NATURAL LANGUAGE PROCESSING

WITH DEEP LEARNING

University of Guilan

Lecturer: Javad PourMostafa

NLP981

BOW VECTORIZATION

- **BOW is a sum of sparse one-hot-encoded vectors corresponding to each particular word.**

	I	Like	Play	Tennis	Outside	Enjoy
I	1	0	0	0	0	0
+						
like	0	1	0	0	0	0
+						
tennis	0	0	0	1	0	0
=						
I like tennis	1	1	0	1	0	0

NEURAL VECTORIZATION

- Dense-based method
- Less columns (about 300)
- Pre-trained and online NN
- A good example of pre-trained is **word2vec embedding**
- Word2vec property:
  - **Words that have similar context tend to have collinear vectors**

	x_1	x_2	x_3
I	**0.4**	**0.1**	**0.3**
		+	
like	**0.2**	**-0.1**	**0.2**
		+	
tennis	**-0.9**	**0.7**	**0.4**
		=	
	-0.3	**0.7**	**0.9**

1-D CONVOLUTIONAL LAYER

- How do we make 2-grams?
- **Don't have Word2vec embedding for token pairs**
- **Use convolutional layer**

	x_1	x_2	x_3
I	**0.4**	**0.1**	**0.3**
like	**0.2**	**-0.1**	**0.2**
tennis	**-0.9**	**0.7**	**0.4**

Convolutional kernel

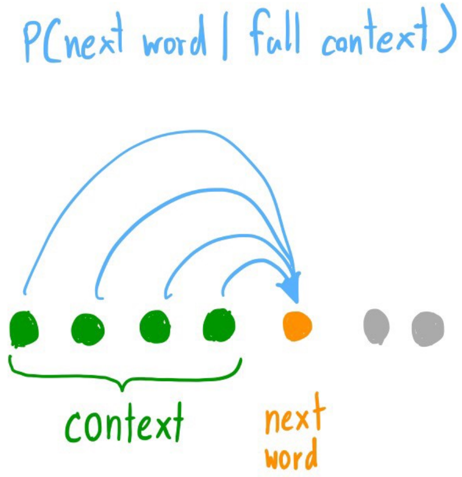
0.3	**0.1**	**0.3**
0.2	**-0.1**	**0.2**

→ **0.31**

Note:
this conv filter is highly activated for 2-grams entitled "**preferences in racket-based sports**"

LANGUAGE MODELING

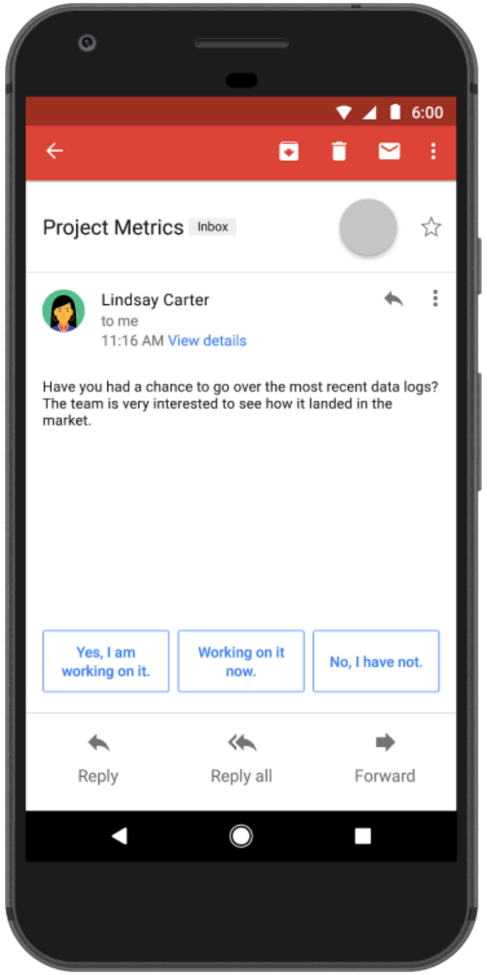
- This is the ….
  - **The house**
  - **Rat**
  - **Did**
  - **malt**



- The probability of the next word?
  - **P(the house | this is the) = ?**

WHERE DO WE NEED LANGUAGE MODELING?

- Smart Reply
- Suggestion in messengers
- Spelling correction
- Machine translation
- Speech recognition
- Handwriting recognition
- …



TOY CORPUS, 4-GRAMS

- This is the house that Jack built
- This is the malt
- That lay in the house that jack built
- This is the rat
- That ate the malt
- That lay in the house that jack built
- This is the cat
- That killed the rat
- That ate the malt
- That lay in the house that jack built

* $P(house \mid this\ is\ the) =$
* $\frac{C(this\ is\ the\ house)}{C(this\ is\ the\ \ldots)} =$
* $\frac{1}{4}$

TOY CORPUS, BI-GRAMS

- This is the house that Jack built
- This is the malt
- That lay in the house that jack built
- This is the rat
- That ate the malt
- That lay in the house that jack built
- This is the cat
- That killed the rat
- That ate the malt
- That lay in the house that jack built

- $P(Jack \mid that) =$
- $\frac{C(that\ Jack)}{C(that \ldots)} =$
- $\frac{4}{10}$

PROBABILITY OF A SEQUENCE OF WORDS

- $W = (w_1, w_2, w_3, \ldots, w_n)$, $P(w_5 \mid w_1, w_2, w_3, w_4)$
- Chain Rule:

$$\mathbf{P}(\boldsymbol{w_1^n}) = \boldsymbol{P}(\boldsymbol{w_1})\, \boldsymbol{P}(\boldsymbol{w_2} \mid \boldsymbol{w_1})\, \mathbf{P}(\boldsymbol{w_3} \mid \boldsymbol{w_1^2}) \ldots \boldsymbol{P}(\boldsymbol{w_n} \mid \boldsymbol{w_1^{n-1}}) = \prod_{k=1}^{n} \boldsymbol{P}(\boldsymbol{w_k} \mid \boldsymbol{w_1^{k-1}})$$

- Example:

$$P(\text{“}Its\ water\ is\ so\ transparent\text{”}) =$$

$$P(its) \times$$

$$P(water \mid its) \times$$

$$P(is \mid its\ water) \times$$

$$P(so \mid its\ water\ is) \times$$

$$P\ (transparent \mid its\ water\ is\ so)$$

PROBABILITY OF A SEQUENCE OF WORDS (CONT'D)

- Last term is too long, how to deal with that?!
- Solution?
  - Markov Assumption and intuition of the n-gram model
- "According to Markov property , we can predict the probability of some future state without looking too far in the past".

$$p(w_i \mid w_1 \dots w_{i-1}) = p(w_i \mid w_{i-n+1} \dots w_{i-1})$$

- Should not care about all the history!
- So Just approximate the history by the last few words

BIGRAM MODEL

- Approximates by using **only** the conditional probability of the **preceding word**
- **E.g.**
- $P(the \mid Walden\ Pond's\ water\ is\ so\ transparent\ that) \approx P(the \mid that)$

$$P(W_n \mid W_1^{n-1}) \approx P(W_n \mid W_{n-1})$$

- Main Idea: Markov assumption

GENERALIZE THE BIGRAM TO N-GRAM

- We can generalize bigram to the trigram
  - **Which looks two words into the past**
- Thus to the n-gram
  - **Which looks n-1 words into the past**
  - Thus the **general** equation

$$P(W_n | W_1^{n-1}) \approx P(W_n | W_{n-N+1}^{n-1})$$

- **We can compute the probability of a complete word sequence (bigram model)**

$$P(W_1^n) \approx \prod_{k=1}^{n} P(W_k \mid W_{k-1})$$

MAXIMUM LIKELIHOOD ESTIMATION (MLE)

- How do we estimate these bigram or n-gram probabilities?
  - By **MLE**

1. **Get the MLE estimate** for the parameters of an n-gram model
   - How? **Counts from corpus**
2. **Normalizing** the counts
   - Lie between **0 and 1**, How?
   - **Dividing by some total count**

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$ For bigram model

EXAMPLE OF BIGRAM USING MINI-TRAINING CORPUS

- <S> I am Sam </S>
- <S> Sam I am </S>
- <S> I do not like green eggs and ham </S>
- Add fake tokens (<S> and </S>)
  - <S> gives us the bigram context of the first word.
  - </S> we need a true probability distribution.
- Some bigram probabilities (**relative frequency**):

$$P(\texttt{I}|\texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam}|\texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am}|\texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>}|\texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam}|\texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do}|\texttt{I}) = \tfrac{1}{3} = .33$$

GENERATIVE MODEL

- An N-gram model can be seen as a probabilistic **automata** for generating sentences.
- How?
  - Initialize sentence with N–1 <s> symbols
  - Until </s> is generated do:

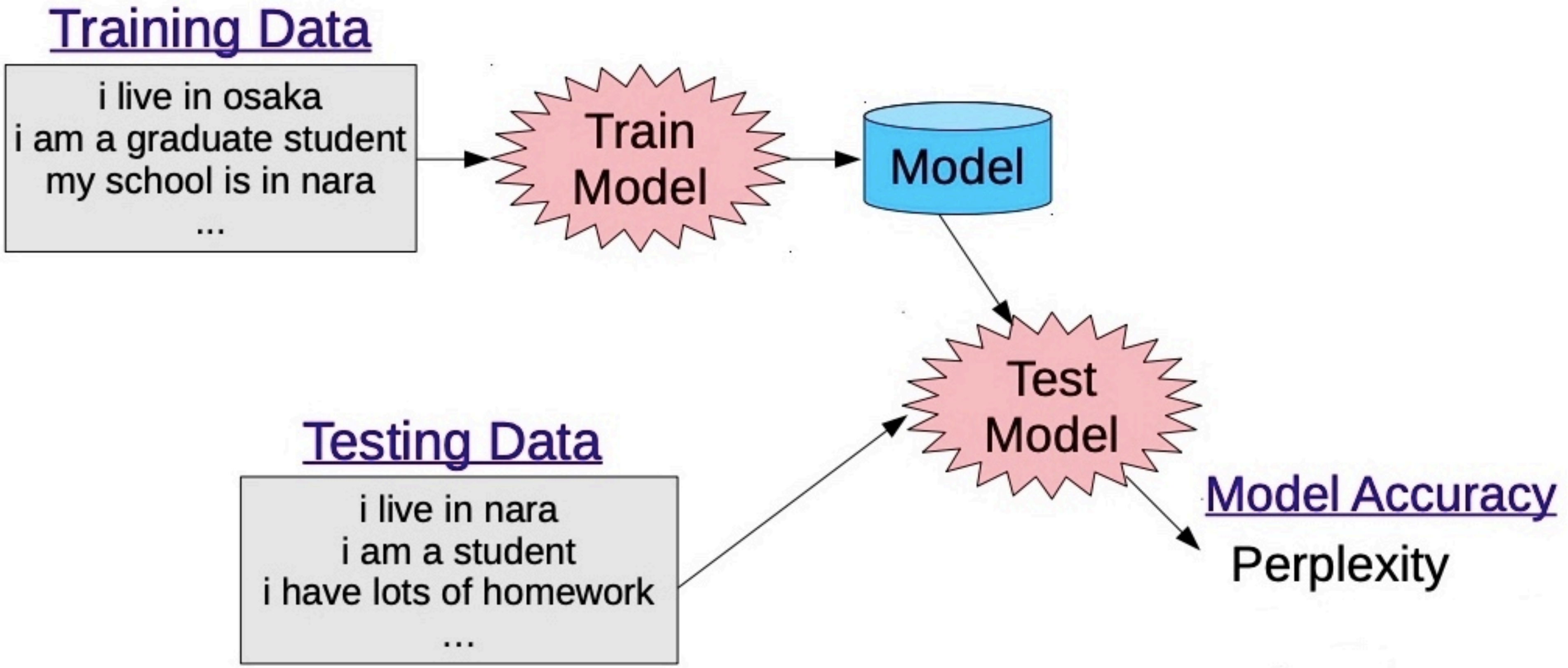
    Stochastically pick the next word based on the conditional probability of each word given the previous N –1 words.

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}}\, P(T \mid M(\lambda))$$

EVALUATING LANGUAGE MODELS

- Ideally, evaluate use of model in end application (**extrinsic**, **in vivo**)
  - Realistic
  - Often very expensive
- Evaluate on ability to model test corpus (**intrinsic**)
  - Less realistic
  - Cheaper

INTRINSIC EVALUATION



PERPLEXITY (PP)

- Call **hold-out (data) perplexity**
- Is the inverse probability of the test set
- Measure of how well a model "fits" the test data
- Lower perplexity is better because the greater conditional probability is better
- Perplexity of W(test set) with a bigram model: (N is the number of words in test sentence)

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

PERPLEXITY (PP) EXAMPLE

- Train Corpus:
  - This is the house that Jack built.
- Test Corpus:
  - This is the malt.
- Perplexity of bigram language model?

$$P(malt \mid the) = \frac{C(the\ malt)}{C(the)} = 0$$

$$P(W\ test\ data) = 0$$

$$PP = infinite$$

UNKNOWN OR OOV

- The percentage of OOV words that appear in test set
  - **OOV rate**
- Solution?
  - Build a vocabulary
    - How? By min-frequency
  - Substitute OOV words with pseudo-word called <UNK> (train and test)
  - Compute counts as usual for all tokens

PERPLEXITY WITHOUT OOV WORDS

- Train Corpus:
  - This is the house that Jack built.
- Test Corpus:
  - This is Jack.
- Perplexity of bigram language model?

$$P(Jack \mid is) = \frac{C(is\ Jack)}{C(is)} = 0$$

$$P(W\ test\ data) = 0$$

$$\mathrm{PP} = infinite$$

SMOOTHING OR DISCOUNTING

- What do we do with words that are in our vocabulary but appear in a test set in an unseen context?
- Stop zero probability by a modification
- Ways to do smoothing:
  1. **Add-one smoothing**
  2. **Add-k smoothing**
  3. **Stupid backoff**
  4. **Kneser-Ney smoothing**

LAPLACE SMOOTHING

- Idea:
  - **Add one** to all the **bigram counts**, before we normalize them into probabilities.
- Then, all the counts that used to be zero will have a count of 1, the counts of 1 will be 2 and so on.
- Laplace does not work well in n-gram models.
- But a good baseline for other smoothing methods

LAPLACE SMOOTHING (CONT'D)

- Recall **unsmoothed maximum likelihood estimate** of **unigram** probabilities:

$$P(w_i) = \frac{c_i}{N}$$

- Add-one smoothing:
  - Just adds one to each count

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

- Add-one smoothing **bigram** probability:

$$P_{Laplace}(W_n|W_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

ADD-ONE SMOOTHING EXAMPLE

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

ADD-K SMOOTHING

- Idea:
  - **Pull some probabilities from frequent bigram and to infrequent ones**
- Tune k with **test data**

$$P_{Add-k}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

- Simple and the most popular smoothing !

STUPID BACKOFF / KATZ BACKOFF

- Longer n-grams are better but data is not always enough!
- Idea:
  - **We only back off to a lower-order n-gram if we have zero evidence for a higher-order n-gram.**

$$P_{\mathrm{BO}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

INTERPOLATION SMOOTHING

- Idea:
  - **Have a combination of several n-grams models**
- E.g.:
  - The trigram probability by mixing together unigram, bigram and trigram each weighted by a λ

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

such that the λs sum to 1:

$$\sum_i \lambda_i = 1$$

KNESER-NEY SMOOTHING

- One of the most commonly used and best performing n-gram smoothing methods
- Known as **absolute discounting**
- **Idea:**
  - **Have the probability of the words proportional to how many different context can go before the word**

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}\, v)} + \lambda(w_{i-1})P(w_i)$$

KNESER-NEY SMOOTHING (CONT'D)

- I can't see without my reading ……………….
- Glasses vs. Kong?
- Which one is predicted in **unigram model?**
- **Hong Kong** is more frequent word.
- A standard unigram model will assign Kong a higher probability than glasses.
- **So instead of $P(w)$, we'd like to use $P_{CONTINUATION}$**

$$P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}|$$

- A frequent word (Kong) occurring in only one context (Hong) will have a low continuation probability

KNESER-NEY SMOOTHING (CONT'D)

- The average of d is 0.75
- λ is a normalizing constant used to distribute probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}|$$

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- **Interpolated Kneser-Ney** smoothing for bigram:

$$P_{\mathrm{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\mathrm{CONTINUATION}}(w_i)$$