Phase 1: N-gram Language Models

RANDHIR KUMAR

May 4, 2025

1 Core Concepts

1.1 Definition

An **n-gram model** predicts the next word using the previous $(n-1)$ words based on probability.

1.2 Key Properties

- **Order**: Unigram (1), Bigram (2), Trigram (3), etc.
- **Markov Assumption**: Only the last $(n-1)$ words matter:

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-n+1},\ldots,w_{k-1})$$

2 Mathematical Formulation

2.1 Maximum Likelihood Estimation

Probability of a word given context:

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^{i})}{C(w_{i-n+1}^{i-1})}$$

where $C(\cdot)$ is the count in training data.

2.2 Perplexity

Model quality metric:

$$PP(W) =$$

3 Implementation (Python)

```python
from collections import defaultdict, Counter

class NGramModel:
    def __init__(self, n=2):
        self.n = n
        self.counts = defaultdict(Counter)

    def train(self, text):
        tokens = text.split()
        for i in range(len(tokens)-self.n+1):
            context = tuple(tokens[i:i+self.n-1])
            word = tokens[i+self.n-1]
            self.counts[context][word] += 1

    def predict_next(self, context):
        return self.counts[context].most_common(1)[0][0]
```

4 Limitations & Solutions

Problem	Solution
Sparse counts	Add-λ smoothing
Unknown words	UNK token
Long dependencies	Higher n or neural models

Appendix

Sample Output

```
Input: "I love"
Bigram Prediction: "coding" (P=0.6)
```