Course series: Deep Learning for Machine Translation

Language Modeling

Lecture # 2

Hassan Sajjad and Fahim Dalvi
Qatar Computing Research Institute, HBKU

Recap: Automatic Translation System

- Translation model
  - learn word level and phrase level translation
- Language model
  - fluency model
  - learn to generate fluent translations
- Decoder
  - translation generation component
  - how to produce a translation from a trained translation model and language model

Recap: Automatic Translation System

- Translation model
  - learn word level and phrase level translation
- Language model
  - fluency model
  - learn to generate fluent translations
- Decoder
  - translation generation component
  - how to produce a translation from a trained translation model and language model

Language Model

You shall know a word by the company it keeps

—Firth, J. R. 1957:11

Language Model

A few applications of language model

- Machine Translation
  - he goes vs. he go

Language Model

A few applications of language model

- Machine Translation
  - he goes vs. he go
- Speech recognition
  - speaker recognition vs. speak are cognition

Language Model

A few applications of language model

- Machine Translation
  - he goes vs. he go
- Speech recognition
  - speaker recognition vs. speak are cognition
- Spell checking
  - from vs. form

Language Model

Fill in the blank:

... a __________ ...

car	cars	water	cat

Language Model

Fill in the blank:

... a _________ ...

car and *cat* both work

car	cars	water	cat

Language Model

Fill in the blank:

... a _________ ...

car and *cat* both work

car	cars	water	cat

John is driving a _________ ...

Language Model

Fill in the blank:

... a _________ ...

car and *cat* both work

car	cars	water	cat

John is driving a _________ ...

only *car* works here

Language Model

Fill in the blank:

... a _________ ...

car and *cat* both work

car	cars	water	cat

John is driving a _________ ...

Similarly, machines use the context to predict the next words

Language Model

You chose “driving a car” because you’ve seen that phrase more frequently

“driving a cat” is not a common phrase

Language Model

Fill in the blank:

This _______ is going at 100 km/hours

car

bicycle

Car at 100km/hours is more probable than a bicycle

Language Model

Language model defines
"how probable a sentence is"

Language Model

Let’s look at the example again

How probable is:

John is driving a car vs. John is driving a cat

In other words, what is the probability to predict cat or car given the context “John is driving a”

Language Model

Calculate probability of a sequence of words using Chain Rule

$$
\begin{aligned}
p(\text{John is driving a car}) = \; & p(\text{John}) \cdot p(\text{is}|\text{John}) \\
& \cdot p(\text{driving}|\text{John is}) \\
& \cdot p(\text{a}|\text{John is driving}) \\
& \cdot p(\text{car}|\text{John is driving a})
\end{aligned}
$$

Language Model

But...

$$p(\text{car}|\text{John is driving a}) \text{ vs. } p(\text{car}|\text{driving a}) \text{ vs. } p(\text{car}|\text{a})$$

- long context is infrequent to find
- requires more memory to keep

Ngram based LM

- Limit the context to fewer words

$$p(\text{car}|\text{John is driving a})$$

$$\Big\downarrow \text{ remove words that appear too far back}$$

$$p(\text{car}|\text{driving a})$$

Ngram based LM

- Limit the context to fewer words

$$p(\text{car}|\text{John is driving a}) \longrightarrow p(\text{car}|\text{driving a})$$

Unigram: $p(\text{car})$

Bigram: $p(\text{car}|\text{a})$

Trigram: $p(\text{car}|\text{driving a})$

Ngram based LM

Probability of a sequence with bigram context

$$
\begin{aligned}
p(\text{John is driving a car}) = p(\text{John}) \cdot p(\text{is}|\text{John}) \\
\cdot p(\text{driving}|\text{John is}) \\
\cdot p(\text{a}|\text{is driving}) \\
\cdot p(\text{car}|\text{driving a})
\end{aligned}
$$

Ngram based LM

Probability of a sequence with bigram context

$$p(\text{John is driving a car}) = p(\text{John}) \cdot p(\text{is}|\text{John})$$
$$\cdot p(\text{driving}|\text{John is})$$
$$\cdot p(\text{a}|\text{is driving})$$
$$\cdot p(\text{car}|\text{driving a})$$

Formally, Ngram approximation of a sequence is given by:

$$p(w_1 w_2 .. w_n) = \prod_{k=1}^{n} p(w_k | w_{k-N+1}^{k-1})$$

Count-based LM

How do we estimate probabilities?

Count-based LM

Estimate probabilities from a corpus

$$p(\text{John}) = \frac{c(\text{John})}{\text{Total number of tokens in corpus}}$$

$$p(\text{is}|\text{John}) = \frac{c(\text{John is})}{c(\text{John})}$$

where c stands for count

Count-based LM - Example

Corpus

<s> I am Sam </s>

<s> Sam likes tea </s>

<s> Dan does not like green eggs and ham </s>

<s> Sam eats ham </s>

<s> Dan likes cats </s>

<s> tea and biscuits go together </s>

<s> a pack of biscuits </s>

Count-based LM - Example

Corpus

<s> I am Sam </s>

<s> Sam likes tea </s>

<s> Dan does not like green eggs and ham </s>

<s> Sam eats ham </s>

<s> Dan likes cats </s>

<s> tea and biscuits go together </s>

<s> a pack of biscuits </s>

and	2	not	1
tea	2	a	1
am	1	like	1
likes	2	Sam	3
go	1	of	1
</s>	7	eats	1
ham	2	together	1
<s>	7	green	1
Dan	2	cats	1
does	1	biscuits	2
I	1	pack	1
eggs	1		

Unigram Counts

Count-based LM - Example

Corpus

<s> I am Sam </s>

<s> Sam likes tea </s>

<s> Dan does not like green eggs and ham </s>

<s> Sam eats ham </s>

<s> Dan likes cats </s>

<s> tea and biscuits go together </s>

<s> a pack of biscuits </s>

Bigram Counts

<s> a	1	and biscuits	1
likes cats	1	Sam </s>	1
eggs and	1	cats </s>	1
ham </s>	2	<s> tea	1
Dan likes	1	and ham	1
a pack	1	likes tea	1
go together	1	does not	1
biscuits go	1	I am	1
tea and	1	green eggs	1
like green	1	Sam eats	1
Dan does	1	not like	1
of biscuits	1	together </s>	1
<s> I	1	Sam likes	1
biscuits </s>	1	<s> Sam	2
pack of	1	eats ham	1
am Sam	1	<s> Dan	2
tea </s>	1		

Count-based LM - Example

Calculate bigram probabilities

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

Unigram Counts

and	2	not	1
tea	2	a	1
am	1	like	1
likes	2	Sam	3
go	1	of	1
</s>	7	eats	1
ham	2	together	1
<s>	7	green	1
Dan	2	cats	1
does	1	biscuits	2
I	1	pack	1
eggs	1		

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

Bigram Counts

<s> a	1	and biscuits	1
likes cats	1	Sam </s>	1
eggs and	1	cats </s>	1
ham </s>	2	<s> tea	1
Dan likes	1	and ham	1
a pack	1	likes tea	1
go together	1	does not	1
biscuits go	1	I am	1
tea and	1	green eggs	1
like green	1	Sam eats	1
Dan does	1	not like	1
of biscuits	1	together </s>	1
<s> I	1	Sam likes	1
biscuits </s>	1	<s> Sam	2
pack of	1	eats ham	1
am Sam	1	<s> Dan	2
tea </s>	1		

Count-based LM - Example

Calculate probability of a sentence

$$p(\texttt{<s>}\ \text{Dan likes tea}\ \texttt{</s>})$$

Count-based LM - Example

Calculate probability of a sentence

$$p(\texttt{<s>}\ \text{Dan likes tea}\ \texttt{</s>}) = \frac{c(\texttt{<s>}\ \text{Dan})}{c(\texttt{<s>})} \cdot \frac{c(\text{Dan likes})}{c(\text{Dan})} \cdot \frac{c(\text{likes tea})}{c(\text{likes})} \cdot \frac{c(\text{tea}\ \texttt{</s>})}{c(\text{tea})}$$

Count-based LM - Example

Calculate probability of a sentence

$$p(\texttt{<s>}\ \text{Dan likes tea}\ \texttt{</s>}) = \frac{c(\texttt{<s>}\ \text{Dan})}{c(\texttt{<s>})}$$

$$\cdot \frac{c(\text{Dan likes})}{c(\text{Dan})} \cdot \frac{c(\text{likes tea})}{c(\text{likes})} \cdot \frac{c(\text{tea}\ \texttt{</s>})}{c(\text{tea})}$$

= 0.286 x 0.5 x 0.5 x 0.5

= 0.036

Count-based LM - Example

Calculate probability of a sentence

$$p(\texttt{<s>}\ \text{Dan likes tea and biscuits}\ \texttt{</s>})$$

Count-based LM - Example

Calculate probability of a sentence

$$p(\texttt{<s>}\ \text{Dan likes tea and biscuits}\ \texttt{</s>})$$

= 0.286 x 0.5 x 0.5 x 0.5 x 0.5 x 0.5

= 0.0089

Unknown Words/Sequences

$$p(\texttt{<s>}\ \text{Dan likes ham}\ \texttt{</s>}) = \frac{c(\texttt{<s>}\ \text{Dan})}{c(\texttt{<s>})} \cdot \frac{c(\text{Dan likes})}{c(\text{Dan})} \cdot \frac{c(\text{likes ham})}{c(\text{likes})} \cdot \frac{c(\text{ham}\ \texttt{</s>})}{c(\text{ham})}$$

Unknown Words/Sequences

$$p(\texttt{<s>}\ \text{Dan likes ham}\ \texttt{</s>}) = \frac{c(\texttt{<s>}\ \text{Dan})}{c(\texttt{<s>})} \cdot \frac{c(\text{Dan likes})}{c(\text{Dan})} \cdot \boxed{\frac{c(\text{likes ham})}{c(\text{likes})}} \cdot \frac{c(\text{ham}\ \texttt{</s>})}{c(\text{ham})}$$

Unknown

probability of unknown is zero

probability of the entire sentence becomes zero!

Intuition of Smoothing

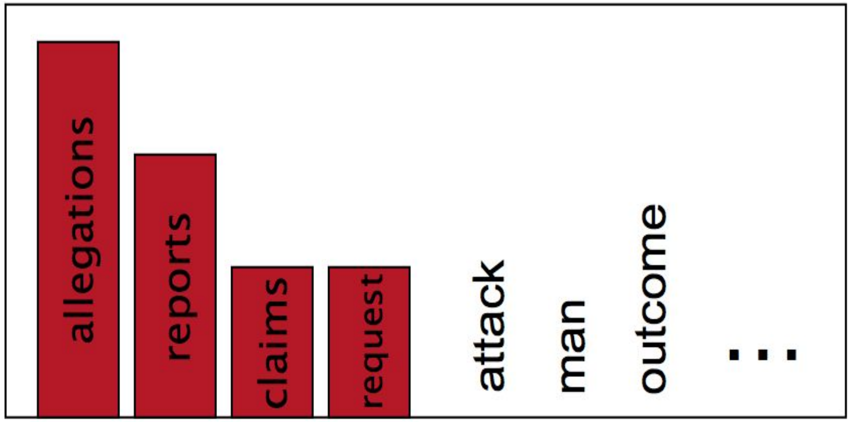
- Steal probability mass from known words
- Assign it to unknown words

From D. Jurafsky, Stanford https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf

Intuition of Smoothing

Sparse statistics:

- 3 allegations, 2 reports, 1 claims, 1 request
- 7 total



From D. Jurafsky, Stanford https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf

Intuition of Smoothing

Sparse statistics:

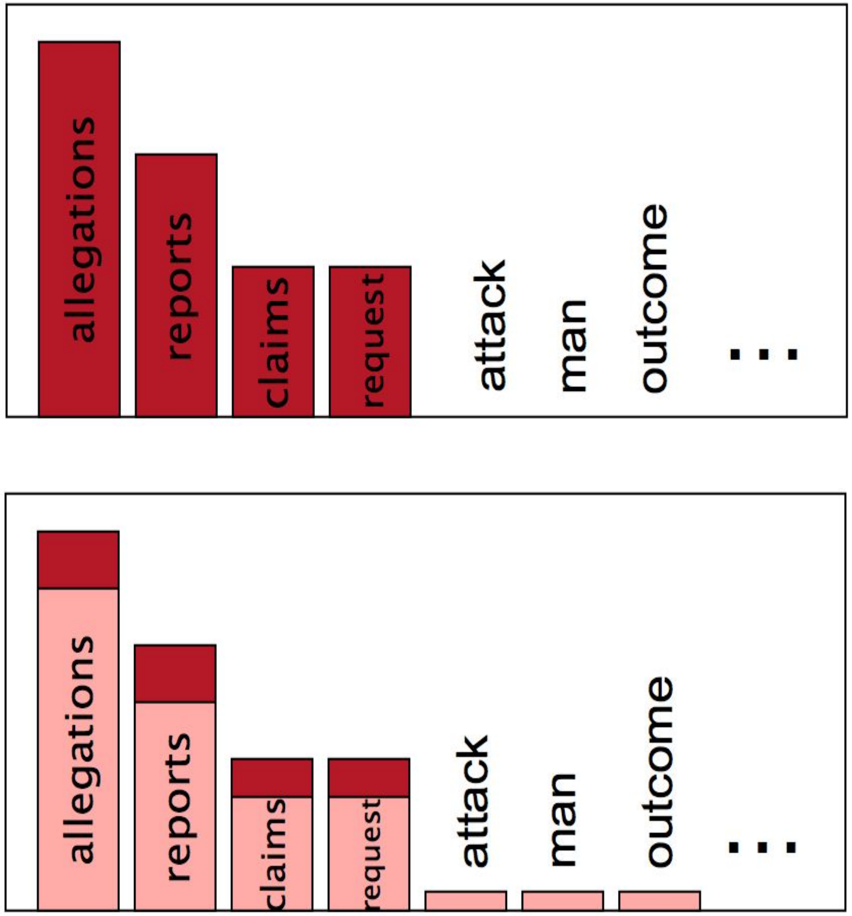
3 allegations, 2 reports, 1 claims, 1 request

7 total

Steal probability mass to generalize better

2.5 allegations, 1.5 reports, 0.5 claims, 0.5 request, **2 other**

7 total



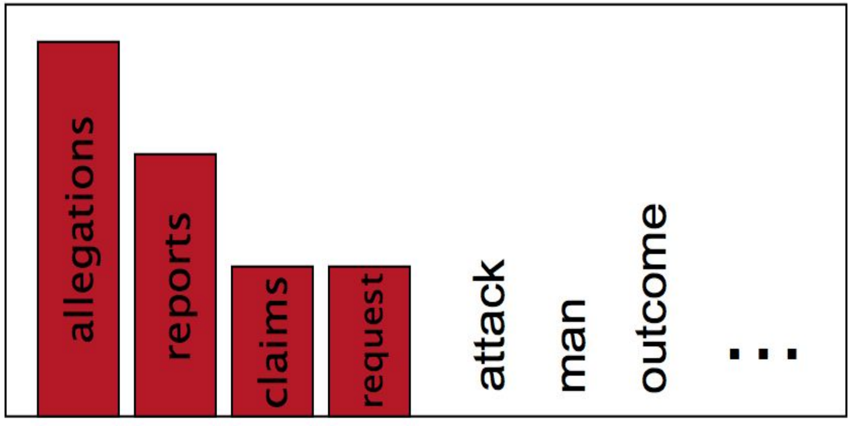
From D. Jurafsky, Stanford https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf

Intuition of Smoothing

Sparse statistics:

3 allegations, 2 reports, 1 claims, 1 request

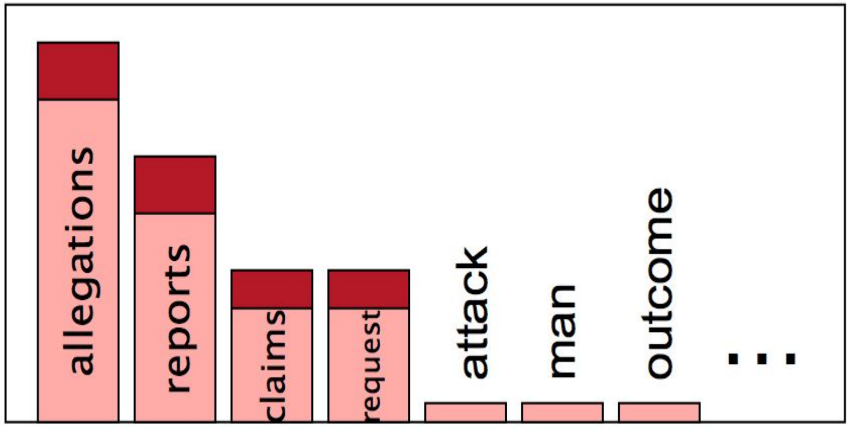
7 total



Steal probability mass to generalize better

2.5 allegations, 1.5 reports, 0.5 claims, 0.5 request, **2 other**

7 total



Now we have count of 2 additional words that we can split into unknown words

From D. Jurafsky, Stanford https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf

Add-one (Laplace) Smoothing

- Add one to all counts!
  - assume we saw each word in corpus one more time
  - then we saw unseen words once in the corpus

Smoothing of unigrams

$$p(w_i) = \frac{c(w_i) + 1}{N + V}$$

Here V is the total number of **unique** words in our corpus

N is total number of words in our corpus

Add-one (Laplace) Smoothing

- Add one to all counts!
  - assume we saw each word in corpus one more time
  - then we saw unseen words once in the corpus

Smoothing of bigrams

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

Here V is the total number of unique words in our corpus

Add-one (Laplace) Smoothing

Smoothing of Ngrams

$$p(w_n|w_{n-N+1}^{n-1}) = \frac{c(w_{n-N+1}^{n-1}, w_n) + 1}{c(w_{n-N+1}^{n-1}) + V}$$

Here V is the total number of possible (N-1) grams

Smoothing

Several smoothing methods,

- Kneser-Ney
- Good Turing
- …

Evaluating LM

- Does our LM give higher score to good sentences compared to bad sentences?
  - frequently observed vs. rare sentences

Train	the corpus used to build LM
Test	unseen data to see LM performance
Evaluate	scoring metric to evaluate LM

Evaluating LM

- Does our LM give higher score to good sentences compared to bad sentences?
  - frequently observed vs. rare sentences

Train	the corpus used to build LM
Test	unseen data to see LM performance
Evaluate	scoring metric to evaluate LM

Why do we need unseen data to test?

Evaluating LM

Test	unseen data to *see* LM performance
Development	unseen data to *tune* LM performance

Test set contains examples that will come from the real world. We assume that we don't have a test while training a model

Evaluating LM

Test	unseen data to *see* LM performance
Development	unseen data to *tune* LM performance

Hold out a few examples from your training set to test your LM!

Evaluating LM

How good is the LM in scoring a test set?

- Extrinsic evaluation
  - use LM in another application, such as machine translation and observe the performance improvement
  - expensive
- Intrinsic evaluation
  - test the quality of LM on unseen data
  - perplexity

Evaluation: Perplexity

How good is the LM in scoring a test set?

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

$$ppl(s) = p(w_1\ w_2\ \dots\ w_N)^{-1/N}$$

- higher the probability, lower the perplexity
- lower is better

LM closing remarks

Language models can learn different styles

- formal
- informal (spoken, sms, chat)

So if we have multiple LMs, how do you know which one to use?

- check perplexity on a dev set
- an LM built on formal sentences would have lower **ppl** on formal corpus than informal

Python & Numpy

[See accompanying notebook]

iPython/Jupyter notebook Installation

https://www.anaconda.com/download/#download

or

http://jupyter.readthedocs.io/en/latest/install.html