Discrete and Continuous Language Models

Ashish Vaswani

University of Southern California
Information Sciences Institute
Statistical Machine Translation (SMT)

source

target
Statistical Machine Translation (SMT)

source

la kato sidis sur mato

target

the cat sat on a mat
Statistical Machine Translation (SMT)

source

“la kato sidis sur mato”

“the cat sat on a mat”

target

“वह बिल्ली चटाई पर बैठी”

“the cat sat on a mat”
Statistical Machine Translation (SMT)

source

set $a$ as the sigmoid of $y$

कह बिल्ली चटाई पर बैठी

la kato sidis sur mato

target

$a = \frac{1}{1 + e^{-y}}$

the cat sat on a mat

the cat sat on a mat
Statistical Machine Translation (SMT)

source

\[ a = \frac{1}{1 + e^{-y}} \]

वह बिल्ली चटाई पर बैठी

la kato sidis sur mato

target

set \( a \) as the sigmoid of \( y \)

the cat sat on a mat

the cat sat on a mat
Statistical Machine Translation (SMT)

source

loop over all people

\[ a = \frac{1}{1 + e^{-y}} \]

वह बिल्ली चटाई पर बैठी

la kato sidis sur mato

target

for (int i=0; i>personlist.size(); i++)

set a as the sigmoid of y

the cat sat on a mat

the cat sat on a mat
Language models in Machine Translation

Diagram:
- Translation grammars
- Decoder
- Language model

Input sentence flows through the diagram.

Translation grammars → Decoder → Language model
Language models in Machine Translation

\[ \text{sigmoid of } y, \quad \frac{1}{1 + e^{-y}} \]

*la kato*, the cat

*चटाई पर बैठी*, sat on a mat
Language models in Machine Translation

Translation grammars → Decoder → Language model

Input sentence

SMT Parameters

1.5 billion

0.9 billion

la kato, the cat

चटाई पर बैठी, sat on a mat

sigmoid of y, $\frac{1}{1 + e^{-y}}$
Applications of Language Models

Natural Language Applications

- Machine Translation
- Speech Recognition
- Spelling Correction
- Document Summarization
- Sentence Completion

Programming Languages

- Code Completion
- Retrieval of code from natural language
- Retrieval of natural language from code snippets
- Identifying buggy code
Outline

What are Language Models?
- Definition
- Markovization
Outline

What are Language Models?
- Definition
- Markovization

Discrete Language Models
- Estimating Probabilities
- Maximum Likelihood Estimation
- Overfitting
- Smoothing
Outline

What are Language Models?
- Definition
- Markovization

Discrete Language Models
- Estimating Probabilities
- Maximum Likelihood Estimation
- Overfitting
- Smoothing

Continuous Language Models
- Feed Forward Neural Network LM
- Recurrent Neural Network LM
  A. Vanilla RNN
  B. Long Short Term Memory RNN
- Visualizing LSTMs
What are Language Models?
Language Models

the cat sat on the mat
Language Models

\[ P(\text{the cat sat on the mat} ) \]
Chain Rule

\[ P(\text{the cat sat on the mat}) \]

\[ = P(\text{the}) P(\text{cat } | \text{the}) P(\text{sat } | \text{the cat}) P(\text{on } | \text{the cat sat}) \]

\[ P(\text{the } | \text{the cat sat on}) P(\text{mat } | \text{the cat sat on the}) \]
Chain Rule

\[ P(\text{the cat sat on the mat}) = P(\text{the}) \cdot P(\text{cat} | \text{the}) \cdot P(\text{sat} | \text{the cat}) \cdot P(\text{on} | \text{the cat sat}) \cdot P(\text{the} | \text{the cat sat on}) \cdot P(\text{mat} | \text{the cat sat on the}) \]

\[ P(w_1, w_2, \ldots, w_n) = \prod_i P(w_i | w_1, w_2, \ldots, w_{i-1}) \]
n-gram Language Model

\[
P(\text{the cat sat on the mat })
\]

\[
= P(\text{the}) P(\text{cat | the}) P(\text{sat | cat}) P(\text{on | sat}) P(\text{the | on}) P(\text{mat | the})
\]
P( the cat sat on the mat )

= P( the ) P( cat | the ) P( sat | cat ) P( on | sat )
P( the | on ) P( mat | the )
Bigram Language Model

\[
P(\text{the cat sat on the mat}) = P(\text{the}) P(\text{cat} | \text{the}) P(\text{sat} | \text{cat}) P(\text{on} | \text{sat}) P(\text{the} | \text{on}) P(\text{mat} | \text{the})
\]
Bigram Language Model

\[
P( \text{the cat sat on the mat} )
\]

\[
= P( \text{the} ) P( \text{cat | the} ) P( \text{sat | cat} ) P( \text{on | sat} ) P( \text{mat | the} )
\]
Trigram Language Model

\[ P( \text{the cat sat on the mat} ) \]

\[ = P( \text{the} ) \cdot P( \text{cat | the} ) \cdot P( \text{sat | the cat} ) \cdot P( \text{on | cat sat} ) \]

\[ P( \text{the | sat on} ) \cdot P( \text{mat | on the} ) \]
Trigram Language Model

\[ P( \text{the cat sat on the mat} ) \]
\[ = P( \text{the} ) P( \text{cat | the} ) P( \text{sat | the cat} ) P( \text{on | cat sat} ) \]
\[ P( \text{the | sat on} ) P( \text{mat | on the} ) \]

\[ P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i | w_{i-k+1}, w_2, \ldots, w_{i-1}) \]
Discrete Space Language Models
Estimating Probabilities

\[
P( \text{the cat sat on the mat} )
\]

\[
= P( \text{the} ) P( \text{cat | the} ) P( \text{sat | cat} ) P( \text{on | sat} )
\]

\[
P( \text{the} ) P( \text{on | the} ) P( \text{mat | the} )
\]

\[
P(\text{on | sat}) = \frac{\text{count(sat on)}}{\text{count(sat)}}
\]
Estimating Probabilities

\[
P(\text{the cat sat on the mat})
\]
\[
= P(\text{the}) P(\text{cat | the}) P(\text{sat | cat}) P(\text{on | sat})
\]
\[
P(\text{the | on}) P(\text{mat | the})
\]

\[
P(\text{on | sat}) = \frac{c(\text{sat on})}{c(\text{sat})}
\]
Estimating Probabilities

\[ P(on \mid sat) = \frac{c(sat \ on)}{c(sat)} \]

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \ w_i)}{c(w_i)} \]
Estimating Probabilities

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \ w_i)}{c(w_i)} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>
Estimating Probabilities

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \ w_i)}{c(w_i)} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[ P(\text{cat} \mid \text{the}) = \frac{3}{6} = 0.5 \]
Estimating Probabilities

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \ w_i)}{c(w_i)} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[ P(\text{cat} \mid \text{the}) = \frac{3}{6} = 0.5 \]

\[ P(\text{dog} \mid \text{the}) = \frac{1}{6} = 0.167 \]
Estimating Probabilities

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \ w_i)}{c(w_i)} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[ P(\text{cat} \mid \text{the}) = \frac{3}{6} = 0.5 \]
\[ P(\text{dog} \mid \text{the}) = \frac{1}{6} = 0.167 \]
\[ P(\text{mat} \mid \text{the}) = \frac{1}{6} = 0.167 \]
Estimating Probabilities

\[
P(w_i | w_{i-1}) = \frac{c(w_{i-1} \ w_i)}{c(w_i)}
\]

<s> the cat sat on the mat </s>
\[ P(\text{cat} | \text{the}) = \frac{3}{6} = 0.5 \]

<s> the dog sat on the cat </s>
\[ P(\text{dog} | \text{the}) = \frac{1}{6} = 0.167 \]

<s> the cat caught the mouse </s>
\[ P(\text{mat} | \text{the}) = \frac{1}{6} = 0.167 \]

\[ P(\text{the} | \text{on}) = \frac{2}{2} = 1.0 \]
Maximum Likelihood Estimation

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} \, w_i)}{c(w_i)} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[ P(\text{cat} \mid \text{the}) = \frac{3}{6} = 0.5 \]
\[ P(\text{dog} \mid \text{the}) = \frac{1}{6} = 0.167 \]
\[ P(\text{mat} \mid \text{the}) = \frac{1}{6} = 0.167 \]
\[ P(\text{the} \mid \text{on}) = \frac{2}{2} = 1.0 \]
Maximum Likelihood Estimation (MLE)

\[ P( \text{<s> the cat sat on the mat </s>} ) = \]
Maximum Likelihood Estimation (MLE)

\[
P( <s> \text{ the cat sat on the mat } </s> ) = P(\text{the} \mid < s >) \times \]

Maximum Likelihood Estimation (MLE)

\[
P(\text{the cat sat on the mat}) = P(\text{the } | < s >) \times P(\text{cat } | \text{the})
\]
Maximum Likelihood Estimation (MLE)

\[
P(\text{<s> the cat sat on the mat </s>}) = P(\text{the} | <s>) \times P(\text{cat} | \text{the}) \times P(\text{sat} | \text{cat}) \times
\]
Maximum Likelihood Estimation (MLE)

\[
P( \text{<s> the cat sat on the mat </s>}) = \quad P(\text{the | <s>}) \times \\
P(\text{cat | the}) \times \\
P(\text{sat | cat}) \times \\
P(\text{on | sat}) \times
\]
Maximum Likelihood Estimation (MLE)

\[ P(\text{<s> the cat sat on the mat </s>}) = P(\text{the} | \text{<s>}) \times P(\text{cat} | \text{the}) \times P(\text{sat} | \text{cat}) \times P(\text{on} | \text{sat}) \times P(\text{the} | \text{on}) \times \]
Maximum Likelihood Estimation (MLE)

\[
P( \text{<s> the cat sat on the mat </s>}) = \quad P(\text{the} | \text{< s >}) \times \\
P(\text{cat} | \text{the}) \times \\
P(\text{sat} | \text{cat}) \times \\
P(\text{on} | \text{sat}) \times \\
P(\text{the} | \text{on}) \times \\
P(\text{mat} | \text{the}) \times \\
\]
Maximum Likelihood Estimation (MLE)

\[ P( \text{the cat sat on the mat} ) = P(\text{the} | < s >) \times P(\text{cat} | \text{the}) \times P(\text{sat} | \text{cat}) \times P(\text{on} | \text{sat}) \times P(\text{the} | \text{on}) \times P(\text{mat} | \text{the}) \times P(< /s > | \text{mat}) \]
Maximum Likelihood Estimation (MLE)

\[
P(\text{<s> the cat sat on the mat </s>}) = P(\text{the} | < s >) \times P(\text{cat} | \text{the}) \times P(\text{sat} | \text{cat}) \times P(\text{on} | \text{sat}) \times P(\text{the} | \text{on}) \times P(\text{mat} | \text{the}) \times P(< /s > | \text{mat})
\]

\[
= 0.084
\]
MLE with higher order n-grams?

\[
P( <s> <s> \text{ the cat sat on the mat } </s> ) = P(\text{the }|<s><s>) \times \\
P(\text{cat }|<s>\text{ the}) \times \\
P(\text{sat }|\text{ the cat}) \times \\
\ldots \\
P(</s>|\text{ the mat}) \\
= 0.33
\]
MLE with higher order n-grams?

If you increase the n-gram order, you will improve your training data probability with MLE.
MLE with higher order n-grams?

If you increase the n-gram order, you will improve your training data probability with MLE

\[ P(\text{DATA})_{n\text{-gram}} \geq P(\text{DATA})_{(n-1)\text{-gram}} \]
MLE with higher order n-grams?

If you increase the n-gram order, you will improve your training data probability with MLE

\[ P(\text{DATA})_{n-\text{gram}} \geq P(\text{DATA})_{(n-1)-\text{gram}} \]

Log Sum Inequality

\[ a_i \log \frac{a_i}{b_i} \geq \left( \sum_{i=1}^{n} a_i \right) \log \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i} \]
Comparing Language Models
Data Splits

• Divide your data into train, development, and test.
• Train on the training set.
• Adjust hyperparameters on the dev set (length of n-grams).
• Test on the test set.
Extrinsic Evaluation

• Imagine you have two models A and B.
• Use them on an end-to-end task: Machine Translation, speech recognition.
• Evaluate the accuracy on the task.
• Improve language model.
• This process can take a while.
Intrinsic Evaluation

Log Likelihood

Trigram Model: \[
\sum_{i=1}^{\text{number of tokens}} \log P(w_i \mid w_{i-2}, w_{i-1})
\]

Bigram Model: \[
\sum_{i=1}^{\text{number of tokens}} \log P(w_i \mid w_{i-1})
\]

Higher is better
Intrinsic Evaluation

Cross Entropy

Trigram Model: \[ \frac{-1}{\text{number of tokens}} \sum_{i=1}^{\text{number of tokens}} \log P(w_i | w_{i-2}, w_{i-1}) \]

Bigram Model: \[ \frac{-1}{\text{number of tokens}} \sum_{i=1}^{\text{number of tokens}} \log P(w_i | w_{i-1}) \]

Lower is better
Intrinsic Evaluation

Perplexity

Trigram Model:

Bigram Model:

Lower is better
MLE and Overfitting
MLE recap

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1} w_i)}{c(w_i)} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[
\begin{align*}
P(\text{cat} \mid \text{the}) &= \frac{3}{6} = 0.5 \\
P(\text{dog} \mid \text{the}) &= \frac{1}{6} = 0.167 \\
P(\text{mat} \mid \text{the}) &= \frac{1}{6} = 0.167 \\
P(\text{the} \mid \text{on}) &= \frac{2}{2} = 1.0
\end{align*}
\]
MLE recap

\[ P(w_i \mid w_{i-1})_{\text{MLE}} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[ P(\text{cat} \mid \text{the})_{\text{MLE}} = \frac{3}{6} = 0.5 \]
\[ P(\text{dog} \mid \text{the})_{\text{MLE}} = \frac{1}{6} = 0.167 \]
\[ P(\text{mat} \mid \text{the})_{\text{MLE}} = \frac{1}{6} = 0.167 \]
\[ P(\text{the} \mid \text{on})_{\text{MLE}} = \frac{2}{2} = 1.0 \]
Overfitting

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

- You can achieve very low perplexity on training.
- But test data can be different from training.
Overfitting

You can achieve very low perplexity on training.

But test data can be different from training.
Overfitting

TEST

<s> the dog caught the cat </s>

TRAIN

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[
P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3}
\]

\[
P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0
\]
Overfitting

TEST

<s> the dog caught the cat </s>

TRAIN

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

Test probability is 0!

\[
P(\text{caught} | \text{cat})_{\text{MLE}} = \frac{1}{3}
\]

\[
P(\text{caught} | \text{dog})_{\text{MLE}} = 0
\]
Generalization

We need to train models that generalize well.

- Train n-gram models were ’n’ is small
- Train n-gram models on large collections of data.
Tokens Vs Types

Number of Types

WSJ
Eclipse

Millions of tokens

0 1 2 3 4 5 6 7 8

0 40000 80000 120000 160000
n-gram Language Models

{l-gram

sharon
talks
bush
iPhone
mercurial
...

Number of unique l-grams in Language: \(60 \times 10^3\)
n-gram Language Models

2-gram

sharon talks bush iPhone mercurial ...
sharon talks bush iPhone mercurial ...

Number of unique 2-grams in Language: $3.6 \times 10^9$
n-gram Language Models

3-gram

sharon talks bush iPhone mercurial ...

sharon talks bush iPhone mercurial ...

sharon talks bush iPhone mercurial ...

Number of unique 3-grams in Language: \(2.16 \times 10^{14}\)
How about Google n-grams?

Number of unique 3-grams: \( 2.16 \times 10^{14} \)
How about Google n-grams?

Number of unique 3-grams: $2.16 \times 10^{14}$
How about Google n-grams?

Running text in Google n-grams
\[
\frac{\text{Number of unique 3-grams in Language:}}{2.16 \times 10^{14}} = \frac{1 \times 10^{12}}{2.16 \times 10^{14}} = 0.0046
\]
The Web?

\[
\frac{\text{Number of words in the indexed web}}{\text{Number of unique 3-grams in Language:}} = \frac{4.4927 \times 10^{14}}{2.16 \times 10^{14}} = 2.07
\]
The Web?

\[
\begin{align*}
\frac{\text{Number of words in the indexed web}}{\text{Number of unique 3-grams in Code:}} &= \frac{4.4927 \times 10^{14}}{3.375 \times 10^{15}} = 0.133
\end{align*}
\]
Better Solution: Smoothing

• Just collecting a lot of data is not enough.

• Smoothing methods differ in how they move probability from seen n-grams to unseen n-grams.

• Neural network language models have a more elegant solution for smoothing.
Better Solution: Smoothing

\[
\begin{align*}
\text{TEST} \\
\langle s \rangle \text{ the dog caught the cat } \langle /s \rangle \\
\text{Test probability is 0!}
\end{align*}
\]

\[
\begin{align*}
\text{TRAIN} \\
\langle s \rangle \text{ the cat sat on the mat } \langle /s \rangle \\
\langle s \rangle \text{ the dog sat on the cat } \langle /s \rangle \\
\langle s \rangle \text{ the cat caught the mouse } \langle /s \rangle \\

P(\text{caught} \mid \text{cat})_{\text{MLE}} &= \frac{1}{3} \\

P(\text{caught} \mid \text{dog})_{\text{MLE}} &= 0
\end{align*}
\]
Better Solution: Smoothing

TEST

<s> the dog caught the cat </s>

Test probability is > 0

TRAIN

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[
P(\text{caught} \mid \text{cat})_{\text{smooth}} = \frac{1 - x}{3}
\]

\[
P(\text{caught} \mid \text{dog})_{\text{smooth}} = \frac{x}{3}
\]
Data Preprocessing

- Split data into train/dev/test
- Get word frequencies on train and keep top V most frequent words
- Replace the remaining words with <unk>. This accounts for Out Of Vocabulary (OOV) words at test time.
Add One (Laplace) Smoothing

- Pretend that we saw every word once more than it did
- Add one to all all n-gram counts, seen or unseen

MLE estimate of probabilities:

\[ P(w_i \mid w_{i-1})_{MLE} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]
Add One (Laplace) Smoothing

- Pretend that we saw every word once more than it did
- Add one to all all n-gram counts, seen or unseen

**MLE estimate of probabilities:**

\[
P(w_i \mid w_{i-1})_{\text{MLE}} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}
\]

**Add-1 estimate of probabilities:**

\[
P(w_i \mid w_{i-1})_{\text{Add-1}} = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}
\]
Add One (Laplace) Smoothing

TEST

<s> the dog caught the cat </s>

TRAIN

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[
P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3}
\]

\[
P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0
\]
Add One (Laplace) Smoothing

\[
P(\text{caught} \mid \text{dog})_{\text{Add-1}} = \frac{c(\text{dog, caught}) + 1}{c(\text{dog}) + |v|} = \frac{0 + 1}{1 + 8} = 0.11
\]

\[
P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3}
\]

\[
P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0
\]
Add One (Laplace) Smoothing

\[
P(\text{caught} \mid \text{cat})_{\text{Add-1}} = \frac{c(\text{cat, caught}) + 1}{c(\text{cat}) + 1} = \frac{1 + 1}{3 + 8} = 0.18
\]

\[
P(\text{caught} \mid \text{dog})_{\text{Add-1}} = \frac{c(\text{dog, caught}) + 1}{c(\text{dog}) + 1} = \frac{0 + 1}{1 + 8} = 0.11
\]

\[
P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3}
\]

\[
P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0
\]
Add One (Laplace) Smoothing

\[
P(\text{caught} \mid \text{cat})_{\text{Add-1}} = \frac{c(\text{cat, caught}) + 1}{c(\text{cat}) + |v|} = \frac{1 + 1}{3 + 8} = 0.18 \\
P(\text{caught} \mid \text{dog})_{\text{Add-1}} = \frac{c(\text{dog, caught}) + 1}{c(\text{dog}) + |v|} = \frac{0 + 1}{1 + 8} = 0.11
\]

**TEST**

<s> the dog caught the cat </s>

**TRAIN**

<s> the cat sat on the mat </s>
<s> the dog sat on the cat </s>
<s> the cat caught the mouse </s>

\[
P(\text{caught} \mid \text{cat})_{\text{MLE}} = \frac{1}{3} \\
P(\text{caught} \mid \text{dog})_{\text{MLE}} = 0
\]
Add One (Laplace) Smoothing

- Add-1 smoothing can smooth excessively.
- Not used for language modeling.
Good Turing Smoothing

• Move probability mass from n-grams that occur $c + 1$ times to those that occur $c$ times.

• Move probability from n-grams occurring once to unseen n-grams

Let $n_c$ be the number of n-grams seen $c$ times. (Frequency of Frequency)

For every n-gram with count $c$, new count $c^* = (c + 1) \frac{n_{c+1}}{n_c}$
Good Turing Smoothing
(Josh Goodman)

You’re fishing in a lake and there are 8 species of fish
• carp, perch, whitefish, trout, salmon, eel, catfish, bass
You’re fishing in a lake and there are 8 species of fish
  • carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish
  • 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel
You’re fishing in a lake and there are 8 species of fish
  • carp, perch, whitefish, trout, salmon, eel, catfish, bass
You catch 18 fish
  • 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout?
You’re fishing in a lake and there are 8 species of fish
  • carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish
  • 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout? \( \frac{1}{18} \)
Good Turing Smoothing (Josh Goodman)

You’re fishing in a lake and there are 8 species of fish
  • carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish
  • 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout? \( \frac{1}{18} \)

How likely is the next species new?
Good Turing Smoothing  
(Josh Goodman)

You’re fishing in a lake and there are 8 species of fish
- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish
- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout? \( \frac{1}{18} \)

How likely is the next species new? \( \frac{n_1}{N} = \frac{3}{18} \)
Good Turing Smoothing (Josh Goodman)

You’re fishing in a lake and there are 8 species of fish
• carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish
• 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout? \( \frac{1}{18} \)

How likely is the next species new? \( \frac{n_1}{N} = \frac{3}{18} \)

In that case, what is the probability of the next species being trout?
You’re fishing in a lake and there are 8 species of fish
- carp, perch, whitefish, trout, salmon, eel, catfish, bass

You catch 18 fish
- 10 carp, 3 perch, 2 whitefish, 1 trout, salmon and eel

How likely is the next species a trout? \( \frac{1}{18} \)

How likely is the next species new? \( \frac{n_1}{N} = \frac{3}{18} \)

In that case, what is the probability of the next species being trout? \( < \frac{1}{18} \)
Good Turing Smoothing (Josh Goodman)

\[ P^{*}_{GT} \text{(things with zero frequency)} = \frac{n_1}{18} = \frac{3}{18} \quad , \quad c^* = (c + 1) \frac{n_{c+1}}{n_c} \]

Probability of Unseen (bass or catfish)  Probability seen once (bass or catfish)
Good Turing Smoothing
(Josh Goodman)

\[ P_{\ast GT}(\text{things with zero frequency}) = \frac{n_1}{18} = \frac{3}{18}, \quad c^* = (c + 1) \frac{n_{c+1}}{n_c} \]

Probability of Unseen (bass or catfish)  Probability seen once (bass or catfish)

\[ P_{MLE}(\text{bass}) = \frac{0}{18} \]

\[ P_{\ast GT}(\text{bass}) = \frac{3}{18} \]
Good Turing Smoothing
(Josh Goodman)

\[ P_{GT} \text{ (things with zero frequency)} = \frac{n_1}{18} = \frac{3}{18}, \quad c^* = (c + 1) \frac{n_{c+1}}{n_c} \]

Probability of Unseen (bass or catfish)  Probability seen once (bass or catfish)

\[ P_{MLE}(\text{bass}) = \frac{0}{18} \quad P_{MLE}(\text{trout}) = \frac{1}{18} \]
\[ P_{*GT}(\text{bass}) = \frac{3}{18} \]
\[ c^*(\text{trout}) = (1 + 1) \frac{1}{3} = \frac{2}{3} \]
\[ P_{GT} * (\text{trout}) = \frac{2}{3} \cdot \frac{1}{18} = \frac{1}{27} \]
Backoff and interpolation

If the context is less frequent, then use shorter contexts.

Backoff:

• If n-gram is frequent, use it.
• else, use lower order n-gram.

Interpolation

• Mix all n-gram contexts.
• Works better in practice than backoff.
Interpolation

Simple interpolation:

\[ P_{ interp}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i) \]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \)
Interpolation

Simple interpolation:

\[ \text{Pinterp}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i) \]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \)

Conditioning interpolation weights on context:

\[ \text{Pinterp}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 (w_{i-2}, w_{i-1})P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 (w_{i-2}, w_{i-1})P(w_i \mid w_{i-1}) + \lambda_3 (w_{i-2}, w_{i-1})P(w_i) \]
Interpolation

• Jeninek Mercer
• Absolute discounting
• Modified Kneser Ney
Performance on the Penn Treebank Dataset

- Good Turing 3-gram: 140
- Good Turing 5-gram: 147.5
- Keneser Ney 3-gram: 155
- Keneser Ney 5-gram: 162.5

Perplexity
Cache Language Models

• Work well for code.

• Intuition is that recently used words (variables) are likely to appear.

On the “Naturalness” of Buggy Code. (Ray et al., 2015)
Cache Language Models

• Work well for code.

• Intuition is that recently used words (variables) are likely to appear.

\[ P_{\text{CACHE}}(w_i \mid \text{history}) = \lambda P(w_i \mid w_{i-2}, w_{i-1}) + (1 - \lambda) \frac{c(w \in \text{history})}{|\text{history}|} \]

On the “Naturalness” of Buggy Code. (Ray et al., 2015)
Cache Language Models

Stefan Fiott, 2015 (Masters Thesis)
General approach for training n-gram models

• Collect n-gram counts
• Smooth for unseen n-grams
• Large number of parameters
• Fast probability computation
So Far…

- Standard n-gram
  - Model Size: X
  - Training Time: ✓
  - Query Time: ✓
Continuous Space Language Models
Feed Forward Neural Network Language Models
Motivation for Neural Network Language Models

TEST

<s> ibm was bought </s>

TRAIN

<s> apple was sold </s>

\[ c(\text{ibm was bought}) = 0 \]
Motivation for Neural Network Language Models

<table>
<thead>
<tr>
<th>TEST</th>
<th>TRAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;s&gt;</code> ibm was bought <code>&lt;/s&gt;</code></td>
<td><code>&lt;s&gt;</code> apple was sold <code>&lt;/s&gt;</code></td>
</tr>
</tbody>
</table>

\[ c(\text{ibm was bought}) = 0 \]

\[ \text{bought} \approx \text{sold} \]

\[ \text{ibm} \approx \text{apple} \]
Neural Probabilistic Language Models

Bengio et al., 2003

$P(\text{sat} \mid \text{the cat})$
Neural Probabilistic Language Models

$$P(\text{sat} \mid \text{the cat})$$
Neural Probabilistic Language Models

\[ P(\text{sat} \mid \text{the cat}) \]
Neural Probabilistic Language Models

\[ P(\text{sat} \mid \text{the cat}) \]

hidden: \[ F_{h1} = \varphi \left( \sum_{j=1}^{n-1} C^j D_{u_j} \right) \]

input embeddings

input words

\[ \text{the} \quad u_1 \]

\[ \text{cat} \quad u_2 \]
Neural Probabilistic Language Models

\[ P(\text{sat} \mid \text{the cat}) \]

hidden: \( \mathbf{F}^{h_1} = \varphi \left( \sum_{j=1}^{n-1} \mathbf{C}^i \mathbf{D}_{u_j} \right) \)

input embeddings

input words
P(sat | the cat)

hidden: $F^{h_2} = \varphi \left( MF^{h_1} \right)$

hidden: $F^{h_1} = \varphi \left( \sum_{j=1}^{n-1} C^j D_{u_j} \right)$

input embeddings

input words
Neural Probabilistic Language Models

\[ p(w | u) = \exp \left( D'_w F^{h_2} \right) \]

hidden: \[ F^{h_2} = \varphi (MF^{h_1}) \]

hidden: \[ F^{h_1} = \varphi \left( \sum_{j=1}^{n-1} C^j D_{u_j} \right) \]

input embeddings

input words

\[ P(\text{sat} | \text{the cat}) \]

\[ \text{the} \rightarrow u_1 \]

\[ \text{cat} \rightarrow u_2 \]
Neural Probabilistic Language Models

\[ \text{normalization: } Z(u) = \sum_{w' \in V} p(w' | u) \]

\[ p(w | u) = \exp \left( D'_w F^{h_2} \right) \]

hidden: \( F^{h_2} = \varphi \left( MF^{h_1} \right) \)

hidden: \( F^{h_1} = \varphi \left( \sum_{j=1}^{n-1} c^j D_{u_j} \right) \)

\[ \text{input embeddings} \]

\[ \text{input words} \]
Neural Probabilistic Language Models

P(sat | the cat)  

normalization:  \( Z(u) = \sum_{w' \in V} p(w' | u) \)

\[ p(w | u) = \exp \left( D'_w F^{h_2} \right) \]

hidden:  \( F^{h_2} = \varphi \left( MF^{h_1} \right) \)

hidden:  \( F^{h_1} = \varphi \left( \sum_{j=1}^{n-1} C^i D_{u_j} \right) \)

input embeddings

input words

Nair and Hinton, 2010

\( \varphi(x) = \max(0, x) \)
Neural Probabilistic Language Models

\[ p(w \mid u) = \exp \left( D_w^F h_2 \right) \]

\[ \text{normalization: } Z(u) = \sum_{w' \in V} p(w' \mid u) \]
Neural Probabilistic Language Models

\[
P(w \mid u) = \frac{\exp \left( D'_w F^{h_2} \right)}{Z(u)}
\]
Maximum Likelihood Training

$$\theta_{ML} = \arg \max_\theta P(w \mid u)$$
Maximum Likelihood Training

$$\theta_{ML} = \arg \max_{\theta} \log P(w \mid u)$$
Maximum Likelihood Training

$$\theta_{ML} = \arg \max_\theta \log \left( \frac{\exp \left( D'_w F^{h2} \right)}{Z(u)} \right)$$
Maximum Likelihood Training
Maximum Likelihood Training
Maximum Likelihood Training

- Perform stochastic gradient descent:
  - Compute $P(w|u)$ using Forward Propagation
  - Compute gradient with Backward Propagation
- Very slow training and decoding times
Maximum Likelihood Training

true distribution $P_{true}(w|u)$

observed data $uw$

training data

The cat sat
Noise Contrastive Estimation

The cat sat
Noise Contrastive Estimation

\[
\frac{1}{1+k} \rightarrow \text{true distribution } P_{\text{true}}(w|u) \rightarrow \text{observed data } u \rightarrow \text{training data}
\]

\[
\frac{k}{1+k} \rightarrow \text{noise data } u \rightarrow \text{noise distribution } q(w)
\]

The cat sat

The cat pig
The cat hat
The cat on
...

The cat
Noise Contrastive Estimation
Noise Contrastive Estimation

$k$ noise samples
Noise Contrastive Estimation

$k$ noise samples
Noise Contrastive Estimation

\[
\frac{1}{1 + k} \frac{1}{1 + k} \]

true distribution
\( P_{\text{true}}(w | u) \)

observed data
\( uw \)

training data
noise data
\( uw \)

noise distribution
\( q(w) \)

The cat sat
The cat pig
The cat hat
The cat on
...

Vaswani, Zhao, Fossum and Chiang, 2013
Noise Contrastive Estimation

\[
\frac{1}{1 + k} \quad \frac{1}{1 + k}
\]

true distribution \( P_{\text{true}}(w|u) \)

observed data \( uw \)

noise data \( uw \)

noise distribution \( q(w) \)

training data

The cat sat

The cat pig

The cat hat

The cat on

\[ P(w \text{ is true} \mid uw) \]

Vaswani, Zhao, Fossum and Chiang, 2013
Noise Contrastive Estimation

\[
\frac{1}{1+k} \quad \text{true distribution} \quad P_{\text{true}}(w|u) \quad \text{observed data} \quad uw \quad \text{training data} \quad k \\
1 + k \quad \text{noise data} \quad uw \quad \text{noise distribution} \quad q(w)
\]

The cat sat

\[
P(w \text{ is true} \mid uw) = \frac{P(w \mid u)}{P(w \mid u) + kq(w)}
\]

The cat pig
The cat hat
The cat on
...

Vaswani, Zhao, Fossum and Chiang, 2013
Noise Contrastive Estimation

\[ \frac{1}{1+k} \]

true distribution
\[ P_{\text{true}}(w|u) \]

observed data
\[ \mathbf{u}w \]

training data

noise data
\[ \mathbf{u}w \]

noise distribution
\[ q(w) \]

\[ \frac{k}{1+k} \]

The cat sat

The cat pig

The cat hat

The cat on

\[ P(w \text{ is true} \mid \mathbf{u}w) \]

\[ \frac{P(w \mid u)}{P(w \mid u) + kq(w)} \]

\[ P(\overline{w}_j \text{ is noise} \mid \mathbf{u}\overline{w}_j) \]

Vaswani, Zhao, Fossum and Chiang, 2013
Noise Contrastive Estimation

The cat sat
The cat pig
The cat hat
The cat on
...

\[
\begin{align*}
P(w \text{ is true } | \, &uw) \quad P(w \text{ is true } | \, &uw) \\
P(w \, | \, &u) \quad P(w \, | \, &u) \\
\frac{P(w \, | \, &u)}{P(w \, | \, &u) + kq(w)} \quad \frac{P(w \, | \, &u)}{P(w \, | \, &u) + kq(w)} \\
\end{align*}
\]

Vaswani, Zhao, Fossum and Chiang, 2013
Noise Contrastive Estimation

\[
\begin{array}{c}
\frac{1}{1+k} \\
\text{true distribution} \quad P_{\text{true}}(w|u) \\
\text{observed data} \quad uw \\
\text{noise data} \quad \bar{w}_j \\
\text{noise distribution} \quad q(w) \\
\frac{k}{1+k}
\end{array}
\]

\[
\begin{align*}
P(C = 0 \mid uw) &= \frac{P(w \mid u)}{P(w \mid u) + kq(w)} \\
&P(C = 1 \mid u\bar{w}_j) = \frac{q(\bar{w}_j)}{P(\bar{w}_j \mid u) + kq(w)}
\end{align*}
\]

Vaswani, Zhao, Fossum and Chiang, 2013

The cat sat

The cat pig

The cat hat

The cat on

...
\[ L = \log P(C = 0 \mid \mathbf{u}w) + \sum_{j=1}^{k} \log P(C = 1 \mid \mathbf{u}\bar{w}_j) \]

\[ = \log \frac{P(w \mid \mathbf{u})}{P(w \mid \mathbf{u}) + kq(w)} + \sum_{j=1}^{k} \log \frac{q(\bar{w}_j)}{P(\bar{w}_j \mid \mathbf{u}) + kq(w)} \]
For each training example \((u, w)\):

- generate \(k\) noise samples
- train model to classify real examples and noise samples
Advantages of NCE

• Much Faster training time.

• You can significantly speed up test time by encouraging the model to have a normalization constant of 1.
Better perplexity

- Log Bilinear
- Kneser Ney
- NPLM with MLE
- NPLM with NCE
- RNN

Perplexity

- 120
- 127.5
- 135
- 142.5
- 150
NNLM on the Android Dataset

- Dataset of Android Apps
- 11 million tokens
- 90/10 split for Training set and Development
- 50k Vocabulary
- Cross entropy of 2.95 on Dev.

Results from Saheel Godane
So Far…

<table>
<thead>
<tr>
<th></th>
<th>Standard n-gram</th>
<th>NPLM MLE</th>
<th>NPLM NCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Size</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Training Time</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Query Time</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Faster training times

![Graph showing training time vs vocabulary size for different models.]

- CSLM
- NPLM (k=1000)
- NPLM (k=100)
- NPLM (k=10)
## Nearest Neighbors

<table>
<thead>
<tr>
<th>doctor</th>
<th>hospital</th>
<th>apple</th>
<th>bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>physician</td>
<td>medical</td>
<td>ibm</td>
<td>purchased</td>
</tr>
<tr>
<td>dentist</td>
<td>clinic</td>
<td>intel</td>
<td>sold</td>
</tr>
<tr>
<td>pharmacist</td>
<td>hospitals</td>
<td>seagate</td>
<td>procured</td>
</tr>
<tr>
<td>psychologist</td>
<td>hospice</td>
<td>hp</td>
<td>scooped</td>
</tr>
<tr>
<td>neurosurgeon</td>
<td>mortuary</td>
<td>netflix</td>
<td>cashed</td>
</tr>
<tr>
<td>veterinarian</td>
<td>sanatorium</td>
<td>kmart</td>
<td>reaped</td>
</tr>
<tr>
<td>physiotherapist</td>
<td>orphanage</td>
<td>tivo</td>
<td>fetched</td>
</tr>
</tbody>
</table>
2-dim projection
Perplexity is robust to learning rate
Other approaches

• Class based softmax

• Hierarchical Softmax

• Automatically discovering the right size of the network (Murray and Chiang, 2015)

• NCE has been used in modeling Code and Language (Allamanis et al., 2015)
Recurrent Neural Network Language Models
Feed Forward NNLM

The Cat

Context Vector

Sat

The

Cat
Recurrent NNLMs
Recurrent NNLMs

The cat sat on a mat

Context Vector

output

input
Recurrent NNLMs

The cat sat on a

cat

Context Vector

The cat sat on a
Recurrent NNLMs

The cat sat on a mat.
Recurrent NNLMs

The cat sat on a mat.
Recurrent NNLMs

The cat sat on a mat.

A mouse caught a...
Recurrent NNLMs
Recurrent NNLMs

\[ \sigma(x) \]

\[ P(a) \]
Recurrent NNLMs

\[ \sigma(x) \]

\[ P(a) \]

0.25

-0.1

on
Recurrent NNLMs

\[ \sigma(x) \]

\[ P(a) \]

\[ \begin{array}{c}
0.25 \\
-0.1 \\
0.54 \\
0.54
\end{array} \]
Recurrent NNLMs

\[ P(a|\text{context}) = 0.9 \]

\[ \tanh(x) \]
Training Recurrent NNLMs

Forward Propagation

\[ P(cat) \rightarrow P(sat) \rightarrow P(on) \rightarrow P(a) \rightarrow P(mat) \]

The cat sat on a

TIME
Training Recurrent NNLMs

Back Propagation through time

\[ \frac{\partial \log P(\text{cat})}{\partial \theta} \]
\[ \frac{\partial \log P(\text{sat})}{\partial \theta} \]
\[ \frac{\partial \log P(\text{on})}{\partial \theta} \]
\[ \frac{\partial \log P(\text{a})}{\partial \theta} \]
\[ \frac{\partial \log P(\text{mat})}{\partial \theta} \]

Context Vector
Context Vector
Context Vector
Context Vector
Context Vector

The cat sat on a

TIME
Vanishing gradients

Back Propagation through time

\[
\begin{align*}
\frac{\partial \log P(\text{cat})}{\partial \theta} & \to \frac{\partial \log P(\text{sat})}{\partial \theta} & \frac{\partial \log P(\text{on})}{\partial \theta} & \frac{\partial \log P(a)}{\partial \theta} & \frac{\partial \log P(\text{mat})}{\partial \theta}
\end{align*}
\]

The cat sat on a
Vanishing gradients

\[
\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))
\]

Back Propagation through time

\[
\frac{\partial \log P(\text{cat})}{\partial \theta} \quad \frac{\partial \log P(\text{sat})}{\partial \theta} \quad \frac{\partial \log P(\text{on})}{\partial \theta} \quad \frac{\partial \log P(\text{a})}{\partial \theta} \quad \frac{\partial \log P(\text{mat})}{\partial \theta}
\]

The cat sat on a
Vanishing gradients

\[
\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))
\]

Back Propagation through time

\[
\frac{\partial \log P(\text{cat})}{\partial \theta} \quad \frac{\partial \log P(\text{sat})}{\partial \theta} \quad \frac{\partial \log P(\text{on})}{\partial \theta} \quad \frac{\partial \log P(a)}{\partial \theta} \quad \frac{\partial \log P(\text{mat})}{\partial \theta}
\]

\[
\sigma(x)(1 - \sigma(x)) \quad \ldots \quad \sigma(x)(1 - \sigma(x))
\]
Solution: Long Short Term Memory

\[ P(a) \]

-0.1

0.25

0.149

-0.1

on
Solution: Long Short Term Memory
Solution: Long Short Term Memory

\[ P(a) \]

\[ h_t \]

\[ c_t \]

\[ c_{t-1} \]

\[ f_t \]

\[ \sigma \]

\[ h_{t-1} \]

\[ c'_t \]

\[ \tanh \]

\[ \sigma \]

\[ h_{t-1} \]

\[ o_t \]

\[ i_t \]

\[ \tanh \]

\[ \text{LSTM BLOCK} \]
Forward Propagation in LSTM Block

\[ P(a) \]

\[
P_t = f_t c_{t-1} + h_{t-1}
\]

\[
\sigma_t = \sigma(h_t)
\]

\[
\tan_t = \tanh(c_t)
\]

\[
h_t = o_t \sigma_t + h_{t-1}
\]

\[
c_t = i_t \tan_t + c_{t-1}
\]

\[
f_t = \sigma(f_t)
\]

\[
i_t = \sigma(i_t)
\]

\[
c_{t-1} = \sigma(c_{t-1})
\]

\[
h_{t-1} = \sigma(h_{t-1})
\]
Forward Propagation in LSTM Block

\[ P(a) \]

\[ h_t \]

\[ c_t \]

\[ c_{t-1} \]

\[ h_{t-1} \]

inputs

outputs

on
I. Forgetting memory

\[ P(a) \]

\[ c_{t-1} \]

\[ h_t \]

\[ c_t \]

\[ h_{t-1} \]

on
I. Forgetting memory

Forgetting memory

\[ P(a) \]

\[ h_t \]

\[ c_{t-1} \]

Forget Gate

\[ f_t \]

\[ \sigma \]

\[ h_{t-1} \]

\[ c_t \]
1. Forgetting memory

\[ P(a) \]

\[ h_t \]

\[ c_t \]

Forget Gate

\[ f_t \]

\[ h_{t-1} \]

\[ c_{t-1} \]
2. Adding new memories

\[ P(a) \]

\[ h_t \]

\[ c_{t-1} \]

\[ f_t \]

\[ \sigma \]

\[ h_{t-1} \]

\[ c_t \]

\[ \text{on} \]
2. Adding new memories

\[ P(a) \]

\[ h_t \]

\[ c_t \]

\[ c_{t-1} \times \]

\[ f_t \sigma \]

\[ h_{t-1} \]

\[ c'_t \text{ tanh} \]

Input Gate
2. Adding new memories

\[ P(a) \]

\[ h_t \]

\[ c_t \]

\[ c_{t-1} \]

\[ f_t \]

\[ \sigma \]

\[ h_{t-1} \]

\[ c'_t \]

\[ \text{tanh} \]

\[ i_t \]

\[ \sigma \]

\[ h_{t-1} \]

Input Gate
2. Adding new memories

$$P(a)$$

$$h_t$$

Input Gate

$$f_t$$

$$h_{t-1}$$

$$c_{t-1}$$

$$c_t$$

$$i_t$$

$$c_t'$$

$$\sigma$$

$$\sigma$$

$$\tanh$$

$$\times$$
3. Updating the memory

\[ P(a) \]

\[ h_t \]

\[ c_{t-1} \rightarrow \times \rightarrow c_t \]

\[ f_t \rightarrow \sigma \rightarrow h_{t-1} \]

\[ c'_t \rightarrow \times \rightarrow c_t \]

\[ i_t \rightarrow \sigma \rightarrow h_{t-1} \]

\[ \text{on} \]
3. Updating the memory

\[ P(a) \]

\[ h_t \]

\[
\begin{align*}
    c_t &= c_{t-1} \\
    f_t &= \sigma \left( h_{t-1} \right) \\
    i_t &= \sigma \left( h_{t-1} \right) \\
    c_t' &= \tanh \left( f_t c_{t-1} + i_t c_t \right) \\
    h_t &= h_{t-1} + c_t' \times i_t \\
    \end{align*}
\]
3. Updating the memory

\[ P(a) \]

\[ h_t \]

Cell Value

\[ \sigma \]

\[ f_t \]

\[ c_{t-1} \]

\[ h_{t-1} \]

\[ c_t \]

\[ c_t \]

\[ i_t \]

\[ h_{t-1} \]

\[ h_{t-1} \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]

\[ \sigma \]
4. Calculating hidden state

\[ P(a) \]

\[ h_t \]

\[ c_t \]

\[ f_t \]

\[ i_t \]

\[ h_{t-1} \]

\[ c_{t-1} \]

\[ h_{t-1} \]

\[ h_{t-1} \]
4. Calculating hidden state

$$P(a)$$

\[ h_t = \tanh(c_t - 1 + h_{t-1}) \]

\[ c_t = \sigma(f_t \times h_{t-1}) \]

\[ i_t = \sigma(c'_t) \]

\[ c_t = c_t \times c'_t \]

\[ f_t = \sigma(f_t) \]

\[ i_t = \sigma(i_t) \]
4. Calculating hidden state

\[ P(a) \]

\[ h_t = \sigma \]

\[ h_t' \]

\[ c_t \]

\[ f_t \]

\[ c_t' \]

\[ \sigma \]

\[ \tanh \]

\[ o_t \]

\[ h_{t-1} \]

\[ c_{t-1} \]

\[ c_t \]
5. Computing probabilities

\[ P(a) \]

\[
\begin{align*}
    h_t &= \sigma(h_{t-1}) \\
    c_t &= f_t \cdot c_{t-1} + i_t \cdot c'_t \\
    o_t &= \sigma(h_t) \\
    h_t &= \tanh(c_t) \\
    c_{t-1} &= \sigma(h_{t-1}) \\
    h_{t-1} &= \tanh(h_t) \\
    \end{align*}
\]
No more vanishing gradient

$$\frac{\partial \log P(a)}{\partial c_{t-1}}$$

\[ h_t \]

\[ o_t \]

\[ c_{t-1} \]

\[ f_t \]
No more vanishing gradient
No more vanishing gradient
No more vanishing gradient
Improved Perplexity on Penn Treebank

Log Bilinear
Kneser Ney
NNLM with MLE
NNLM with NCE
RNN
LSTM

Perplexity

Zaremba et al., 2014
LSTM Successes in NLP

• Language Modeling
• Neural Machine Translation
• Natural Language Parsing
• Tagging
Large Vocabulary LSTMs

• Language Modeling

• Neural Machine Translation

• Natural Language Parsing
Visualizing Simple LSTMs

Joint work with Jon May
Encoder Decoder Framework for Sequence Sequence to Learning

Language Modeling:
Input: English
Output: English

cat sat on a mat

the cat sat on a
Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation
Input: Esperanto
Output: English

acxeto kato sur mato
the cat sat on a mat
Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation
Input: Esperanto
Output: English

acxeto kato sur mato
ENCODER

cat sat on a mat
the cat sat on a mat
Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation
Input: Esperanto
Output: English

ACXETO KATO SUR MATO
ENCODER

Cat Sat On A Mat
DECODER
Encoder Decoder Framework for Sequence Sequence to Learning

Machine Translation
Input: Esperanto
Output: English

Encoder Decoder Framework:

**Encoder**
- acxeto
- kato
- sur
- mato

**Decoder**
- the
- cat
- sat
- on
- a
- mat

**Sentence Vector**

Input: Esperanto
Output: English
Visualization Approach

- Train on input/output sequences
- Look at LSTM internals for test input/output sequences
Counting a Symbol

Input: 

\[ a \ a \ a \ a \ a \ \]

\[ a \]

\[ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \]

Output: 

\[ a \ a \ a \ a \ a \]

\[ a \]

\[ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \ a \]

Input \#(a) = Output \#(a)
Counting a Symbol

Input: a a a a a

Output: a a a a a

Input #(a) = Output #(a)
Counting a Symbol

Input: a a a a a a

Output: a a a a a a

Input #(a) = Output #(a)
Counting:

Cell Values

124
Counting:

Cell Values

Time

cell magnitude

-25
-20
-15
-10
-5
0
5

0 10 20 30 40 50

Time

125
Counting:

Cell Values

126
Counting:

Cell Values

Time

127
Counting a or b

Input:

a a a a a
b
a a a a a a a a a
b b b b b b b b b b b

Input #(a) = Output #(a)
OR
Input #(b) = Output #(b)

Output:

a a a a a
b
a a a a a a a a a
b b b b b b b b b b b
Counting a or b

Counting a

Counting b
Counting a or b

Counting a

Time

Counting b

Time
Counting a or b

Time

Counting a

Counting b

Time
Counting a or b

Output embeddings

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>&lt;/s&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>-5.25</td>
<td>-3.5</td>
<td>-1.75</td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a and b are represented by green bars, while </s> is represented by a red bar.
Counting a or b

Output embeddings

’a’ and ‘b’ have flipped embeddings
Counting a or b

Counting a

Output embeddings

Time

hidden states values

magnitude
Counting a or b

\[ p(\text{symbol}) \propto \text{embedding}(\text{symbol}) h^T \]
Counting a or b

Counting a

‘a’ region

Output embeddings

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.25</td>
</tr>
<tr>
<td>3.5</td>
</tr>
<tr>
<td>1.75</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>-1.75</td>
</tr>
<tr>
<td>-3.5</td>
</tr>
<tr>
<td>-5.25</td>
</tr>
</tbody>
</table>

-5.25 -3.5 -1.75 0 1.75 3.5 5.25
Counting a

'\text{a}' region

\text{a} \quad \text{b} \quad \langle/s\rangle

Output embeddings

\begin{tabular}{llll}
\text{a} & \text{b} & \langle/s\rangle \\
-5.25 & -3.5 & -1.75 & 0 & 1.75 & 3.5 & 5.25
\end{tabular}
Counting a or b

Output embeddings

Counting b

Time

Output embeddings

Time
Counting a or b

Counting b

‘b’ region

Output embeddings

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>&lt;/s&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5.25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-3.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
Counting a or b

Counting a

'a' region

</s>

'а' region

</s>
Counting a or b

Counting b

‘b’ region

Symbol probabilities from Decoder

P(b)
P(<s>)
P(a)

decoding time

Probability

magnitude

hidden states values

1.0
0.5
0.0
-0.5
-1.0

0.0
0.2
0.4
0.6
0.8
1.0

0
5
10
15
20
25
30
35
40

136
Counting log(a)

Input:

a a a a a

a a a a a a a a

Output:

5*log(5) number of a’s

5*log(9) number of a’s
Counting $\log(a)$
Counting log(a)
Counting $\log(a)$
Counting $\log(a)$
Downloadable Tools
Downloadable Tools

http://www.speech.sri.com/projects/srilm/
Feed Forward Neural Language Model

http://nlg.isi.edu/software/nplm/

Neural Probabilistic Language Model Toolkit

NPLM is a toolkit for training and using feedforward neural language models (Bengio, 2003). It is fast even for large vocabularies (100k or more): a model can be trained on a billion words of data in about a week, and can be queried in about 40 μs, which is usable inside a decoder for machine translation.

NPLM is written by Ashish Vaswani, with contributions from David Chiang and Victoria Fossum. It is distributed under the MIT open-source license.

- Latest stable version: nplm-0.3.tar.gz
Faster RNNLM (HS/NCE) toolkit

In a nutshell, the goal of this project is to create an rnnlm implementation that can be trained on huge datasets (several billions of words) and very large vocabularies (several hundred thousands) and used in real-world ASR and MT problems. Besides, to achieve better results this implementation supports such praised setups as ReLU+DiagonalInitialization [1], GRU [2], NCE [3], and RMSProp [4].

How fast is it? Well, on One Billion Word Benchmark [8] and 3.3GHz CPU the program with standard parameters (sigmoid hidden layer of size 256 and hierarchical softmax) processes more than 250k words per second in 8 threads, i.e. 15 millions of words per minute. As a result an epoch takes less than one hour. Check Experiments section for more numbers and figures.

The distribution includes ./run_benchmark.sh script to compare training speed on your machine among several implementations. The scripts downloads Penn Tree Bank corpus and trains four models: Mikolov’s rnnlm with class-based softmax from rnnlm.org, Edrenkin’s rnnlm with HS from Kaldi project, faster-rnnlm with hierarchical softmax, and faster-rnnlm with noise contrastive estimation. Note that while models with class-based softmax can achieve a little lower entropy then models hierarchical softmax, their training is infeasible for large vocabularies. On the other hand, NCE speed doesn’t depend on the size of the vocabulary. What's more, models trained with NCE is comparable with class-based models in terms of resulting entropy.

https://github.com/yandex/faster-rnnlm
Training LSTMs

http://nlg.isi.edu/software/EUREKA.tar.gz

NCE based training

https://github.com/isinlp/Zoph_RNN

GPU based training
Training LSTMs

http://nlg.isi.edu/software/EUREKA.tar.gz

NCE based training

https://github.com/isi-nlp/Zoph_RNN

GPU based training
RNNLM Toolkit

http://rnnlm.org/

Python Easy Neural Network Extruder

This is a library that tries to make creating neural networks as easy as possible by being as similar to Python/NumPy as possible. It borrows heavily from Theano and CNN. If you find it useful, or want to help improve it, please let me (David Chiang) know!

You need to have NumPy (tested with 1.9).

- Having SciPy (tested with 0.14) helps.
- There is also experimental GPU support, which requires the development version of libgpuarray.

The library is in pure Python and doesn't need to be built. Just make sure that the penne package directory (the one containing __init__.py) is in your Python path and import the package (note, don't import * from both numpy and penne, as they have many symbols in common):

Basic examples - very useful for quick introduction (training, evaluation, hyperparameter selection, simple n-best list rescoring, etc.) - 35MB
Advanced examples - includes large scale experiments with speech lattices (n-best list rescoring, ...) - 235MB, by Stefan Krombink
Slides from my presentation at Google - pdf
RNNLM is now integrated into Kaldi toolkit! Check this.
Penne

https://bitbucket.org/ndnlp/penne

Python Easy Neural Network Extruder

This is a library that tries to make creating neural networks as easy as possible by being as similar to Python/NumPy as possible. It borrows heavily from Theano and CNN. If you find it useful, or want to help improve it, please let me (David Chiang) know!

You need to have NumPy (tested with 1.9).

- Having SciPy (tested with 0.14) helps.
- There is also experimental GPU support, which requires the development version of libgpuarray.

The library is in pure Python and doesn’t need to be built. Just make sure that the penne package directory (the one containing __init__.py) is in your Python path and import the package (note, don’t import * from both numpy and penne, as they have many symbols in common):
Contributors and Collaborators
Thanks!