Language Modeling

ML4SE

Denis Litvinov

November 22, 2021

Table of Contents

Problem Statement

Probability of a sequence of tokens w_i:

$$P(w_1, .., w_T) = P(w_1)P(w_2|w_1)...P(w_T|w_1, ..., w_{T-1})$$

Perplexity

Common quality metric language modeling is perplexity (smaller is the better):

$$Q(w_1, .., w_T) = P(w_1, .., w_T)^{-\frac{1}{T}}$$

Perplexity of a corpus of text:
As

$$C = s_1, .., s_m$$

then

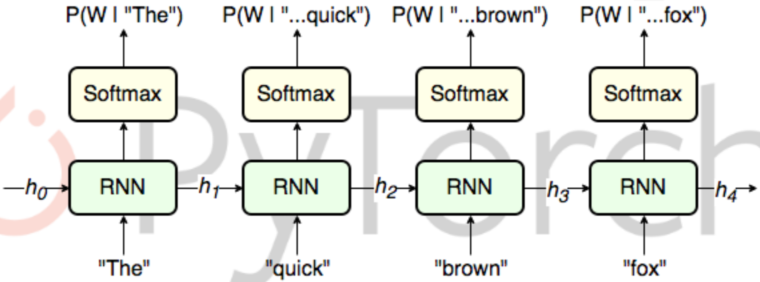
$$P(c) = \prod_{i=1}^{m} P(s_i)$$

$$Q(C) = (\prod_{i=1}^{m} P(s_i))^{-\frac{1}{m}} = 2^{\log_2(\prod_{i=1}^{m} P(s_i))^{-\frac{1}{m}}} =$$

$$= 2^{-\frac{1}{m}\log_2(\prod_{i=1}^{m} P(s_i))} = 2^{-\frac{1}{m}\sum_{i=1}^{m} log_2 P(s_i)}$$

Autoregressive Model

In general, model is autoregressive if it predicts future values based on past values.
RNN is a autoregressive model



N-gram Language Models

N-gram language model = (finite horizon):

$$P(w_t|w_1, ..., w_{t-1}) = P(w_t|w_{t-n+1}, ..., w_{t-1})$$

$$P(w_1, .., w_T) = \prod_{t=1}^{T} P(w_t|w_{t-n+1}, ..., w_{t-1})$$

How to estimate $P(w_t|w_{t-n+1}, ..., w_{t-1})$?

N-gram Language Models

$$P(w_t|w_{t-n+1},...,w_{t-1}) = \frac{count(w_{t-n+1},...,w_{t-1},w_t)}{\sum_{\hat{w}} count(w_{t-n+1},...,w_{t-1},\hat{w})}$$

OR

$$P(w_t|w_{t-n+1},...,w_{t-1}) = \frac{count(w_{t-n+1},...,w_{t-1},w_t)^{\frac{1}{\tau}}}{\sum_{\hat{w}} count(w_{t-n+1},...,w_{t-1},\hat{w})^{\frac{1}{\tau}}}$$

where *tau* is called temperature. Higher temperature means more flat probability distribution.
Problems with naive approach?

Laplace smoothing

$$P(w_t|w_{t-n+1},...,w_{t-1}) = \frac{count(w_{t-n+1},...,w_{t-1},w_t) + \delta}{\sum_{\hat{w}}[count(w_{t-n+1},...,w_{t-1},\hat{w}) + \delta]}$$

where δ accounts for missing n-grams

Tokenization

Text

“The cat sat on the mat.”

↓

Tokens

“the”, “cat”, “sat”, “on”, “the”, “mat”, “.”

What are the problems with word tokenization?

Special Tokens

We usually add special tokens:

- $< bos >$ begin of sentence. The first word in new generated sequence is conditioned on $< bos >$
- $< eos >$ end of sentence.
- $< pad >$ padding token. Usually pad_token_id = 0 for convinience
- $< unk >$ unknown token for out-of-vocabulary words

```
'<bos>', 'the', 'cat', 'sat', 'on', 'a', 'mat', '.', '<eos>'

'<bos>', 'I', 'see', 'a', 'dog', 'called', '<unk>', '<eos>',
'<pad>'
```

Byte Pair Encoding

Problem of text tokenization:

1. If tokenized per word => len(sequence) is moderate, but vocab_size » 1
2. If tokenized per chars => vocab_size is moderate, but len(sequence) » 1

Long sequences mean long training time,
Big vocab size means more weighs in the models and potential overfitting.
BPE, among others, is the "middle ground", trying to balance sequence length and vocab size by merging frequent n-grams into new tokens.

Byte Pair Encoding

```
AABCDCEABFABCB => vocab_size=6 => AB=X
AXCDCEXFXCB => vocab_size=7 => XC=Y
AYDCEXFYB => vocab_size=8
```

sketch for algorithm:
while (vocab) < max_vocab_size:

1. count bigram frequencies
2. take most common bigram w_i, w_j
3. substitute it with a new token in the text
4. update vocab

WordPiece and Others

from huggingface docs:

```
WordPiece ... is very similar to BPE.
In contrast to BPE, WordPiece does not choose the most frequent
symbol pair, but the one that maximizes the likelihood of the
training data once added to the vocabulary.
```

Tokens A and B are merged together into a new token AB if $\frac{p(''AB'')}{p('A')p('B')}$ is greater than for any other symbol pair.

Other popular tokenization algorithms:

- Unigram
- SentencePiece
- BPE Dropout
- ...

Problem Statement

Inference for Language Models: we would like to generate a new sequence of tokens from conditioned on some input.
And usually we want the generated sequence to be the most probable for the input, and at the same time diverse enough not to collapse into the same mode.

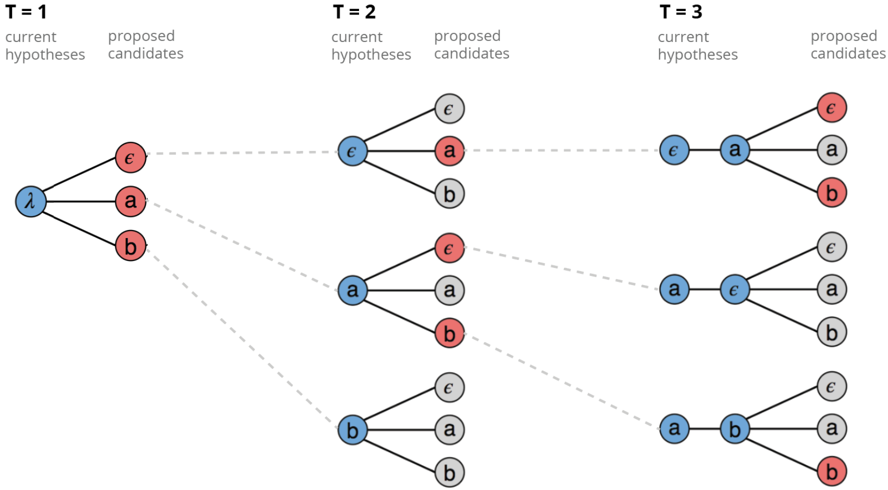
```
What's the time? -> I don't know.
How are you? -> I don't know.
...
```

Argmax

$$w_t = argmaxP(w_t | w_{t-n+1}, ..., w_{t-1})$$

However, greedy selecting the tokens not necessarily mean producing the most probable sequence.

Beam search: greedy, but smarter



Standard beam search algorithm with an output

Sampling

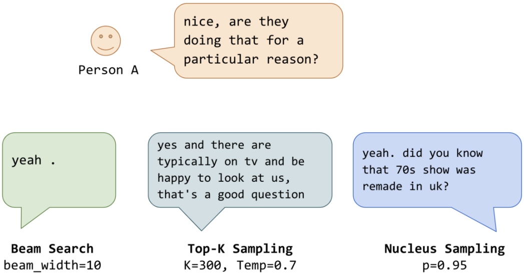
Sample token from distribution

$$w_t \sim P(w_t | w_{t-n+1}, ..., w_{t-1})$$

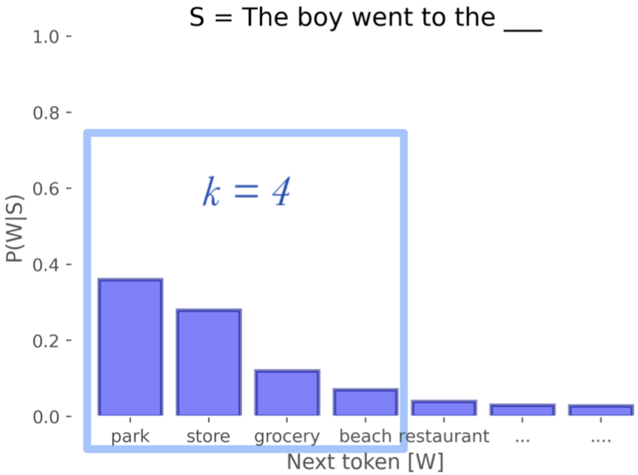
With temperature, controlling "the randomness" of generated sequence

$$w_t \sim P(w_t | w_{t-n+1}, ..., w_{t-1}; \tau)$$

using softmax with temperature $p(w = i) = \frac{e^{z_i / \tau}}{\sum_j e^{z_j / \tau}}$

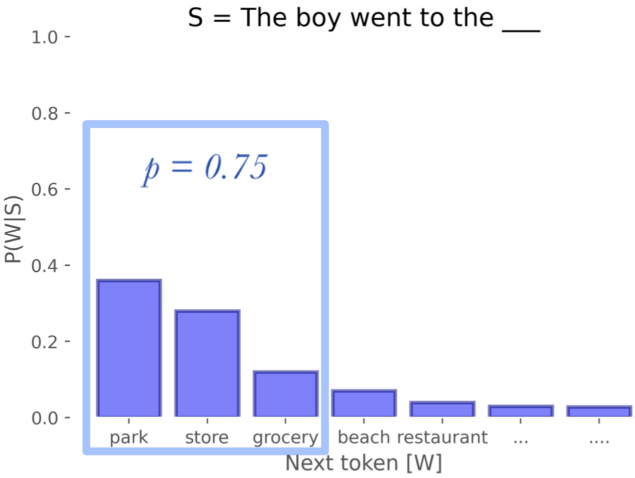


Top-k


S = The boy went to the ___
1.0
0.8
0.6
0.4
0.2
0.0
P(W|S)
k = 4
park
store
grocery
beach
restaurant
...
....
Next token [W]

Top-p

Nucleus sampling [https://arxiv.org/abs/1904.09751]



About Overfitting

- We usually consider Language Modeling as **self-supervised task**.
- Usually we don't have limit on a dataset size: all texts are at our disposal! Hence big transformers dominate the area.
- That's why we usually don't observe overfitting in language models.
- Though, if we want to calculate test loss properly, random dataset splitting is not enough. We need to check on the percentage of overlapping n-grams (n=5,6,7..) in the documents from train and test set.