Foundations of NLP

Week 1 - Statistical Language Modeling

NLP Course 2025

September 20, 2025

Using Optimal Readability Template

Course Overview

Foundations

- Language basics
- Probability theory
- N-gram models
- Evaluation metrics

Neural Methods

- Word embeddings
- RNNs and LSTMs
- Seq2Seq models
- Attention mechanism

Modern NLP

- Transformers
- Pre-training
- Fine-tuning
- Applications

What is Language Modeling?

Core Objective

- Assign **probabilities** to sequences of words
- Predict the next word given context
- Model the structure of language

Applications

- Machine Translation
- Speech Recognition
- Text Generation
- Spell Correction

Language models capture the statistical patterns in text

$$P(w_1, w_2, ..., w_n)$$

Probability Fundamentals

Chain Rule

$$P(w_1^n) = P(w_1) \times P(w_2|w_1) \\ \times P(w_3|w_1^2)...P(w_n|w_1^{n-1})$$

Markov Assumption

$$P(w_i|w_1^{i-1}) \approx P(w_i|w_{i-k+1}^{i-1})$$

N-gram Models

- **Unigram**: $P(w_i)$
- **Bigram**: $P(w_i|w_{i-1})$
- **Trigram**: $P(w_i|w_{i-2}, w_{i-1})$

Trade-off: Context vs Sparsity

N-gram Model Comparison

Model	Context	Parameters	Sparsity	Perplexity
Unigram	0 words	V	Low	250
Bigram	1 word	V^2	Medium	120
Trigram	2 words	V^3	High	85
4-gram	3 words	V^4	Very High	**72**

V = Vocabulary size (typically 10,000-50,000)

Implementation Example

```python
class BigramModel:
    def __init__(self):
        self.counts = defaultdict(
            lambda: defaultdict(int)
        )
        self.vocab = set()

    def train(self, text):
        words = text.split()
        for i in range(len(words)-1):
            w1, w2 = words[i], words[i+1]
            self.counts[w1][w2] += 1
            self.vocab.update([w1, w2])

    def probability(self, w1, w2):
        if w1 not in self.counts:
            return 0.0
        total = sum(self.counts[w1].values())
        return self.counts[w1][w2] / total
```

Key Components

- **counts**: Store bigram frequencies
- **vocab**: Track unique words
- train(): Build frequency table
- probability(): Calculate $P(w_2|w_1)$

Complexity

- Time: O(n) for training
- Space: $O(V^2)$ worst case
- Query: O(1) lookup

Evaluation Metrics

Perplexity

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

- **Lower is better**
- Geometric mean of inverse probability
- Typical ranges: 50-500

Interpretation

- PP $= 100$: On average, choosing from 100 equally likely words
- PP $= 10$: Very good model
- PP $= 1000$: Poor model

Cross-Entropy

$$H(P, Q) = -\sum_x P(x) \log Q(x)$$

- Measures information loss
- Related: $PP = 2^H$
- Used in neural models

Other Metrics

- BLEU: Machine translation
- ROUGE: Summarization
- Word Error Rate: Speech

Smoothing Techniques

The Zero Probability Problem

Add-One (Laplace)

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

- Simple to implement
- Over-smooths
- Poor performance

Good-Turing

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

- Better estimates
- Complex theory
- Works well

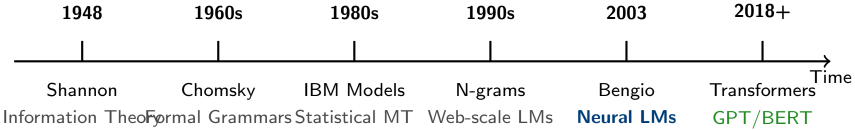
Kneser-Ney

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

- State-of-the-art
- Interpolation-based
- Best results

Smoothing redistributes probability mass to unseen events

Historical Perspective



From Rules → Statistics → Deep Learning

Week 1 Summary

- Language modeling assigns **probabilities** to text
- N-gram models use Markov assumption for tractability
- Sparsity is the main challenge
- Smoothing techniques handle unseen events
- Evaluation via perplexity and cross-entropy

Next Week: Neural Language Models & Word Embeddings

Key Takeaway