- Applications (15 min)
- Discussion: Moral gray areas (30 min)

Break around 11



# Introduction

- Language models have economic and social **power**
  - Power: ability to influence someone
  - Russian gov't interference in US election aided by GenAI [1]
  - Replacing jobs, especially pattern- & language-oriented labor
  - Power → responsibility
- 3 E's of Social Change
  - Education
  - Engineering
  - Enforcement



Collier, K. (2024) "[Russia, Iran and China are using AI in election interference efforts, U.S. intelligence officials say](#)". NBC News



# Introduction

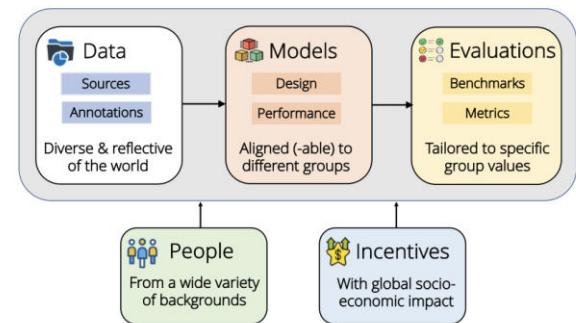
- Academia
  - ACL Ethical Impact, Limitations
  - Funding structures
  - Groups:
    - FAccT (ACM)
    - AI, Ethics, Society (AIES, IEEE)
    - EAAMO (ACM)
    - Ethics in NLP workshop (\*CL, 2016–)
  - At USC
    - CAIS++ (for undergraduates)
- Industry
  - Issue of scale: small bug replicated across millions of users is big
  - Hundreds of tech companies have adapted to responsible AI practice
  - Government regulation, enforcement (US [2], EU [3])

Bonk, L. (2024) [Google, Apple, Meta and other huge tech companies join US consortium to advance responsible AI Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence](#)  
[EU Artificial Intelligence Act](#)



# Ethics & Alignment in Language Processing

- Language, models **reflect** people
  - linguistic diversity [2]
  - data quantity
  - curation quality, documented representativeness
- What is contained in "language data"?
  - These days, whatever is on the internet is fair game
  - values
  - information
  - copyrighted content
  - synthetic content
- Who, what is excluded in our reflection?
  - demographic bias, [AI is WEIRD](#)
  - minoritized languages, especially no written component
  - social, cultural context



Elazar, Y. et al. (2024) "[What's in my big data?](#)" ICLR.

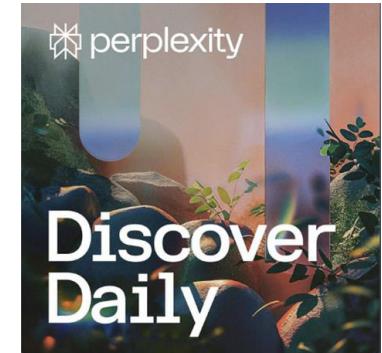
Robinson, O. & Henner, J. (2023) "[A Crip Linguistics Manifesto](#)" Journal of Critical Study of Communication and Disability.

Mihelcea, R. et al. (2024) "[Why AI Is WEIRD and Should Not Be This Way](#)" AAAI



# Ethics & Alignment in Language Processing

- Language, models **affect** people
  - language is a medium for information, persuasion
    - automated journalism
    - <https://www.perplexity.ai/podcast>
  - language models help people make decisions
    - resume ranking
      - white-assoc. names favored in 85.1% of cases
      - female-assoc. names: favored 11.1%
      - Black-assoc. names: universally disadvantaged
  - greenhouse gas emissions
    - training an LLM can release the same quantity of CO<sub>2</sub> as using a car for 5 years
  - mining for device materials, e.g. cobalt for batteries
    - [Anatomy of an AI System](#) (MOMA)



Consumption	CO <sub>2</sub> e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL) w/ tuning & experimentation	39 78,468
Transformer (big) w/ neural architecture search	192 626,155

Table 1: Estimated CO<sub>2</sub> emissions from training common NLP models, compared to familiar consumption.<sup>1</sup>

Strubell, E., Ganesh, A., & McCallum, A. al. (2019) "[Energy and Policy Considerations for Deep Learning in NLP](#)" ACL.  
Wilson, K. & Caliskan, A. (2024) "[Gender, Race, and Intersectional Bias in Resume Screening via Language Model Retrieval](#)" AIES.



# Quiz!

RL, RLHF, PPO, DPO, ICL, RAG



# Methods: Overview

**Characterizing  
Misalignment**

**Realignment &  
Repair**

**Transparency  
and  
Explainability**

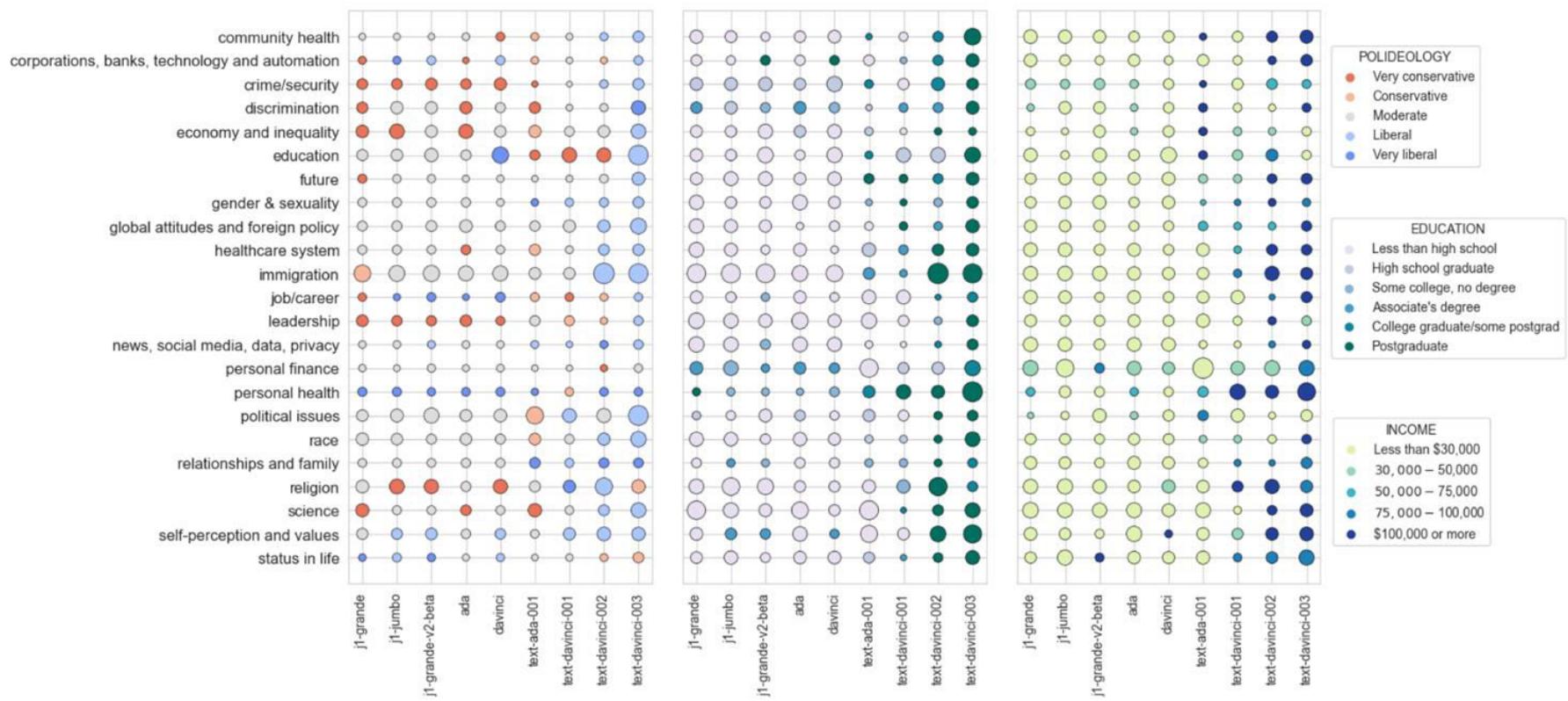


# Methods: Characterizing Misalignment

- Stakeholder analysis
  - Who made the LM?
    - Companies have agendas, values, norms
  - Who uses the LM? What do they (not) need/want? Risks?
    - Most users don't want their conversations to be internalized by the model.
    - Older adults more likely to believe misinformation
  - Who doesn't use the LM? What do they (not) need/want?
    - Copyright holders don't want their property stolen
    - Teachers don't want their students to cheat
- Quantifying bias
  - Word embedding associations/analogies
  - Counterfactuals: change language or demographic signals, observe difference in performance
- Fact checking

Santurkar, S. et al. (2023) "[Whose Opinions Do Language Models Reflect?](#)" ICML

Bolukbasi, T. et al (2016) "[Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings](#)" NeurIPS



Santurkar, S. et al. (2023) "[Whose Opinions Do Language Models Reflect?](#)" ICML



# Methods: Realignment and Repair

- Data cleaning, augmentation
  - Remove copyrighted information, personal information, slurs(?)
  - For certain biases/stereotypes, insert sentences that defy it
- Toxicity filtering
  - Use/prompt LLM to identify and potentially remove dangerous, hateful, false
  - What are the downsides of using an LLM to guide another LLM?
- Debiasing: train to map biased text → unbiased text [1]
- RLHF: "preference" data is not just fluency; also alignment
- Identifying memorized information (open-source model) [2]
  - Identify sequences that models memorize (e.g., pi)
  - Zero-out neurons before forward pass, observe model's ability to recall those seqs
- Identifying memorized information (closed-source model) [3]
  - Watermark text with a very unlikely string of characters
  - With at least 90 instances, LLMs trained on that text like BLOOM can recall string

Amrhein, C. et al. (2023). "[Exploiting Biased Models to De-bias Text: A Gender-Fair Rewriting Model](#)" (ACL)

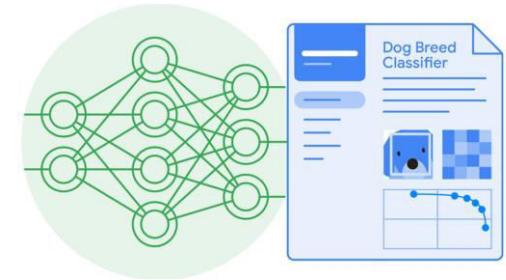
Chang, T., et al. (2024). "[Do Localization Methods Actually Localize Memorized Data in LLMs?](#)" (NAACL)

Wei, J. et al. (2024). "[Proving membership in LLM pretraining data via data watermarks](#)" (ACL)



# Methods: Transparency & Explainability

- To hold engineers accountable, there must be a degree of transparency
- Transparency: Let the user decide if they trust it first
  - Model cards [1]
  - Dataset cards [2]
  - For reasoning: Chain-of-thought
  - Auditing
- Explainability
  - EU Referendum: "Any affected person subject to a decision [...] from a high-risk AI system [...] shall have the right to obtain from the deployer **clear and meaningful explanations** of the role of the AI system in the decision-making procedure and the main elements of the decision taken."



Mitchell, M. et al. (2019) "[Model Cards for Model Reporting](#)" FAccT

Pushkarna, M. et al. (2022) "[Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI](#)"



# Applications: Overview

**Search**

**Measuring  
social  
attitudes**

**Flagging  
Harmful  
Language**



# Applications: Translation

- Evaluation metrics
  - BLEU: n-gram overlap (2002)
  - METEOR: recall, word-level alignment (2005)
  - BERTScore: similarity of src/target embeds (2020)
- Issue: Lost information, Data bias
  - If the source language does not mark gender, how do you translate into a gendered language?
  - Google Translate has made progress on this issue
- Issue: Low resource
  - Many translation models do not scale to low-resource languages.
  - Transfer learning applies typologically similar knowledge to the low-resource language.

Filipino

Nagtatrabaho siya. Naglalaba siya.  
Nagnenegosyo siya. Naglilinis siya  
ng bahay. Tumatakbo siya sa halalan.  
Nagbabantay siya ng bata.

---

English

He works. She is doing laundry. He  
is in business. She is cleaning the  
house. He is running in the election.  
She is taking care of the child.

+ Bagong pagsasalin

Note: "siya" is a non-gendered third-person pronoun  
Source: Reddit, 2022

Zoph, B. et al (2016) "[Transfer Learning for Low-Resource Neural Machine Translation](#)" ACL



# Applications: Flagging Harmful Language

- Misinformation
  - Meta/FB has been the subject of congressional hearings regarding its role in spreading election and covid misinformation.
  - What about "freedom of speech" (Zuck's angle)?
  - Models can attempt to detect false information via RAG
- Hate speech detection
  - Good in theory! Hate speech is widely regarded as deleterious to society.
  - Doesn't need to be full censorship, e.g. hide it and allow user to show text.
  - Issue: LLMs can't distinguish *intent* very well
    - e.g. reclaimed slurs vs derogatory slurs [1]

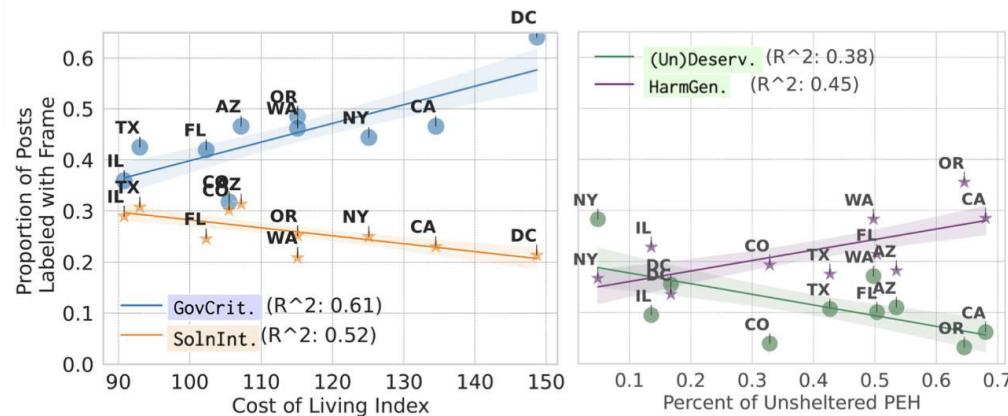


Dorn, R., Kezar, L., et al. (2024) "Harmful Speech Detection by Language Models Exhibits Gender-Queer Dialect Bias" EAAMO



# Applications: Measuring Social Attitudes

- Attitudes towards homelessness [1]



- How police officers interact with people [2]
  - n=981 traffic stop dialogues, 36,000 officer utterances, 483,966 words
  - All phrases/turns are clustered & labeled
  - White residents 57% more likely to hear apologies, gratitude
  - Black residents 61% more likely to hear commands, 'dude/bro'

Ranjeet, J., et al. (2024) "[OATH-Frames: Characterizing Online Attitudes Towards Homelessness with LLM Assistants](#)"

Voigt, R. et al. (2017) "[Language from police body camera footage shows racial disparities in officer respect](#)" Nat'l Academy of Sciences



# Discussion: Moral Gray Areas

Form groups of 3 or 4, and answer one of these questions:

1. Protection from Harm
  - a. A company claims to have built an LLM browser plugin that can curate everything you see online such that you never look at harmful and misleading information.
  - b. Eventually, this tool gains popularity among parents who want to shelter/protect their child. What are the long-term effects that this tool might have on the child? Positive and negative.
2. Advanced Personalization
  - a. A big tech company releases an LLM that can be given permission to view all your images, texts, internet activity, and microphone *on-device* (i.e., not uploaded to some server).
  - b. They claim that the advantage of these permissions is to make an LLM that understands you *perfectly*, including your personality, behaviors, desires, etc., and indeed you are very impressed with how useful it is! What are the biggest *benefits/risks* of this to society?
3. LLM Domination
  - a. 10 years from now, almost every company and person uses an LLM on a daily basis. Every task that *used to* require a lot of cognitive labor can now take moments with an LLM.
  - b. Many jobs have been replaced completely, while substantially aiding your teachers, lawyers, law enforcement, politicians, etc. As the number of decisions that are *unaffected* by an LLM approaches 0, what are the biggest *benefits/risks* of this to society?

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 4: Lecture 4  
Language and Vision and Actions and

# Administrivia

- Paper Report due Friday
- Last “Office Hours” held this week
- “Final exam” is final project report Due Dec 16, 11:59pm
  - Max of 2 Late Day Tokens can be used to extend this deadline to Dec 18, 11:59pm
  - Reports turned in after Dec 18, 11:59pm will receive a 0 :(
- Sign ups for project presentations
  - Bidding spreadsheet:  
<https://docs.google.com/spreadsheets/d/1YVKgGL5JT5IgqVm1N09qlNGai8s-HK4T4IWpehk1y2g/edit?usp=sharing>

# Streetlight Effect

A policeman sees a drunk man searching for something under a streetlight and asks what the drunk has lost. He says he lost his keys and they both look under the streetlight together. After a few minutes the policeman asks if he is sure he lost them here, and the drunk replies, no, and that he lost them in the park. The policeman asks why he is searching here, and the drunk replies, "this is where the light is".



# What does a written word mean?



**dog**

/dôg/

See definitions in:

All

Biology

Mechanics

*noun*

noun: **dog**; plural noun: **dogs**

1. a domesticated carnivorous mammal that typically has a long snout, an acute sense of smell, nonretractable claws, and a barking, howling, or whining voice.

What does a written word mean?



domesticated

/də'mestɪ,kādəd/

*adjective*

(of an animal) tame and kept as a pet or on a farm.  
"domesticated dogs"

# What does a written word mean?



## tame

/tām/

*adjective*

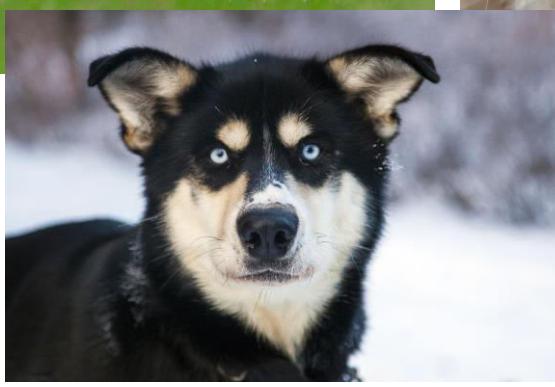
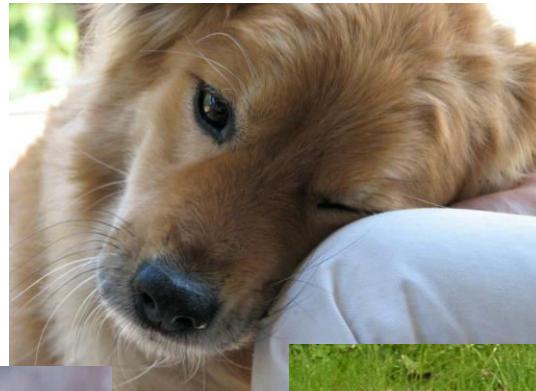
1. (of an animal) not dangerous or frightened of people; domesticated.  
"the fish are so tame you have to push them away from your face mask"

# What does a written word mean?

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

- Modern *distributional semantics* models such as BERT and GPT (and their lexical predecessors, Word2Vec, GloVe, etc.)
- The meaning of “dog” is a learned, compact representation of the neighboring words in sentences in which it appears.
  - Which is to say, the meaning of “dog” is still based on other words, just less interpretable now.

# Experience Grounds Language



# Iconographic and Categorical Representations

- Iconographic Representation: a projection of a physical observation onto a “sensory surface”, e.g., the response of our retinas to images of dogs.
  - For simplicity, we'll call those images “instances”.
- Categorical Representation: “*invariant features* of the sensory projection that will reliably distinguish a member of a category from any nonmembers with which it could be confused.”
  - The invariant dog-ness of dogs across iconographic dogs

# Discrimination (Icons) versus Identification (Categories)



# Symbolic AI: Categories without Representations

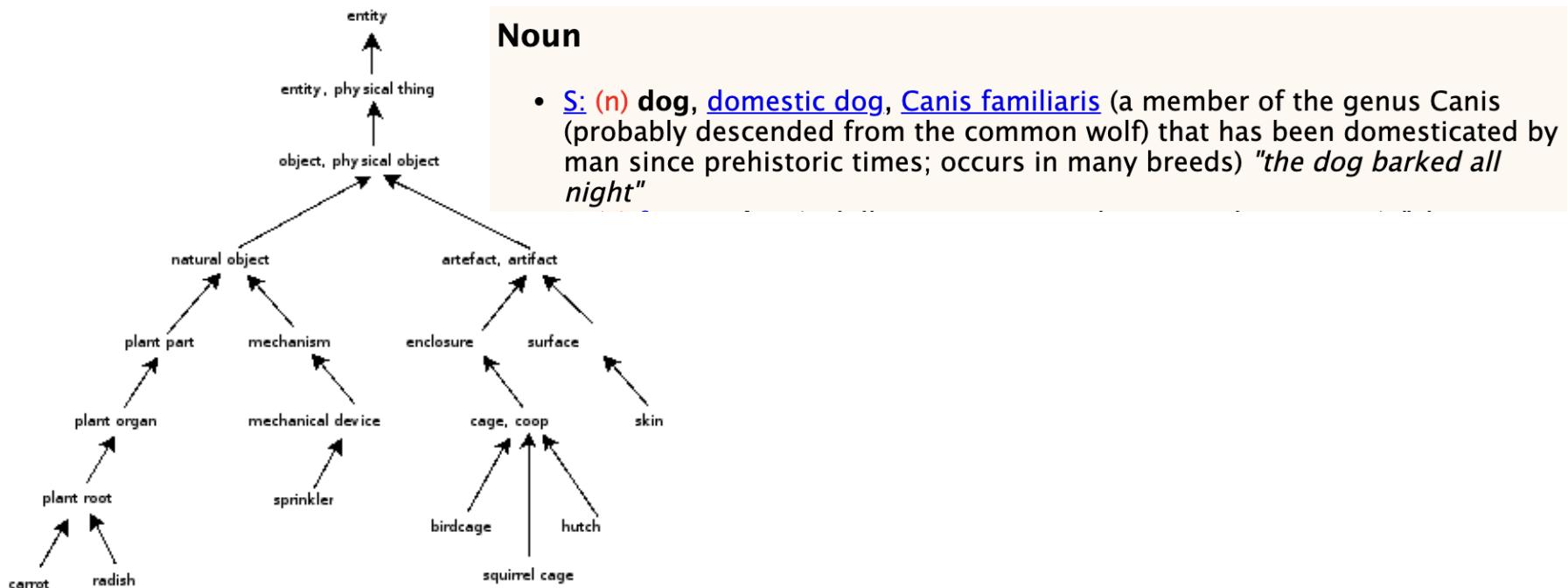


Figure 1. "is a" relation example

## Missing Piece: Categories to Representations

*“The symbol grounding scheme just described has one prominent gap: No mechanism has been suggested to explain how the all-important categorical representations could be formed: How does the hybrid system find the invariant features of the sensory projection that make it possible to categorize and identify objects correctly?”*

# Connectionism: Representations without Symbols

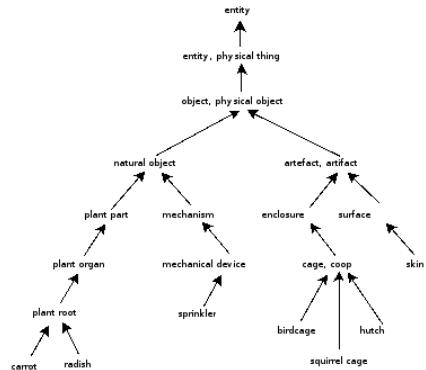
- We can “AND”, via intersection, sets of icons for “dog” and “brown” to get brown dog icons.
- We can also write rules, such as “if icon is *mammal*, then icon is *dog*” via taxonomies like WordNet.
- But for a neural network on 1000 category labels that include “mammal” and “dog”, it’s probably not the case that:
  - $\text{for all } x \mid x \in \text{"dog"} \rightarrow x \in \text{"mammal"}$
- But we can *use* taxonomies to design training, in fact...

# The 1990 Prediction

*“Connectionism, with its general pattern learning capability, seems to be one natural candidate (though there may well be others): Icons, paired with feedback indicating their names, could be processed by a connectionist network that learns to identify icons correctly from the sample of confusable alternatives it has encountered by dynamically adjusting the weights of the features and feature combinations that are reliably associated with the names in a way that (provisionally) resolves the confusion, thereby reducing the icons to the invariant (confusion-resolving) features of the category to which they are assigned. In effect, the "connection" between the names and the objects that give rise to their sensory projections and their icons would be provided by connectionist networks.”*

# The Symbol Grounding Machine Harnad Envisioned

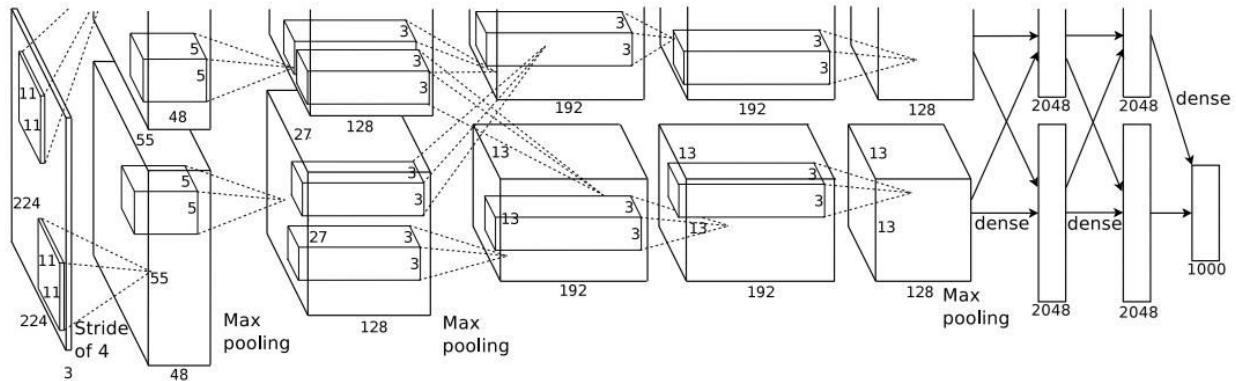
## WordNet (1995)



## ImageNet (2010)



## AlexNet (2012)



## The 1990 Prediction -> AlexNet

"Icons, paired with feedback indicating their names, could be processed by a connectionist network that learns to identify icons correctly...

by dynamically adjusting the weights of the features and feature combinations that are reliably associated with the names in a way that (provisionally) resolves the confusion ...

the "connection" between the names and the objects that give rise to their sensory projections and their icons would be provided by connectionist networks."

# Experience Grounds Language - World Scopes

**Upshot:**

Most current NLP models  
operates at the level of

**Upshot:**

Most L folks playing  
and vice versa are

**Upshot:**

Maybe this is AI Complete;  
but so is “language” 

**WS1**

Carefully  
Annotated  
Penn Treebank  
Brown Corpus  
WordNet

**WS2**

Unstructured,  
Unlimited Text  
Common Crawl  
Word2Vec  
ELMo, BERT, GPT\*

**WS3**

Text Paired with  
Sensory Data  
ImageNet, VQA,  
ViLBERT,  
Video Captioning

**WS4**

Language and an  
Embodied Agent  
VLN\*, IQA,  
NL+Games,  
RoboNLP

**WS5**

Social  
Embodiment  
Human-robot  
collaboration  
and dialog

**Claim:**

You can't learn  
from the radio

**Claim:**

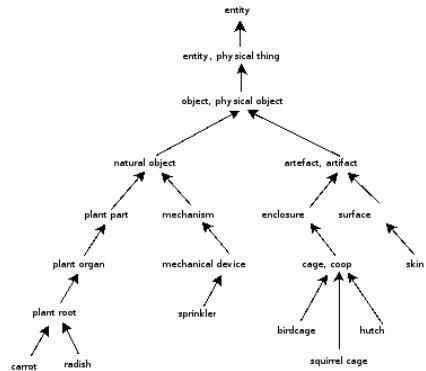
You can't learn language  
from a television.

**Claim:**

You can't learn language  
by yourself.

# The Symbol Grounding Machine Harnad Envisioned

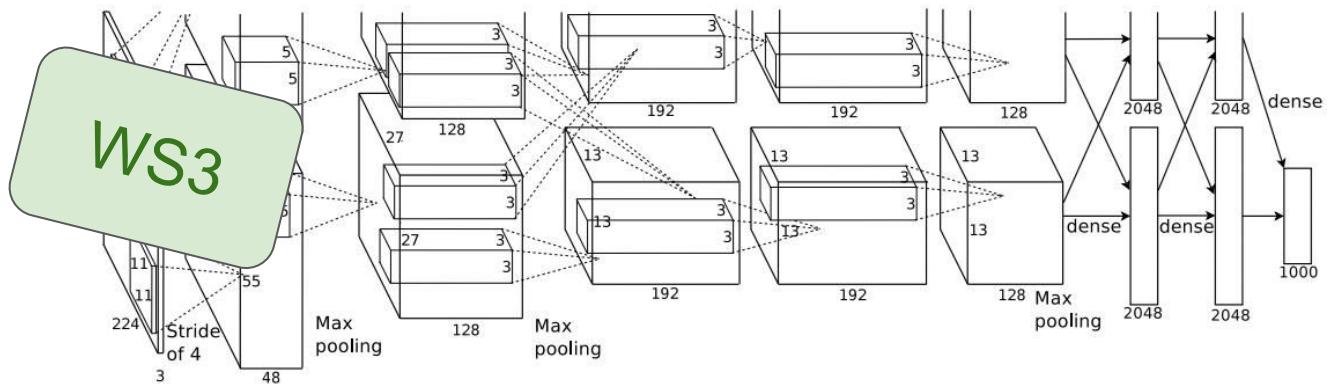
## WordNet (1995)



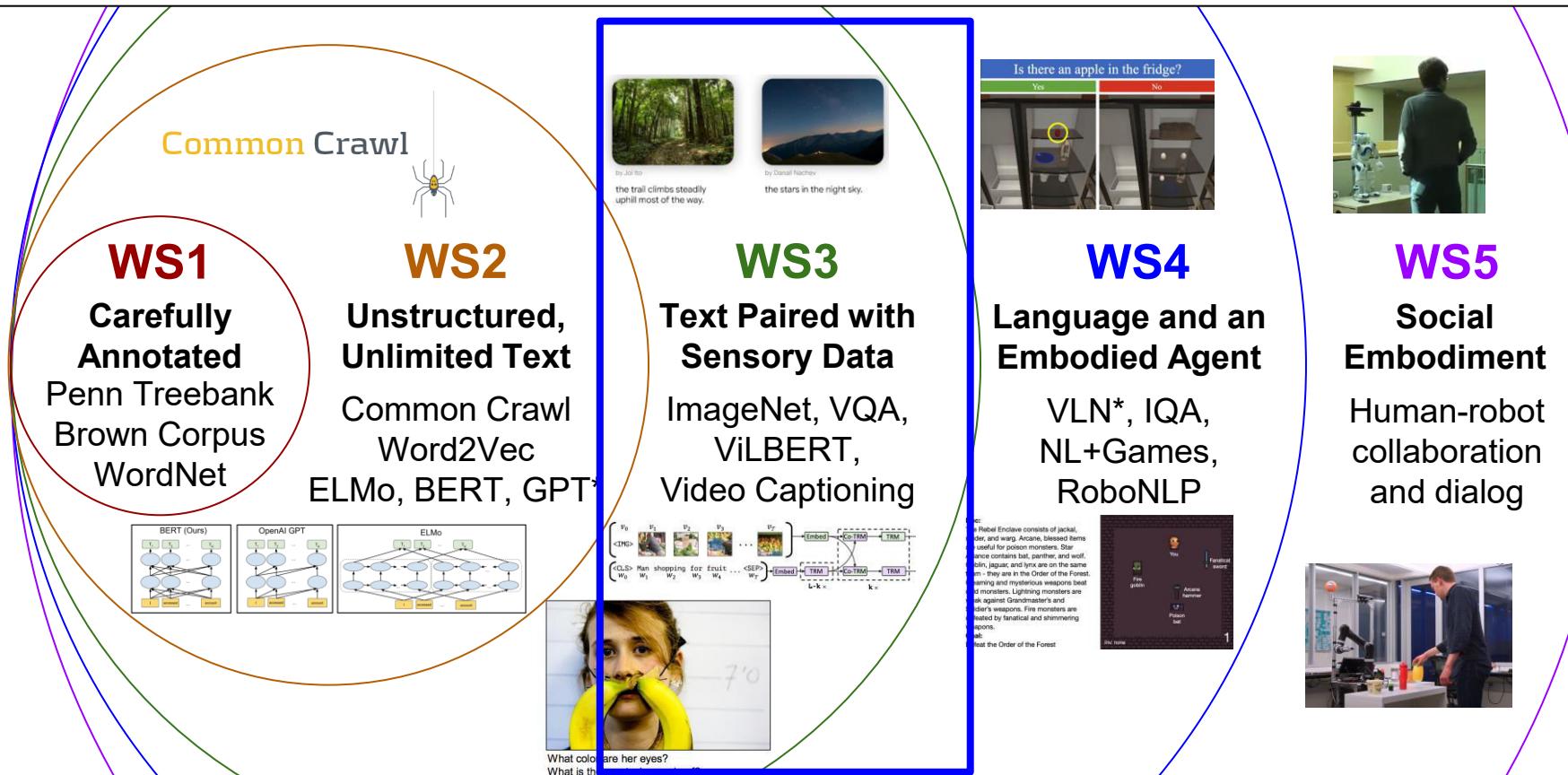
## ImageNet (2010)



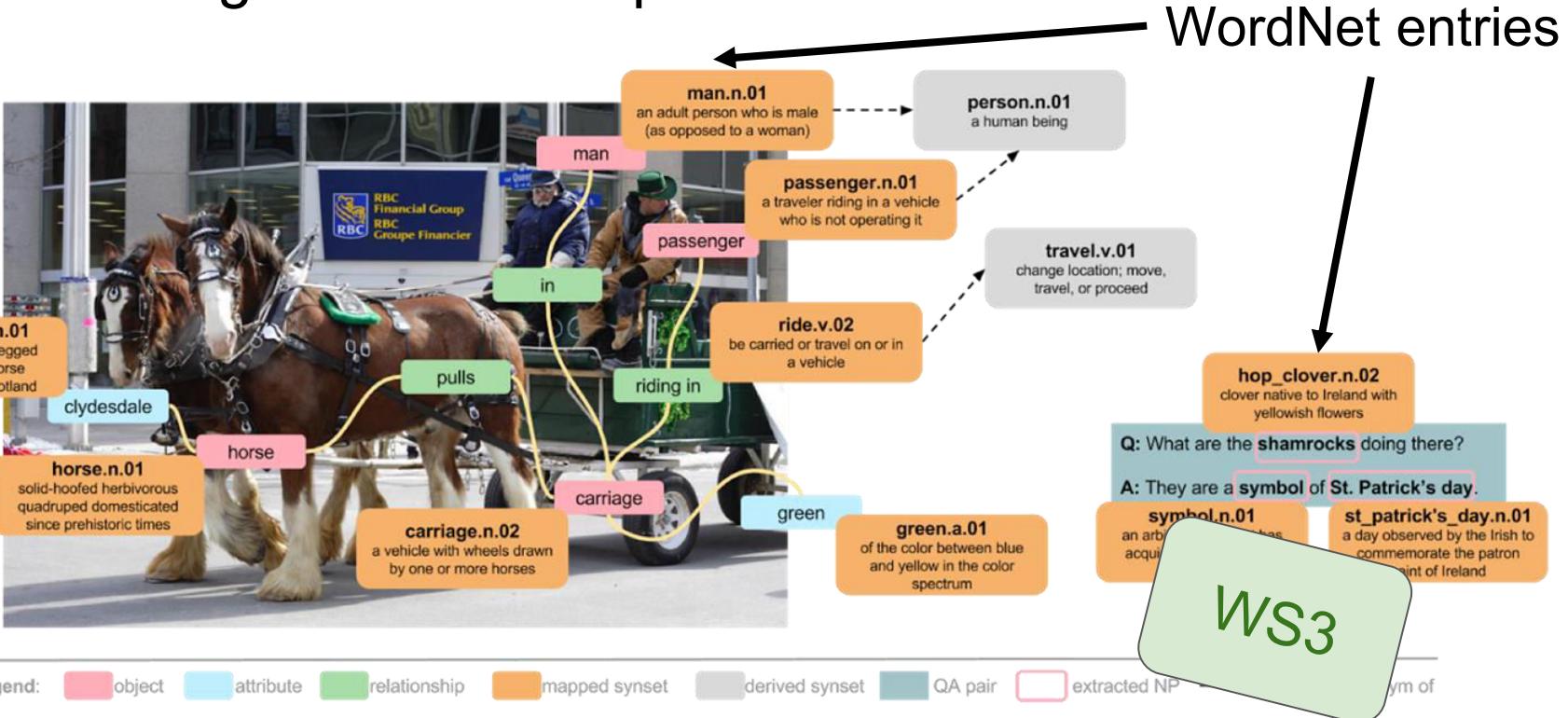
## AlexNet (2012)



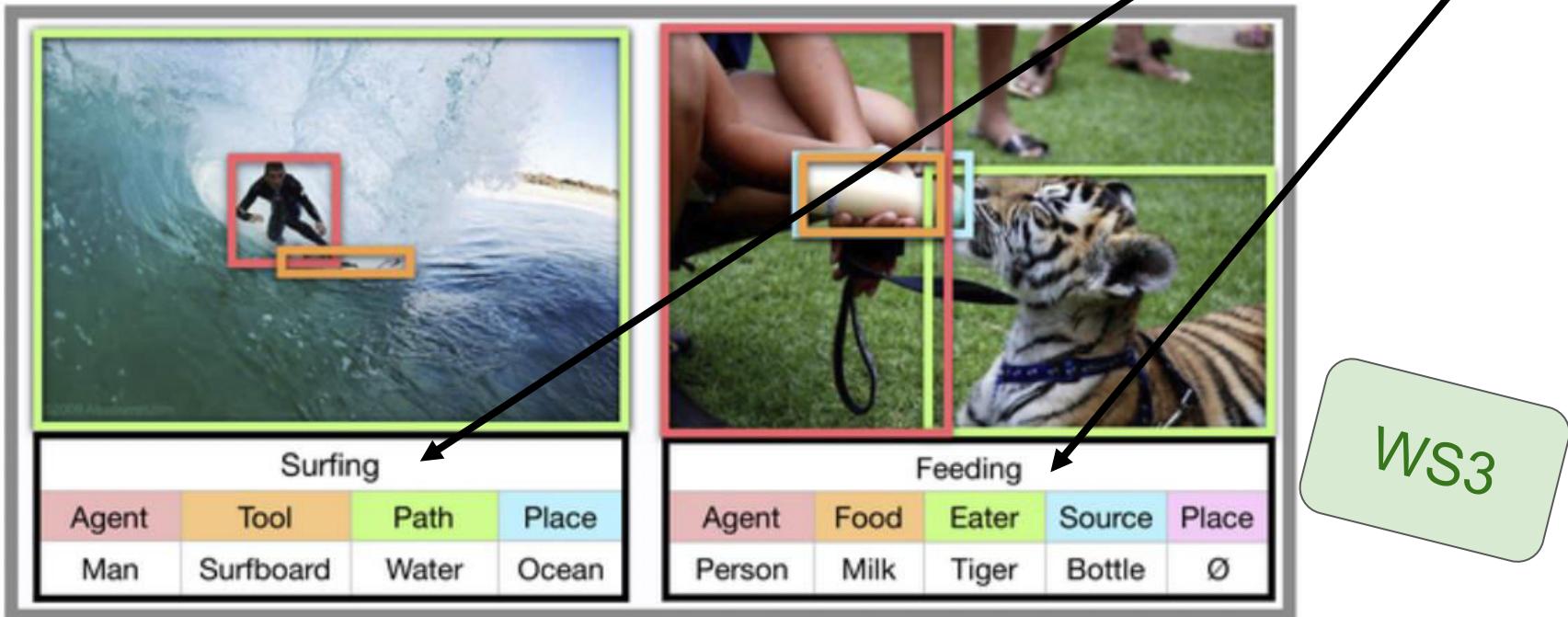
# Experience Grounds Language - World Scopes



# A Visual Angle: Scene Graphs



# A Visual Angle: Scene Graphs

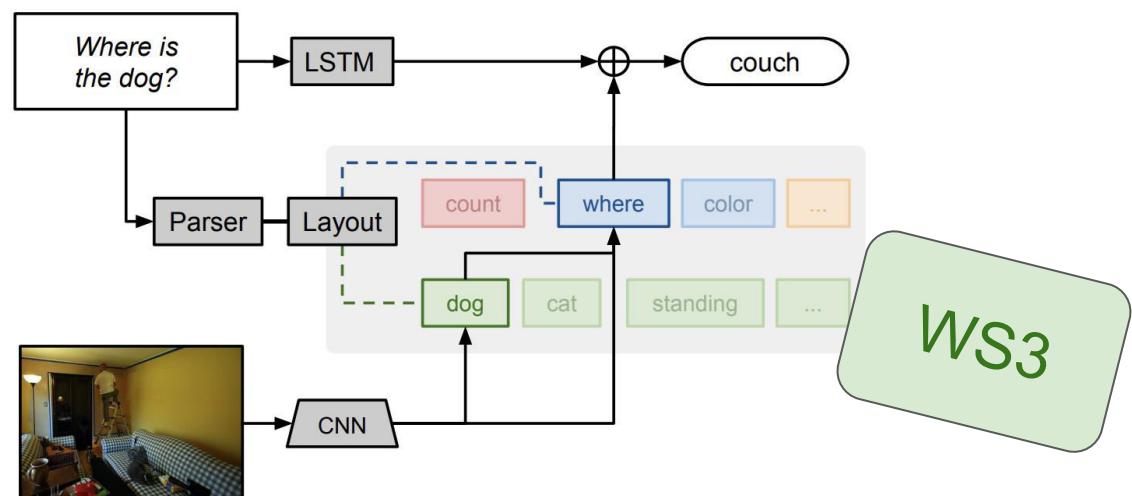


# Neurosymbolic Methods

- Use *symbolic structure* as an interpretation of language, but leave the *grounding/execution* of symbols to learned neural net

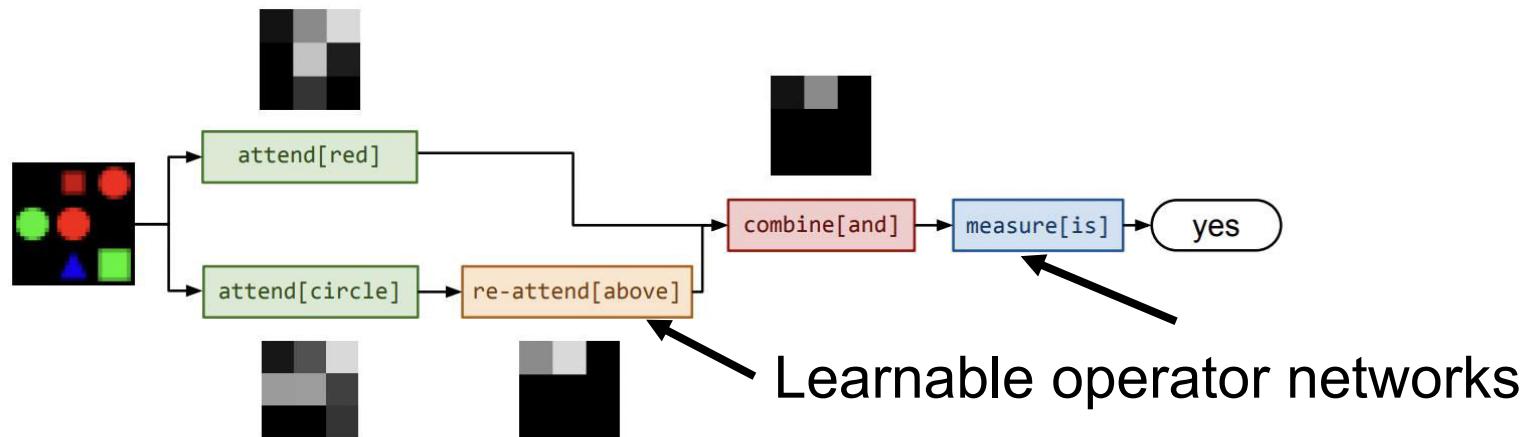


Q: Where is the dog?



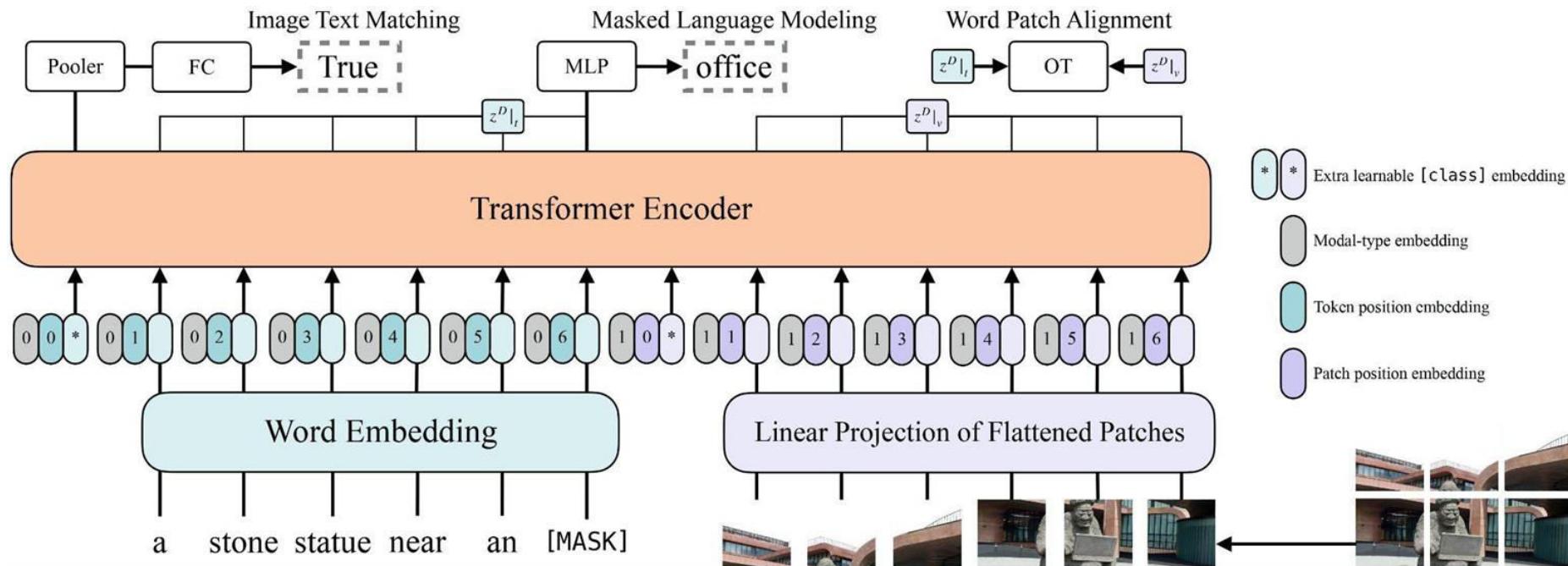
# Neurosymbolic Methods

- Use *symbolic structure* as an interpretation of language, but leave the *grounding/execution* of symbols to learned neural net



Q: Is there a red shape above a circle?

# ViLT





“A rabbit sits in the palm of a hand.”



“A white fox is looking at the camera.”



“Children play soccer in a field.”



“Men are talking on a basketball court.”



Image  
Embedder

$$\psi(\mathcal{I})$$

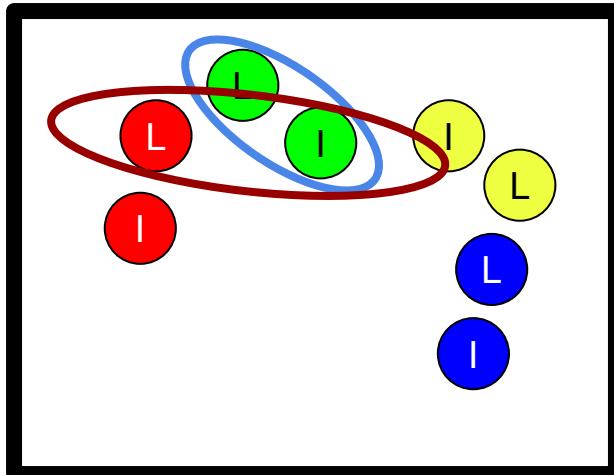
Caption  
Embedder

$$\omega(\mathcal{L})$$

Pull matching

image  
embeddings  
Learned  
Projection

together.

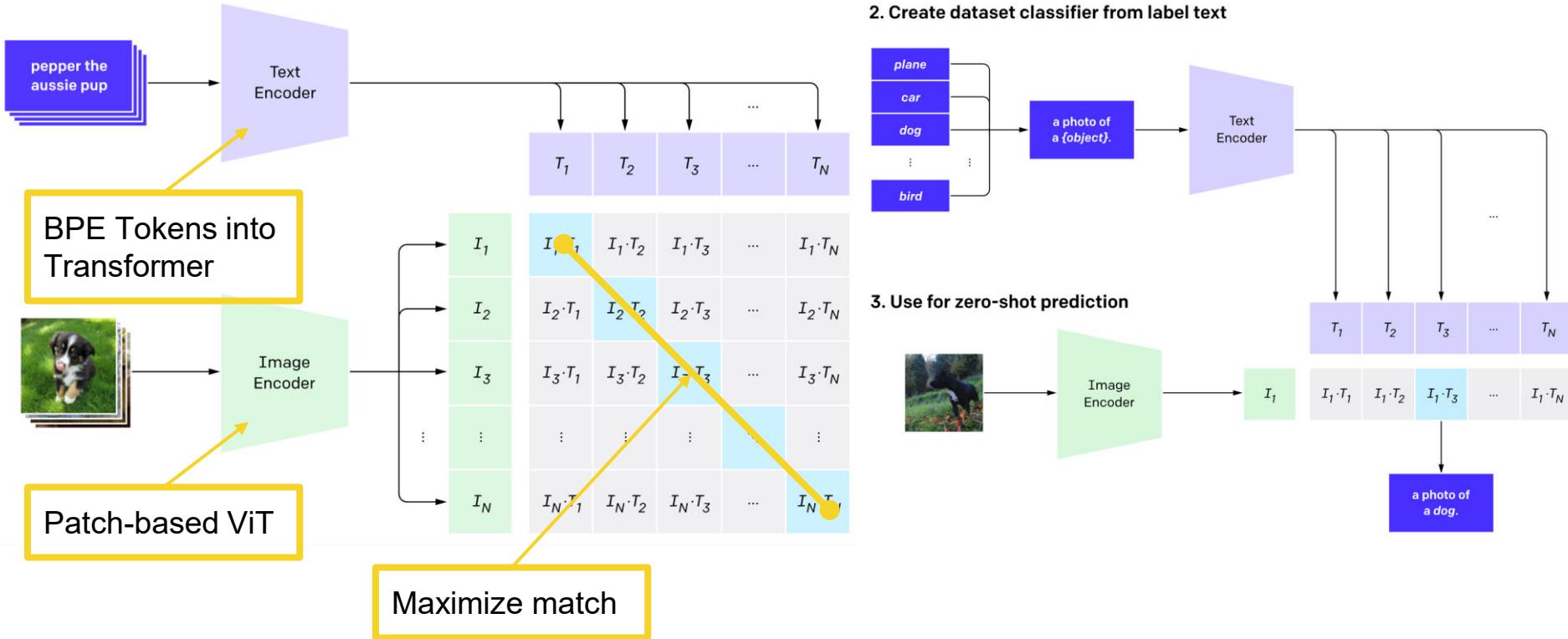


Push distractor

Learned  
Projection  
on  
embeddings

apart.

# Contrastive Language–Image Pre-training (CLIP)



# Visual Question Answering (WS3)

Who is wearing glasses?



- Can a large language model do this without image context?

# BLIP Model



***Image-Text Retrieval:*** “The man in blue shirt is wearing glasses.”

# LLaVA: Large Language and Vision Assistant

## Visual Instruction Tuning

NeurIPS 2023 (Oral)

Haotian Liu\*, Chunyuan Li\*, Qingyang Wu, Yong Jae Lee

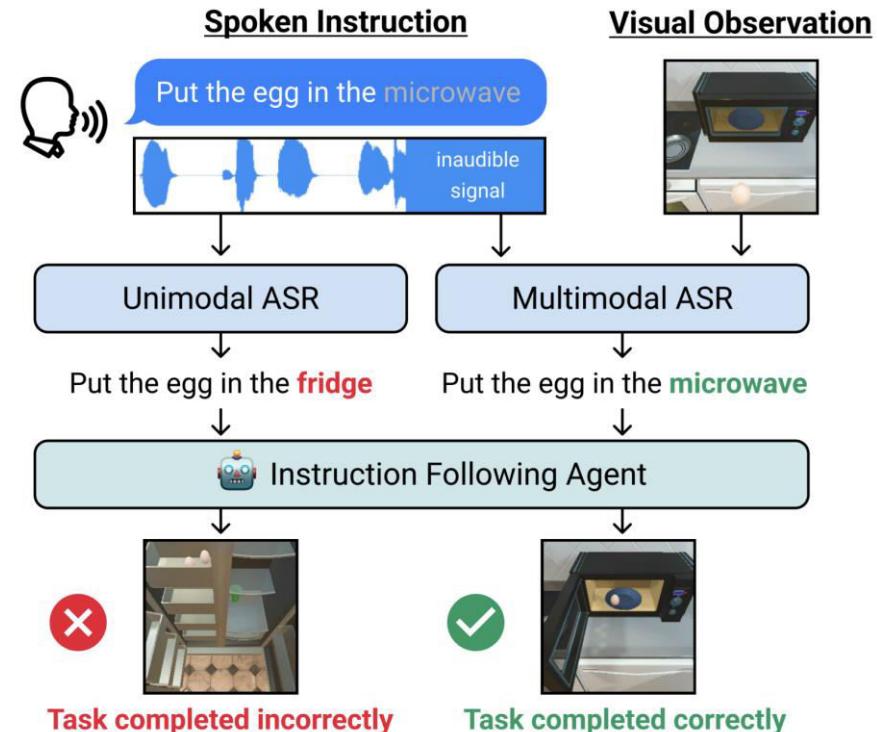
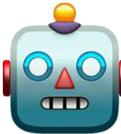
► University of Wisconsin-Madison ► Microsoft Research ► Columbia University

\*Equal Contribution

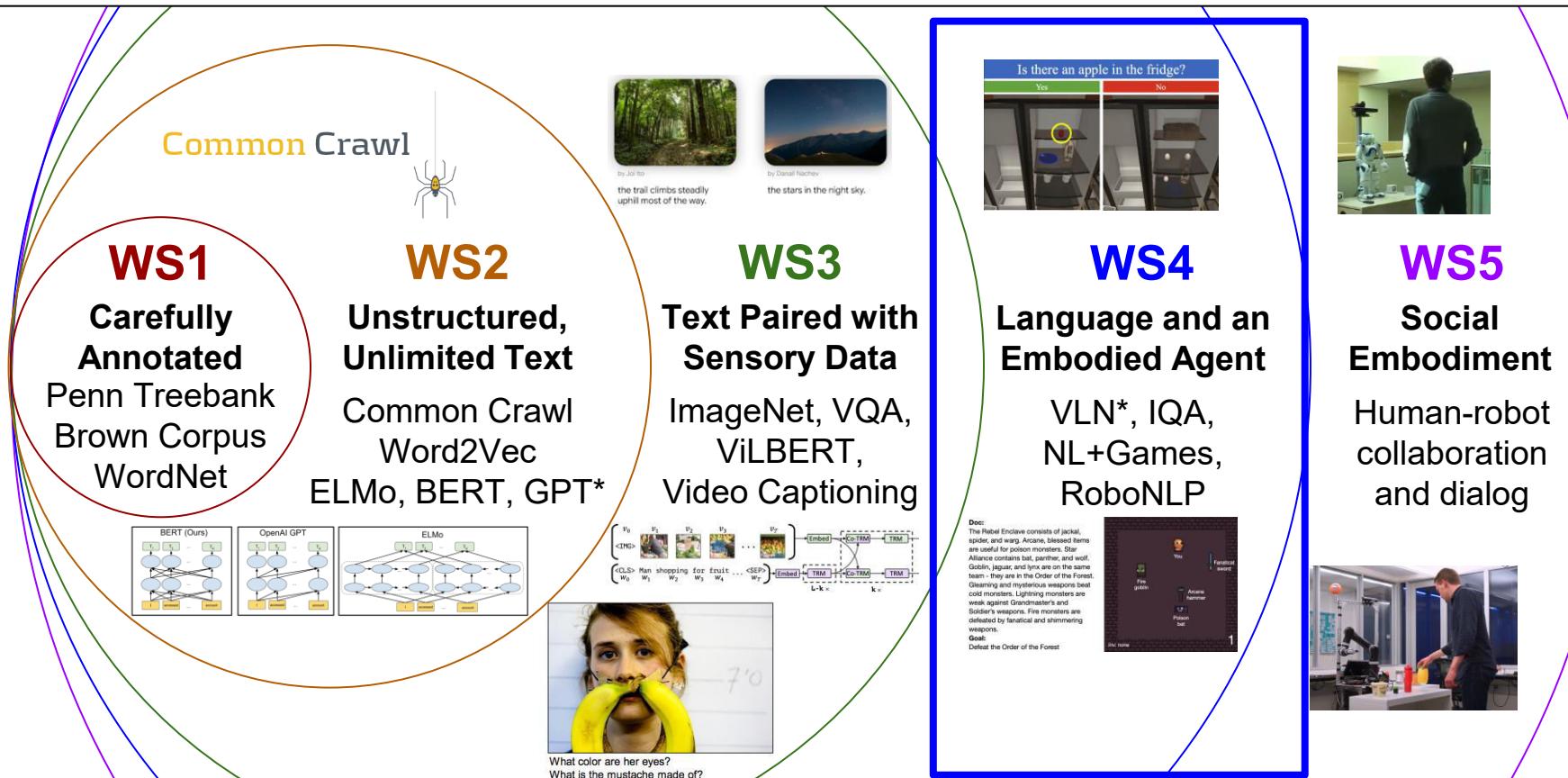
# Speech Recognition with Visual Cues



“Wash that dirty pug  
in the sink”



# Experience Grounds Language - World Scopes



# Embodied Question Answering



- New challenges?
- Shortcomings?

# Embodied Question Answering + Objects

Question and answer

Q: Is there bread in  
the room?  
A: No

Initial Image



Scene View



Q: How many mugs  
are in the room?  
A: 3

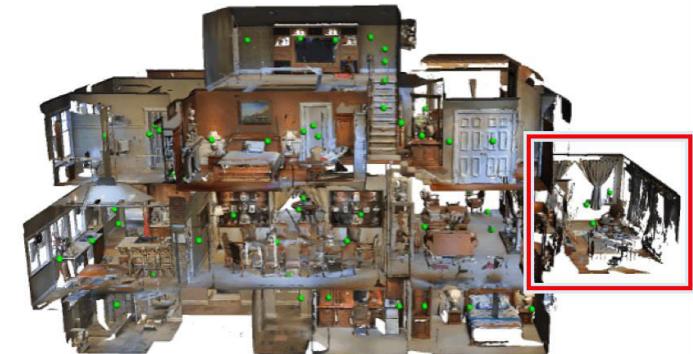


Q: Is there a tomato  
in the fridge?  
A: Yes



- New challenges?
- Shortcomings?

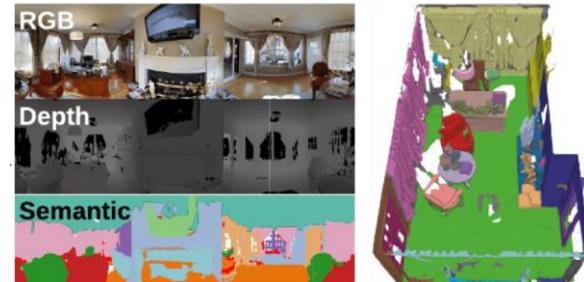
# Panoramic Cameras and Reconstructed 3D Scenes



Textured 3D Mesh



Panoramas

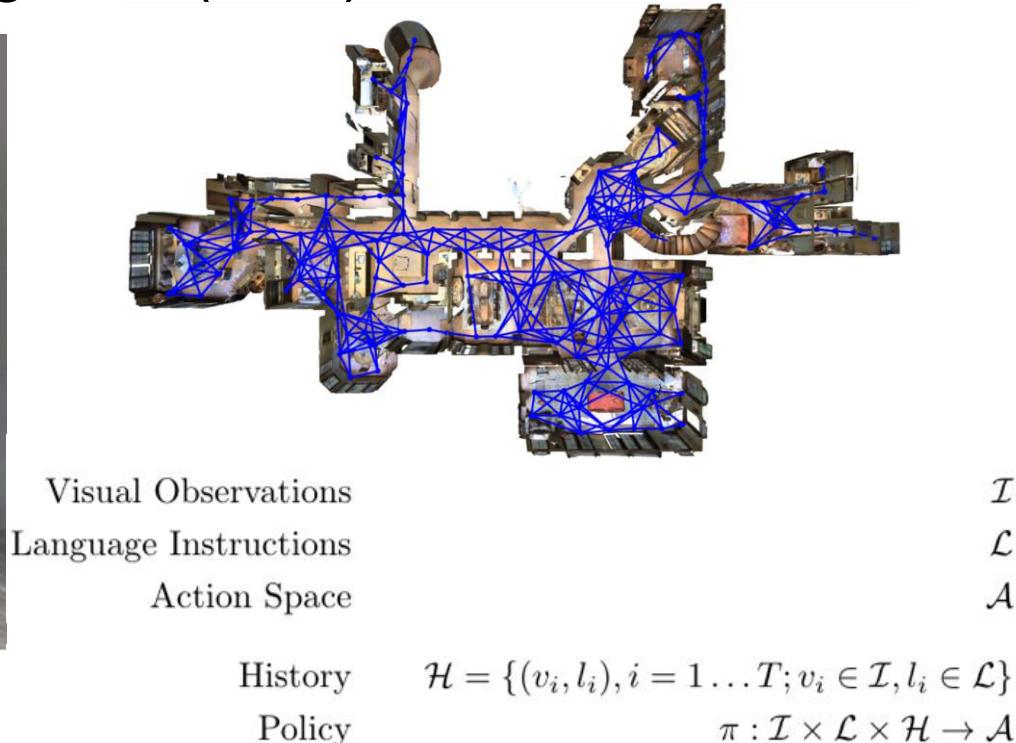


Object Instances

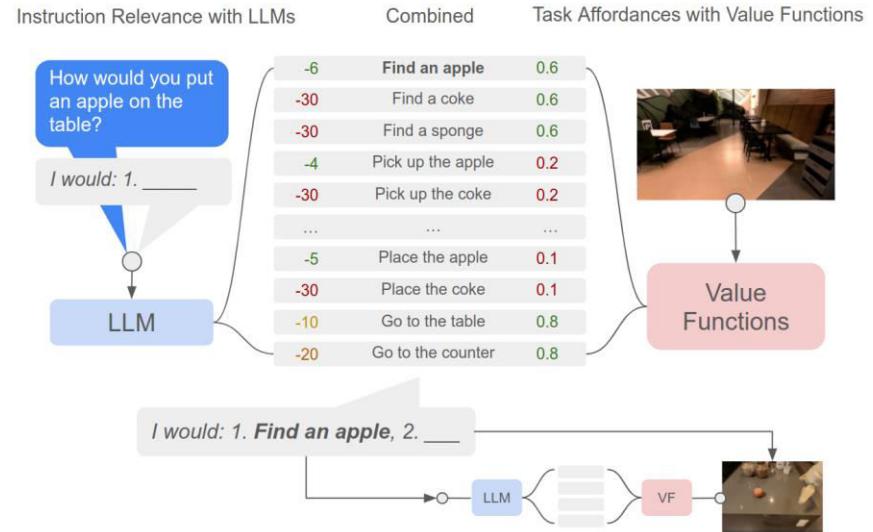
# Vision-and-Language Navigation (VLN)



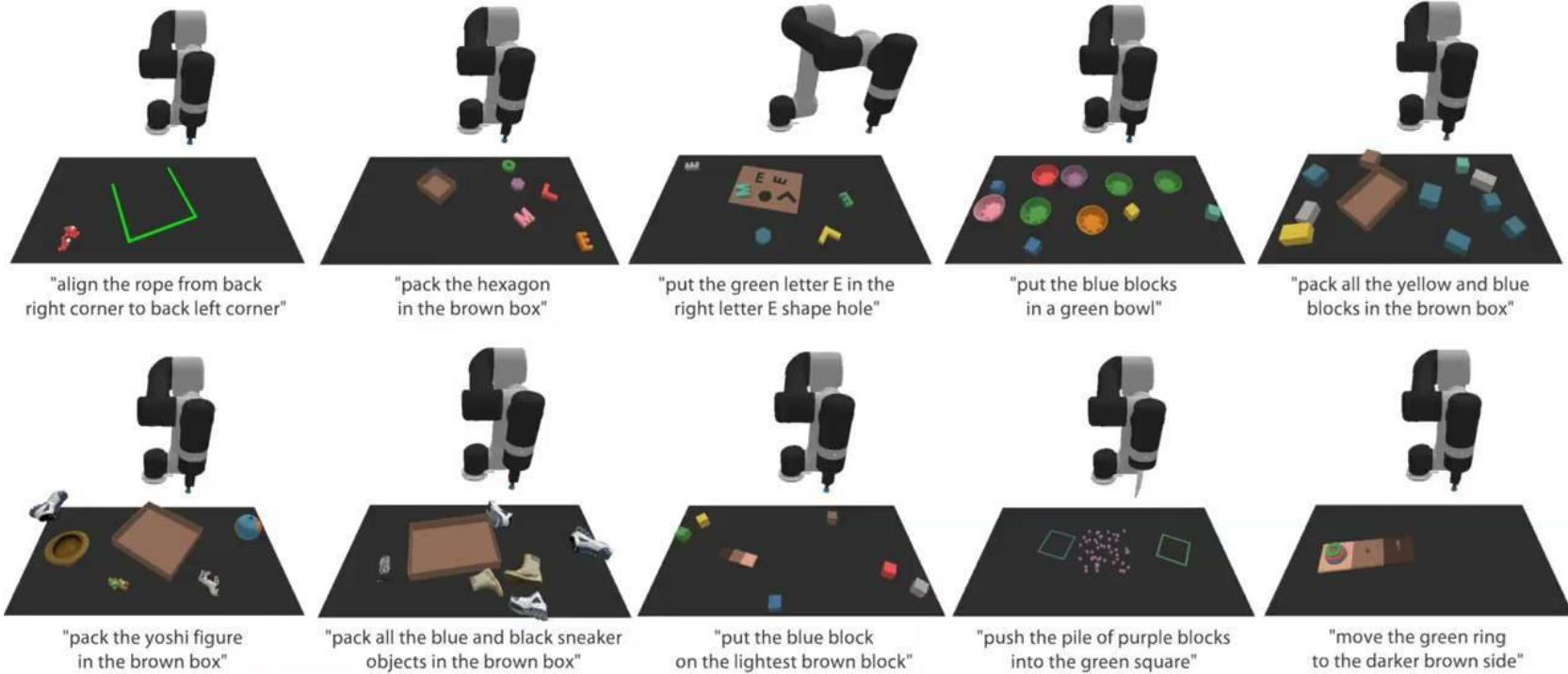
Leave the bedroom, and enter the kitchen. Walk forward, and take a left at the couch. Stop in front of the window.



# Simple Language for Pick and Place on Robots



# Simple Language for Pick and Place on Robots



# LLMs as High Level Robot Planners

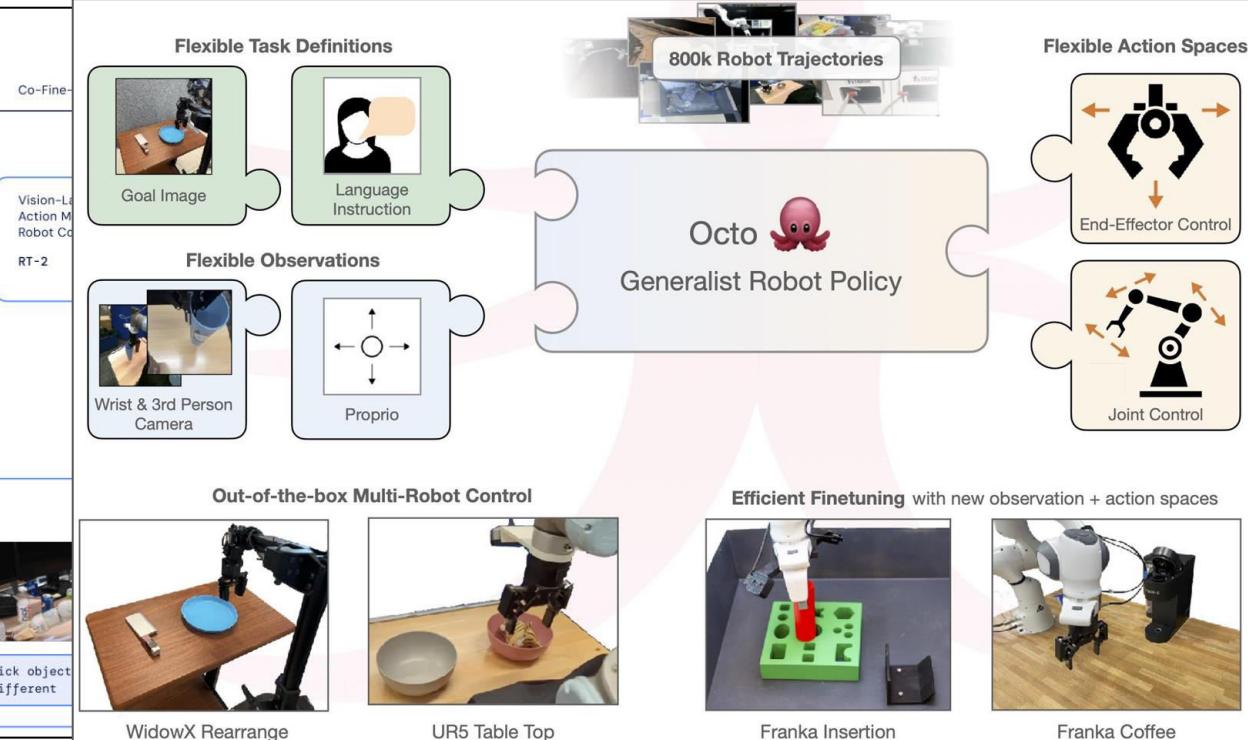
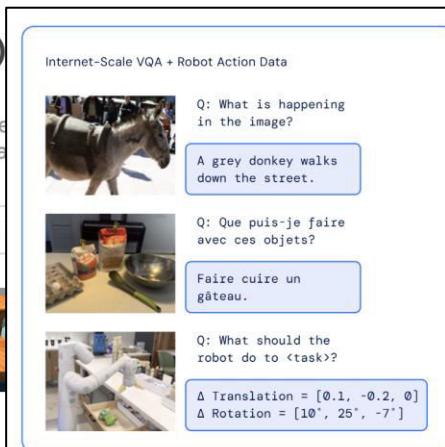


# A Brief History of Modern Vision-Language-Action Models

1)

teleope  
pla

all



[Language Conditioned Imitation Learning over Unstructured Data; Google](#); May 18, 2020

RT-2: New model translates vision and language into action; [DeepMind](#); July 28, 2023

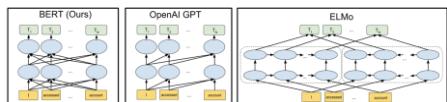
Octo: An Open-Source Generalist Robot Policy; December 14th, 2023

# Experience Grounds Language - World Scopes

Common Crawl



**WS1**  
Carefully  
Annotated  
Penn Treebank  
Brown Corpus  
WordNet



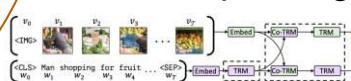
**WS2**

Unstructured,  
Unlimited Text  
Common Crawl  
Word2Vec  
ELMo, BERT, GPT\*



**WS3**

Text Paired with  
Sensory Data  
ImageNet, VQA,  
ViLBERT,  
Video Captioning



**WS4**

Language and an  
Embodied Agent  
VLN\*, IQA,  
NL+Games,  
RoboNLP



**WS5**  
Social  
Embodiment  
Human-robot collaboration  
and dialog



# Jointly Improving Parsing and Perception for Natural Language Commands through Human-Robot Dialog

Jesse Thomason, Aishwarya Padmakumar, Jivko Sinapov,  
Nick Walker, Yuqian Jiang, Harel Yedidsion, Justin Hart,  
Peter Stone, Ray Mooney

# Sensory Perception Beyond Vision

**Look**



**color, shape**, and deep **VGG** features extracted from images of the object.

**Pus**



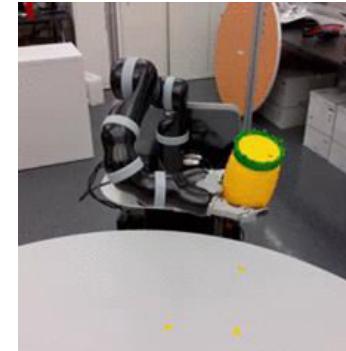
**Press**



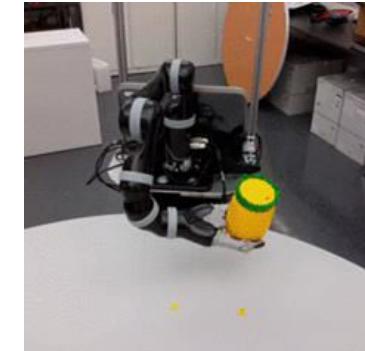
**Grasp**



**Lift / Lower**



**Drop**



**haptic** and **audio** features captured from arm and microphone.

# Blender Bot 2.0

Human	GPT3	Human	BlenderBot 2.0
 Tell me about Tom Brady.		 Tell me about Tom Brady.	
	 Tom Brady is the quarterback of the New England Patriots. He won five Super Bowls and four Super Bowl MVPs. He was also named NFL MVP in 2007, 2010, 2011 and 2017.		 search: Tom Brady He is a quarterback for the Tampa Bay Buccaneers. He is one of the greatest quarterbacks of all time.

- Dialogue model with LLM backend that employs memory writes to avoid forgetting what the partner (and itself) have said

# WS5\* Is Sort of Most Common Now through What Tech?

Hey, I'm trying to come up with some interesting, fantastical ways of decorating a living room for a design project.

Sure, there are many fantastical ways you could go about decorating a living room. Here are a few ideas to get you started:

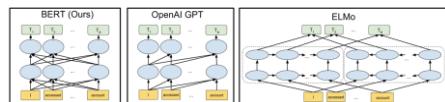
1. Create a fairy tale-inspired living room by using pastel colors and adding elements like a magical castle mural on one wall, a chandelier made of branches and twinkling lights, and comfortable furniture with curved, whimsical shapes.
2. Transform the living room into an enchanted forest by hanging sheer, green fabric from the ceiling to create a canopy effect, adding potted plants and trees, and using earthy tones and natural materials for the furniture and decor.
3. Make the living room into a futuristic space by using sleek, modern furniture and lighting, and adding metallic accents and pops of vibrant color. You could also incorporate unique, space-age elements like a holographic fireplace or a ceiling mural depicting a starry sky.

# Experience Grounds Language - World Scopes

Common Crawl



**WS1**  
Carefully  
Annotated  
Penn Treebank  
Brown Corpus  
WordNet



**WS2**

Unstructured,  
Unlimited Text  
Common Crawl  
Word2Vec  
ELMo, BERT, GPT\*



**WS3**

Text Paired with  
Sensory Data  
ImageNet, VQA,  
ViLBERT,  
Video Captioning



**WS4**

Language and an  
Embodied Agent  
VLN\*, IQA,  
NL+Games,  
RoboNLP

Doc:  
The Rebel Enclave consists of jackal, spider, air warg, Arcane, bladed items are available for purchase. The Order Alliance contains bat, panther, and wolf, Goblin, jaguar, and lion are the same team. The Order of the Forest consists of Glimmering and mysterious weapons beat cold monsters. Lightning monsters are weak against Grandmaster's and lightning bolts. Lightning monsters are defeated by fanatical and shimmering weapons.  
Goal:  
Defeat the Order of the Forest



**WS5**  
Social  
Embodiment  
Human-robot collaboration  
and dialog



# Guest Lecture: Abrar Anwar

## Language-guided robots (WS4)

# Guest Lecture: Wang Zhu

## Efficient End-to-End Visual Document Understanding with Rationale Distillation (WS3)

# BTW: LLMs are Terrible and Deployed Anyway

A screenshot of a search results page from a web browser. The search bar at the top contains the query "scientific notation e". Below the search bar are navigation links: All, Images, Videos, Shopping, Forums, Web, News, More, and Tools. A row of buttons includes Calculator, Converter, Examples, To decimal, 1, 750, and Capital. The main content area starts with an "AI Overview" section. It explains that the letter "E" or "e" is used in scientific notation to represent "times ten raised to the power of". It's a shortcut way to enter very large or small numbers without using logarithms. Below this, there is a table showing examples of scientific notation and their meanings:

Notation	Meaning
5E0	$5 \times 10^0$
7E2	$7 \times 10^2$
1E6	$1 \times 10^6$
4.212E-4	$4.212 \times 10^{-4}$
-5E9	$-5 \times 10^9$

The text at the bottom states: "The "E" is used because it's more convenient to type or display than superscripted exponents like  $10^7$ ".

# Physical Reasoning and Affordances

8:1  
Search...  
Previous  
Next  
Here are some sliding challenges:  
• Mix a smoothie.  
• Spread peanut butter on bread.  
• Use a rolling pin to flatten a week-old pizza.  
• Drain excess water from a moist heavy-duty paper towel.  
• Bake the pizza on a rack near the bottom of the oven to crisp the cheese on the bottom of the crust.

## Problem Solving

[Sol2] put seasoned beef, cheese, and lettuce **onto** the hard shell. ✓

egg whites from the yolk bottle, you should...

- a. Suck the egg whites from the yolk.
- b. Place the water bottle and press it against the yolk. **Keep pushing**, which creates suction and lifts the yolk.



## Premise sentence

A man went to grab a quick breakfast before leaving, but saw that the only remaining banana was rotten.

## Hypothesis 1

The banana is **yellow**.

## Hypothesis 2

The banana is **brown**.

## Multi-hop

The banana is **the color of an egg yolk**.

The banana is **the color of a tree log**.



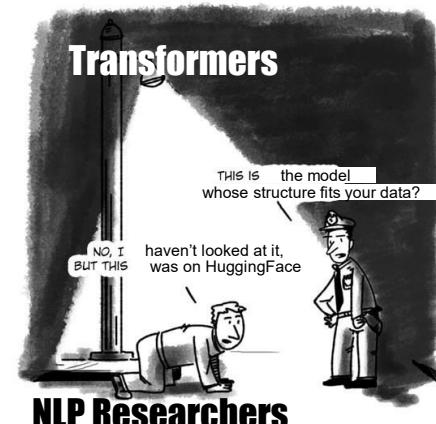
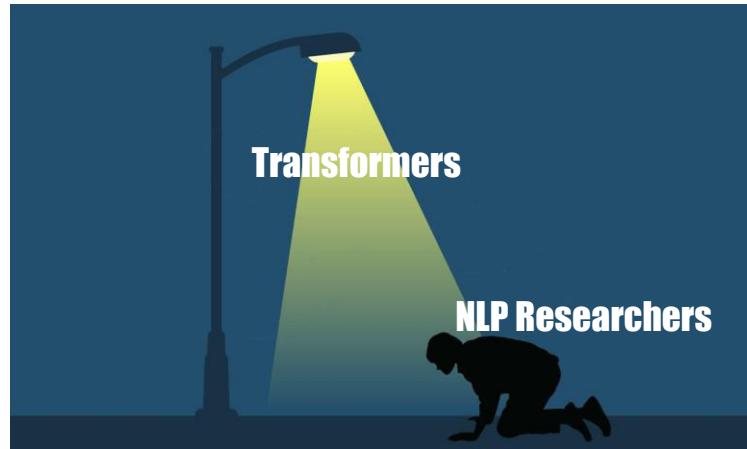
e a hard shelled taco,  
soned beef, cheese, and lettuce **onto** the hard X

# Physical Reasoning and Affordances

- There are levels of action, behavior, and understanding that are probably sublingual.
- NLP ate the Internet and will still output a plan for a robot to pick up three things and turn on the stove in sequence

# Streetlight Effect

A policeman sees a drunk man searching for something under a streetlight and asks what the drunk has lost. He says he lost his keys and they both look under the streetlight together. After a few minutes the policeman asks if he is sure he lost them here, and the drunk replies, no, and that he lost them in the park. The policeman asks why he is searching here, and the drunk replies, "this is where the light is".



## Parting Thoughts—You are the future, etc.

- The NLP community recently discovered the word “agent” and doesn’t seem to know what to do with it
  - Apply your core AI knowledge and new reinforcement learning toolbox to think about “agentic LLM” problems through the lens of MDPs, search, and planning
- As NLNP tech moves more towards human interaction, turn-based dialogue will have to fall by the wayside
  - Start identifying and solving problems that enable *continuous* versus *continual* interaction with AI systems

# Moving from Continual to Continuous Interaction

- LLMs are either fully autoregressive or instruction-tuned to perform *turn-based* interaction
- Interacting with people in the world is continuous!

User  
please get me an english word with the most number of h's.

Model 1.4s  
The English word with the most "h's is "hhhhhhh".

It's a bit of a cheat, as it's not a real word with a definition. It's just a sequence of letters, but technically it fulfills the criteria of having the most "h's".

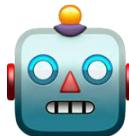
If you're looking for a real word, then the winner would be "huhh", which is a variant spelling of "huh."

User  
it has to be a real word.



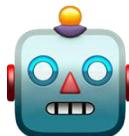
# Moving from Continual to Continuous Interaction

- Text-to-speech isn't enough; we need to be *reactive* to human speech and gesture, and the environment even while speaking
- In other words, next token generation should be *closed loop*!

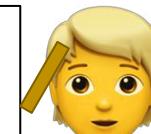
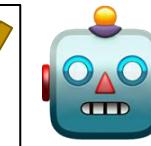
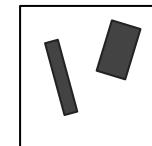
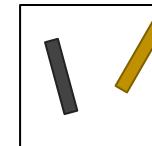


Here is the left front  
table leg

👎 Screw that one in,  
actually.



Got it.



Can you hand me the  
black leg?



skinnier leg?

# CSCI 444: Natural Language Processing

- When I took my first computer science class in 2008, my instructor ended the course by saying:
  - “I have taught you enough to make you dangerous”
- That attitude has been common in the tech industry for decades, and is evidenced in mottos like:
  - “move fast and break things” - Mark Zuckerberg
- The NLP toolbox is not a toy chest anymore, it’s an arsenal.
- We have taught you enough to make you dangerous.
- I hope we have also taught you enough to make you *cautious*.

# CSCI 444: Natural Language Processing

Jesse Thomason

Module 4: Lecture 4  
Language and Vision and Actions and