Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Language Models

Tobias Scheffer

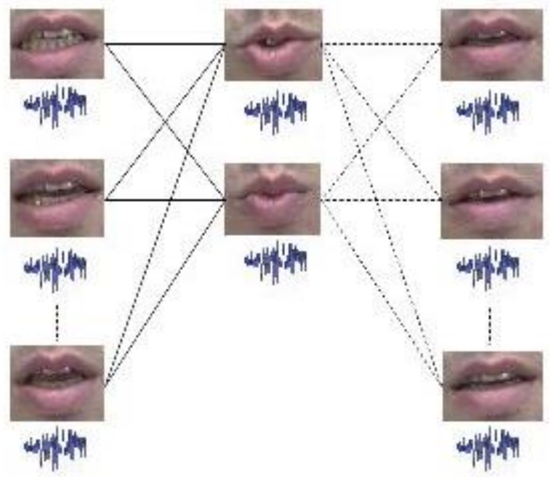
Stochastic Language Models

- A stochastic language model is a probability distribution over words.
- Given a string of words, $w_1, \ldots, w_m$, a language model assigns a probability $p(w_1, \ldots, w_m)$.
- “Words” $w_1, \ldots, w_m$ can be words, letters, keystrokes.
- Useful for many (most) NLP tasks.
  - Speech recognition,
  - Spell checking, auto-corrrect, auto-complete,
  - Machine translation,
  - Text classification,
  - Many non-standard NLP problems.

Language Models: Why?

- Speech recognition
  - Acoustic model + language model.

Acoustic model



→ Most likely words

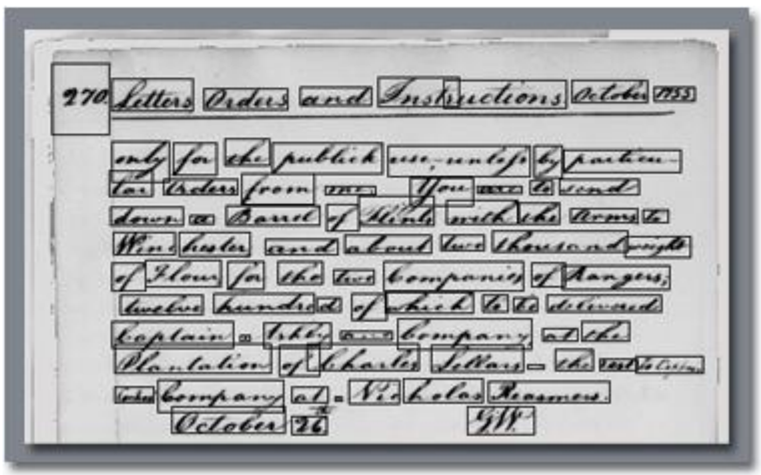
Language model

$$P(\text{I saw a tree}) = \cdots$$
$$P(\text{Eyes awe entry}) = \cdots$$

Language Models: Why?

- Hand-written text recognition:
  - Pattern-recognition model + language model

Pattern recognition model



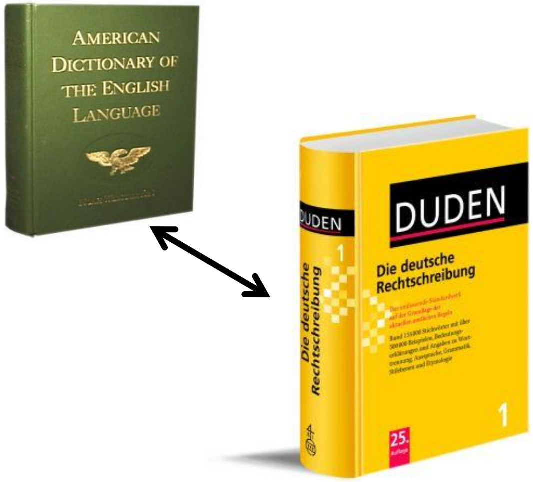
Language model

$$P(\text{I saw a tree}) = \cdots$$
$$P(\text{1 saw a free}) = \cdots$$

Language Models: Why?

- Machine translation
  - Translation model + language model

Translation model



Language model

$$P(\text{I saw a tree}) = \cdots$$
$$P(\text{I saw one tree}) = \cdots$$

Language Models: Why?

- Auto-correct, auto-complete
  - Keyboard model + language model

Keyboard model

Language model

$$P(\text{I saw a tree}) = \cdots$$
$$P(\text{O saq a trew}) = \cdots$$

The n-Gram Model

- Basic tool for language modeling.
- Based on the Markov assumption of order $n-1$:
  - $p(X_t|X_{t-1},\ldots,X_1) = p(X_t|X_{t-1},\ldots,X_{t-n+1})$

- n-gram model:

$$
\begin{aligned}
&p(X_1,\ldots,X_T) \\
&= p(X_1)\ldots p(X_T|X_{T-1},\ldots,X_{T-n+1}) \\
&= P(X_1)\ldots p(X_{n-1}|X_{n-2},\ldots,X_1)\prod_{t=n}^{T} p(X_t|X_{t-1},\ldots,X_{t-n+1})
\end{aligned}
$$

Categorical distributions

The n-Gram Model

- See lecture on "basic models".
- Inference: determine $p(w_1, \ldots, w_T)$.
- Parameter estimation:
  - Determine $\theta_{x_1,\ldots,x_m}$ by counting occurances.
  - Laplace smooting for regularization.
  - Laplace smooting assigns positive probability to unseen n-grams.

Implementing the n-Gram Model

- Probabilities of word sequences decrease exponentially in T.
- Fall below floating-point precision quickly.
- Instead of $\theta_{x_1,\ldots,x_m}$, use parameters $\theta_{x_m|x_1,\ldots,x_m}$.
- Do all calculations using logarithmic values.
  - $\log \prod_t p(w_t|w_{t-1},,,,.w_{t+n-1}) = \sum_t \log p(w_t|w_{t-1},,,,.w_{t+n-1})$

n-Gram Model: Long-Term Dependencies

- "The computer that I just installed the new operating system on crashed".
- Small values of n:
  - Better estimates of n-gram probabilities but lack of context.
- Increasing n to 4, 5, 6, …
  - There will be increasingly many combinations of word n-grams that have never been observed.

Linear Interpolation

- Simple interpolation:
$$P(w_n|w_{n-1},\dots,w_1)$$
$$= \lambda_n P_n(w_n|w_{n-1},\dots,w_1) + \dots + \lambda_2 P_2(w_n|w_{n-1}) + \lambda_1 P_1(w_n)$$
- Context-dependent interpolation weights:
$$P(w_n|w_{n-1},\dots,w_1)$$
$$= \lambda_n(w_{n-1},w_{n-2}) P_n(w_n|w_{n-1},\dots,w_1) + \cdots$$
$$+ \lambda_2(w_{n-1},w_{n-2}) P_2(w_n|w_{n-1}) + \lambda_1(w_{n-1},w_{n-2}) P_1(w_n)$$

Linear Interpolation

- Setting the interpolation coefficients.
- Split training data into 80% training and 20% tuning data.
- Estimate parameters $\theta_{x_1,\ldots,x_m}$ on training part.
- Then, tune parameters λ_i to maximize likelihood of the tuning data.

Resources

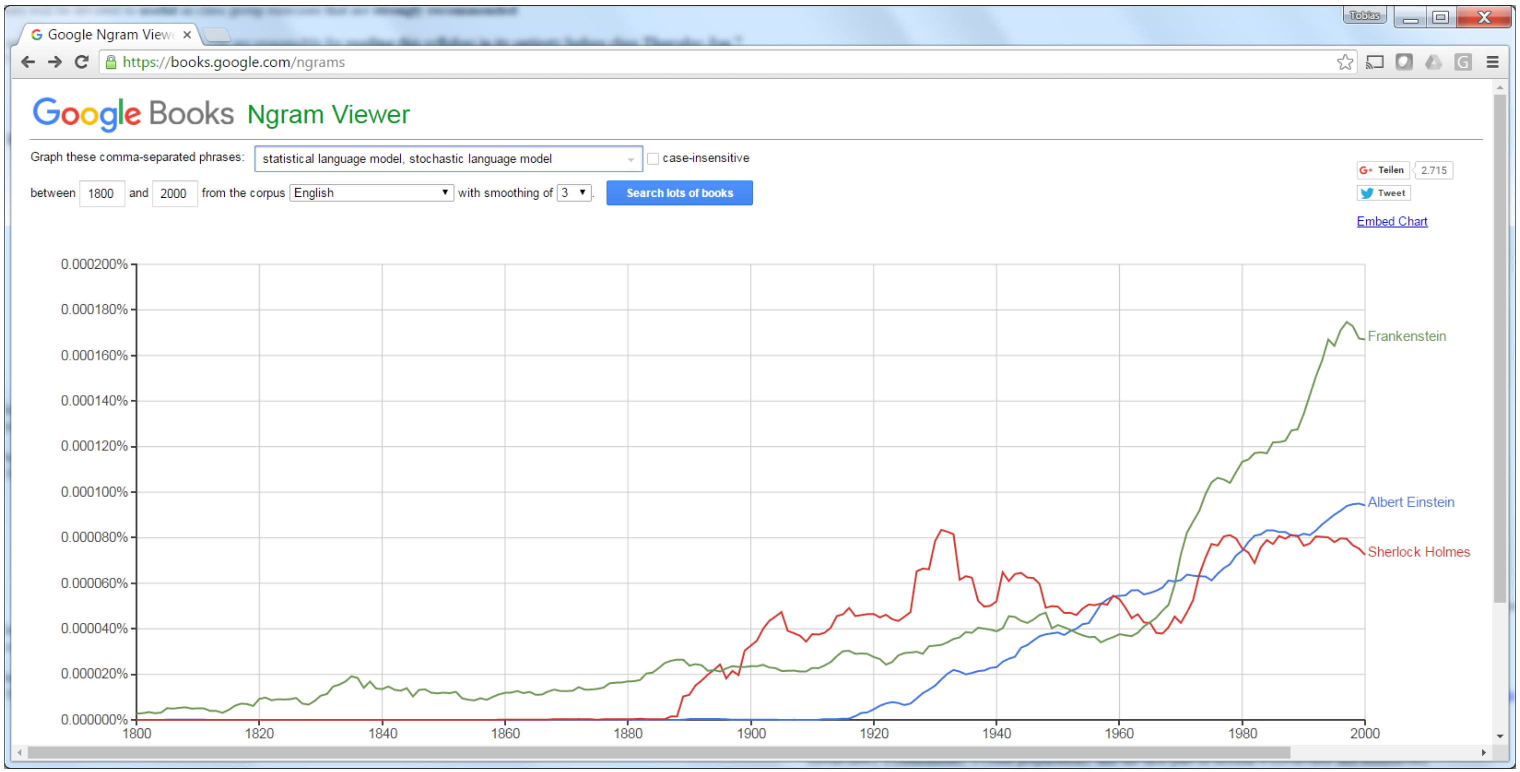
- Google has published a large n-gram corpus.
- http://googleresearch.blogspot.de/2006/08/all-our-n-gram-are-belong-to-you.html

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:    1,024,908,267,229
Number of sentences:    95,119,665,584
Number of unigrams:         13,588,391
Number of bigrams:         314,843,401
Number of trigrams:        977,069,902
Number of fourgrams:     1,313,818,354
Number of fivegrams:     1,176,470,663
```

Resources

- Google has published data on the evolution of n-gram counts over time from Google Books.



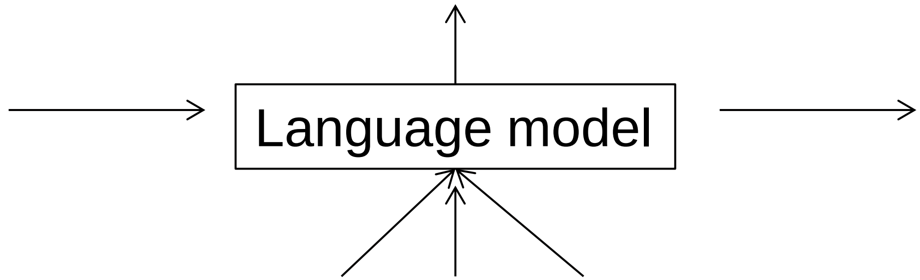
Limitations of the n-Gram Model

- Capability of reflecting contect limited to n words.
- Independent parameter for each n-word combination.
- Semantically similar terms have independent parameters.
- Idea: Improve genralization by treating semantically similar words in a similar way.
  - Linear interpolation is an attempt at improving generalization.
  - Also, n-gram class models are an attempt at improving generalization.
  - Continuous-space language models.

Continuous-Space Language Models

- Predict w_t based on features extracted from $w_{t-1}, \dots, w_{t-n+1}$.
- Maximize $\prod_t P(w_t | w_{t-1}, \dots, w_{t-n+1})$

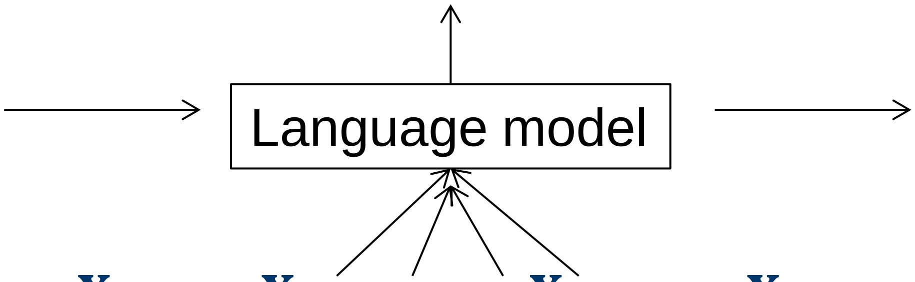
- Output $\dots \mathbf{x}_{t-1} \quad \mathbf{x}_t \quad \mathbf{x}_{t+1} \dots \mathbf{x}_T$



- Input $\mathbf{x}_1 \quad \dots \mathbf{x}_{t-n+1} \quad \dots \quad \mathbf{x}_{t-1} \dots \mathbf{x}_T$.

Continuous-Space Language Models

- Predict w_t based on features extracted from $w_{t-n}, \ldots, w_{t+n}$.
- Maximize $\prod_t \prod_{\substack{j=-n\ldots+n \\ j \neq 0}} P(w_t | w_{t+j})$
- Model looks into the "future".
- Output $\ldots \mathbf{x}_{t-1} \quad \mathbf{x}_t \quad \mathbf{x}_{t+1} \ldots \mathbf{x}_T$

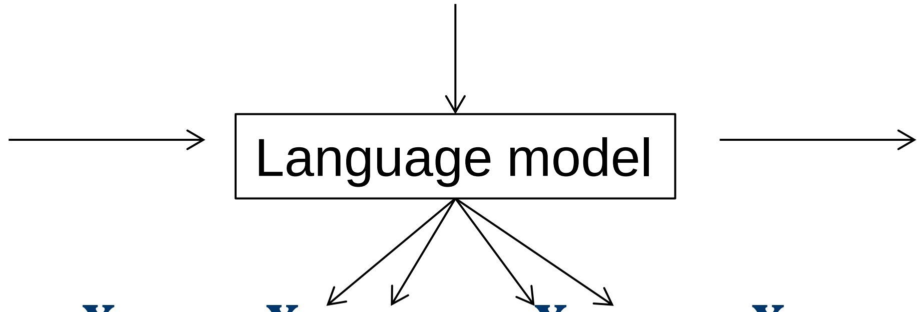


- Input $\mathbf{x}_1 \quad \ldots \quad \mathbf{x}_{t-n} \quad \ldots \quad \mathbf{x}_{t-n} \ldots \mathbf{x}_T$.

Continuous-Space Language Models

- Skip-gram model: Predict $w_{t-n}, \ldots, w_{t-1}, w_{t+1}, w_{t+n}$ based on features extracted from w_t.
- Maximize $\prod_t \prod_{\substack{j=-n\ldots+n \\ j\neq 0}} P(w_{t+j}|w_t)$

- Output $\ldots \mathbf{x}_{t-1} \quad \mathbf{x}_t \quad \mathbf{x}_{t+1} \ldots \mathbf{x}_T$



- Input $\mathbf{x}_1 \quad \ldots \quad \mathbf{x}_{t-n} \quad \ldots \quad \mathbf{x}_{t-n} \ldots \mathbf{x}_T.$

Continuous-Space Language Models

- For a word-level language model, words are usually represented by one-hot coded feature vector.
  - $\mathbf{x}_t = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ ← Hippopotamus

Dictionary: $\begin{bmatrix} \text{Aardvark} \\ \vdots \\ \text{Hippie} \\ \text{Hippopotamus} \\ \text{Hipster} \\ \vdots \\ \text{Zyzzogeton} \end{bmatrix}$

Continuous-Space Language Models

- For a letter-level language model, latters are usually represented by one-hot coded feature vector.
  - $$\mathbf{x}_t = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

“ ”

Dictionary: $\begin{bmatrix} \text{A} \\ \vdots \\ \text{Z} \\ \text{" "} \\ ! \\ \vdots \\ 9 \end{bmatrix}$

Continuous-Space Language Models

- Continuous-space language models are usually implemented by neural networks.
- Forward propagation leads to activation of the hidden units.
- The activation of the hidden units creates the embedding (feature representation) $\phi(\mathbf{x}_t)$ of each word.
- This feature representation $\phi(\mathbf{x}_t)$ is useful for many tasks.
- Words that occur in similar contexts have similar feature representations.

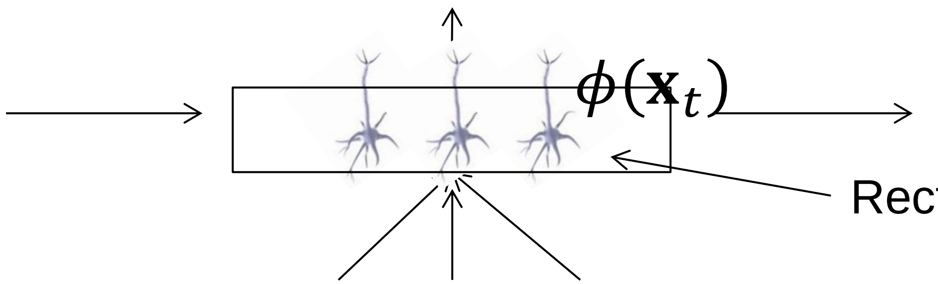
Neural Language Model

Also uses Markov assumption of order $n-1$!

$$\begin{pmatrix} 0.001 \\ \vdots \\ 0.6 \\ 0.1 \\ 0.02 \\ \vdots \\ 0 \end{pmatrix} =$$

Softmax output layer

- Output $\ldots \mathbf{x}_{t-1} \quad \mathbf{x}_t \quad \mathbf{x}_{t+1} \ldots \mathbf{x}_T$



$\phi(\mathbf{x}_t)$

Rectified linear units

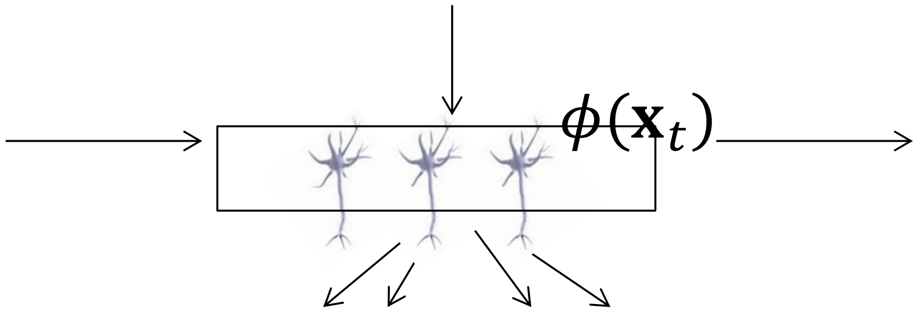
- Input $\mathbf{x}_1 \quad \ldots \mathbf{x}_{t-n+1} \quad \ldots \quad \mathbf{x}_{t-1} \quad \ldots \mathbf{x}_T.$

$$= \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Neural Skip-Gram Language Model

$$\begin{pmatrix}0\\ \vdots\\ 0\\ 1\\ 0\\ \vdots\\ 0\end{pmatrix} =$$

- Output $\ldots\mathbf{x}_{t-1}\quad \mathbf{x}_t\quad \mathbf{x}_{t+1}\ \ldots\ \mathbf{x}_T$



$\phi(\mathbf{x}_t)$

- Input $\mathbf{x}_1\quad \ldots\mathbf{x}_{t-n+1}\ \ldots\ \mathbf{x}_{t-n+1}\ \ldots\ \mathbf{x}_T.$

$$= \begin{pmatrix}0.001\\ \vdots\\ 0.6\\ 0.1\\ 0.02\\ \vdots\\ 0\end{pmatrix} \qquad = \begin{pmatrix}0.01\\ \vdots\\ 0.1\\ 0.2\\ 0.8\\ \vdots\\ 0\end{pmatrix}$$

Language Model with Context Memory

- Additionally provide paragraph ID as input.
- Allows model to lean an embedding for paragraphs in addition to the embedding for words.
- Paragraph vector: one-hot endocing of paragraph ID.
- $\mathbf{p}_t = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

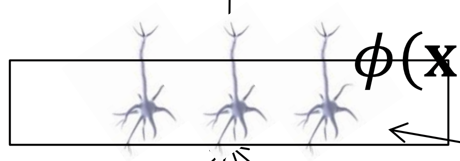
Paragraph 17

Neural Language Model with Context

$$\begin{pmatrix} 0.001 \\ \vdots \\ 0.6 \\ 0.1 \\ 0.02 \\ \vdots \\ 0 \end{pmatrix} =$$

Softmax output layer

- Output $\ldots \mathbf{x}_{t-1} \quad \mathbf{x}_t \quad \mathbf{x}_{t+1} \ldots \mathbf{x}_T$



$\phi(\mathbf{x}_t)$

Rectified linear units

- Input $(\mathbf{x}_1, \mathbf{p}_1) \ldots (\mathbf{x}_{t-n+1}, \mathbf{p}_{t-n+1}) \ldots (\mathbf{x}_{t-1}, \mathbf{p}_{t-1}) \ldots \mathbf{x}_T.$

$$= \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Using Word Embeddings

- Feature vectors $\phi(\mathbf{x}_t)$ can replace one-hot coding of words in many applications.
  - Text classification (aggregate over text),
  - Sentiment analysis,
  - Information extraction,
  - …

Using Language Models

- Auto-correct: find most likely intended sentence $w_1^*, \ldots, w_T^*$ given word entries $x_1, \ldots, x_T$.
  - $w_1^*, \ldots, w_T^* = \arg\max_{w_1,\ldots,w_T} P(w_1, \ldots, w_T | x_1, \ldots, x_T)$
    $= \arg\max_{w_1,\ldots,w_T} P(x_1, \ldots, x_T | w_1, \ldots, w_T) P(w_1, \ldots, w_T)$
    $= \arg\max_{w_1,\ldots,w_T} \left(\prod_t P(x_t | w_t) \right) P(w_1, \ldots, w_T)$

Maximization is Exponential in T

Language model

Learn from corpus of spelling mistakes

Using Language Models

- Auto-correct: find most likely intended sentence $w_1^*, \dots, w_T^*$ given word entries $x_1, \dots, x_T$.
  - $w_1^*, \dots, w_T^* = \arg\max_{w_1,\dots,w_T} P(w_1, \dots, w_T | x_1, \dots, x_T)$
    $= \arg\max_{w_1,\dots,w_T} P(x_1, \dots, x_T | w_1, \dots, w_T) P(w_1, \dots, w_T)$
    $= \arg\max_{w_1,\dots,w_T} \left(\prod_t P(x_t | w_t) \right) P(w_1, \dots, w_T)$

When the language model is based on n-th order Markov assumption, maximization by Viterbi is exponential in n (not T).

Using Language Models

- Auto-correct: find most likely intended sentence $w_1^*, \dots, w_T^*$ given word entries $x_1, \dots, x_T$.
  - $w_1^*, \dots, w_T^* = \arg\max_{w_1,\dots,w_T} P(w_1, \dots, w_T | x_1, \dots, x_T)$
    $= \arg\max_{w_1,\dots,w_T} P(x_1, \dots, x_T | w_1, \dots, w_T)\, P(w_1, \dots, w_T)$
    $= \arg\max_{w_1,\dots,w_T} \left(\prod_t P(x_t | w_t)\right) P(w_1, \dots, w_T)$

If $O(k^n)$ is still too slow, use beam search instead of maximizing over all sequences.

Evaluating Language Models

- Extrinsic evaluation: how well deoes the model perform at the task that it is intended for?
- Machine translation:
  - Apply translation system with different language models.
  - The have human judge score the output sencences.
- Auto completion:
  - Install two models on smartphones.
  - Measure how frequently users accept the proposed completions.

Evaluating Language Models

- Intrinsic evaluation: Does the language model assign a high likelihood to actual sentences?
- Cannot evaluate on training corpus:
  - Has been used to train the model.
  - Does not imply that model will assign high likelyhood to any out-of-corpus sentence.
- Training-Test-Split:
  - Estimate parameters on 80% of the corpus.
  - Evaluate model on remaining 20%.

Evaluating Language Models

- What is a good evaluation measure?
- Log-Likelihood of the test corpus?
  - Possible to compare multiple language models.
  - Not possible to compare sencences because short sentences tend to have a higher likelihood than long one (fewer factors).
- Perplexity of the test corpus:
  - Inverse likelihood, normalized by number of words.
  - $PP(w_1, \dots, w_T) = P(w_1, \dots, w_T)^{-\frac{1}{\mathrm{T}}} = \sqrt[T]{\prod_t \frac{1}{P(w_t|w_{t-1}, \dots)}}$

Evaluating Language Models

- Perplexity of the test corpus:
  - Inverse likelihood, normalized by number of words.
  - $PP(w_1, \dots, w_T) = P(w_1, \dots, w_T)^{-\frac{1}{\mathrm{T}}} = \sqrt[T]{\prod_t \frac{1}{P(w_t|w_{t-1}, \dots)}}$
- Example:
  - $p(w_t) = \frac{1}{10}$
  - $PP(w_1, \dots, w_T) = \left(\left(\frac{1}{10}\right)^T\right)^{-\frac{1}{T}} = \left(\frac{1}{10}\right)^{-1} = 10$
  - Perplexity: average branching factor.

Evaluating Language Models

- Different corpora reflect different distributions.
- A model that has been trained on the Wall Street Journal corpus may assign a log likelihood to sentences from a belletristic corpus.
- If a language model infers $P(w_1, \ldots, w_T) = 0$, then the perplexity is undefined.
  - Sign of overfitting to the training data, lack of regularization.

Summary

- Stochastic language models quantify the likelihood of a sencence.
- Markov assumption of order $n-1$: word w_t only dependent on $w_{t-1}, \ldots, w_{t-n+1}$.
- The n-gram model has a discrete parameter for each n-word combination.
- Continuous-space (neural) language models learn embedding of words into a feature space; this gives better generalization.
- Language models are a useful tools for many applications.