

# Language Modeling

## Introduction to N-grams

Many Slides are adapted from slides by Dan Jurafsky

# Probabilistic Language Models

- Today's goal: assign a probability to a sentence
  - Machine Translation:
    - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
  - Spell Correction
    - The office is about fifteen **minuets** from my house
      - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
  - Speech Recognition
    - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
  - + Summarization, question-answering, etc., etc.!!

Why?

# Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these:

$P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$  is called a **language model**.

- Better: **the grammar** But **language model** or **LM** is standard

# How to compute $P(W)$

- How to compute this joint probability:
  - $P(\text{its, water, is, so, transparent, that})$
- Intuition: let's rely on the Chain Rule of Probability

# Reminder: The Chain Rule

- With 4 variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- The Chain Rule in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2 | x_1)P(x_3 | x_1, x_2) \dots P(x_n | x_1, \dots, x_{n-1})$$

# The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i | w_1 w_2 \dots w_{i-1})$$

P(“its water is so transparent”) =

P(its) × P(water | its) × P(is | its water)

× P(so | its water is) × P(transparent | its water is so)

# How to estimate these probabilities

- Could we just count and divide?

$$P(\text{the l its water is so transparent that}) = \frac{\textit{Count}(\textit{its water is so transparent that the})}{\textit{Count}(\textit{its water is so transparent that})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these

# Markov Assumption

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i \mid w_{i-k} \dots w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-k} \dots w_{i-1})$$



# Markov Assumption



Andrei Markov

- Simplifying assumption:

$P(\text{the lits water is so transparent that}) \approx P(\text{the l that})$

- Or maybe

$P(\text{the lits water is so transparent that}) \approx P(\text{the l transparent that})$

# Simplest case: Unigram model

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

fifth, an, of, futures, the, an, incorporated, a,  
a, the, inflation, most, dollars, quarter, in, is,  
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the

# Bigram model

- Condition on the previous word:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in,  
a, boiler, house, said, mr., gurria, mexico, 's, motion,  
control, proposal, without, permission, from, five, hundred,  
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

# N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
  - because language has **long-distance dependencies**:

“The computer which I had just put into the machine room on the fifth floor crashed.”

- But we can often get away with N-gram models

Still, Most words depend on their previous few words

# Language Modeling

Introduction to N-grams

# Language Modeling

Estimating N-gram  
Probabilities

# Estimating bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{\textit{count}(w_{i-1}, w_i)}{\textit{count}(w_{i-1})}$$

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

# An example

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$



## More examples:

### Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

# Raw bigram counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

# Raw bigram probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

# Bigram estimates of sentence probabilities

$$\begin{aligned} P(\langle s \rangle \text{ I want english food } \langle /s \rangle) &= \\ &P(\text{I} | \langle s \rangle) \\ &\times P(\text{want} | \text{I}) \\ &\times P(\text{english} | \text{want}) \\ &\times P(\text{food} | \text{english}) \\ &\times P(\langle /s \rangle | \text{food}) \\ &= .000031 \end{aligned}$$

# What kinds of knowledge?

- $P(\text{english} | \text{want}) = .0011$
- $P(\text{chinese} | \text{want}) = .0065$
- $P(\text{to} | \text{want}) = .66$
- $P(\text{eat} | \text{to}) = .28$
- $P(\text{food} | \text{to}) = 0$
- $P(\text{want} | \text{spend}) = 0$
- $P(i | \langle s \rangle) = .25$

## Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

# Language Modeling Toolkits

- SRILM
  - <http://www.speech.sri.com/projects/srilm/>

# Google N-Gram Release, August 2006

<http://ngrams.googlelabs.com/>

AUG

3

## All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.



# Language Modeling

Estimating N-gram  
Probabilities

# Language Modeling

Evaluation and  
Perplexity

# Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

# Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

# Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**

# Intuition of Perplexity

- The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and \_\_\_\_\_

The 33<sup>rd</sup> President of the US was \_\_\_\_\_

I saw a \_\_\_\_\_

- Unigrams are terrible at this game. (Why?)

- A better model of a text

- is one which assigns a higher probability to the word that actually occurs

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

....

fried rice 0.0001

....

and 1e-100

# Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest  $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

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

**Minimizing perplexity is the same as maximizing probability**

## Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109



# Language Modeling

Evaluation and  
Perplexity

# Language Modeling

Generalization and  
zeros

# The Shannon Visualization Method

- Choose a random bigram  
(`<s>`, `w`) according to its probability
- Now choose a random bigram  
(`w`, `x`) according to its probability
- And so on until we choose `</s>`
- Then string the words together

```
<s> I
    I want
      want to
        to eat
          eat Chinese
            Chinese food
              food </s>

I want to eat Chinese food
```

# Approximating Shakespeare

## Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have  
Every enter now severally so, let  
Hill he late speaks; or! a more to leg less first you enter  
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

## Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.  
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.  
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

## Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.  
This shall forbid it should be branded, if renown made it empty.  
Indeed the duke; and had a very good friend.  
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

## Quadrigram

King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;  
Will you not tell me who I am?  
It cannot be but so.  
Indeed the short and the long. Marry, 'tis a noble Lepidus.

# Shakespeare as corpus

- $N=884,647$  tokens,  $V=29,066$
- Shakespeare produced 300,000 bigram types out of  $V^2= 844$  million possible bigrams.
  - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare

# The wall street journal is not shakespeare (no offense)

## **Unigram**

Months the my and issue of year foreign new exchange's september were recession ex-  
change new endorsed a acquire to six executives

## **Bigram**

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor  
would seem to complete the major central planners one point five percent of U. S. E. has  
already old M. X. corporation of living on information such as more frequently fishing to  
keep her

## **Trigram**

They also point to ninety nine point six billion dollars from two hundred four oh six three  
percent of the rates of interest stores as Mexico and Brazil on market conditions

# The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
      - But occur in the test set

# Zeros

- Training set:
  - ... denied the allegations
  - ... denied the reports
  - ... denied the claims
  - ... denied the request
- Test set
  - ... denied the offer
  - ... denied the loan

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



# Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

# Language Modeling

Generalization and  
zeros

# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- MLE estimate:

- Add-1 estimate:

$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V + 1}$$

# Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	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

# Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} + 1$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

# Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V + 1}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16



# Add-1 estimation is a blunt instrument

- So add-1 isn't used for N-grams:
  - We'll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.

# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# Language Modeling

Interpolation, Backoff,  
and Web-Scale LMs

# Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram
- Interpolation works better

# Linear Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

# How to set the lambdas?

- Use a **held-out** corpus



- Choose  $\lambda$ s to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for  $\lambda$ s that give largest probability to held-out set:

$$\log P(w_1 \dots w_n \mid M(\lambda_1 \dots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \dots \lambda_k)}(w_i \mid w_{i-1})$$

# Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advanced
  - Vocabulary  $V$  is fixed
  - Closed vocabulary task
- Often we don't know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon  $L$  of size  $V$
    - At text normalization phase, any training word not in  $L$  changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

# Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count  $>$  threshold.
    - Remove singletons of higher-order n-grams
- Efficiency
  - Efficient data structures like tries
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes



# Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants *et al.* 2007)
- No discounting, just use relative frequencies

$$S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ 0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

# N-gram Smoothing Summary

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
  - Stupid backoff

# Language Modeling

Interpolation, Backoff,  
and Web-Scale LMs

# Language Modeling

Advanced:  
Kneser-Ney Smoothing

# Advanced smoothing algorithms

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
- Use the count of things we've **seen**
  - to help estimate the count of things we've **never seen**

# Kneser-Ney Smoothing I (smart backoff)

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading Figtarsissso ?*
  - “Francisco” is more common than “glasses”
  - ... but “Francisco” always follows “San”
- The unigram is useful exactly when we haven't seen this bigram!
- Instead of  $P(w)$ : “How likely is  $w$ ”
- $P_{\text{continuation}}(w)$ : “How likely is  $w$  to appear as a novel continuation?”
  - For each word, count the number of unique bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{\text{CONTINUATION}}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

# Kneser-Ney Smoothing II

- How many times does  $w$  appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalized by the total number of word bigram types

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

# Kneser-Ney Smoothing III

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

$\lambda$  is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

the normalized discount

The number of word types that can follow  $w_{i-1}$   
= # of word types we discounted  
= # of times we applied normalized discount



# Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i | w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i | w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} \textit{count}(\bullet) & \text{for the highest order} \\ \textit{continuationcount}(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

# Language Modeling

Advanced:  
Kneser-Ney Smoothing