Foundations of NLP

Lecture-9

Blue Score, BERT (Bidirectional Encoder Representations)

Mahindra™ University

Global Thinkers. Engaged Leaders.

Recap

- Language modeling
- Recurrent Neural Network and Implementation
- Applications of Recurrent Neural Network
- Language modeling using Long Short-term Memory
- Sequence to Sequence learning
- Attention
- Transformers

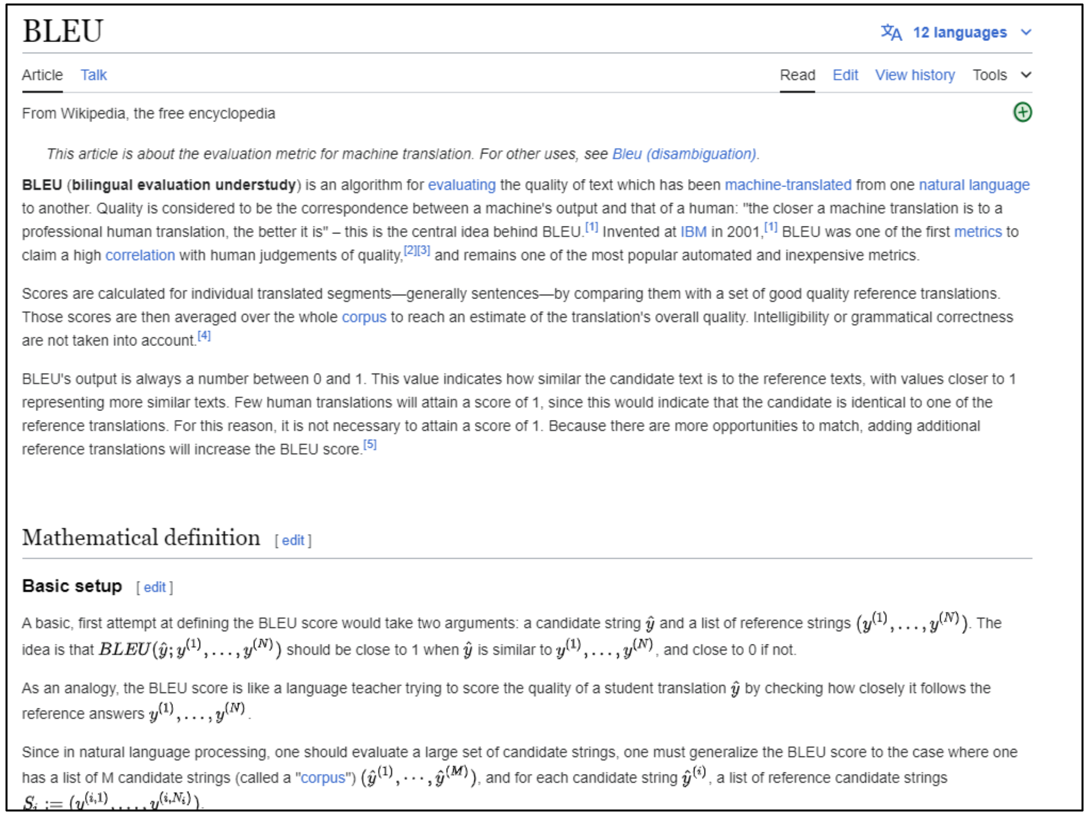
Why?

- You covered Transformers for Machine Translation Task.

How would you evaluate the quality of translations?

How to evaluate Machine Translation?

BLEU (Bilingual Evaluation Understudy) ✓



BLEU

12 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about the evaluation metric for machine translation. For other uses, see Bleu (disambiguation).

BLEU (**bilingual evaluation understudy**) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" – this is the central idea behind BLEU.[1] Invented at IBM in 2001,[1] BLEU was one of the first metrics to claim a high correlation with human judgements of quality,[2][3] and remains one of the most popular automated and inexpensive metrics.

Scores are calculated for individual translated segments—generally sentences—by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Intelligibility or grammatical correctness are not taken into account.[4]

BLEU's output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts. Few human translations will attain a score of 1, since this would indicate that the candidate is identical to one of the reference translations. For this reason, it is not necessary to attain a score of 1. Because there are more opportunities to match, adding additional reference translations will increase the BLEU score.[5]

Mathematical definition [edit]

Basic setup [edit]

A basic, first attempt at defining the BLEU score would take two arguments: a candidate string $\hat{y}$ and a list of reference strings $(y^{(1)}, \dots, y^{(N)})$. The idea is that $BLEU(\hat{y}; y^{(1)}, \dots, y^{(N)})$ should be close to 1 when $\hat{y}$ is similar to $y^{(1)}, \dots, y^{(N)}$, and close to 0 if not.

As an analogy, the BLEU score is like a language teacher trying to score the quality of a student translation $\hat{y}$ by checking how closely it follows the reference answers $y^{(1)}, \dots, y^{(N)}$.

Since in natural language processing, one should evaluate a large set of candidate strings, one must generalize the BLEU score to the case where one has a list of M candidate strings (called a "corpus") $(\hat{y}^{(1)}, \cdots, \hat{y}^{(M)})$, and for each candidate string $\hat{y}^{(i)}$, a list of reference candidate strings $S_i := (y^{(i,1)}, \dots, y^{(i,N_i)})$.

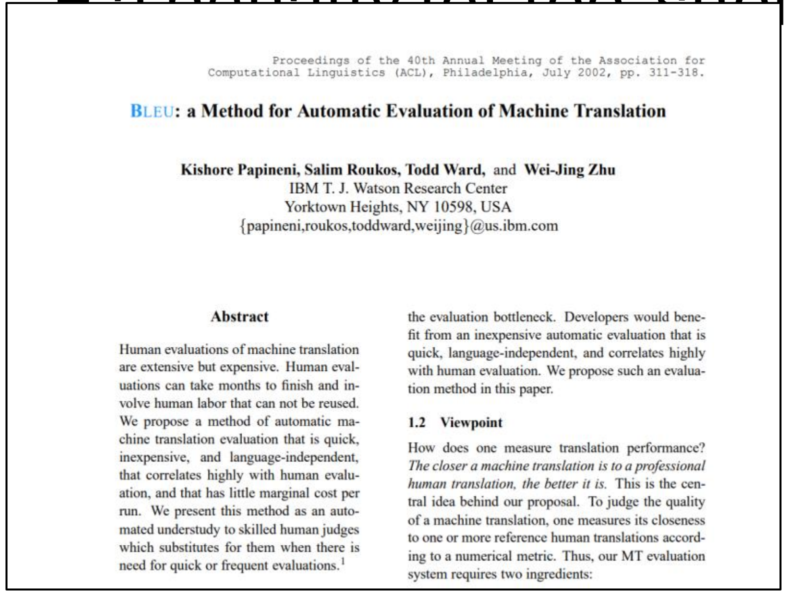
https://en.wikipedia.org/wiki/BLEU

How to evaluate Machine Translation?

BLEU (Bilingual Evaluation Understudy)

BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:

- Geometric mean of n-gram precision (usually for 1, 2, 3 and 4-grams)
- + a penalty for too short system translations



Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 311-318.

BLEU: a Method for Automatic Evaluation of Machine Translation

Kishore Papineni, Salim Roukos, Todd Ward, and **Wei-Jing Zhu**
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
{papineni,roukos,toddward,weijing}@us.ibm.com

Abstract

Human evaluations of machine translation are extensive but expensive. Human evaluations can take months to finish and involve human labor that can not be reused. We propose a method of automatic machine translation evaluation that is quick, inexpensive, and language-independent, that correlates highly with human evaluation, and that has little marginal cost per run. We present this method as an automated understudy to skilled human judges which substitutes for them when there is need for quick or frequent evaluations.[1]

the evaluation bottleneck. Developers would benefit from an inexpensive automatic evaluation that is quick, language-independent, and correlates highly with human evaluation. We propose such an evaluation method in this paper.

1.2 Viewpoint

How does one measure translation performance? *The closer a machine translation is to a professional human translation, the better it is.* This is the central idea behind our proposal. To judge the quality of a machine translation, one measures its closeness to one or more reference human translations according to a numerical metric. Thus, our MT evaluation system requires two ingredients:

Source paper: *"BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002*

https://aclanthology.org/P02-1040.pdf

How to calculate BLEU Score?

French: Le chat est sur le tapis → Source language

Reference: The cat is on the mat → Human-Written Translation

Model

MT Output- Candidate: The cat on mat — Machine generated Translation

$$P = \frac{m}{w_t}$$

m = No of candidate words in the reference

w_t = Total No of words in the candidate

m = 4

$w_t = 4$

$= \frac{4}{4} = 1$

How to calculate BLEU Score? (What can go wrong?)

French: Le chat est sur le tapis

Reference 1: The cat is on the mat

Reference 2: There is a cat on the mat

Reference

MT Output- Candidate: the the the the the the the

m = No of candidate words in the reference

$m = 7$

$w_t = 7$

$$P = \frac{m}{w_t} = \frac{7}{7} = 1$$

How to calculate BLEU Score?

MT The cat is on the mat

MT the cat the cat on the mat

	Combo	Count
the cat	2	1
cat the	1	0
cat on	1	0
on the	1	1
the mat	1	1

Modified

$p = \frac{7}{7} = 1$

2006

$m_t = 7$

$w_t = 7$

$\frac{3}{6}$

→ Bigram

My Neural Machine Translation model:
generates some output

BLEU Score:

How to calculate BLEU Score?

Source paper:*"BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002*

https://aclanthology.org/P02-1040.pdf

BLEU Score

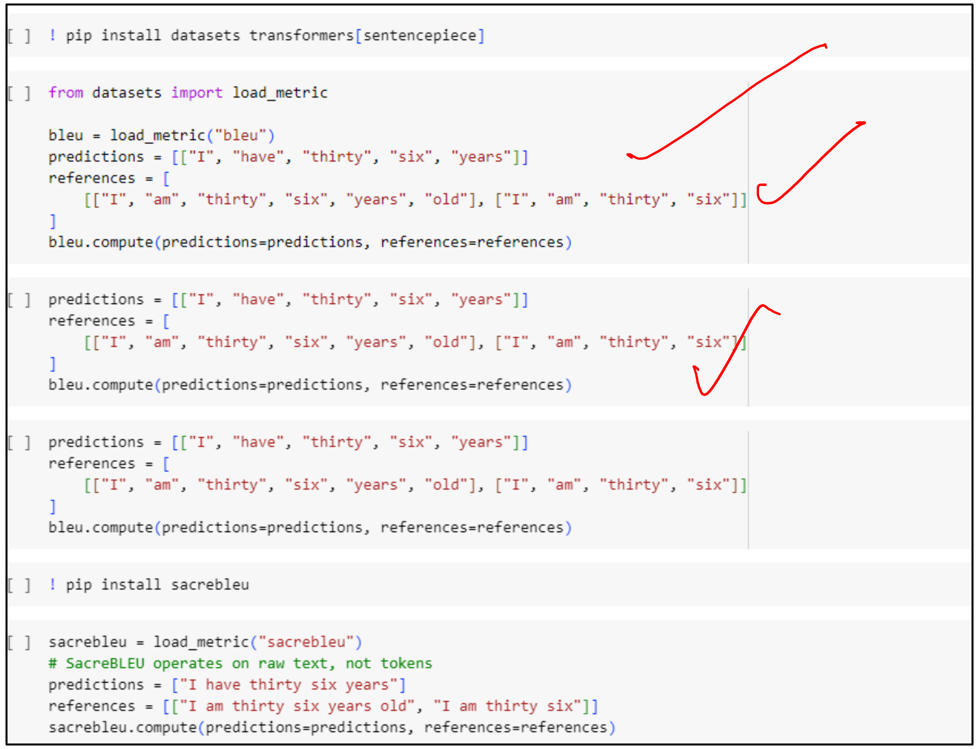
Pros:

- Simple and widely used metric
- BLEU is useful but imperfect

Cons:

- Doesn't incorporate sentence structure. ✓
  - If the word is placed in the wrong place then meaning changes ✓
- Does not take in account semantic similarity of the words ✓
  - There are many valid ways to translate a sentence
- More focused on precision rather than recall
- Struggle with non-english languages.

BLEU Score Python Code



```
[ ] ! pip install datasets transformers[sentencepiece]

[ ] from datasets import load_metric

    bleu = load_metric("bleu")
    predictions = [["I", "have", "thirty", "six", "years"]]
    references = [
        [["I", "am", "thirty", "six", "years", "old"], ["I", "am", "thirty", "six"]]
    ]
    bleu.compute(predictions=predictions, references=references)

[ ] predictions = [["I", "have", "thirty", "six", "years"]]
    references = [
        [["I", "am", "thirty", "six", "years", "old"], ["I", "am", "thirty", "six"]]
    ]
    bleu.compute(predictions=predictions, references=references)

[ ] predictions = [["I", "have", "thirty", "six", "years"]]
    references = [
        [["I", "am", "thirty", "six", "years", "old"], ["I", "am", "thirty", "six"]]
    ]
    bleu.compute(predictions=predictions, references=references)

[ ] ! pip install sacrebleu

[ ] sacrebleu = load_metric("sacrebleu")
    # SacreBLEU operates on raw text, not tokens
    predictions = ["I have thirty six years"]
    references = [["I am thirty six years old", "I am thirty six"]]
    sacrebleu.compute(predictions=predictions, references=references)
```

https://colab.research.google.com/github/huggingface/notebooks/blob/master/course/videos/bleu_metric.ipynb

Performance of Transformer models- Good results

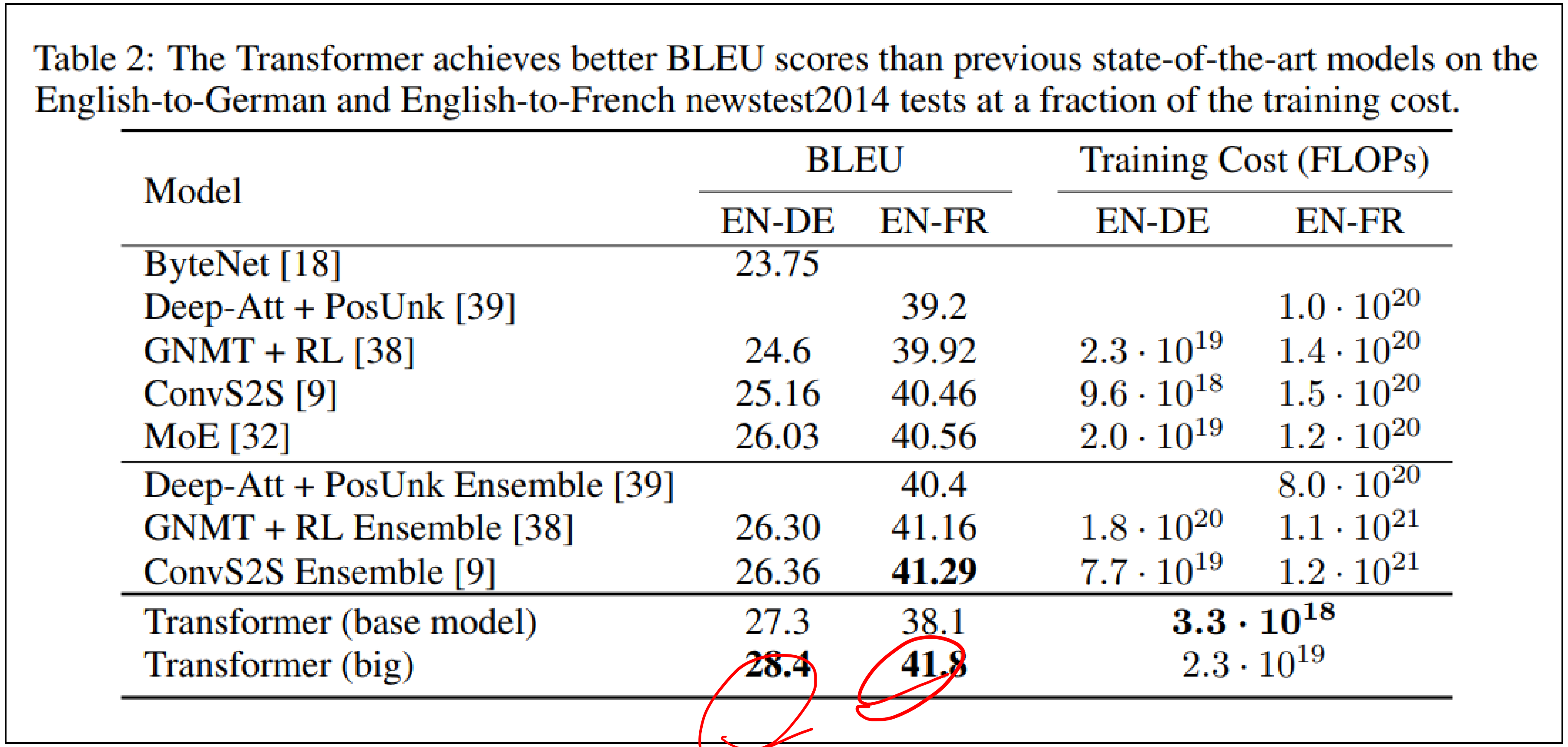


Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

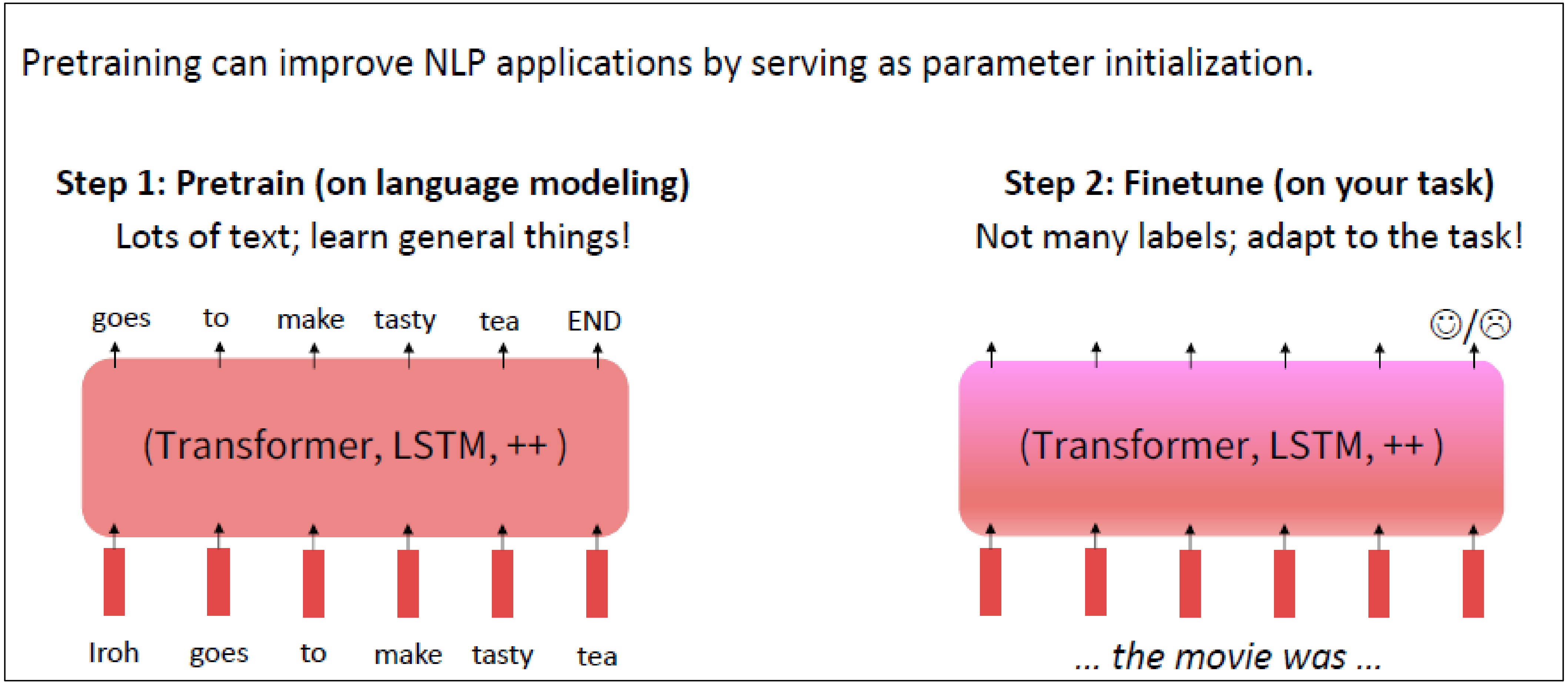
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Pretraining

To create domain-specific Language Models (LLMs), you have two main approaches:

- **Training from Scratch**: Building specialized LLMs entirely from domain-specific data.
- **Continual Pre-training**: Enhancing existing LLMs by further pre-training them with domain-specific data.

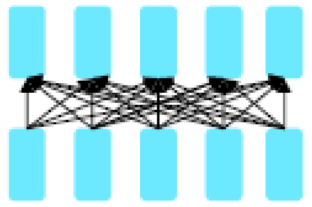
Pretraining and Fine-tuning


Pretraining can improve NLP applications by serving as parameter initialization.

Step 1: Pretrain (on language modeling)
Lots of text; learn general things!
goes
to
make
tasty
tea
END
(Transformer, LSTM, ++)
Iroh
goes
to
make
tasty
tea
Step 2: Finetune (on your task)
Not many labels; adapt to the task!
☺/☹
(Transformer, LSTM, ++)
... the movie was ...

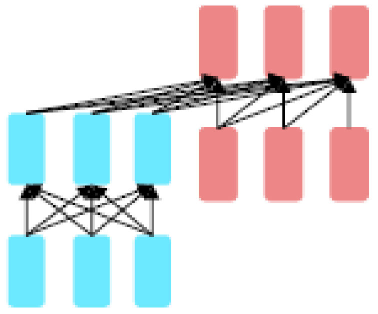
Pretraining for three types of architectures

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



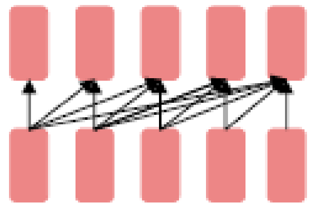
Encoders

- 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

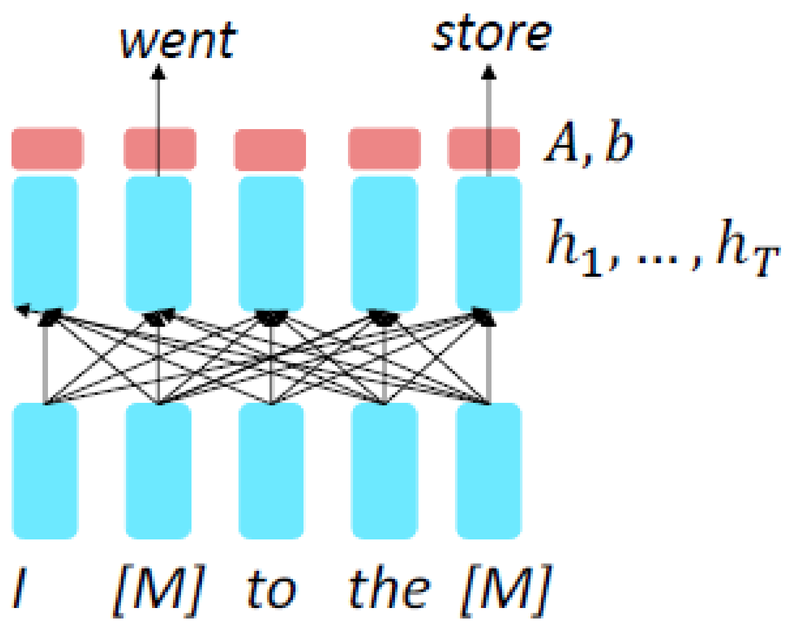
Pretraining Encoders

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**.



Pre-training

Key ideas in pretraining

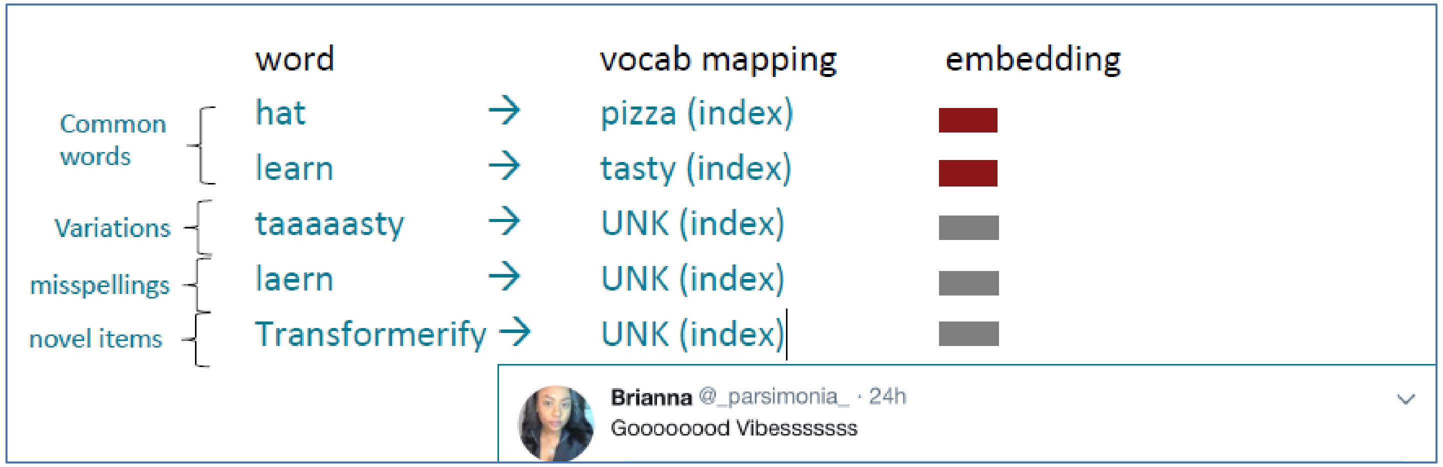
- Make sure your model can process large-scale, diverse datasets
- Don't use labeled data (otherwise you can't scale!)
- Compute-aware scaling

https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture09-pretraining-updated.pdf

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
Goooooood 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.)**

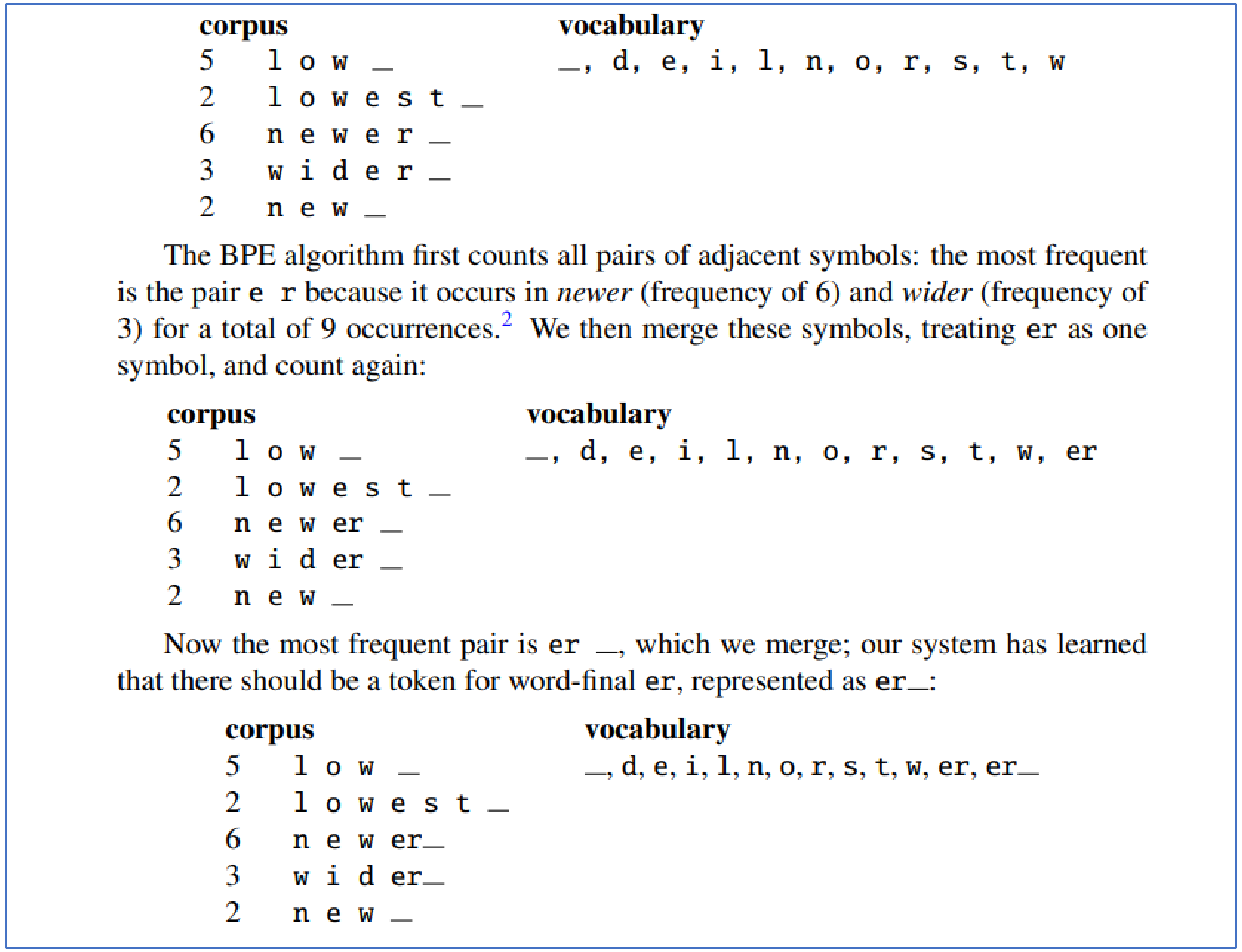
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens).
- At training and testing time, each word is split into a sequence of known subwords.

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_`:

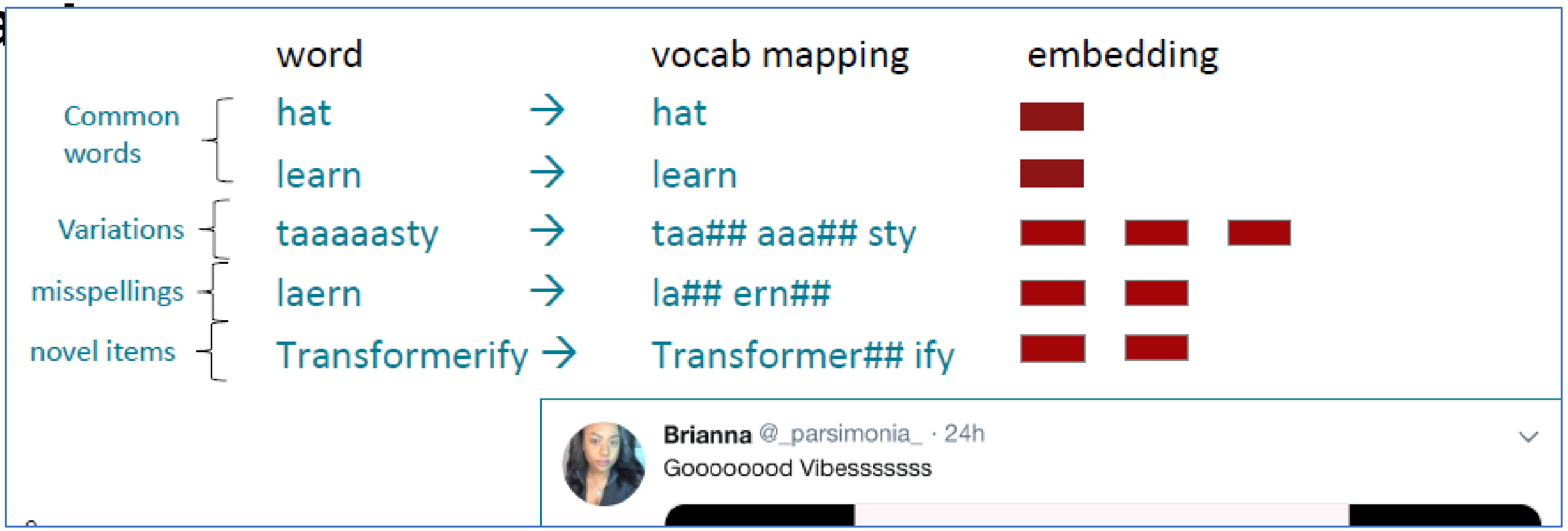
corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

https://web.stanford.edu/~jurafsky/slp3/2.pdf

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 chara**



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 d**istributional 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.

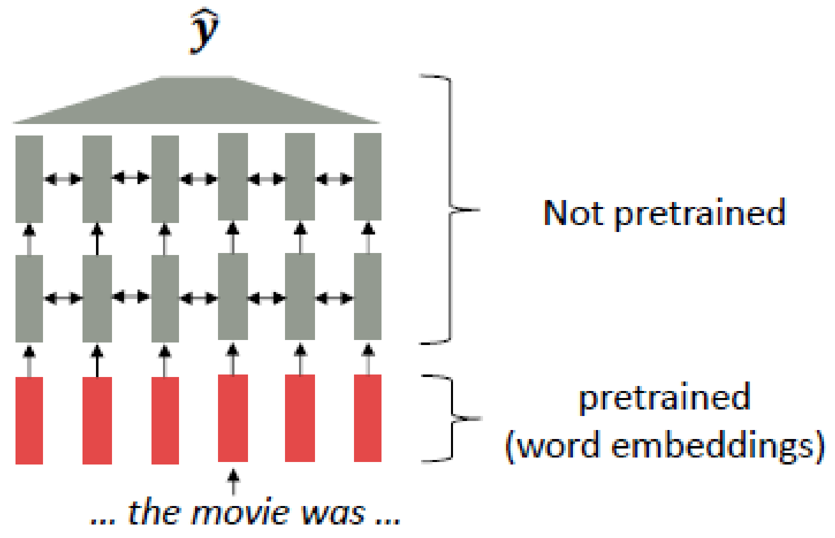
Where were we- Pretrained word embeddings

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:

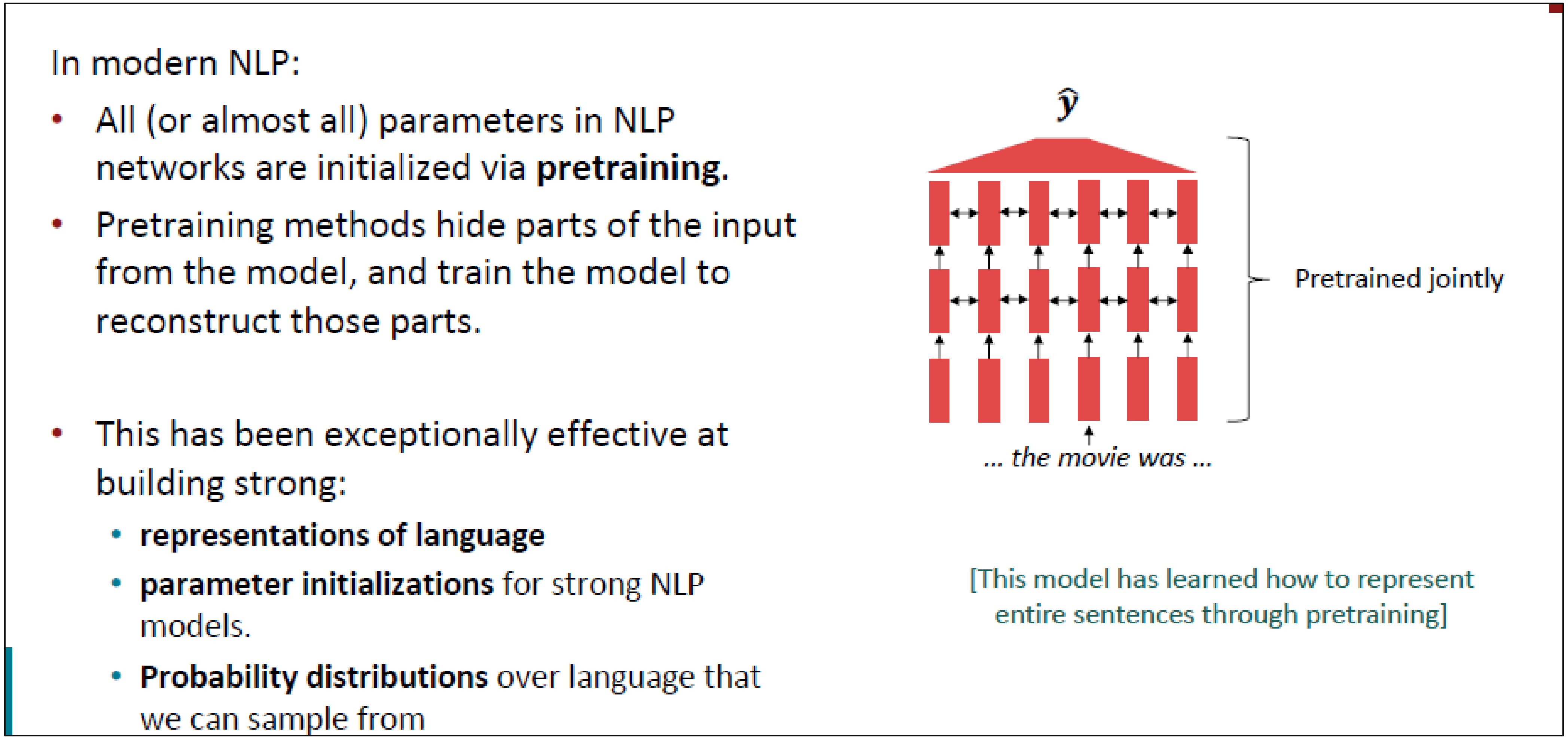
- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomly initialized!



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

How to learn Contextual representations?

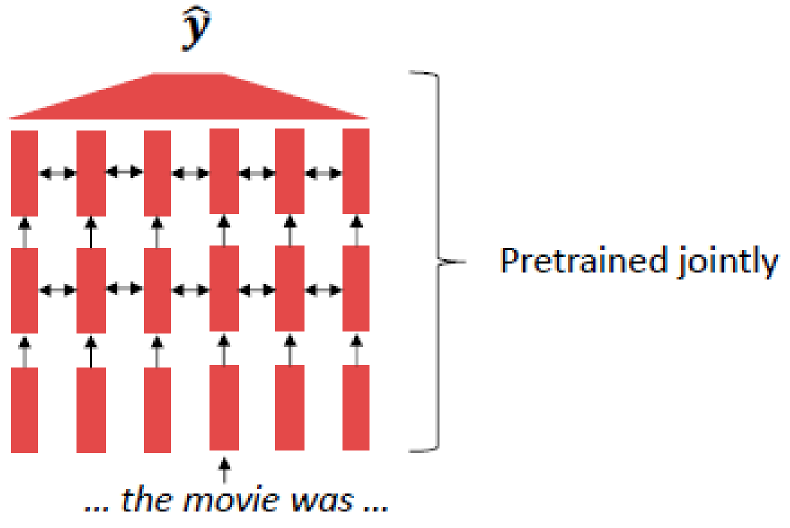
Where are we going- Pretraining whole



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.

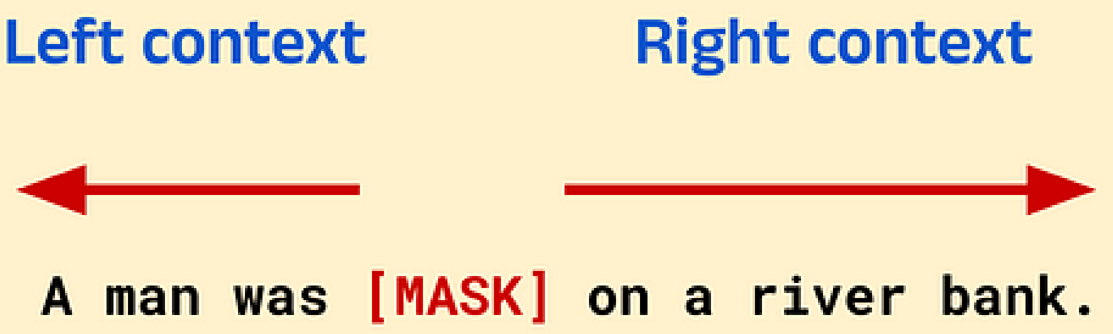
- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from



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

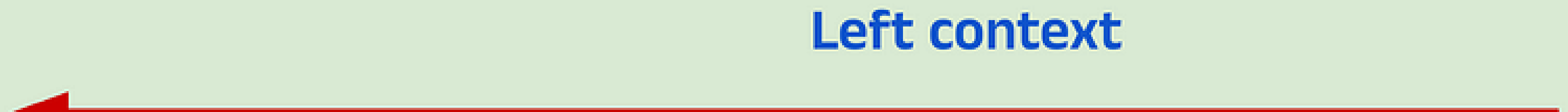
Bidirectional Encoder Representations from Transformers

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



UNIDIRECTIONAL CONTEXT

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

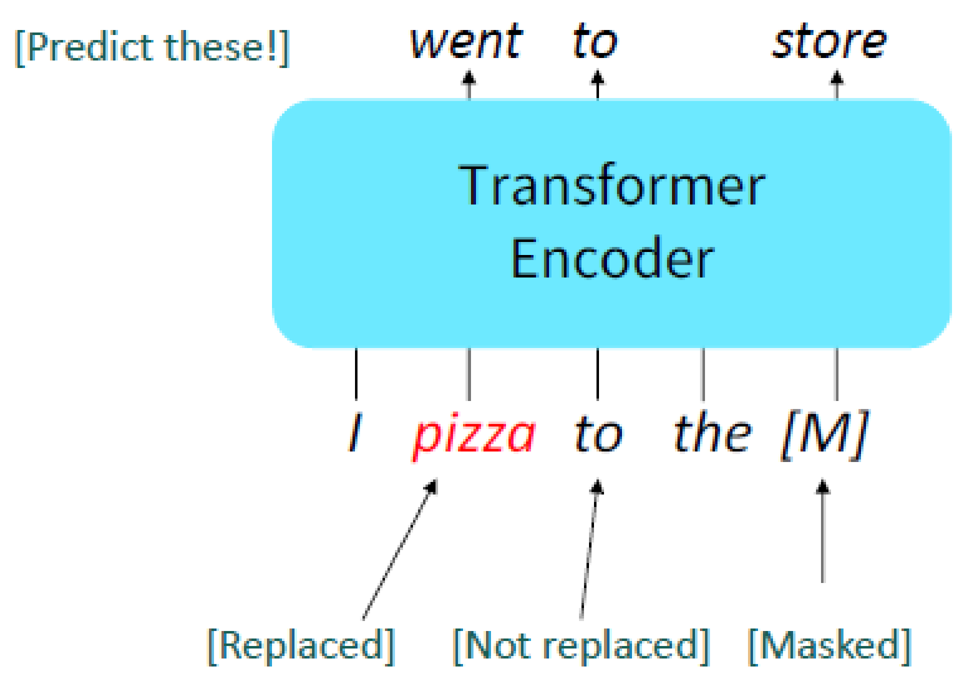


Bidirectional Encoder Representations from Transformers

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!)



https://arxiv.org/pdf/1810.04805

Tokenization Algorithm used by BERT

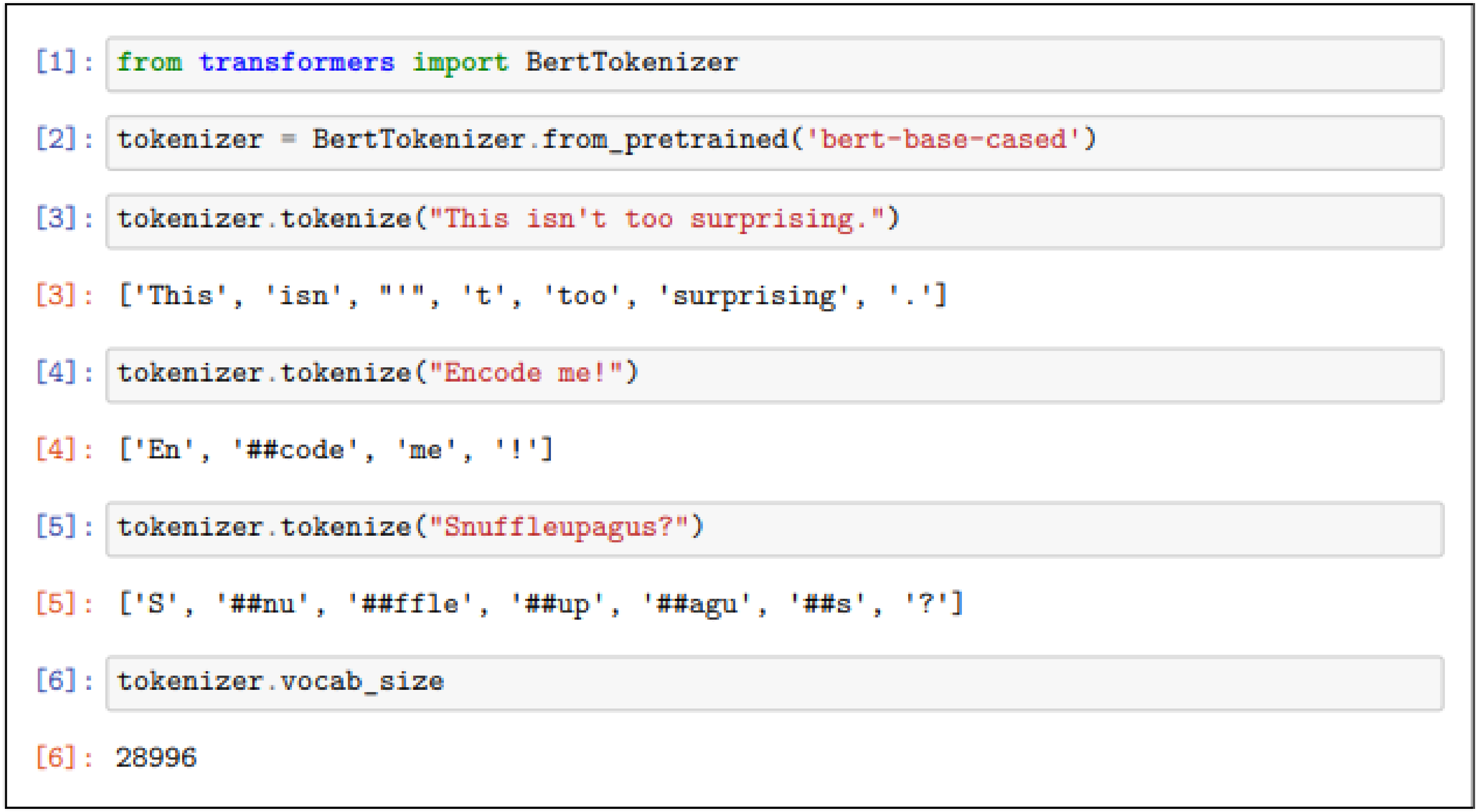
WordPiece is the tokenization algorithm Google developed to pretrain BERT.

WordPiece

Tokenization Process:

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

BERT Tokenizer



```
[1]: from transformers import BertTokenizer

[2]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

[3]: tokenizer.tokenize("This isn't too surprising.")

[3]: ['This', 'isn', "'", 't', 'too', 'surprising', '.']

[4]: tokenizer.tokenize("Encode me!")

[4]: ['En', '##code', 'me', '!']

[5]: tokenizer.tokenize("Snuffleupagus?")

[5]: ['S', '##nu', '##ffle', '##up', '##agu', '##s', '?']

[6]: tokenizer.vocab_size

[6]: 28996
```

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.**

https://web.stanford.edu/~jurafsky/slp3/11.pdf

Pretraining and Fine-tuning

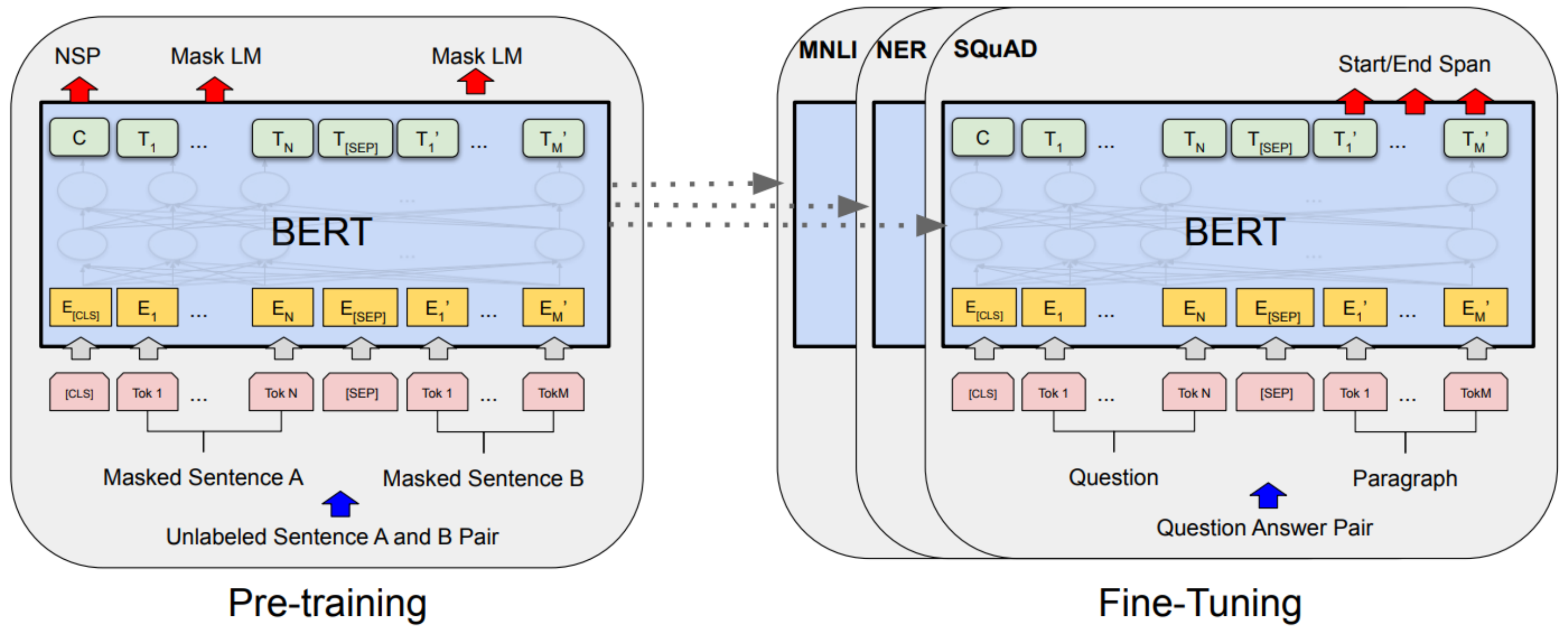


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are 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

BERT Implementation/Code

- google-research/bert: TensorFlow code and pre-trained models for BERT

Difference from Word2vec

- **Out-of-Vocabulary problem**
    - Word2Vec learns embeddings at word level, it can only generate embeddings for words that existed in it's training set (aka it's "vocabulary space"). This is a major drawback to Word2Vec - that it just doesn't support Out-of-Vocabulary words.

    - BERT learns representations at the subword level, so a BERT model will have a smaller vocabulary space than the number of unique words in its training corpus.
    - BERT is able to generate embeddings for words outside of its vocabulary space giving it a near infinite vocabulary.

Difference from Word2vec

- **Embeddings**
  - Word2Vec offers pre-trained word embeddings that anyone can use off-the-shelf.
  - Word2Vec takes a single word as input and outputs a single vector representation of that word.

  - BERT generates contextual embeddings, it takes as input a sequence (usually a sentence) rather than a single word.
  - BERT needs to be shown the context that surrounding words provide before it can generate a word embedding.

Word2Vec vs BERT — Salt Data Labs

Difference from Word2vec

- **Probabilistic/Static vs Contextual Embeddings**
  - Word2Vec offers pre-trained word embeddings that anyone can use off-the-shelf.
  - Word2Vec takes a single word as input and outputs a single vector representation of that word.

  - BERT generates contextual embeddings, it takes as input a sequence (usually a sentence) rather than a single word.
  - BERT needs to be shown the context that surrounding words provide before it can generate a word embedding.

[Word2Vec vs BERT — Salt Data Labs]

Acknowledgments

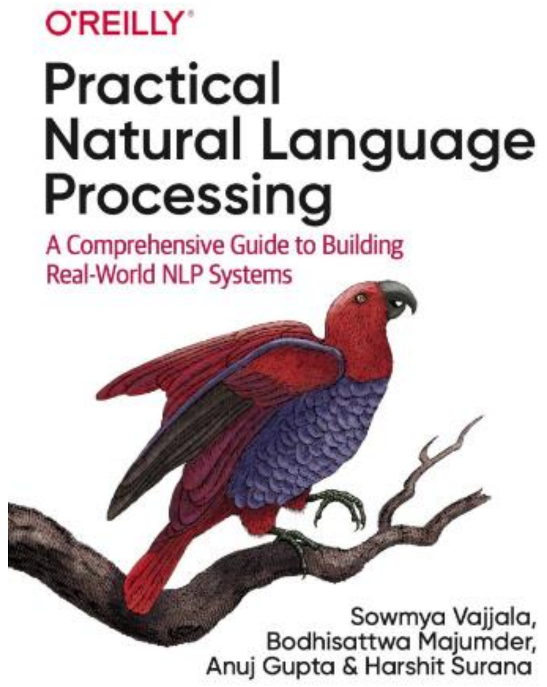
- These slides were adapted from the book SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition
- Some modifications from CS224N presentations and resources found in the WEB by several scholars.

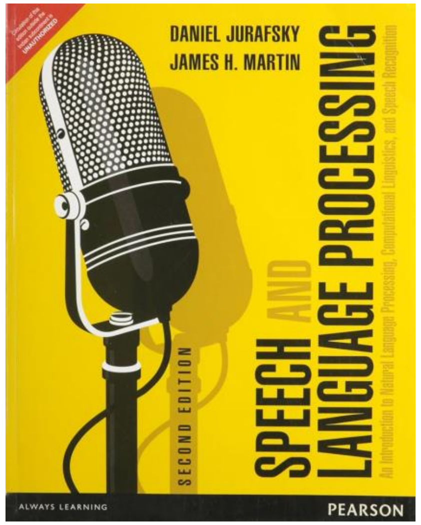
Reference materials

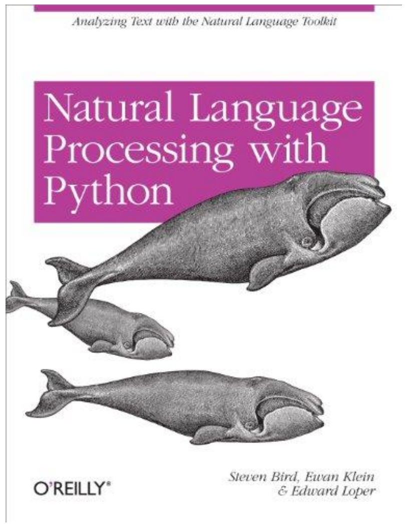
- https://vlanc-lab.github.io/mu-nlp-course/teachings/fall2024-AI-schedule.html

- 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


O'REILLY
Practical Natural Language Processing
A Comprehensive Guide to Building Real-World NLP Systems
Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta & Harshit Surana




Analyzing Text with the Natural Language Toolkit
Natural Language Processing with Python
O'REILLY
Steven Bird, Ewan Klein & Edward Loper