Lec 3 Language Modelling (LM)

The task

Discriminative model — $P(Y|X)$ — conditional

lang model →

Generative model — $P(X)$ standalone prob — $P(X, Y)$

Do with LMs

- scoring
  - "Jane went to the store" → high score
  - "store to Jane went the" → low
- generate
  - sample by $\tilde{x} \sim P(X)$
  - find most probable sentence in prob space

Applying the LM

→ QA — as prediction — see open-llm for rankings on these

"Q: where is CMU located ? A: Pittsburgh." → high score
"Q: where is CMU located ? A: Birmingham." → low score ?

→ Prompt question & gen continuation

"Q: where is CMU located ? A: ____

→ Categorise

"This is great! Star rating ____
"This is great! Star rating 5" → score

→ Correct grammar

replace words, paraphrase etc.

Autoregressive LMs

← GPT, LLAMA, etc.

$$P(X) = \prod_{i=1}^{I} P(x_i \mid x_1, \dots, x_{i-1})$$

x_i: next tkn; $x_1, \dots, x_{i-1}$: context

So the problem is predicting $P(x_i \mid x_1, \dots, x_{i-1})$

Why not just do $P(X)$?
→ the input space becomes $|V|^n$ where V is vocab and n is sentence length

Alternative : [MASK] LMs.
problems : - doesn't give prob of sequence
- hard to do generation

Alternative : energy-based
very advanced

Unigram LMs

↳ potential sols

Assume $x_i \perp x_1, \dots, x_{i-1}$
$\Rightarrow P(x_i) \approx P(x_i \mid x_1, \dots, x_{i-1})$

$$P_{MLE}(x_i) = \frac{c_{train}(x_i)}{\sum_{\hat{x}} c_{train}(\hat{x})}$$

(count)

Problems - $P(\langle \text{unknown word} \rangle) = 0$
↳ segment word
↳ model unknown words by characters

Log space multiplication

$$P(X) = \prod_{i=1}^{|X|} P(x_i) \iff \log P(X) = \sum_{i=1}^{|X|} \log P(x_i)$$

↑ tends to underflow ↑ better

Notation : $\theta_{x_i} := \log P(x_i)$ so unigram has $|V|$ params

n-gram model

Generalised

$$P_{LM}(x_i \mid x_{i-n-1}, \dots, x_{i-1}) := \frac{c(x_{i-n-1}, \dots, x_i)}{c(x_{i-n-1}, \dots, x_{i-1})}$$

Problems - Unseen context seq ⇒ prob is 0
↳ fall back to shorter context
interpolate with shorter gram
so we ensemble different-gram models
$P(x_i \mid x_{i-n+1}, \dots, x_{i-1}) = \lambda P_{3\text{-gram}} \dots$
$+ (1-\lambda) P_{4\text{-gram}}$ etc.

Idea of ensembling

↳ Goodman 1998 smoothing — (for count-based models)
different varieties:
- Additive / Dirichlet
- Discounting
- Kneser-Ney ← state-of-the-art when n-gram was popular;

fallback		bigram		
the	0.5	the the	0	
box	0.3	the box	0 → 1	0.33
is	0.2	the is	0	

↑ if more evidence for these, we weight them more

3 solutions for individual issues but combining is hard and ppl went neural instead

- Doesn't share among similar words
- Not condition on interleaving word
- Long distance depencies not captured

n-gram	neural
fast	usually better performance
learns low-freq better	
scalable	

{ kenlm for n-gram

LM evaluation

Log-likelihood (LL)

$$LL(X_{test}) = \sum_{x \in X_{test}} \log P(x)$$

WLL (per word LL)

LL norm. by # words

NLL

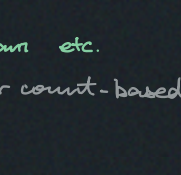
LL but negative

Entropy

$$H(X_{test}) = \frac{1}{\sum_{x \in X_{test}} |x|} - \sum_{x \in X_{test}} \log_2 P(x)$$

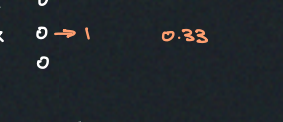
{ related to info theory? prob model can be used to do compression

Ex. vocab dist :



encoding : A ↦ 0
B ↦ 10
C ↦ 110

to make this 2-gram, subdivide each



Perplexity $T := X_{test}$

$$PPL(T) = 2^{H(T)} \approx e^{-WLL(T)}$$

Intuitively : # of samples needed to sample (i.e. guess) from probs to get next tkn right

Note : make sure to divide to account for vocab size etc. before comparing

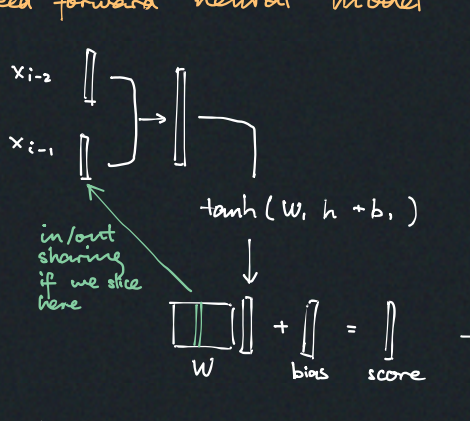
Featurised Log-linear Model

(SGD-optimised)

1. Calculate some feature of context
2. Predict probs for next tkn
3. Gradient descent

Solves ?:
× Doesn't share among similar words
✓ Not condition on interleaving word
× Long distance depencies not captured

Feed forward neural model



Solves ?:
✓ Doesn't share among similar words
✓ Not condition on interleaving word
× Long distance depencies not captured

Other LM desiderata

▸ Calibration : "model knows when it knows"
↳ want $P(\text{a is correct} \mid P_{model} = p) = p$
i.e. if model says 7% right then 7% right
↳ in practice, exact probs are hard, so use buckets
↳ note : calibration & accuracy not necessary high correlation

To compute prob of output i.e. confidence
- prob of answer
  - if single answer → easy
  - if multi answer → ?
- prob of answer + paraphrases of the answer
  var. sum multiple acceptable answer
- sample multiple outputs, count # of correct answers
  - useful if LM doesn't return prob
  - useful for chain of reasoning answers
- ask the model what it thinks

↳ Xiong + 2023 good comparison paper

▸ Efficiency
- Parameter count
- Memory usage
  ↳ peak
  ↳ model load
- Latency (≈ encode time)
  ↳ until first token
  ↳ until finished generating
- Throughput

Efficient NN implementation

- Batching
  ↳ batch size estimate can be # seqs but # tkns recommended
- GPU
- Code
  - Avoid repeating operations
  - Reduce operations
  - Reduce CPU ⇔ GPU movements