9. Language Models

Statistical language models

Problem formulation

Goal: Assign probabilities to a sequence of tokens, e.g.,

- p(the red fox jumped) >> p(the green fox jumped)
- p(colorless green ideas sleep furiously) >> p(furiously sleep ideas green colorless)

Formulation:

- Vocabulary: a set of symbols ν
- Sentence: a finite sequence over the vocabulary $x_1, x_2, ..., x_n \in \nu^n$ where $n \geq 0$
- The set of all sentences (of varying lengths): ν^*
- Assign a probability $p(x)$ to all sentences $x \in \nu^*$

A naive solution

- Training data: a set of N sentences
- Modeling: use a multinomial distribution as our language model $p_s(x) = \frac{\text{count(x)}}{N}$
- Is it a good LM?
  - Most sentences only occur once —sparsity issue
  - Need to restrict the model

Simplification 1: sentence to tokens

Decompose the joint probability using the probability chain rule:

$p(x) = p(x_1, ..., x_n) = p(x_1)p(x_2|x_1)...p(x_n|x_1, ..., x_{n-1})$

- Problem reduced to modeling conditional token probabilities: the red fox → jumped
- The left-to-right decomposition is also called an **autoregressive model**
- This is a classification problem we have seen
- But there is still a large number of contexts

Simplification 2: limited context

Reduce dependence on context by the **Markov assumption**:

- First-order Markov model
  $p(x_i|x_1, ..., x_{i-1}) = p(x_i|x_{i-1})$
  $p(x) = \prod_{i=1}^{n} p(x_i|x_{i-1})$
- Number of contexts: $|\nu|$

- Number of parameters: $|\nu|^2$

Model sequences of variable lengths

Assume each sequence starts with a special start symbol: $x_0 = *$

Assume that all sequences end with a stop symbol STOP, e.g. p(the, fox, jumped, STOP) = p(the | $*$) p(fox | the) p(jumped | fox) p(STOP | jumped).

Without the stop symbol, shorter sentences will always have greater probability.

N-gram LM

- Unigram language model (no context):

  $p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i)$
- Bigram language model ($x_0 = *$):

  $p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i|x_{i-1})$
- n-gram language model:

  $p(x_1, ..., x_m) = \prod_{i=1}^{m} p(x_i|x_{i-n+1}, ..., x_{i-1})$

Parameter estimation

Maximum likelihood estimation over a corpus (a set of sentences):

- Unigram LM

  $p_{\mathrm{MLE}}(x) = \frac{\mathrm{count}(w)}{\sum_{w\in\nu}\mathrm{count}(w)}$
- Bigram LM

  $p_{\mathrm{MLE}}(w|w') = \frac{\mathrm{count}(w,w')}{\sum_{w\in\nu}\mathrm{count}(w,w')}$
- In general, for n-gram LM,

  $p_{\mathrm{MLE}}(w|c) = \frac{\mathrm{count}(w,c)}{\sum_{w\in\nu}\mathrm{count}(w,c)}$

  where $c \in \nu_{n-1}$

Generating text from an n-gram model

1. Initial condition: context = $*$
2. Iterate until next word is STOP:
   a. next word ~ p(· | context[: −(n − 1)])
   b. context ← context + next word

Perplexity

What is the loss function for learning language models?

Held-out likelihood on test data D(negative test loss):

$l(D) = \sum_{i=1}^{|D|} \log p_\theta(x_i | x_{1:i-1})$

Perplexity: $\mathrm{PPL}(D) = 2^{-\frac{l(D)}{|D|}}$

Interpretation: a model of perplexity k predicts the next word by throwing a fair k-sided die.

Summary

Language models: assign probabilities to sentences

N-gram language models:

- Assume each word only conditions on the previous n − 1 words
- MLE estimate: counting n-grams in the training corpus

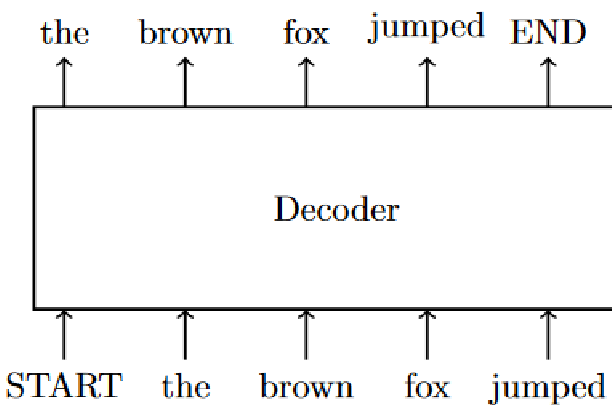
Evaluation by held-out perplexity: how much probability mass does the model assign to unseen text

Challenges:

- **Generalization**: sentences containing unseen n-grams have zero probability
- Much research in n-gram LM is dedicated to **smoothing** methods that allocate probability mass to unseen n-grams

Neural language models

Neural networks solve the generalization problem in n-gram LMs.

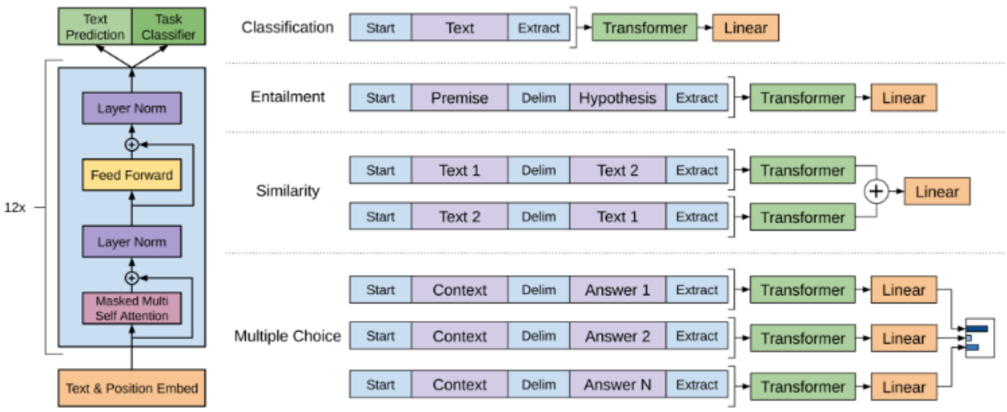


- A decoder-only autoregressive neural language model
- Decoder can be an RNN or a transformer (with causal masking)

Significant improvement in held-out perplexity given similar model sizes.

Recap: language modeling as pretraining

Generative Pre-Training (GPT)



- Pretrained on Bookcorpus; 12 layer decoder-only transformer; learned position embedding; GELU activation
- **Auxiliary LM objective** during finetuning: $L_{\text{task}} + \lambda L_{\text{LM}}$

Ablation studies

- Auxiliary objective only helps on larger datasets
- Pretrained transformer > pretrained LSTM (single layer) > non-pretrained transformer

Zero-shot behaviors

Key insight: if the model has learned to understand language through predicting next words, it should be able to perform these tasks **without finetuning**

Heuristics for zero-shot prediction:

- Sentiment classification: [example] + very + {positive, negative}   (prompting)
- Linguistic acceptability: thresholding on log probabilities
- Multiple choice: predicting the answer with the highest log probabilities

Learning dynamics: zero-shot performance increases during pretraining