

Notes

25 April 2024 06:44

Lecture Outline:

Lecture Plan

1. A brief note on subword modeling
2. Motivating model pretraining from word embeddings
3. Model pretraining three ways
 1. Decoders
 2. Encoders
 3. Encoder-Decoders
4. Interlude: what do we think pretraining is teaching?
5. Very large models and in-context learning

Reminders:
Assignment 5 is out on Thursday! It covers lecture 8 and lecture 9(Today)!
It has ~pedagogically relevant math~

Stanford

Word Structures and Subword modelling

- Finite Vocab assumption:
 - Initially a fixed-known set of words (called as vocab) were used in training. Any new/unknown word that comes in during test time would be mapped to UNK i.e. unknown. A big information loss

Word structure and subword models

Let's take a look at the assumptions we've made about a language's vocabulary.

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.

	word	vocab mapping	embedding
Common words	hat	→ hat (index)	[red]
	learn	→ learn(index)	[red]
Variations	taaaaaasty	→ UNK (index)	[grey]
misspellings	laern	→ UNK (index)	[grey]
novel items	Transformerify	→ UNK (index)	[grey]

Stanford

- Such finite vocab assumption also does not help with languages having a complex word structure – because all the new words, even if they are conjugation/derivation of a known word will be mapped to UNK
- Byte-pair encoding: How do we get information below word level?

The byte-pair encoding algorithm

Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of **parts of words (subword tokens)**.
- At training and testing time, each word is split into a sequence of known subwords.

Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary.

1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
2. Using a corpus of text, find the most common adjacent characters "a,b"; add "ab" as a subword.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

Stanford

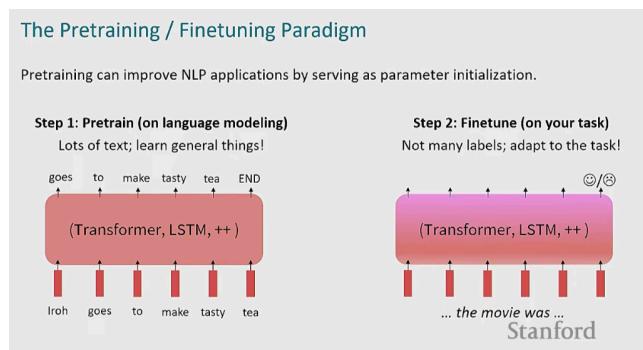
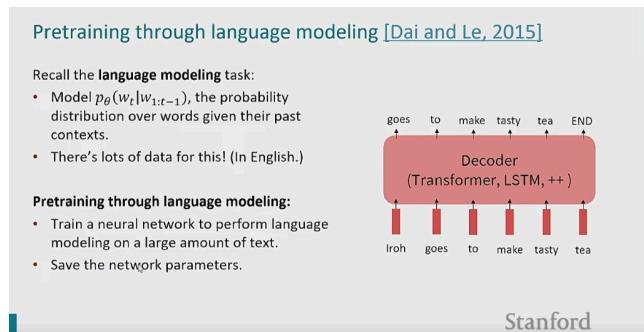
- Most common words end up as subword in vocab but most unknown words are split in sub words (sometimes known, intuitive or sometimes unknown)

Before (end-to-end) pretraining:

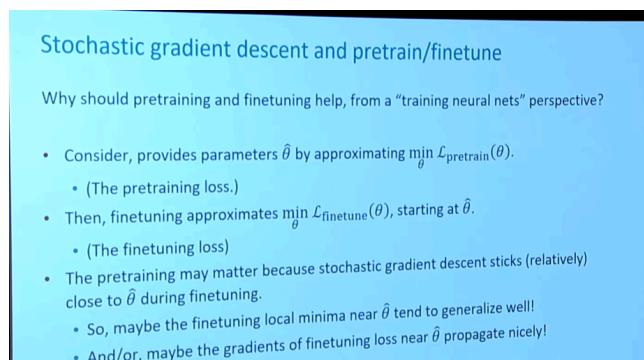
- We used learned **word embeddings** – the context of a particular use case was missing. I.e. How the words are being used in a sentence in a particular use case was not considered when the word embeddings were learnt.
- This forces user to have a huge training dataset of a particular use case for fine tuning.

(end-to-end) Pretraining:

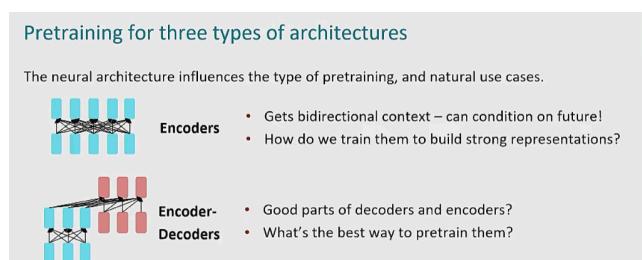
- How do we pretrain?
 - Masked-language modelling – by masking a specific word/part of the input sentence and asking the NN to recover the information
- What does this teach the network?
 - Geo-locations
 - e.g. Stanford University is located in _____, in California
 - Sense of physical world:
 - I put fork _____ the table
 - Entity relation
 - The woman walked across the street by checking for traffic over _____ shoulder
 - AND so on, we can teach the network to do sentiment analysis,

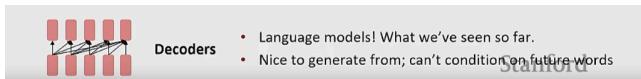


- Why to use pretrained NN parameters as starting point for fine tuning?
 - It offers just a better starting point for finding the minima in loss landscape



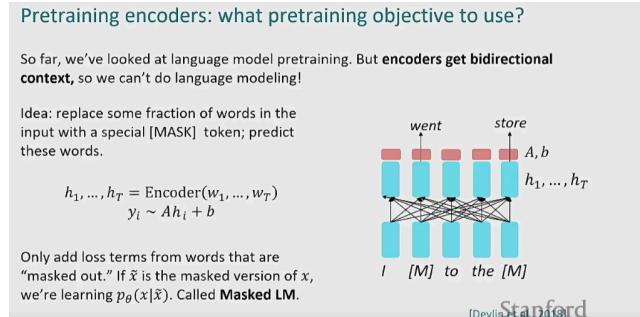
Three ways of pre-training:



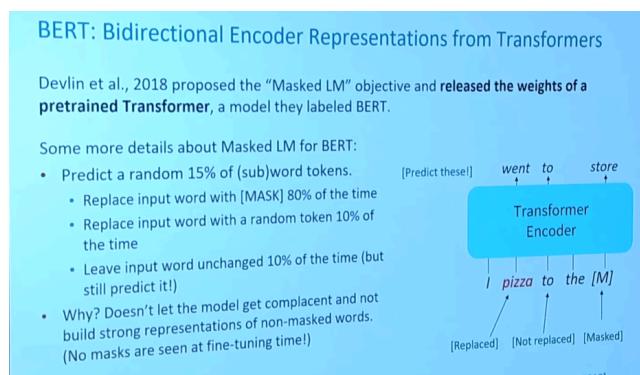


Encoder:

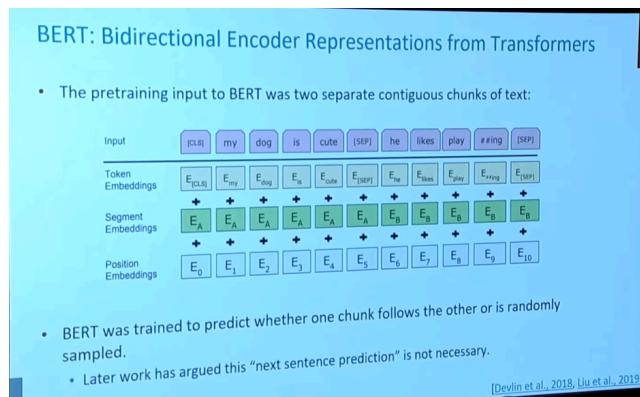
- As encoder have bi-directional context (remember RNN/LSTM lecture?) -- we can't do language modelling with them
- So – Hide some words from the model and ask model to predict the words



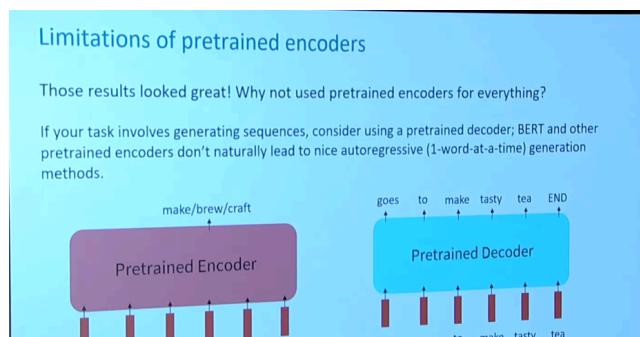
- e.g. of an (all) encoder model – **BERT** i.e. Bidirectional Encoder Representation from Transformers



- Adding long-distance dependency/coherence to BERT – by feeding two segments A and B as input and asking model if segment B is logical continuation of A

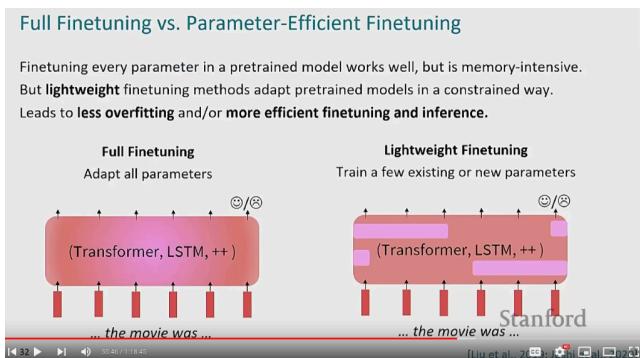


- **Limitations of (all) encoder models:** BERT is trained for filling the blanks, hence it works best when broader context is available and some words are to be predicted. This is not ideal for e.g. Sentence Generation (ChatGPT) like application

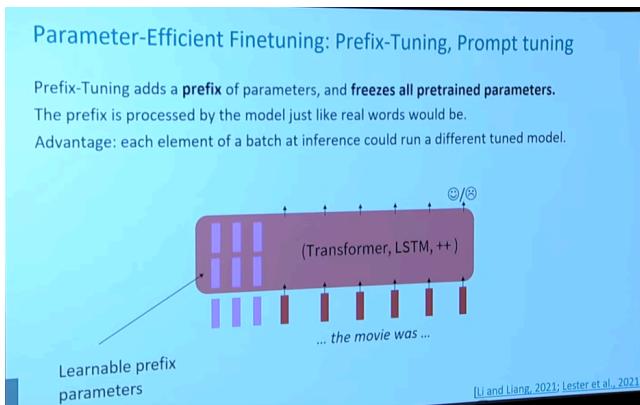




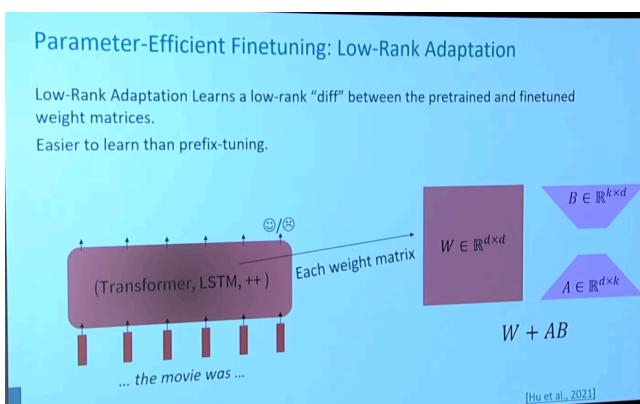
- Extensions of BERT:
 - RoBERT: trained BERT for long(er) duration and removed next sentence prediction.
 - SpanBERT: masking multiple contiguous words instead of masking individual words sporadically
- Parameter Efficient fine tuning: AS WE ARE TALKING ABOUT FINE TUNING
 - How to efficiently fine tune a pretrained model?
 - Instead of changing all the parameters during fine tuning, how about we (smartly) choose some parameters for fine tuning?



- An example is prefix tuning:

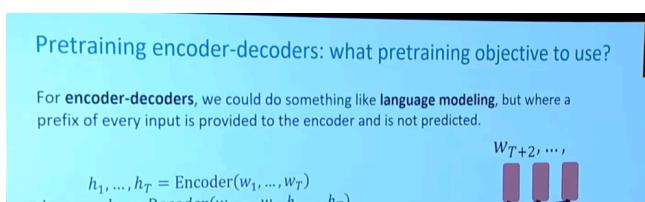


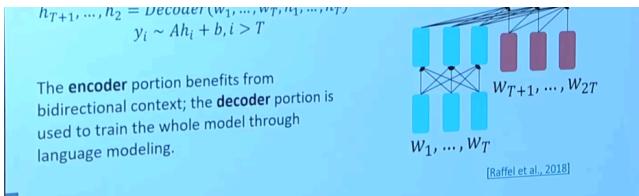
- (the famous) LoRA – Low Rank Matrix Adaption



Encoder – Decoder:

- As encoder has bi-directional context here, we can use decoder for language modelling and then train whole model (encoder+decoder) by using the loss of decoder





- o Masked LM also works for encoder – decoder model

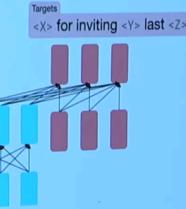
Pretraining encoder-decoders: what pretraining objective to use?

What [Raffel et al., 2018](#) found to work best was **span corruption**. Their model: T5.

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

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

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



Decoder:

Pretraining decoders

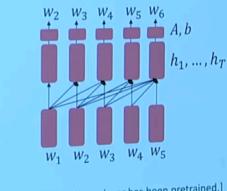
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks where the output is a sequence with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

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

$$w_t \sim Ah_{t-1} + b$$



Where A, b were pretrained in the language model!

- o The Famous GPT:

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

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
 - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

- o GPT-2 -- Just a scaled version of original GPT. Trained on ~9B words of text and contains ~1.5B parameters
- o GPT3 -- A very large, 175B parameter model. Trained on ~300B words of text. Showed ability of in-context learning without explicit inputs or gradient step.

GPT-3, In-context learning, and very large models

So far, we've interacted with pretrained models in two ways:

- Sample from the distributions they define (maybe providing a prompt)
- Fine-tune them on a task we care about, and take their predictions.

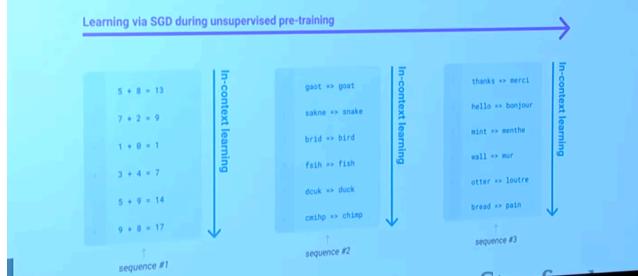
Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.
GPT-3 has 175 billion parameters.

- In-context learning: learning to do a task from very minimal specific inputs or almost no explicit inputs. I.e. the model learnt to perform a task by looking at non-explicit inputs.
- The ability of in-context learning appeared only in large models (trained on large data) --- TO ME, this is a very sophisticated pattern recognition, without explicit training.

GPT-3, In-context learning, and very large models

Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

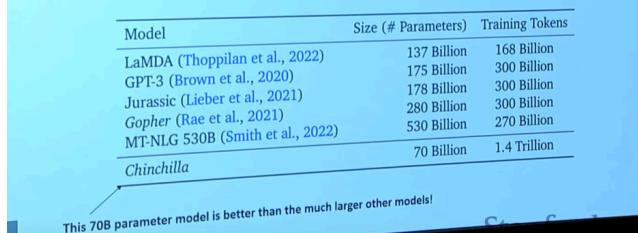


- Relation of number of parameters vs words in training data:

Scaling Efficiency: how do we best use our compute

GPT-3 was **175B parameters** and trained on **300B tokens** of text.

Roughly, the cost of training a large transformer scales as **parameters*tokens**. Did OpenAI strike the right parameter-token ratio to get the best model? No.



- Chain of thought prompting: (after pre-training)
 - Providing task specification (decomposition of steps to find answer) in the prompt

The prefix as task specification and scratch pad: chain-of-thought

