

Natural Language Processing

CS 3216/UG, AI 5203/PG

Week-11

Pre-training, Fine-tuning, BERT, MLM

Pre-training






Key ideas in pretraining

- Make sure your model can process large-scale, diverse datasets
- Don't use labeled data (otherwise you can't scale!)
- Compute-aware scaling

Word Structure and sub words

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	→	pizza (index)	
	learn	→	tasty (index)	
Variations	taaaaasty	→	UNK (index)	
misspellings	laern	→	UNK (index)	
novel items	Transformerify	→	UNK (index)	

**Brianna** @_parsimonia_ · 24h
Gooooooooo Vibesssssss

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 similar methods (WordPiece, SentencePiece) are used in pretrained models, like BERT, GPT.

Byte-pair Encoding

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

The BPE algorithm first counts all pairs of adjacent symbols: the most frequent is the pair `e r` because it occurs in *newer* (frequency of 6) and *wider* (frequency of 3) for a total of 9 occurrences.² We then merge these symbols, treating `er` as one symbol, and count again:

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _	
6 n e w er _	
3 w i d er _	
2 n e w _	






Now the most frequent pair is `er _`, which we merge; our system has learned that there should be a token for word-final `er`, represented as `er_`:


corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2 l o w e s t _	
6 n e w er_	
3 w i d er_	
2 n e w _	

Word Structure and sub words

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, **words are split into as many subwords as they have characters.**

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



Brianna @_parsimonia_ · 24h
Gooooood Vibesssssss

Motivation word meaning and Context

Recall the adage we mentioned at the beginning of the course:

“You shall know a word by the company it keeps” (J. R. Firth 1957: 11)

This quote is a summary of **distributional semantics**, and motivated word2vec. But:

“... the complete meaning of a word is always contextual,
and no study of meaning apart from a complete context
can be taken seriously.” (J. R. Firth 1935)

Consider I **record** the **record**: the two instances of **record** mean different things.

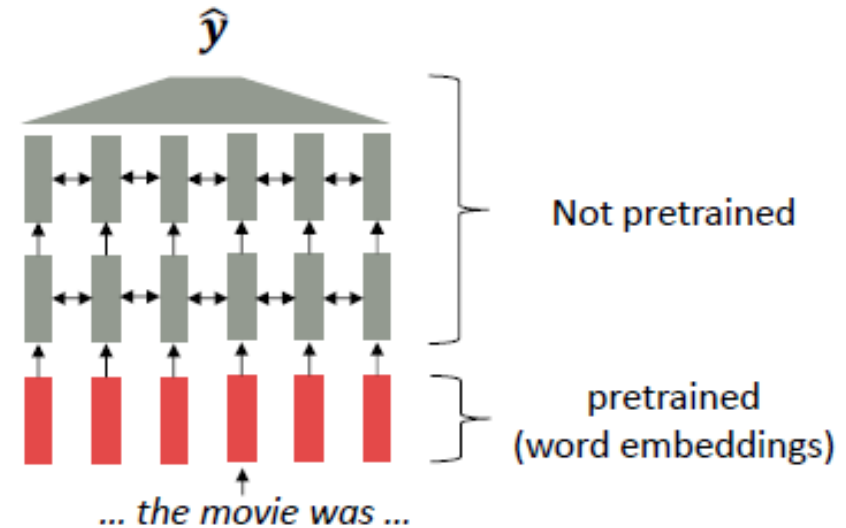
Where were we- Pretrained word embeddings

Circa 2015:

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.

Some issues to think about:

- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomly initialized!



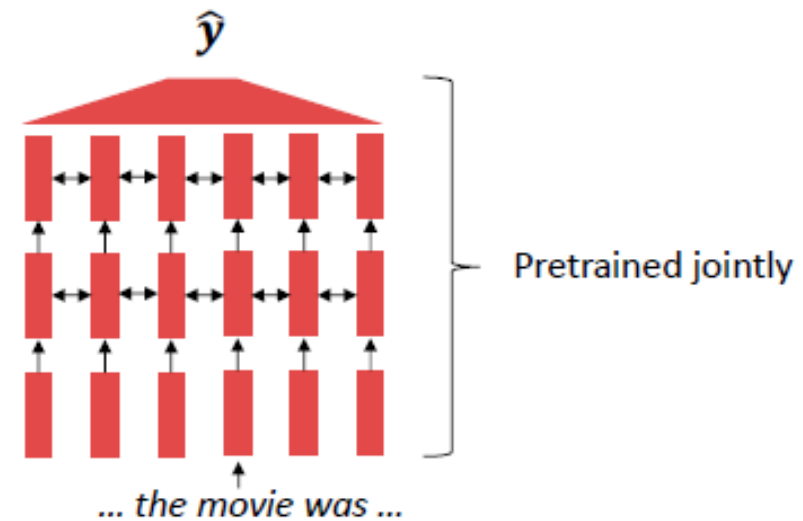
[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

How to learn Contextual representations?

Where are we going- Pretraining whole

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



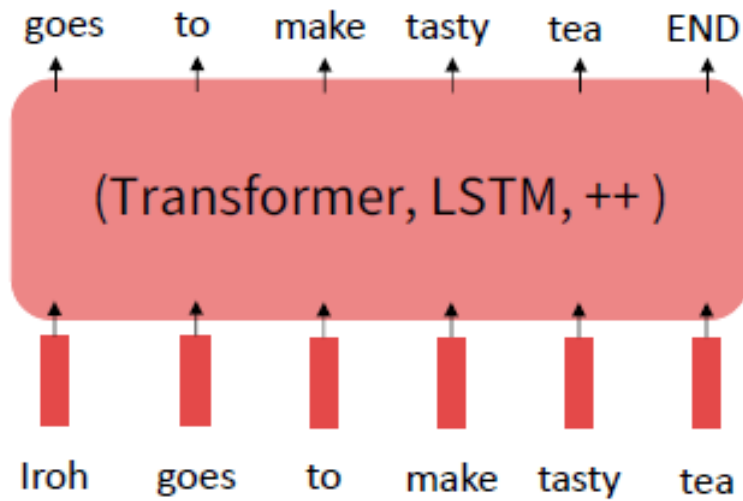
[This model has learned how to represent entire sentences through pretraining]

Pretraining and Fine-tuning

Pretraining can improve NLP applications by serving as parameter initialization.

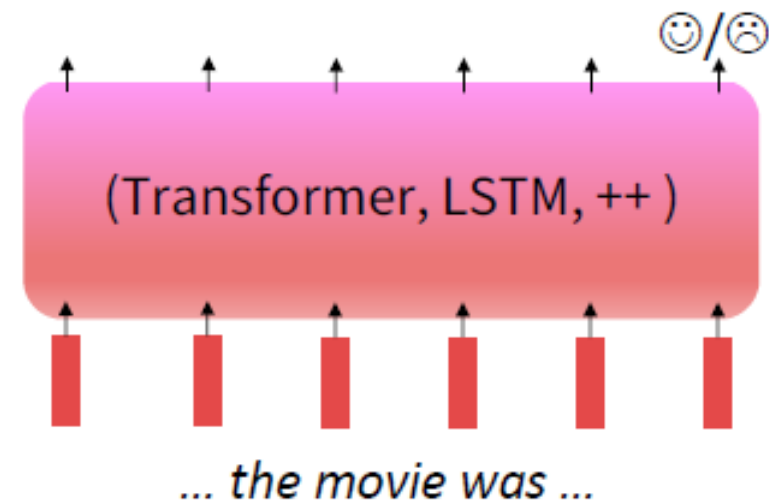
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



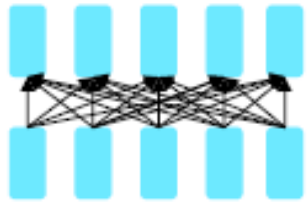
Step 2: Finetune (on your task)

Not many labels; adapt to the task!



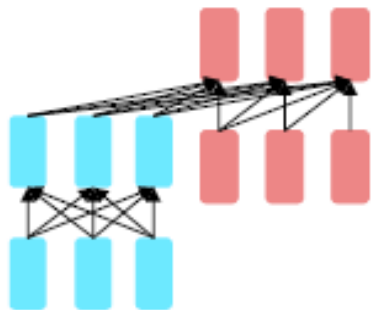
Pretraining for three types of architectures

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



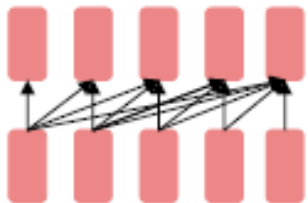
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



**Encoder-
Decoders**

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



Decoders

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

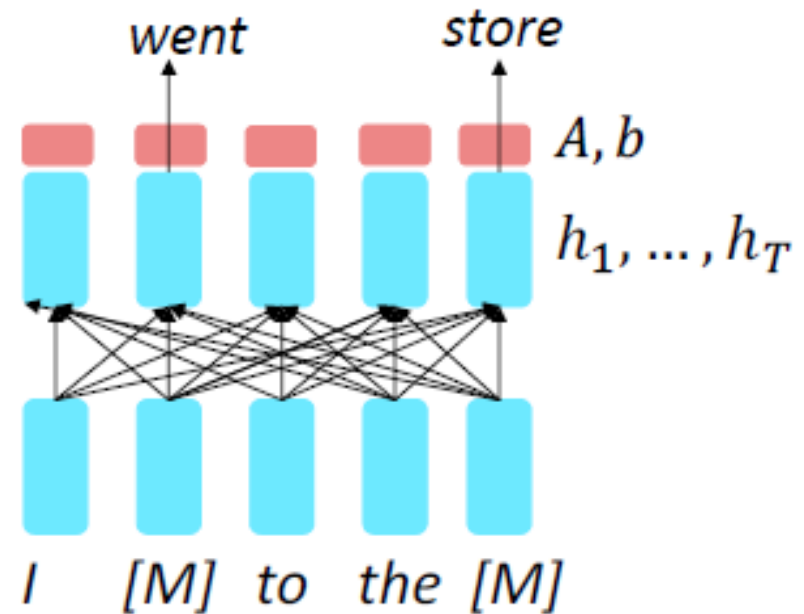
Pretraining Encoders

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.

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

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

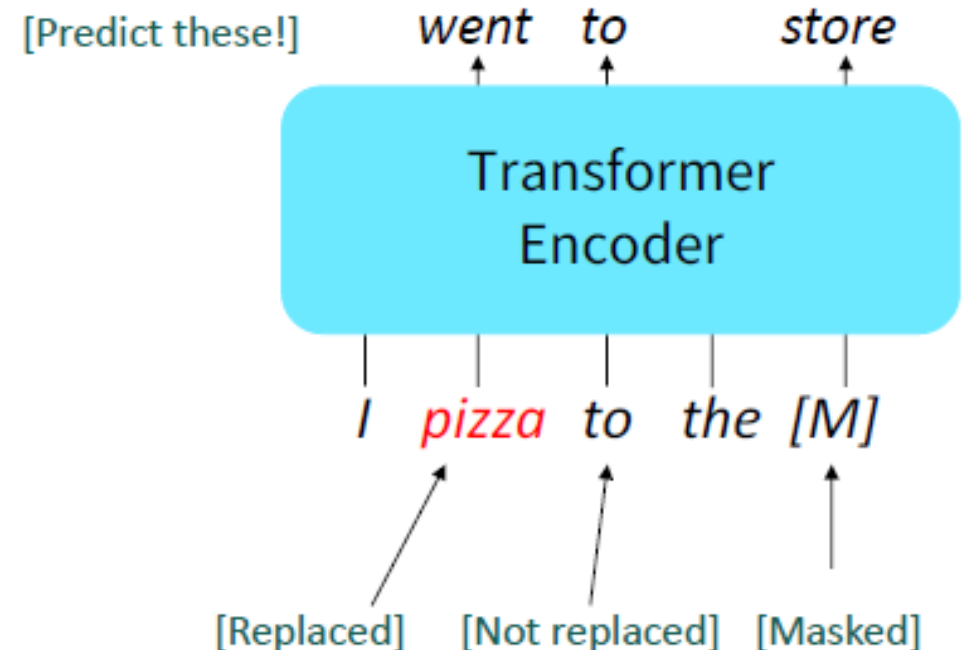


Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the “Masked LM” objective and **released the weights of a pretrained Transformer**, a model they labeled BERT.

Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
 - Replace input word with [MASK] 80% of the time
 - Replace input word with a random token 10% of the time
 - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



When predicting words within a sequence, all of the surrounding words can be used to gain contextual information.

Left context

Right context



A man was [MASK] on a river bank.

UNIDIRECTIONAL CONTEXT

When predicting future words, only the previous words can be used to gain contextual information.

Left context



Tokenization Algorithm used by BERT

WordPiece is the tokenization algorithm Google developed to pretrain BERT.

WordPiece

Tokenization Process:



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

Masked Language Modeling

Masked Language Modeling (MLM) (Devlin et al., 2019).

MLM uses unannotated text from a large corpus. Here, the model is presented with a series of sentences from the training corpus where a random sample of tokens from each training sequence is selected for use in the learning task.

Once chosen, a token is used in one of three ways:

- **It is replaced with the unique vocabulary token [MASK].**
- **It is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities.**
- **It is left unchanged.**

Pretraining and Fine-tuning

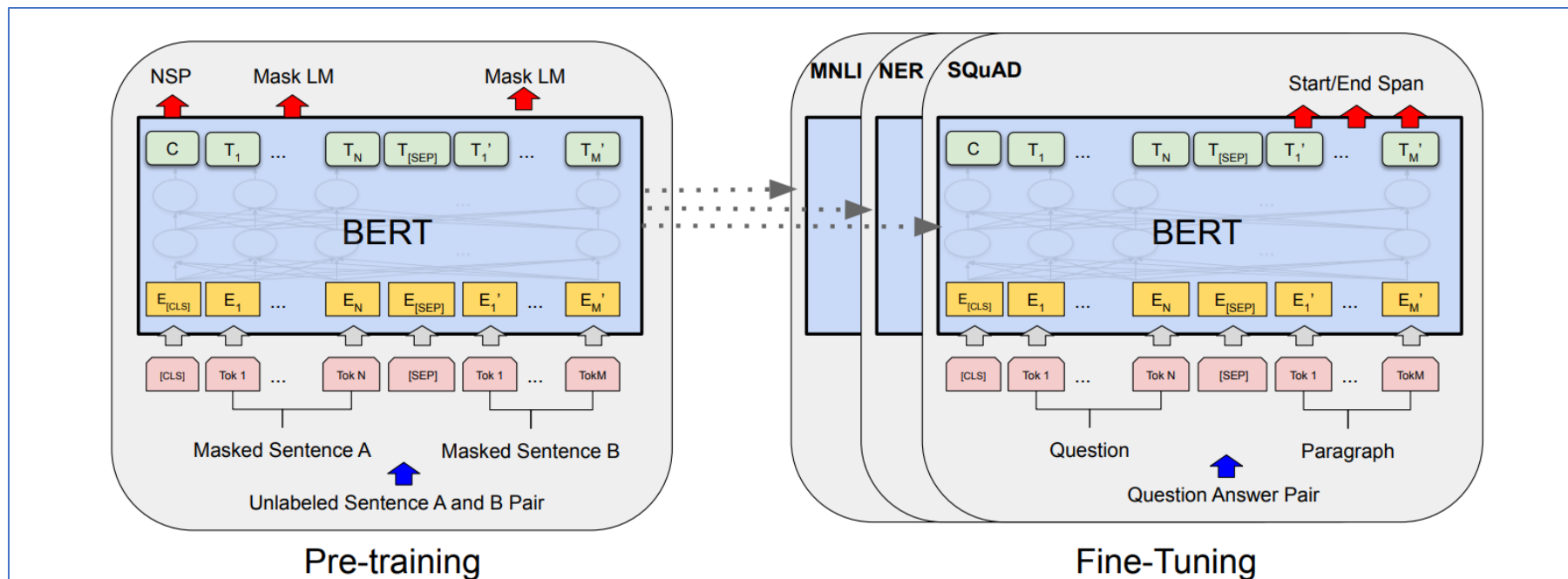


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers)

<https://arxiv.org/pdf/1810.04805>

Pretraining and Fine-tuning

- **Details of BERT Architecture in the below Research Paper [Must Read!!]**

<https://arxiv.org/pdf/1810.04805>

Domain-specific BERT Embeddings

Clinical-BERT Embeddings

<https://aclanthology.org/W19-1909/>

Domain-specific use-cases for Fine-tuned LLMs

- Support issue prioritization
- Fraud detection
- Blog writing
- Lead qualification
- Text classification
- Question answering

Reference materials

- <https://vlanc-lab.github.io/mu-nlp-course/>
- Lecture notes
- (A) Speech and Language Processing by Daniel Jurafsky and James H. Martin
- (B) Natural Language Processing with Python. (updated edition based on Python 3 and NLTK 3) Steven Bird et al. O'Reilly Media

