CS371N Lecture 9
Language Modeling

Announcements

- A1 + response back today
- A2 due Thurs
- Bias response due Thurs
- A3 out Thurs

Recap Binary classification
→ multiclass
→ neural multiclass

How do we tackle "real" problems?
↳ some of these involve generation
↳ some problems are just much harder classification problems

Today
- Language modeling
  - N-gram LMs
  - Neural LMs
  - Next time: Transformer

Language Modeling Model distribution $P(\overline{w})$

"Autocomplete" / predictive text

Predict the next word w_i after a prefix $w_1 \dots w_{i-1}$

$$P(w_i \mid w_1 \dots w_{i-1})$$

$w_i \in$ vocab V

Prob of a sentence of n words:

$$\prod_{i=1}^{n} P(w_i \mid w_1 \dots w_{i-1}) = P(\overline{w})$$

What can these do?

- Anomaly detection: recognize something "weird"
- Train on one author & recognize their style
- Grammatical error correction: wrong sentence $\overline{w}'$ has lower prob than $\overline{w}$

N-gram Language Modeling

In general: $P(\overline{w}) = P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2) \cdots P(w_{10} | w_1 w_2 \ldots w_9)$

n-gram LM: only look at previous n-1 words

$$P(\overline{w}) = \prod_{i=1}^{l} P(w_i | w_{i-n+1} \ldots w_{i-1})$$

Bigram (2-gram):

$$P(\bar{w}) = P(w_1)P(w_2|w_1)\underbrace{P(w_3|w_2)}_{\substack{\text{probability of next}\\ \text{word given prev.}\\ \text{word}}} P(w_4|w_3)\ldots$$

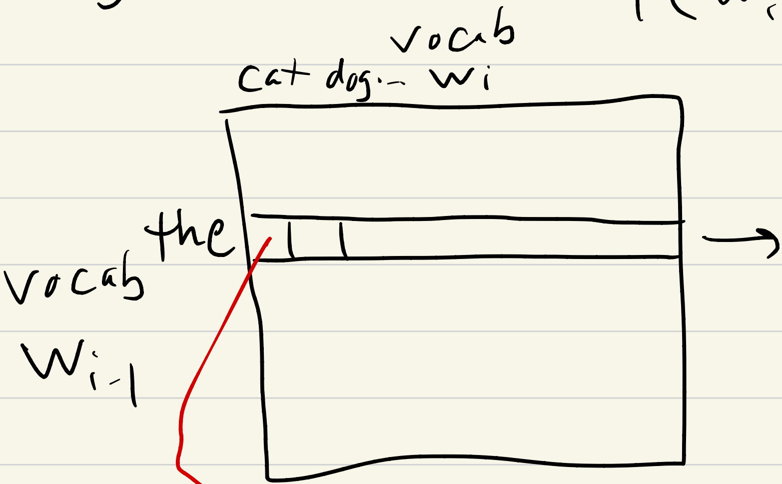
Like skip-gram!

Count-based n-gram LMs

No neural nets

Explicitly represent each n-gram probability with a categorical distr. estimated from data next | curr

2-gram LM: $P(w_i \mid w_{i-1})$



each row sums to 1

$P(\text{cat} \mid \text{the})$: how likely to have cat after the?

Ex I saw the dog ____

$n=2$ (bigram): $P(w_i \mid \text{dog})$

What comes next? Verb (run), noun (treat), ...

Bigram captures these okay?

$\Rightarrow$ ran ✗ (wrong conjugation)

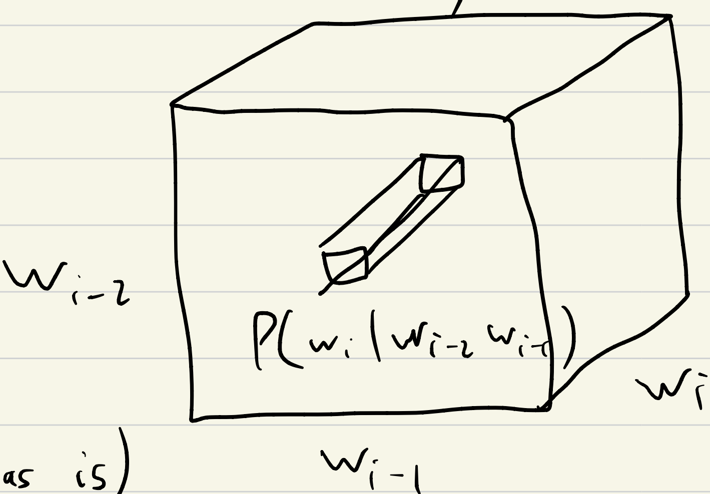
The capital of Texas is ______

Austin   pretty   an... awesome place

$P(w_i \mid \text{is})$: not a good distribution

Trigrams: $P(w_i \mid w_{i-2}\, w_{i-1})$

Parameters:



$P(w \mid \text{Texas is})$

Really need 5-gram here

Parameter estimation

Count + normalize over a large corpus

the cat
the cat
the dog
the snake

corpus

(bigram)

$P(\text{cat} \mid \text{the}) = \frac{2}{4}$

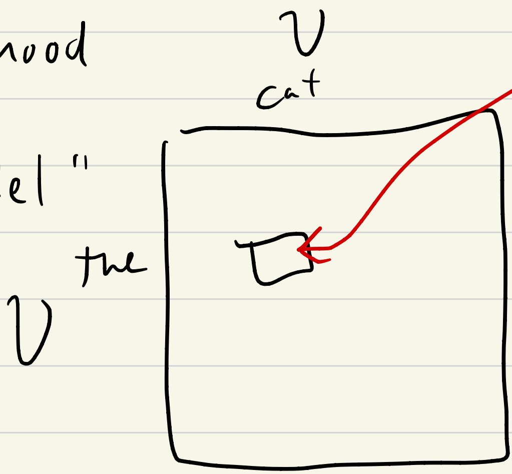
$P(\text{dog} \mid \text{the}) = \frac{1}{4}$

$P(\text{snake} \mid \text{the}) = \frac{1}{4}$

$P(\text{the} \mid \text{the}) = 0$

These parameters maximize dataset log likelihood

Our "model"



matrix instead of $\overline{w}$

Smoothing

For $n \geq 3$, lots of 0s in params

$P(\text{Mavil love going to}) = 0$?

If not observed

Implementing n-gram models in reality involves dealing with this

Backoff, discounting

trigram

Suppose we have $P_3(w_i | w_{i-2}\, w_{i-1})$ estimated from data with MLE

$$P_{3,\text{discounted}}(w_i | w_{i-2}\, w_{i-1})$$

$$= \lambda P_3(w_i | w_{i-2}\, w_{i-1})$$

$$+ (1-\lambda) P_{2,\text{discounted}}(w_i | w_{i-1})$$

Interpolate between bigram + trigram

$$P_{2,\text{discounted}} = \lambda P_2 + (1-\lambda) P_1$$

P_1 has >0 prob for every word

$\Rightarrow$ no more zeroes

<u>Neural Language Modeling</u>

$P(w_i \mid w_1 \dots w_{i-1}) \Rightarrow$ model w/ a neural net

Choices: (1) Do we do n-gram modeling and restrict to n-1 context words? OR look at whole sequence?

(2) What architecture?

Represent each token with vector

$$w_1 \quad w_2 \quad w_3 \quad w_4 \rightarrow w_5$$

$$\bar{v}_1 \quad \bar{v}_2 \quad \bar{v}_3 \quad \bar{v}_4$$

<u>DAN</u> $\bar{v} = \frac{1}{n} \sum_{i=1}^{n} \bar{v}_i$

$$P(w_i) = \text{FFNN}(\bar{v})$$

n-gram

<u>FFNN</u> $\bar{v} = \text{concat}(\underbrace{v_{i-n+1} \cdots v_{i-1}}_{n-1 \text{ words}})$

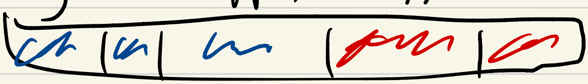
$$P(w_i) = \text{FFNN}(\bar{v})$$

(position-sensitive: model the last token vs. second-to-last differently)

	DAN	FFNN
Advantages	– Dim of $\bar{v}$ is fixed – Considers whole context	– Considers n-1 words in order
Disadvantages	– Ignores order	– No translational invariance

The dog is wagging its ____

The dog is happily wagging its ____



↓

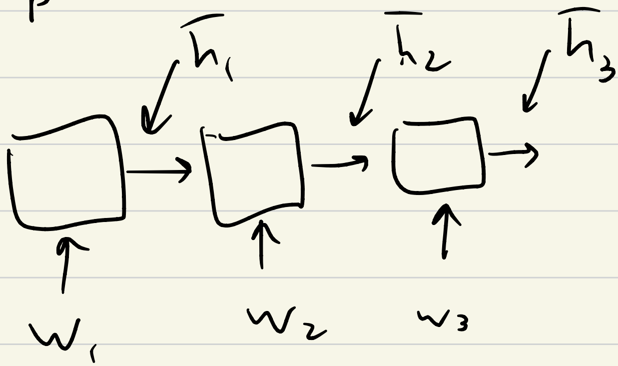
multiply by V $\quad 5d_1 \times d_2$ matrix

<u>Solutions</u> (1) Transformer (next time)

(2) RNN: recurrent neural network

RNN forms a state vector from processing n words

Updates it with the n+1 word



All $\square$ here shared parameters

<u>Problem</u> $O(n)$ sequential computation

"forget" early things in the sequence