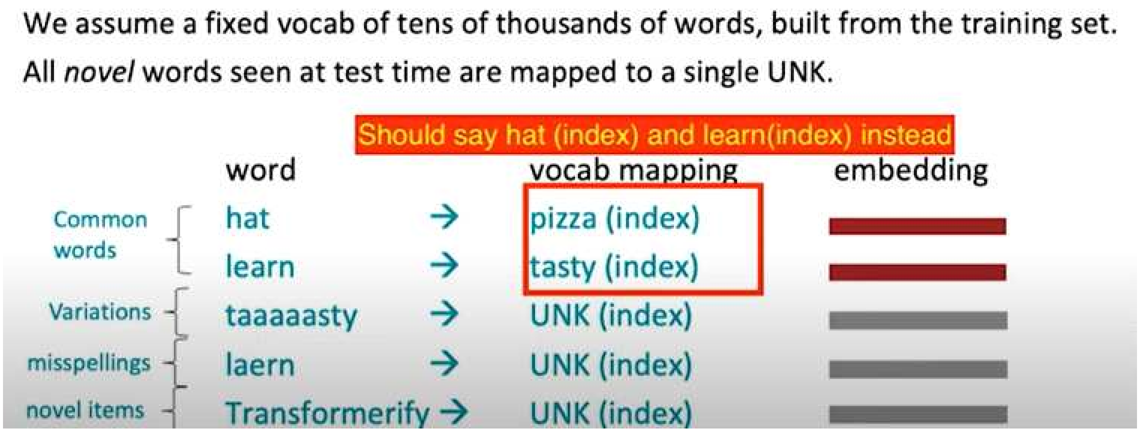
Lec10. Transformers and Pretraining

1. subword modeling



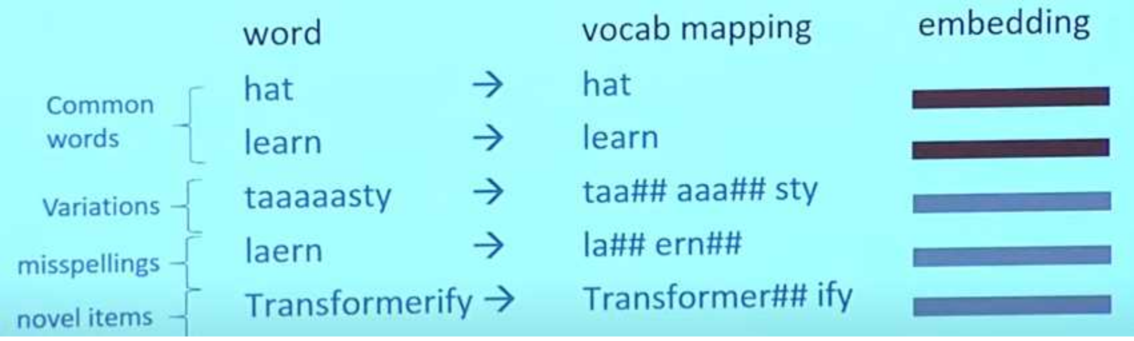
We assume a fixed vocab of tens of thousands of words, built from the training set.
All *novel* words seen at test time are mapped to a single UNK.

Should say hat (index) and learn(index) instead

	word		vocab mapping	embedding
Common words	hat	→	pizza (index)	▬
	learn	→	tasty (index)	▬
Variations	taaaaasty	→	UNK (index)	▬
misspellings	laern	→	UNK (index)	▬
novel items	Transformerify	→	UNK (index)	▬

Variation/misspellings/novel items -> UNK

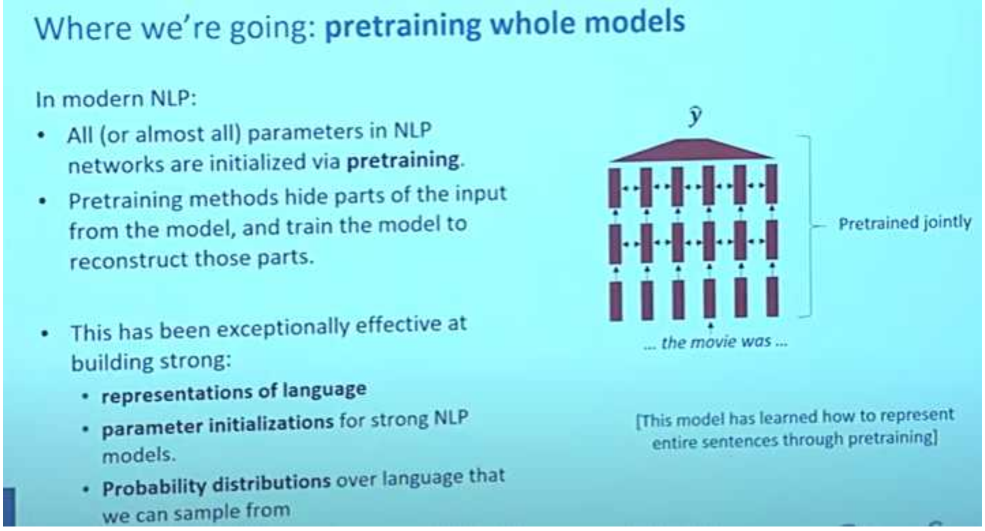
--> The byte-pair encoding algorithm



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

2. Motivating model pretraining from word embeddings

word2vec의 한계 ->



Where we're going: **pretraining whole models**

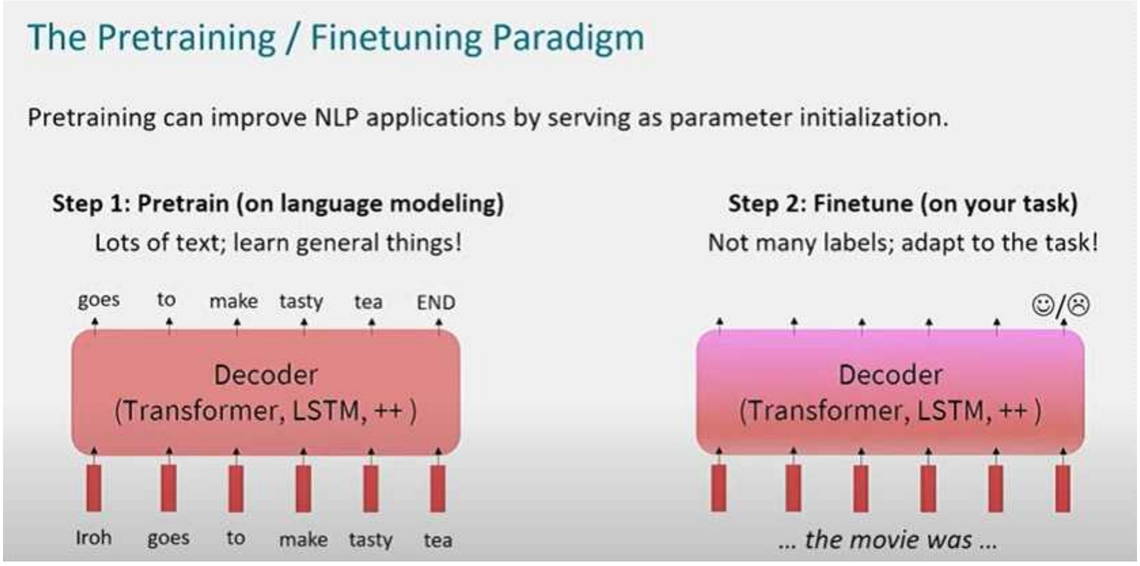
In modern NLP:

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

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from

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

- The Pretraining/ Finetuning Paradigm



The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

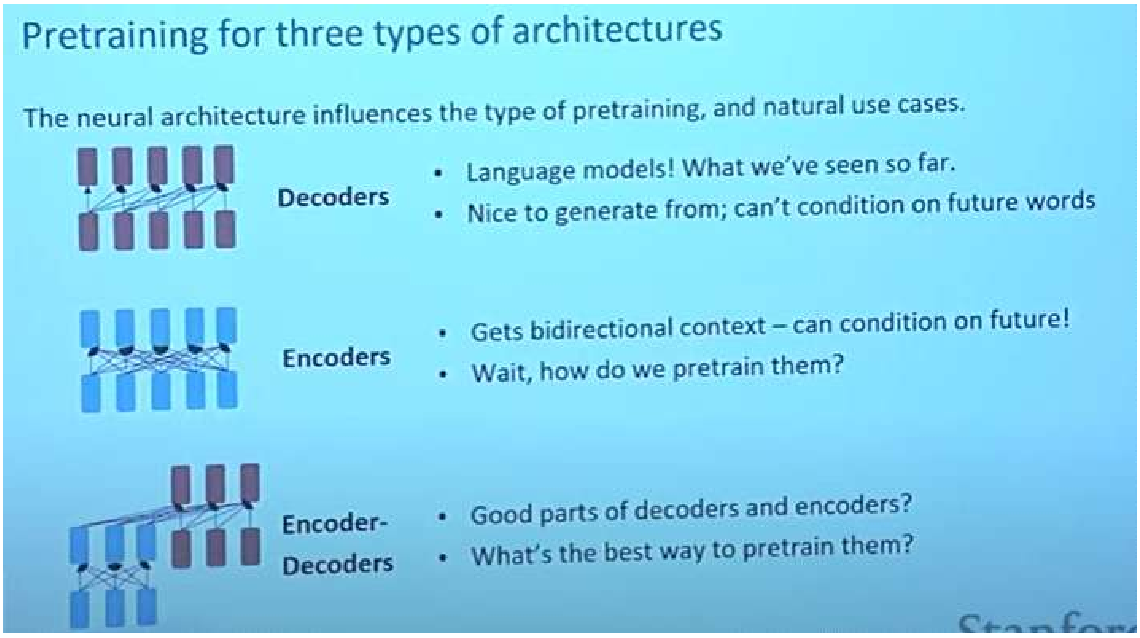
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!

Step 2: Finetune (on your task)

Not many labels; adapt to the task!

3. Model pretraining three ways



Pretraining for three types of architectures

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

Decoders

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

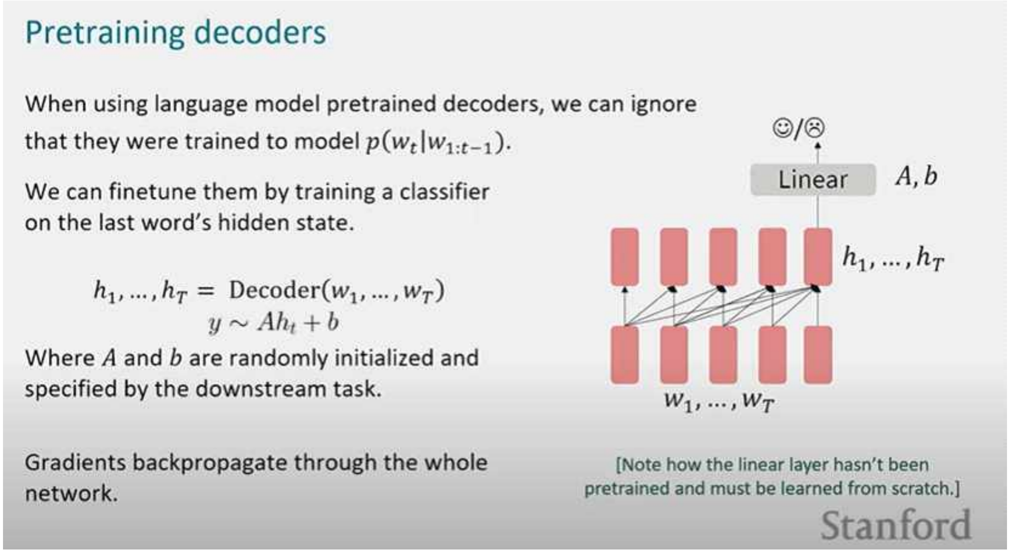
Encoders

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

Encoder-Decoders

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

1. Decoders



Pretraining decoders

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

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

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

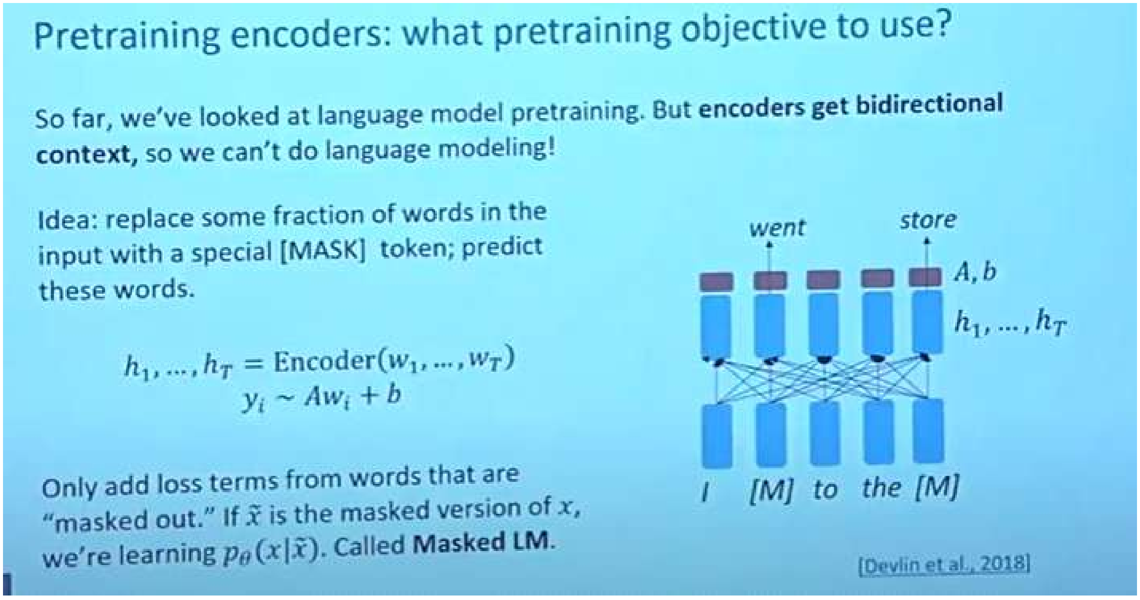
Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.

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

eg. GPT(Generative Pretrained Transformer)
- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus

2. Encoders



Pretraining encoders: what pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

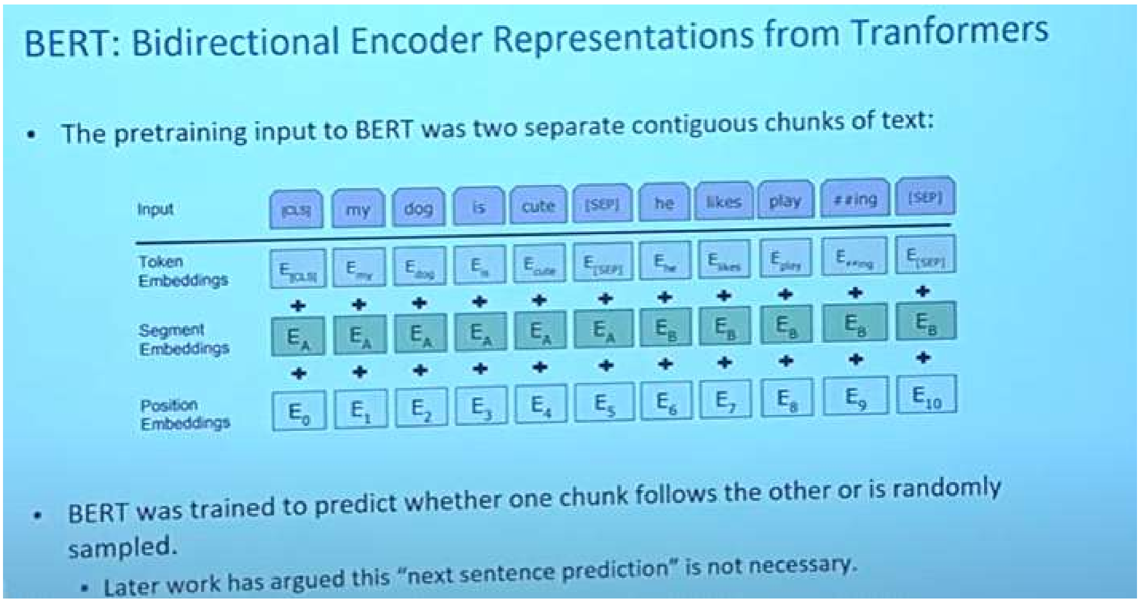
Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

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

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

[Devlin et al., 2018]

eg. BERT(Bidirectional Encoder Representations from Transforms)
- “Masked LM” objective and released the weights of a pretrained Transformer
-



BERT: Bidirectional Encoder Representations from Tranformers

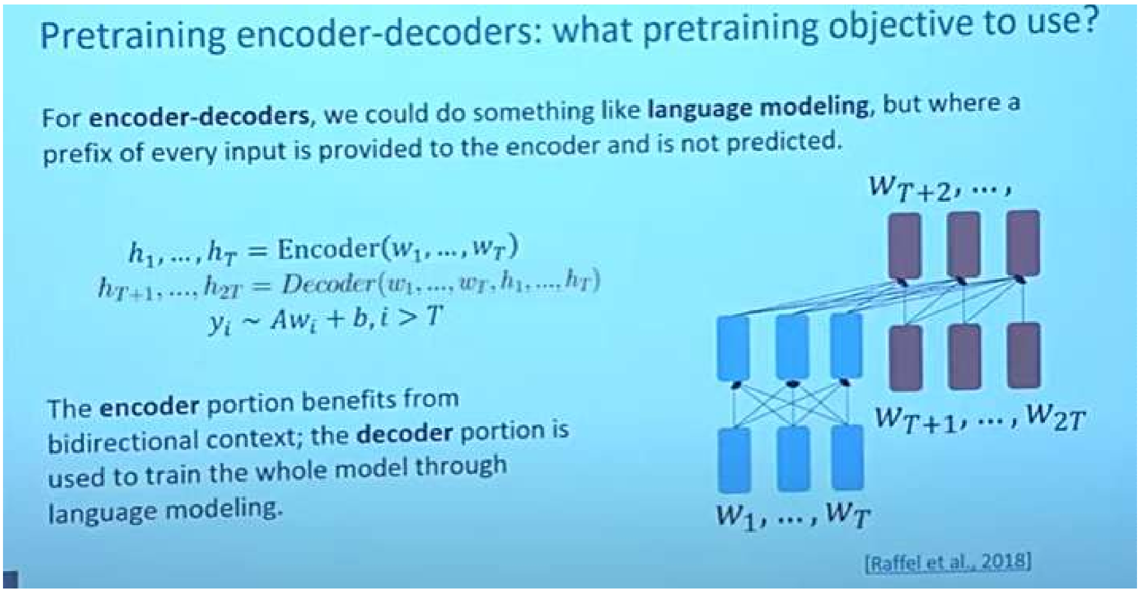
- The pretraining input to BERT was two separate contiguous chunks of text:

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
  - Later work has argued this "next sentence prediction" is not necessary.

- Two models: (BERT-base/large)
- Extensions: RoBERTa, SpanBERT, +++

3. Encoder-Decoder



Pretraining encoder-decoders: what pretraining objective to use?

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

$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$h_{T+1}, \ldots, h_{2T} = \textit{Decoder}(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_i \sim Aw_i + b, i > T$$

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

[Raffel et al., 2018]

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

4. Interlude: what do we think pretraining is teaching?
5. Very large models and in-context learning
- GPT3: has 175 billions parameters.
- -> without gradient steps