Applications: NLP

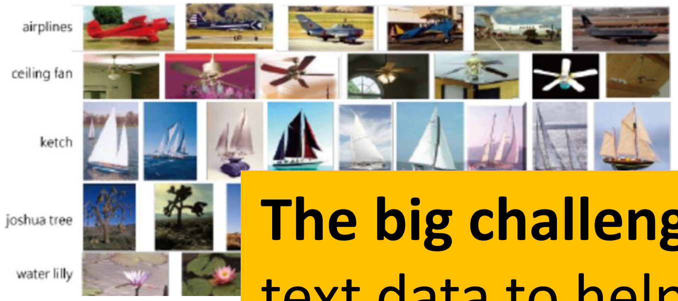
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley

The Big Idea: Unsupervised Pretraining

Deep learning works best when we have a lot of data



The big challenge: how can we use **freely available** and **unlabeled** text data to help us apply deep learning methods to NLP?

Good news: there is plenty of data of text out there!

Bad news: most of it is unlabeled

1,000s of times more data **without** labels (i.e., valid English text in books, news, web) vs. labeled/paired data (e.g., English/French translations)

What can we do with unlabeled data?

learned **representation** of language

[English] → learned **representation** of language → [Spanish]

How can we represent these… representations?

local non-contextual representations

global context-dependent representations

word embeddings **sentence embeddings** **pretrained language models**

Start simple: how do we represent **words?**

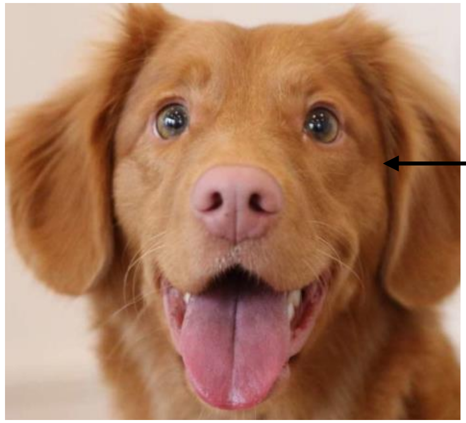
$$x_{1,1} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ 0 \\ 1 \\ 0 \\ \cdot \\ 0 \end{bmatrix}$$

dimensionality = number of possible words

index of this word

This means basically nothing by itself

not great, not terrible…



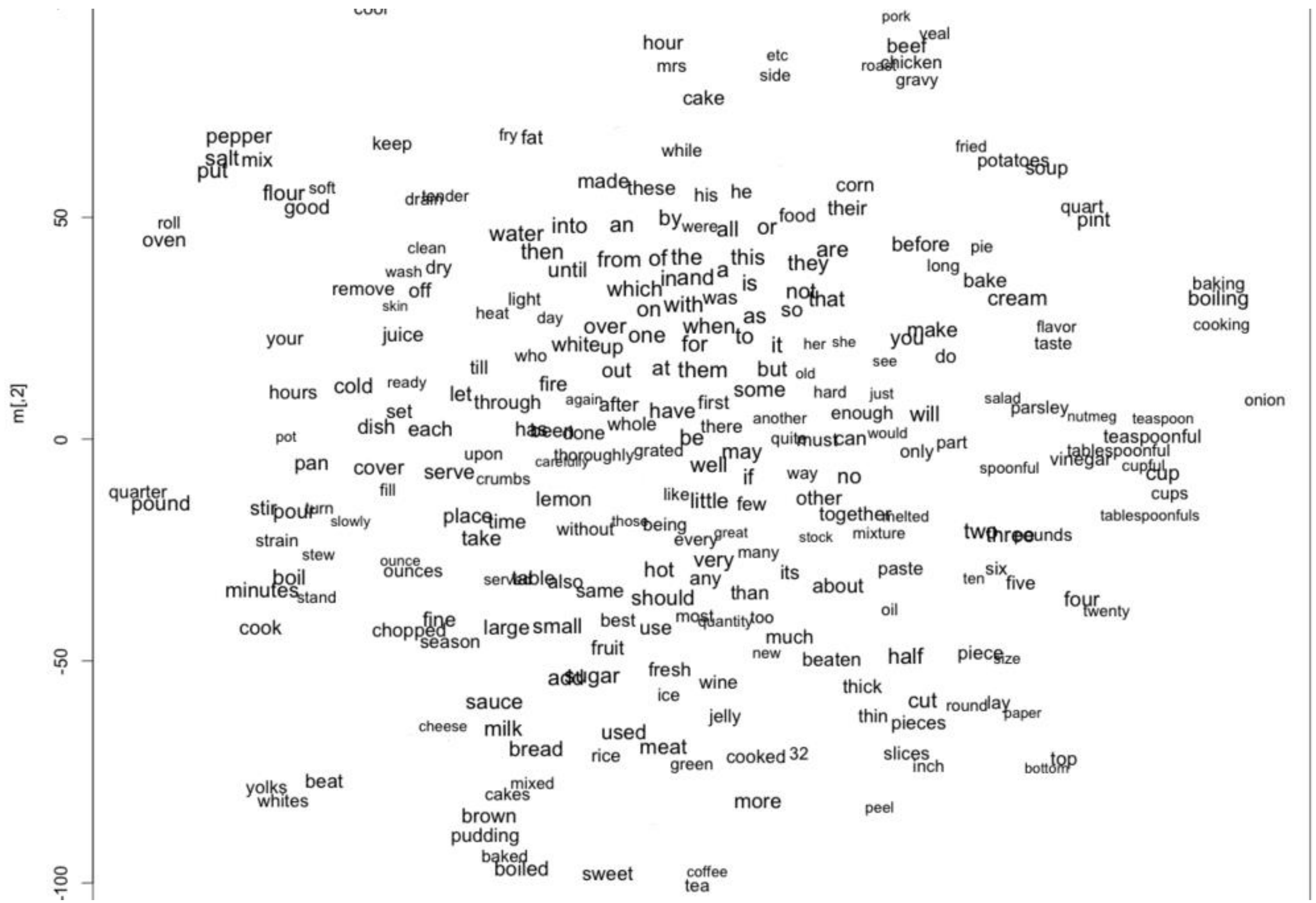
The pixels mean something!

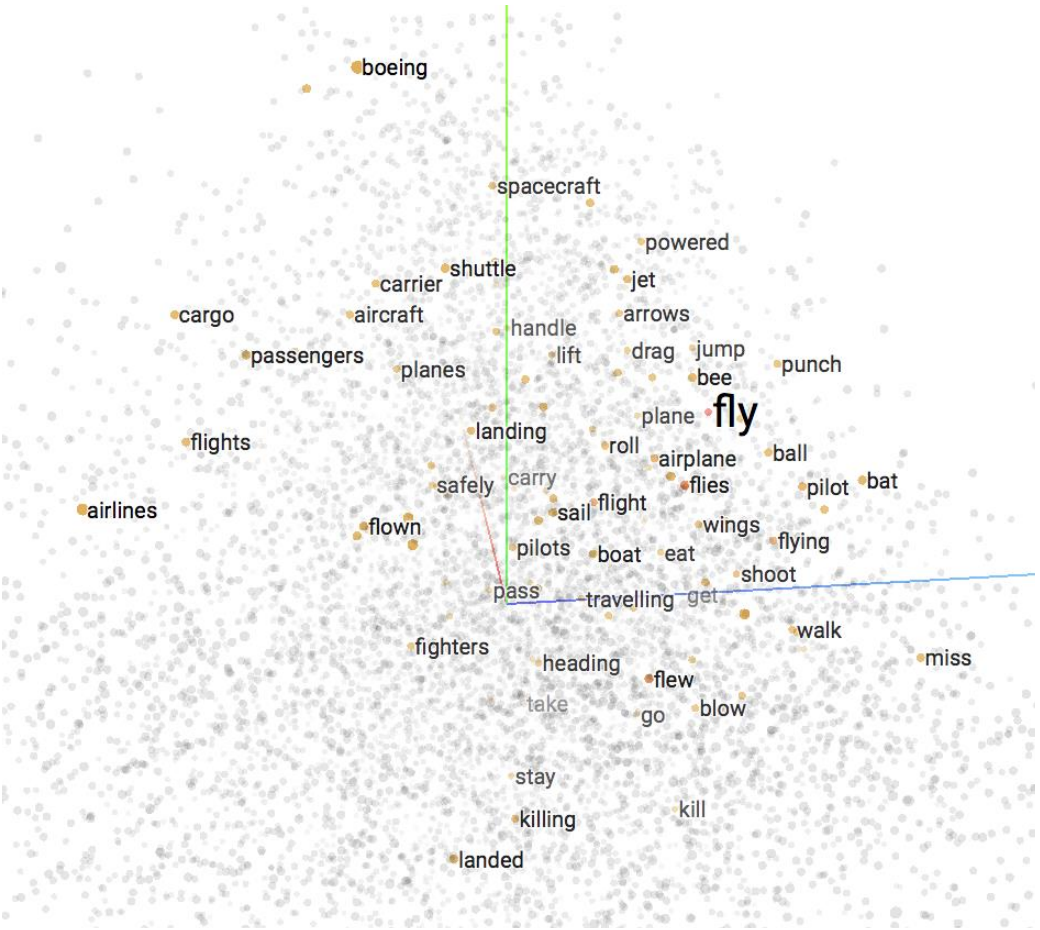
Not much (not a great metric space), but they mean something

Maybe if we had a more meaningful representation of words, then learning downstream tasks would be much easier!

Meaningful = vectors corresponding to **similar** words should be close together

Some examples of **good** embeddings



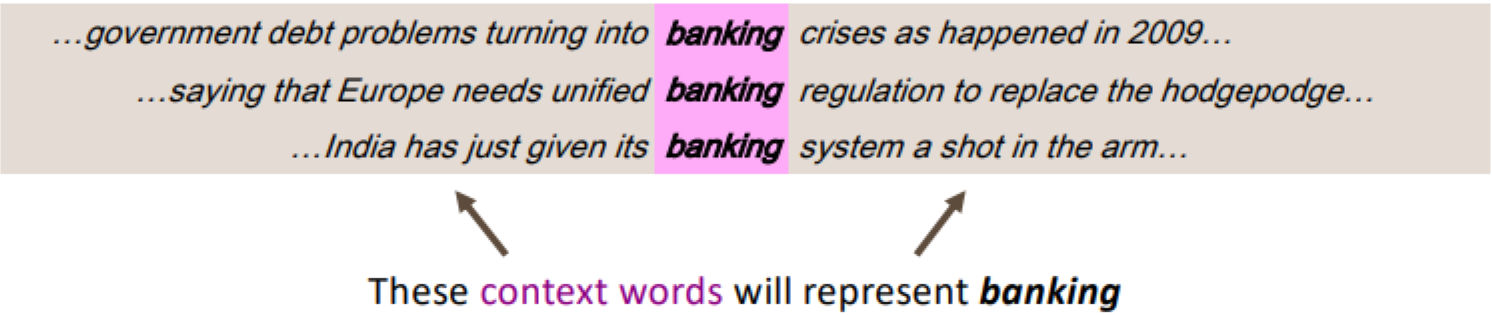


How do we learn embeddings?

Basic idea: the meaning of a word is determined by what **other** words occur in close proximity to it

Put another way: the more interchangeable words are, the more similar they are

Example: Seattle hotel vs. Seattle motel

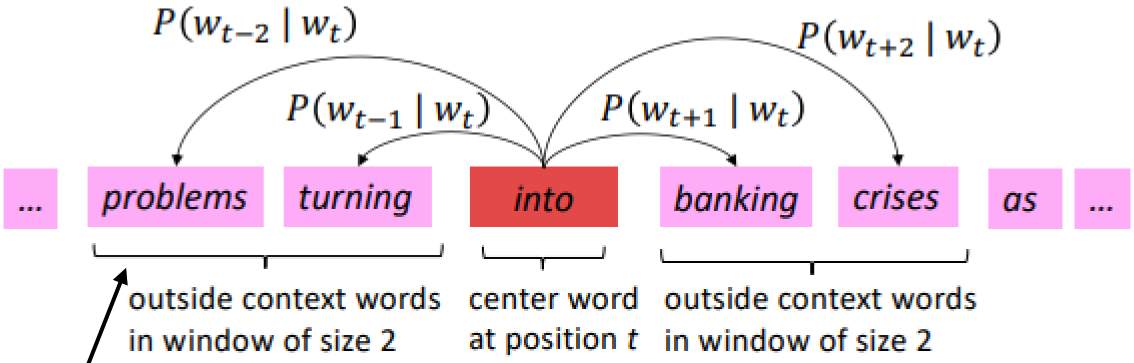


Basic principle: pick a representation for each word such that its neighbors are "close" under this representation

Examples & Images: Christopher Manning, CS224n

More formally…

Can we **predict** the neighbors of a word from its **embedding value**?



"context word" o

"center word" c

(learned) vector representation of o

(learned) vector representation of c

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

all possible words (vocabulary)

looks a bit like a **logistic regression** model

how to train? $$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c)$$

u and v vectors for all possible words

all possible c and o combinations

e.g., for each word c, pick all words o that are within 5 step

Examples & Images: Christopher Manning, CS224n

word2vec

$$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c) \qquad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$\theta = \begin{bmatrix} v_{\text{aardvark}} \\ v_{\text{a}} \\ \ldots \\ v_{\text{zebra}} \\ u_{\text{aardvark}} \\ u_{\text{a}} \\ \ldots \\ u_{\text{zebra}} \end{bmatrix}$$

- Why two vectors? Makes optimization easier
- What to do at the end? Average them
- This then gives us a representation of words!

Mikolov et al. “Linguistic Regularities in Continuous Space Word Representations.” 2013.

Making word2vec tractable

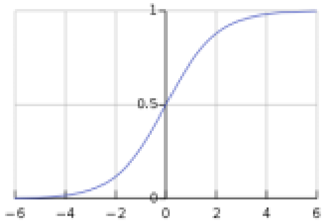
$$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c)$$

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Problem: the vocabulary might be **huge**

denominator might be **really costly** to compute

Another idea: what if we instead have a **binary** classification problem ("is this the right word or not")?



$$p(o \text{ is the right word}|c) = \sigma(u_o^T v_c) = \frac{1}{1 + \exp(-u_o^T v_c)}$$

This is not enough! Why?

$$p(o \text{ is the wrong word}|c) = \sigma(-u_o^T v_c) = \frac{1}{1 + \exp(u_o^T v_c)}$$

$$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \left(\log p(o \text{ is right}|c) + \sum_{w} \log p(w \text{ is wrong}|c) \right)$$

randomly chosen "negatives"

Mikolov et al. "Linguistic Regularities in Continuous Space Word Representations." 2013.

Making word2vec tractable: summary

$$p(o \text{ is the right word}|c) = \sigma(u_o^T v_c) = \frac{1}{1 + \exp(-u_o^T v_c)}$$

$$p(o \text{ is the wrong word}|c) = \sigma(-u_o^T v_c) = \frac{1}{1 + \exp(u_o^T v_c)}$$

$$\arg \max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \left(\log p(o \text{ is right}|c) + \sum_w \log p(w \text{ is wrong}|c) \right)$$

$$\arg \max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \left(\log \sigma(u_o^T v_c) + \sum_w \log \sigma(-u_w^T v_c) \right)$$

Intuition: push v_c *toward* u_o and *away* from other vectors u_w

Mikolov et al. “Linguistic Regularities in Continuous Space Word Representations.” 2013.

word2vec examples

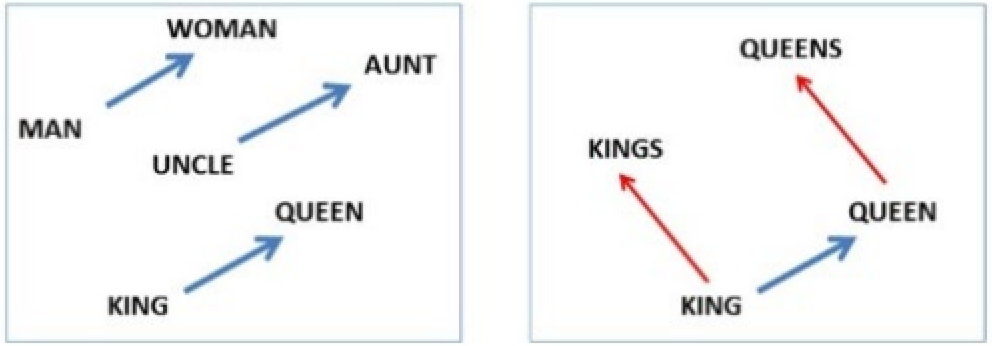
$$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c) \qquad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Algebraic relations:

vec(“woman")−vec(“man") ≃ vec(“aunt")−vec(“uncle")

vec(“woman")−vec(“man") ≃ vec(“queen")−vec(“king")

This is a little bit idealized, most relationships are not nearly this “nice”



Mikolov et al. “Linguistic Regularities in Continuous Space Word Representations.” 2013.

word2vec examples

$$\arg \max_{u_1,\dots,u_n,v_1,\dots,v_n} \sum_{c,o} \log p(o|c) \qquad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2vec model computed from 6 billion word corpus of news articles

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

Mikolov et al. “Linguistic Regularities in Continuous Space Word Representations.” 2013.

word2vec examples

$$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c) \qquad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Mikolov et al. “Linguistic Regularities in Continuous Space Word Representations.” 2013.

Word2vec summary

$$\arg\max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c) \qquad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V} \exp(u_w^T v_c)}$$

What do we do we with this?

- Use as word representation in place of one-hot vectors
- Much more meaningful for downstream applications
- Can train word embeddings on large unlabeled corpus, and then use it as input into a supervised model trained on a much smaller corpus
- Could think of it as a simple type of **unsupervised pretraining**

Mikolov et al. “Linguistic Regularities in Continuous Space Word Representations.” 2013.

Pretrained Language Models

Contextual representations

Word embeddings associate a vector with each word

This can make for a much **better** representation than just a one-hot vector!

However, the vector **does not** change if the word is used in different ways!

Let's play baseball

I saw a play yesterday

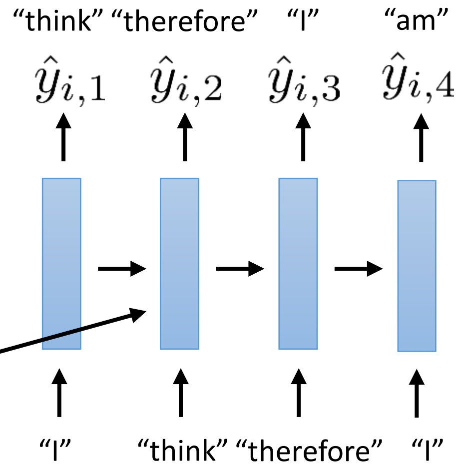
same word2vec representation, even though they mean different things!

Can we learn representations that **depend on context?**

High level idea:

1. Train a **language model**
2. Run it on a sentence
3. Use its **hidden state**

use the hidden state as the representation for downstream tasks



Question 1: how to train the best language model for this?

Question 2: how to use this language model for downstream tasks?

The age of Sesame Street characters

ELMo: bidirectional LSTM model used for context-dependent embeddings

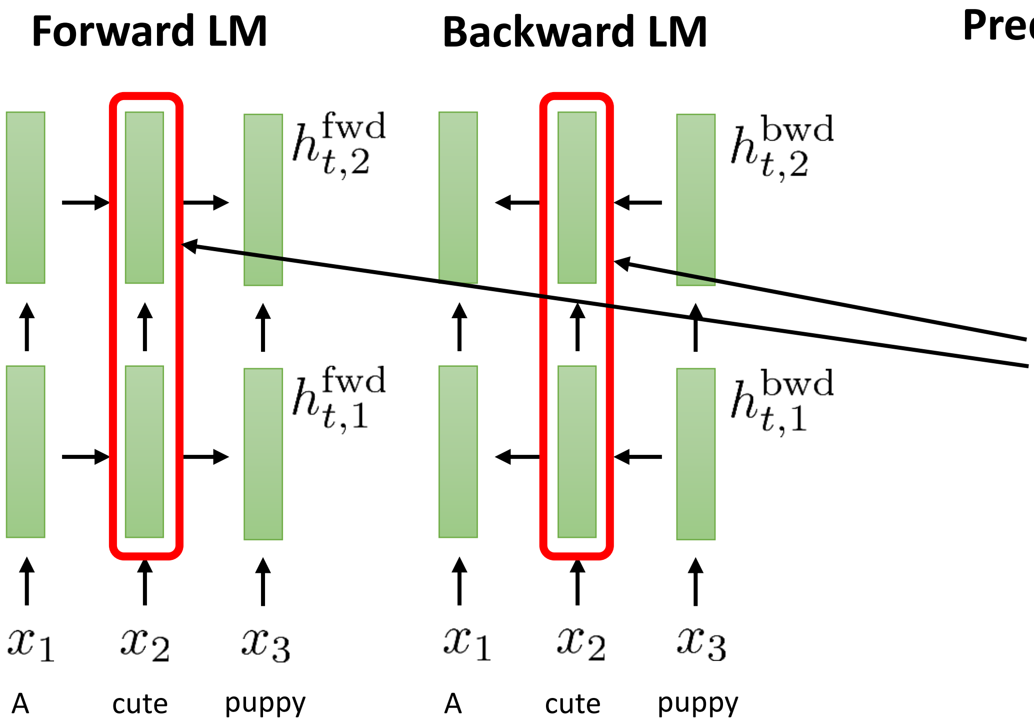
BERT: transformer language model used for context-dependent embeddings

Credit: Jay Alammar: http://jalammar.github.io/illustrated-bert/

ELMo

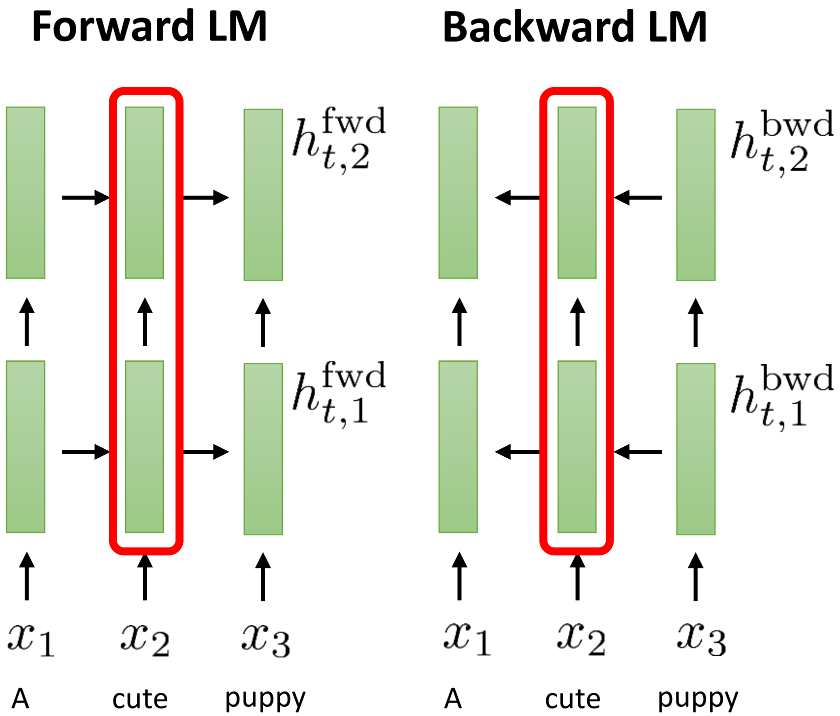
Both the forward and backward LM are trained as language models

Predict the next (or previous) word



together, all these hidden states form
a representation of the word “cute”

Using ELMo



top layer hidden states

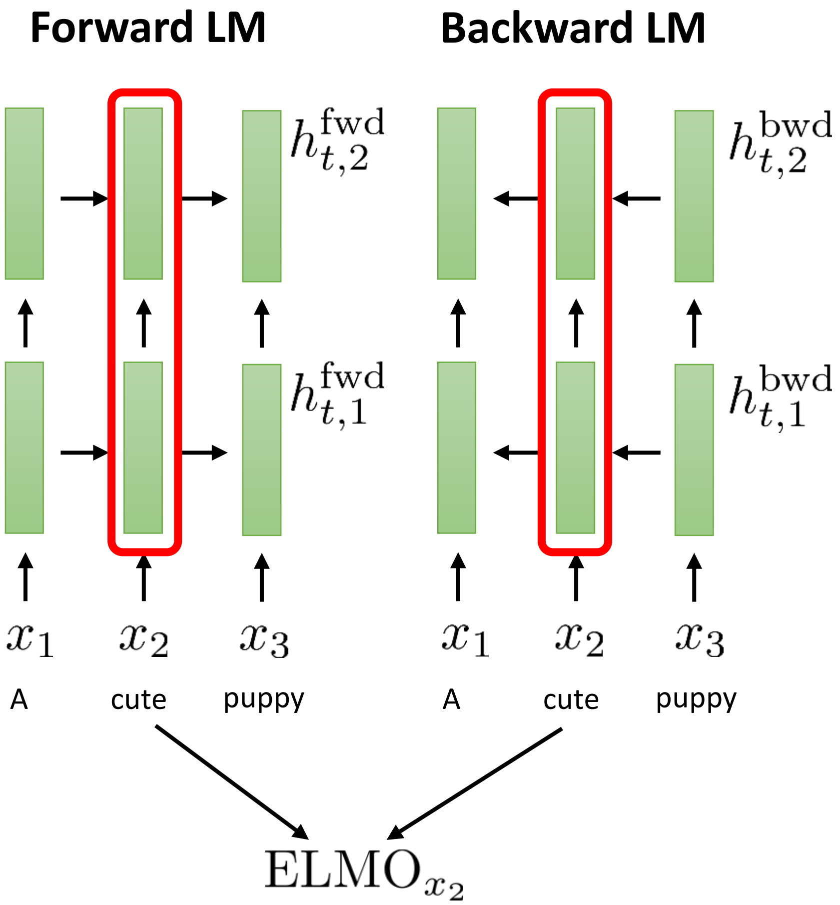
$$\text{simple version: } \mathrm{ELMO}_t = [h_{t,2}^{\text{fwd}}, h_{t,2}^{\text{bwd}}]$$

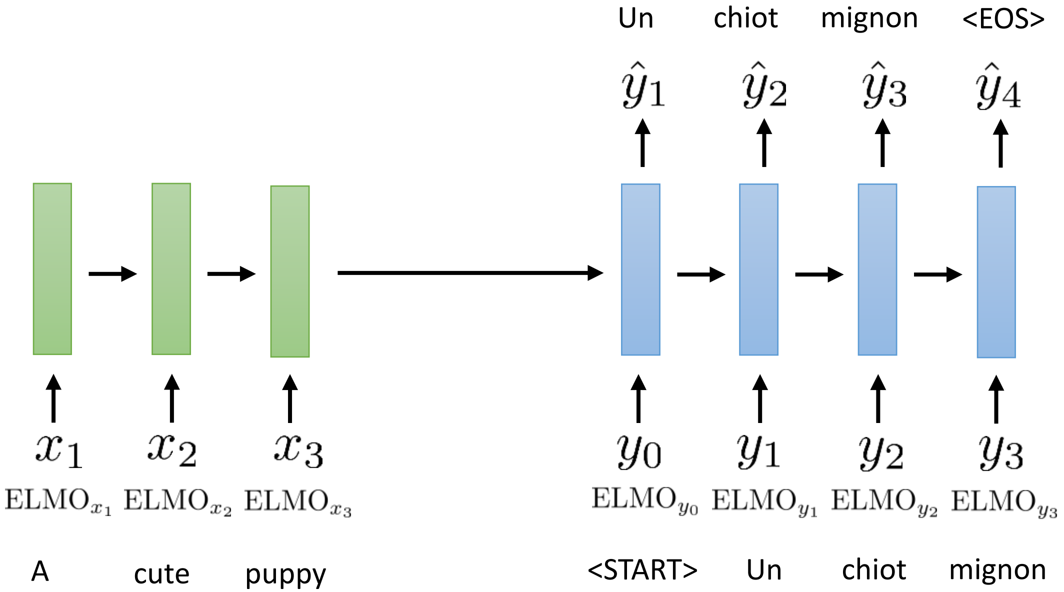
$$\text{complex version: } \mathrm{ELMO}_t = \gamma \sum_{i=1}^{L} w_i [h_{t,i}^{\text{fwd}}, h_{t,i}^{\text{bwd}}]$$

learned task-specific weights

learned as part of **downstream** task!

Using ELMo



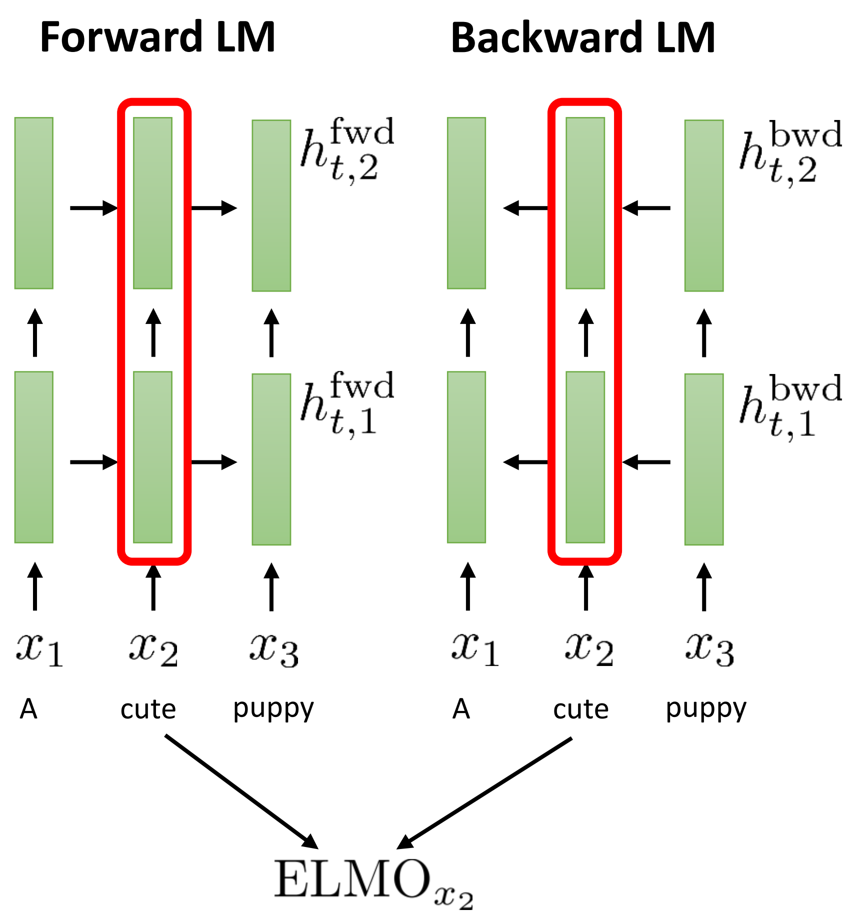


This is just an example, the actual ELMo paper **does not** test on translation, but does test:

- Question answering
- Textual entailment
- Semantic role labeling
- Coreference resolution
- Named entity extraction
- Sentiment analysis

And LMs help with all of these!

ELMo Summary



- Train **forward** and **backward** language models on a large corpus of **unlabeled** text data
- Use the (concatenated) forward and backward LSTM states to represent the word **in context**
- Concatenate the **ELMo representation** to the word embedding (or one-hot vector) as an **input** into a downstream task-specific sequence model
  - This provides a **context specific** and **semantically meaningful** representation of each token

BERT and Friends

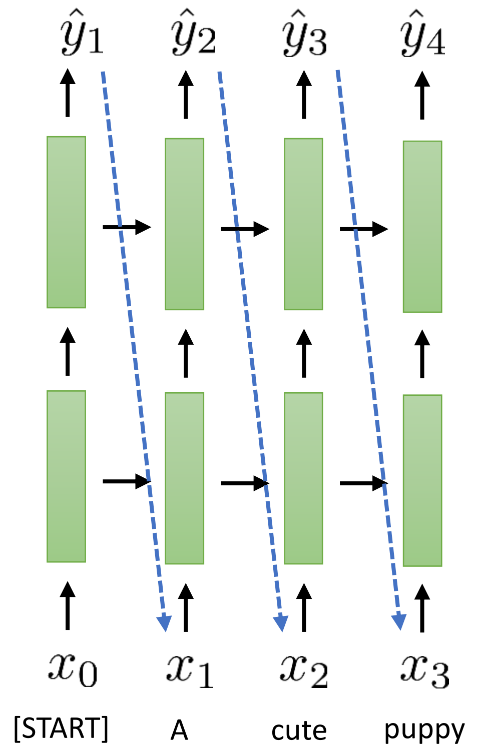
The age of Sesame Street characters

ELMo: bidirectional LSTM model used for context-dependent embeddings

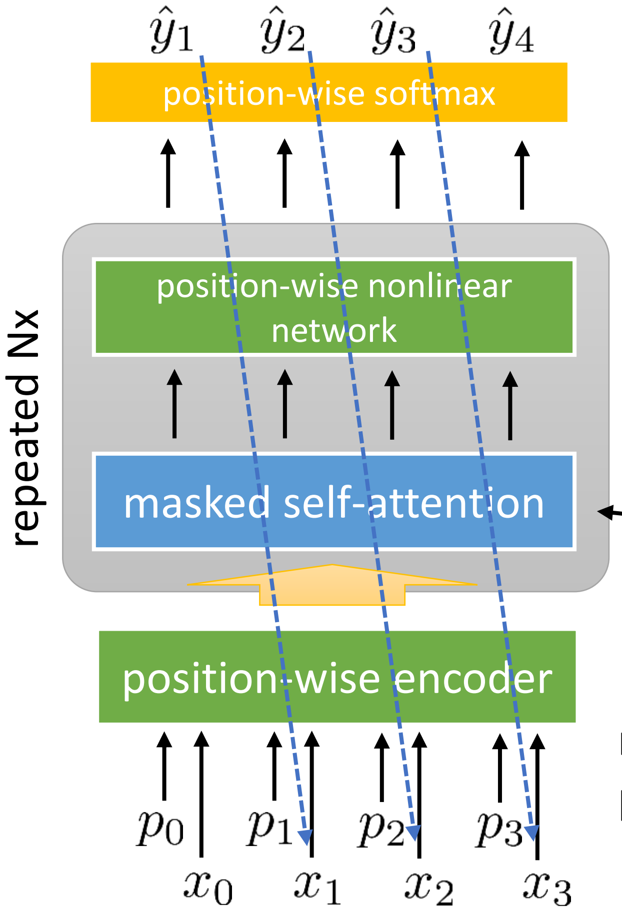
BERT: transformer language model used for context-dependent embeddings

Can we use a transformer instead?

Before:



Now:

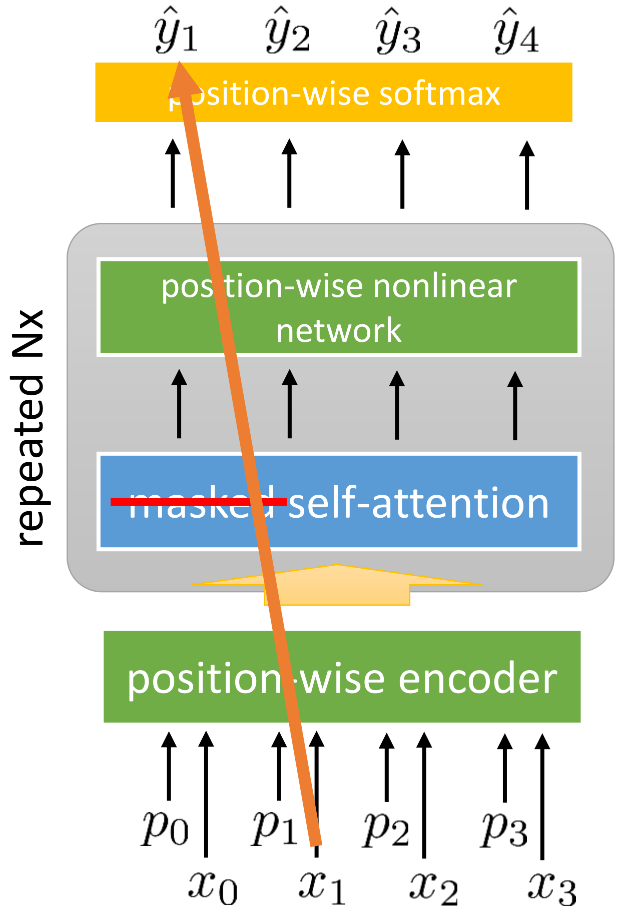


- This model has a **direction** (forward, depends on masking used in self-attention)
- ELMo is **bidirectional**, this isn't
- We could train **two transformers,** and make "transformer ELMo"
- But is there a better way? Can we simply remove the mask in self-attention and have **one** transformer?
  - What would go wrong?

(the decoder part of a transformer)

need masking to have a proper language model

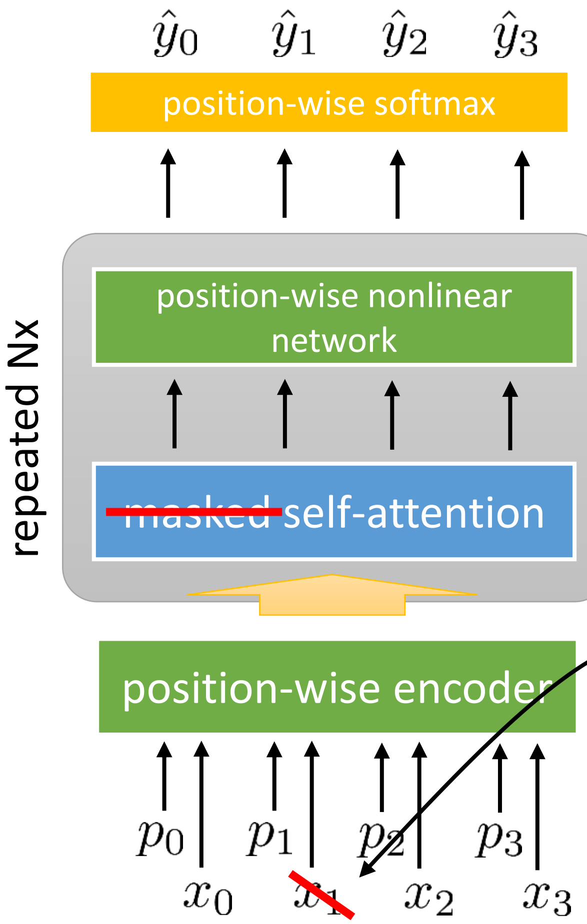
Bidirectional transformer LMs



It's trivially easy to get the right answer, since self-attention can access the "right answer" at time **t** from the input at time **t+1**!

Bidirectional transformer LMs

no need to shift things by one anymore (no masking)



Input: I [MASK] therefore I [MASK]

Output: I think therefore I am

Main idea: needing to predict missing words forces the model to "work hard" to learn a good representation

Without the need for masked self-attention!

This makes it **bidirectional**

BERT is essentially the "encoder" part of a transformer with 15% of inputs replaced with [MASK]

randomly **mask out** some input tokens

mask = replace with [MASK]

Training BERT

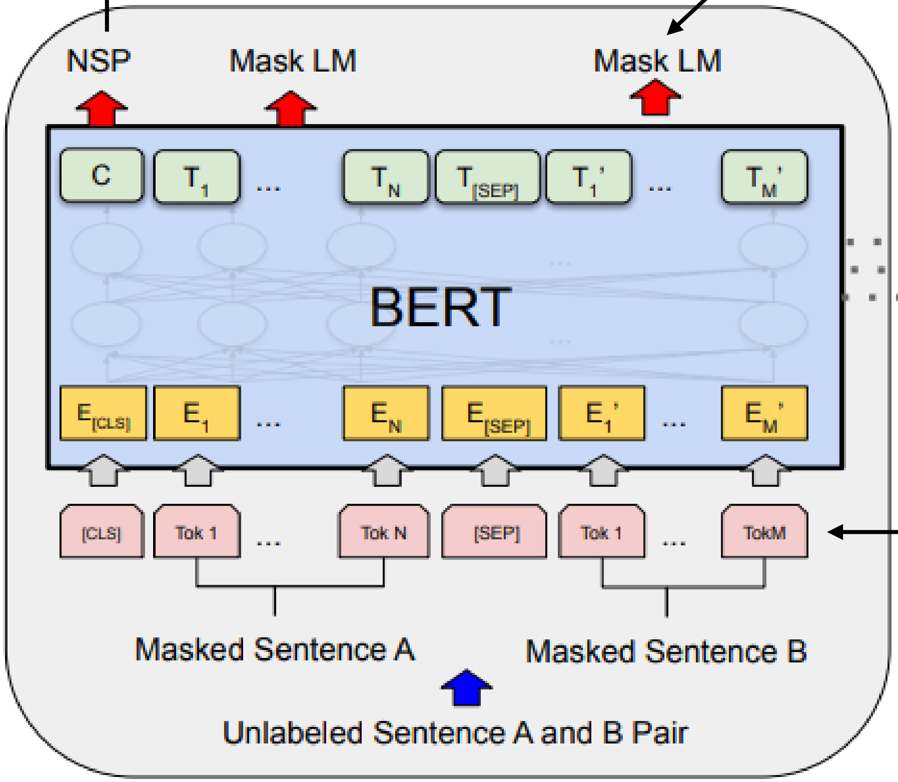
binary classifier output:

does **first sentence** follow the **second sentence,** or precede it?

reconstruct all tokens at each time step
(must predict actual token in place of [MASK])

forces learning context-dependent **word-level** representations

this forces learning **sentence-level** representations



pairs of sentences in the data are transformed in two ways:

1. Randomly replace 15% of the tokens with [MASK]
2. Randomly swap the order of the sentences 50% of the time

input consists of **two** sentences
why?

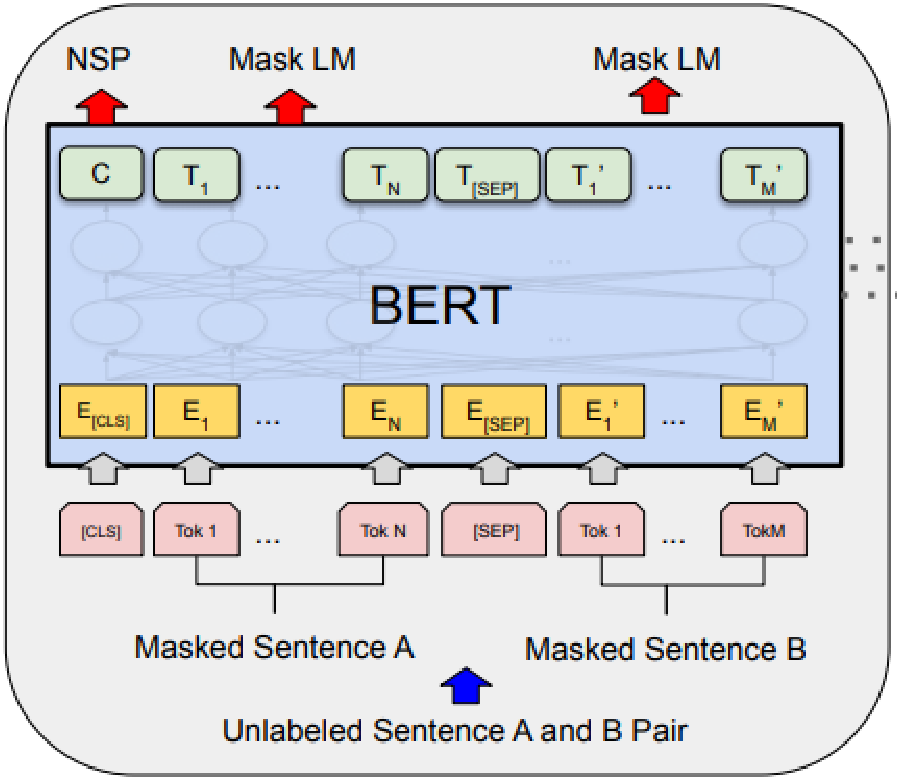
many downstream tasks require processing two sentences:

question answering

natural language inference

Using BERT

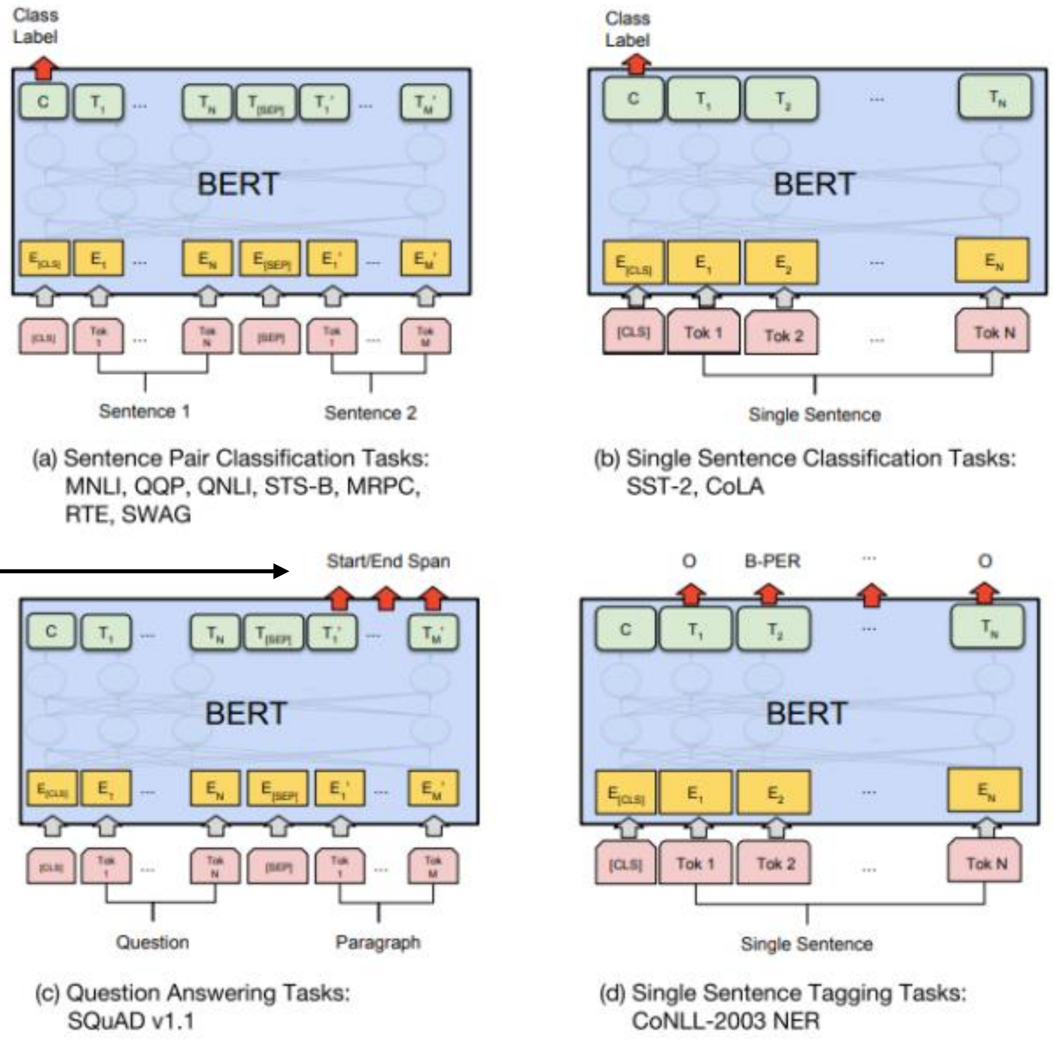
binary classifier output:	entailment classification
~~A before B vs. A after B~~	semantic equivalence (e.g., Quora question pair)
task classification output	sentiment classification



1. Put a cross-entropy loss on **only** the first output (replaces the sentence order classifier)

2. Finetune the **whole** model end-to-end on the new task

Using BERT



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

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

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

classification tasks

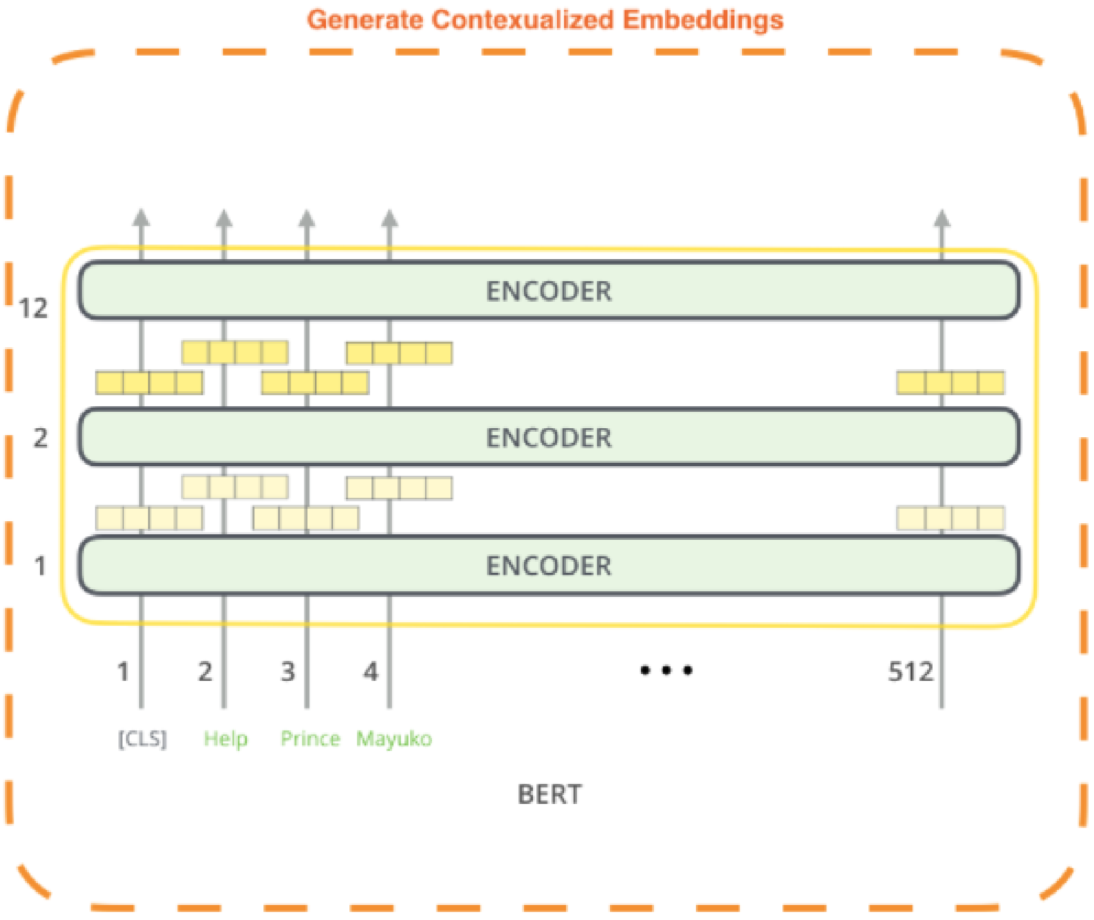
highlight which span of paragraph contains answer

finetune named entity label for **each** position

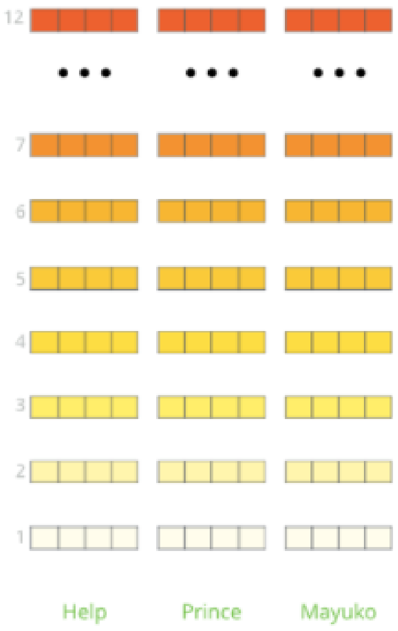
(person name, location, other categories)

Using BERT to get features

We can **also** pull out features, just like with ELMo!



The output of each encoder layer along each token's path can be used as a feature representing that token.

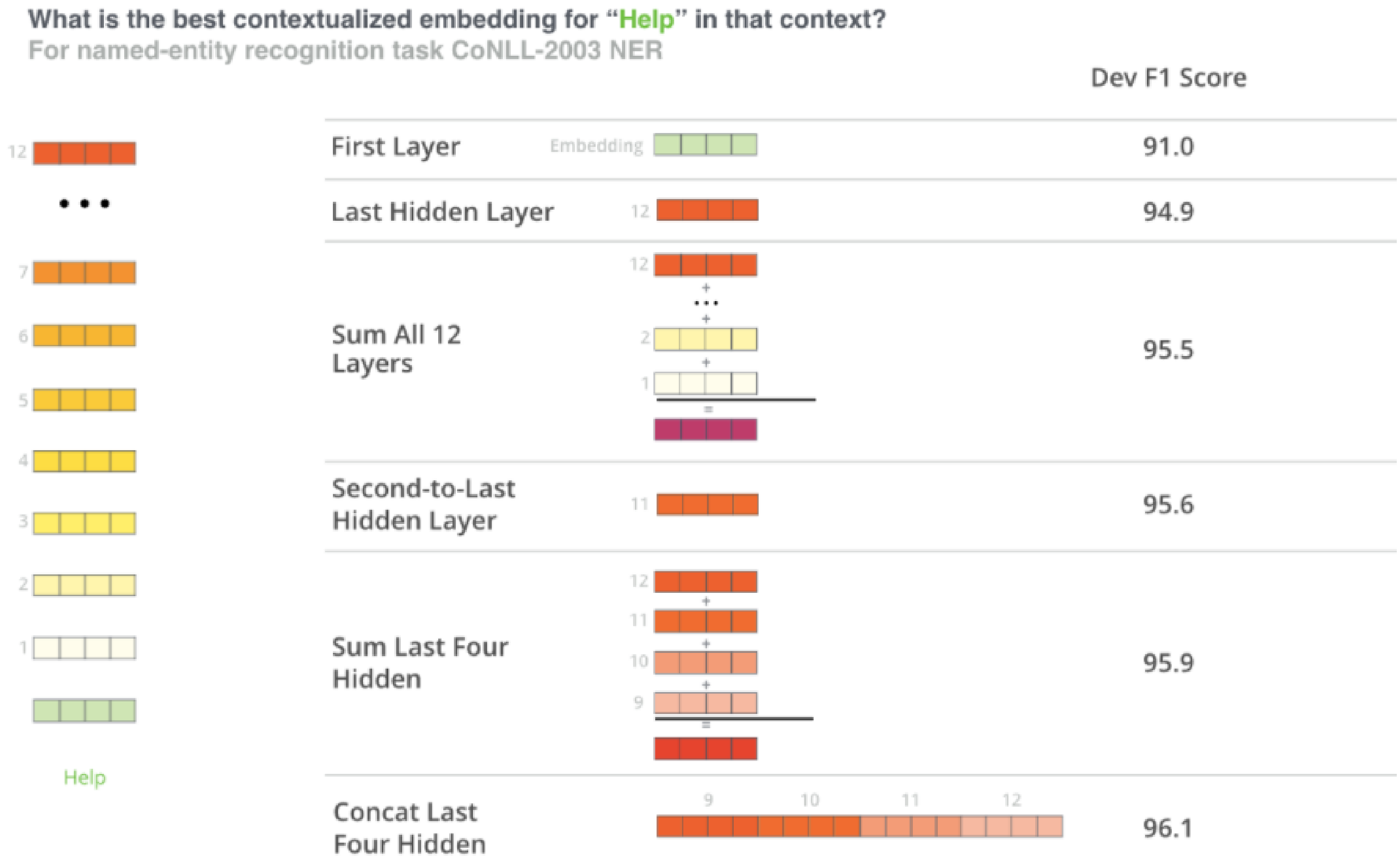


But which one should we use?

Credit: Jay Alammar: http://jalammar.github.io/illustrated-bert/

Using BERT to get features

We can **also** pull out features, just like with ELMo!



Credit: Jay Alammar: http://jalammar.github.io/illustrated-bert/

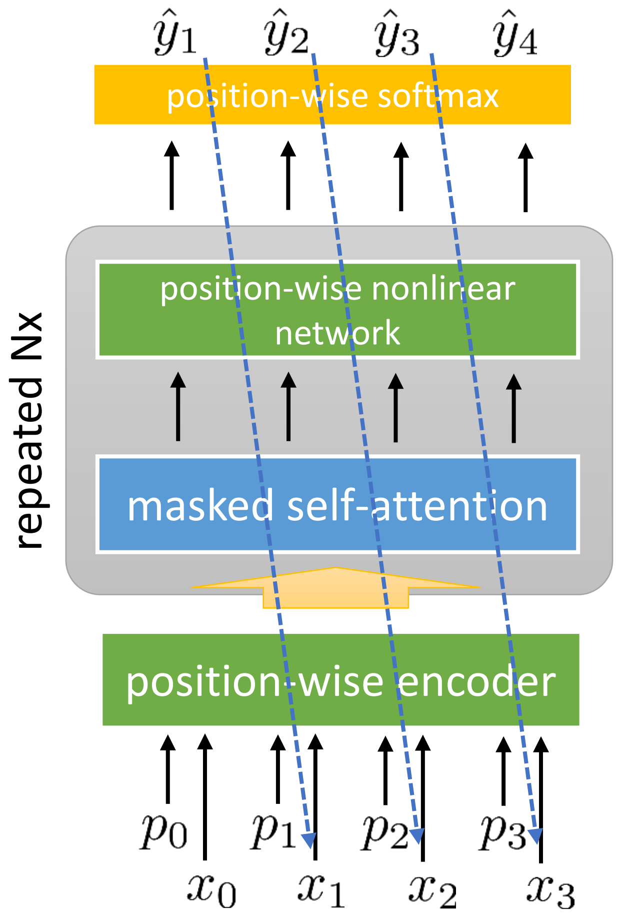
BERT results are extremely good

GLUE test result (battery of varied natural language understanding tasks)

	System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
		392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
	Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
	BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
	OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
12 layers	$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
24 layers	$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

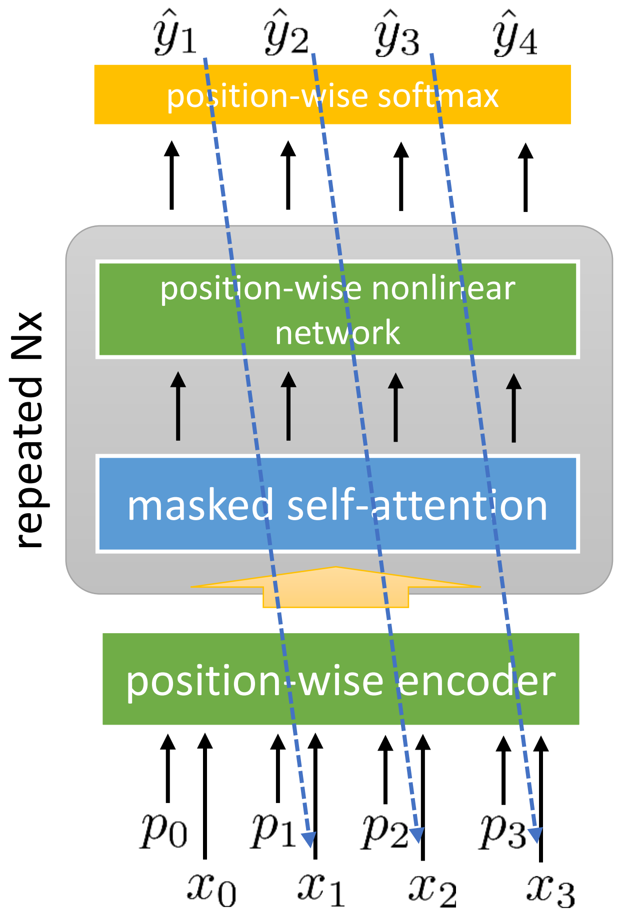
Since then, it has been applied to nearly every NLP task you can imagine, and often makes a **huge** difference in performance

GPT et al.



- One-directional (forward) transformer models do have one **big** advantage over BERT. Can you guess what it is?
- **Generation** is not really possible with BERT, but a forward (masked attention) model can do it!
- GPT (GPT-2, GPT-3, etc.) is a classic example of this

GPT et al.



OpenAI GPT-2 generated text

source

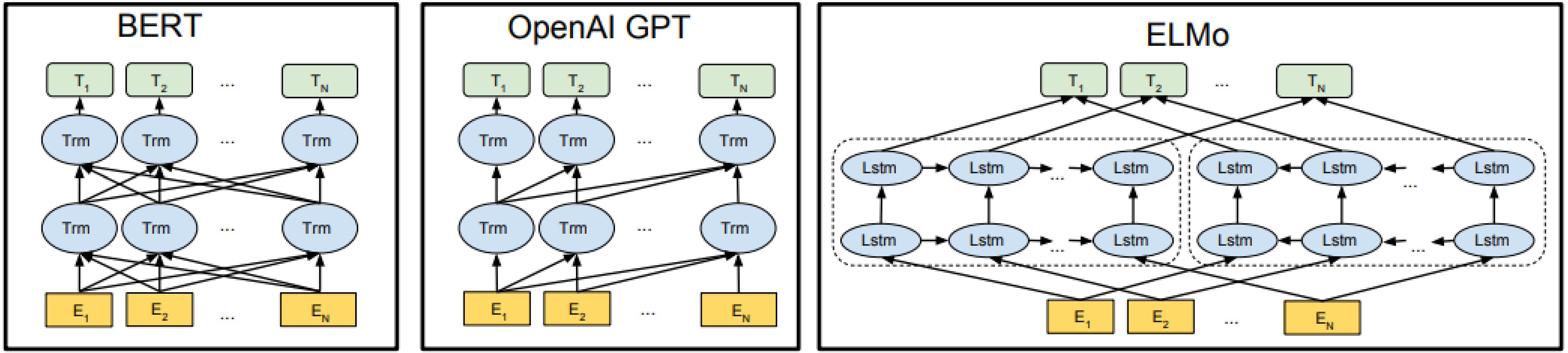
Input: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Output: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pretrained language models summary



bidirectional transformer

+ great representations

- can't generate text

one-directional transformer

+ can generate text

- OK representations

bidirectional LSTM

- OK representations

(largely supplanted by BERT)

Pretrained language models summary

- Language models can be trained on **very large and unlabeled** datasets of text (e.g., Wikipedia text). Often these are 100s or even 1000s of millions of sentences!

- Internal **learned representations** depend on context: the meaning of a word is informed by the **whole sentence!**

- Can even get us representations of entire sentences (e.g., the first output token for BERT)

- Can be used to either **extract representations** to replace standard word embeddings…

- …or directly finetuned on downstream tasks (which means we modify all the weights in the whole language model, rather than just using pretrained model hidden states)

This is **very important** in modern NLP because **it works extremely well!**