Language Modeling

LMs, tokenization, vocabularies

Younes Samih, David Arps: From Static Embeddings to Transformers. HHU 2022

Language Modeling

- A language model (LM) assigns probability to sequences of words
- In practice, the concrete value does not matter:
  - You don't care that P("*I'm levitating"*)=0.00000035
  - You do care that P("*I'm levitating")* > P("sleeping *levitating*")
- So you often care about maximizing P (the most likely scenarios)
- A LM is trained on some dataset, not on all sentences in a language
- The dataset selection influences the models prediction!

What defines a language model?

Triple of

- LM architecture (e.g. neural architecture)
- Training data (language(s), domain(s))
- **LM task**

Language Modeling Tasks

- Next Word prediction
- Masked Language Modeling (Cloze task)
- Next Sentence Prediction
- …

Next word prediction (NWP)

- Widely used: Recurrent models, GPT*, ngrams
- Good for text generation: Similar to writing/reading/listening
- Why artificially restrict attention to previous words for predictions?

Step	Given	Prediction
1	[BOS]	Never
2		
3		
4		
5		

Next word prediction (NWP)

- Widely used: Recurrent models, GPT*, ngrams
- Good for text generation: Similar to writing/reading/listening
- Why artificially restrict attention to previous words for predictions?

Step	Given	Prediction
1	[BOS]	Never
2	[BOS] Never	gonna
3		
4		
5		

Next word prediction (NWP)

- Widely used: Recurrent models, GPT*, ngrams
- Good for text generation: Similar to writing/reading/listening
- Why artificially restrict attention to previous words for predictions?

Step	Given	Prediction
1	[BOS]	Never
2	[BOS] Never	gonna
3	[BOS] Never gonna	give
4		
5		

Next word prediction (NWP)

- Widely used: Recurrent models, GPT*, ngrams
- Good for text generation: Similar to writing/reading/listening
- Why artificially restrict attention to previous words for predictions?

Step	Given	Prediction
1	[BOS]	Never
2	[BOS] Never	gonna
3	[BOS] Never gonna	give
4	[BOS] Never gonna give	you
5		

Next word prediction (NWP)

- Widely used: Recurrent models, GPT*, ngrams
- Good for text generation: Similar to writing/reading/listening
- Why artificially restrict attention to previous words for predictions?

Step	Given	Prediction
1	[BOS]	Never
2	[BOS] Never	gonna
3	[BOS] Never gonna	give
4	[BOS] Never gonna give	you
5	[BOS] Never gonna give you	up

Masked Language Modeling (MLM)

Given:

“In a [MASK] *deluged by irrelevant* [MASK]*, clarity is power.” (Yuval Noah Harari)*

Predict:

“In a world deluged by irrelevant information, clarity is power.” (Yuval Noah Harari)

- Works really well when sentences are processed at once.
- Details: BERT paper (Devlin et al. 2019)

Next Sentence Prediction (NSP)

Given two sentences:

[CLS] *In a hole in the ground there lived a hobbit.* [SEP] *The hobbit was 4 feet high.* [CLS]

Predict if they occur in the training data one after the other:

True/False

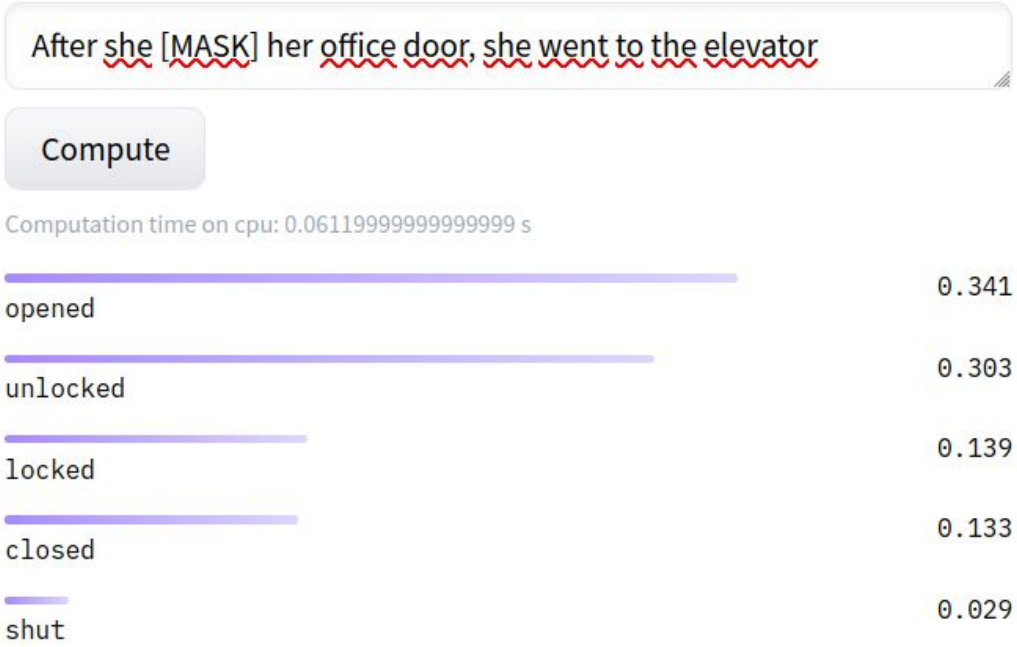
- Used in original BERT paper but is not used that frequently afterwards
- Nice to have for relations between sentences

Autoregressive and Autoencoder LMs

- **Autoregressive**: Every LM prediction depends on previous predictions
- **Autoencoder:** The LM gets a (corrupted) version of a sequence as input, and the goal is to predict/encode this sequence. Example: Masked Language Modeling

Why is Language Modeling so powerful?

- LMs do not require hand-labeled data (self-supervision)
- MLM and NWP combine many domains of linguistic knowledge
  - E.g. morphology, syntax, semantics



Ex. from DistilBERT on https://huggingface.co/distilbert-base-uncased

A bit of “history”: *n*-grams

- pre-neural approach to LM
- method: collect transition frequencies from large corpora
- larger *n* often leads to better results but needs a lot more data
- similar to NWP but the left context is fixed:
  - for bigrams: P(you|I’ve waited here for) ≅ P(you|for)
- **nice**: very fast, model is built by counting from a corpus
- **not nice**: bad at handling rare words, words not seen in training
- **not nice**: can’t keep track of context larger than *n* words
  - Old smartphone: *“Hello the morning the morning the morning…”*

Vocabulary: How to turn text into vectors

- Method 1: Word-based

hello	12	45	43	26	78	532	...
there	43	25	778	43	53	78	...
texas	34	56	23	12	56	74	...
world	342	54	23	5	7	423	...
...	...						

- Problem 1: Not good at rare words, unknown words
- Problem 2: related word forms are treated separately
    - *inform, informs, informing, information* have shared meaning components. But vectors are learned independently

Method 2: Character-based

a	[0.1 0.8 0.4 0.7 …]
b	[0.4 0.9 -0.3 0.1 …]
c	[0.3 -0.2 0.1 -0.9 …]
…	

- Each character gets its own vector
- Small vocabulary: only ~100 characters/vectors
- Nice: there are no unknown characters
- Not nice: Character vectors need to store a lot more context information
- Example: *Y,**o**,u, ,s,h,**o**,u,l,d*…
  - The static vector for “*o”* is always the same and needs to store information about all contexts/words “*o”* occurs in

Best of both worlds: Subword-based methods

- Basic idea: The most frequent subsequences get their own vector

	Text	Alphabet	Most freq. bigram
1.	d,i,e, ,r,**e,n**,t,n,e,r, ,e,s,s,**e,n**, ,d,i,e, ,**e,n**,t,**e,n**	d,e,i,n,s,t	e,n: 4
2.	d,i,e, ,r,EN,t,n,e,r, ,e,s,s,EN, ,d,i,e, ,EN,t,EN	d,e,i,n,s,t,EN	EN,t: 2
3.	d,i,e, ,r,ENT,n,e,r, ,e,s,s,EN, ,d,i,e, ,ENT,EN	d,e,i,n,s,t,EN,ENT	d,i: 2
4.	DI,e, ,r,ENT,n,e,r, ,e,s,s,EN, DI,e, ,ENT,EN	d,e,n,s,t,EN,ENT,DI	DI,e: 2
5	DIE, ,r,ENT,n,e,r, ,e,s,s,EN, ,DIE, ,ENT,EN	d,e,n,s,t,EN,ENT,DIE	

Best of both worlds: Subword-based methods

- Basic idea: The most frequent subsequences get their own vector
- Typical alphabet size: 30K+ vocabulary items
- Nice:
  - Frequent words get their own vector
  - capture related meaning of rare words, related words
  - no unknown words
- Not nice: Arbitrary behavior for some systematic tokens (names, numbers)

Most common subword-based methods

- **WordPiece**: (character level)
  - The vocabulary is initialized with individual characters in the language,
  - then the most frequent combinations of symbols in the vocabulary are iteratively added to the vocabulary.
- **BPE**: (Byte Pair Encoding)
  - data compression algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data.
  - BPE ensures that the most common words are represented in the vocabulary as a single token
  - Rare words are broken down into subword tokens
  - This is in agreement with what a subword-based tokenization algorithm does.

Practical session: Vocabulary

Complete the tasks in the notebook `tokenization.ipynb` on colab or on your computer.