N-Gram Model Formulas

- **Word sequences**
  \[ w^n_i = w_i...w_n \]

- **Chain rule of probability**
  \[ P(w^n_i) = P(w_i)P(w_2 | w_i)P(w_3 | w_1)...P(w_n | w_1^{+1}) = \prod_{k=1}^{n} P(w_k | w_1^{k-1}) \]

- **Bigram approximation**
  \[ P(w^n_i) = \prod_{k=1}^{n} P(w_k | w_{k-1}) \]

- **N-gram approximation**
  \[ P(w^n_i) = \prod_{k=1}^{n} P(w_k | w_{k-N+1}) \]

Estimating Probabilities

- N-gram conditional probabilities can be estimated from raw text based on the *relative frequency* of word sequences.
  \[ \text{Bigram: } P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \]
  \[ \text{N-gram: } P(w_n | w_{n-N+1}) = \frac{C(w_{n-N+1}, w_n)}{C(w_{n-N+1})} \]

- To have a consistent probabilistic model, append a unique start (<s>) and end (<s>) symbol to every sentence and treat these as additional words.

Perplexity

- Measure of how well a model “fits” the test data.
- Uses the probability that the model assigns to the test corpus.
- Normalizes for the number of words in the test corpus and takes the inverse.
  \[ PP(W) = \sqrt[n]{\frac{1}{P(w_1...w_n)}} \]
- Measures the weighted average branching factor in predicting the next word (lower is better).

Laplace (Add-One) Smoothing

- “Hallucinate” additional training data in which each possible N-gram occurs exactly once and adjust estimates accordingly.
  \[ \text{Bigram: } P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V} \]
  \[ \text{N-gram: } P(w_n | w_{n-N+1}) = \frac{C(w_{n-N+1}, w_n) + 1}{C(w_{n-N+1}) + V} \]
  where \( V \) is the total number of possible (N-1)-grams (i.e. the vocabulary size for a bigram model).

- Tends to reassign too much mass to unseen events, so can be adjusted to add \( 0<\delta<1 \) (normalized by \( \delta V \) instead of \( V \)).
Interpolation

- Linearly combine estimates of N-gram models of increasing order.

Interpolated Trigram Model:
\[ \hat{P}(w_n | w_{n-2}, w_{n-1}) = \lambda_1 P(w_n | w_{n-2}, w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n) \]

Where: \[ \sum \lambda_i = 1 \]

- Learn proper values for \( \lambda_i \) by training to (approximately) maximize the likelihood of an independent \textit{development} (a.k.a. \textit{tuning}) corpus.

**Interpolation**

Formal Definition of an HMM

- A set of \( N+2 \) states \( S = \{s_0, s_1, s_2, \ldots, s_N, s_F\} \)
  - Distinguished start state: \( s_0 \)
  - Distinguished final state: \( s_F \)
- A set of \( M \) possible observations \( V = \{v_1, v_2, \ldots, v_M\} \)
- A state transition probability distribution \( A = \{a_{ij}\} \)
  \[ a_{ij} = P(q_{i+1} = s_j | q_i = s_i) \quad 1 \leq i, j \leq N \text{ and } i = 0, j = F \]
  \[ \sum_{j=1}^{N} a_{ij} + a_{iF} = 1 \quad 0 \leq i \leq N \]
- Observation probability distribution for each state \( j \)
  \[ b_j(k) = P(v_k \text{ at } t | q_t = s_j) \quad 1 \leq j \leq N \quad 1 \leq k \leq M \]
- Total parameter set \( \lambda = \{A, B\} \)

**Forward Probabilities**

- Let \( \alpha_t(j) \) be the probability of being in state \( j \) after seeing the first \( t \) observations (by summing over all initial paths leading to \( j \)).

\[ \alpha_t(j) = P(o_1, o_2, \ldots, o_t, q_t = s_j | \lambda) \]

**Computing the Forward Probabilities**

- Initialization
  \[ \alpha_1(j) = a_{0j} b_j(o_1) \quad 1 \leq j \leq N \]
- Recursion
  \[ \alpha_t(j) = \left[ \sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij} \right] b_j(o_t) \quad 1 \leq j \leq N, \quad 1 < t \leq T \]
- Termination
  \[ P(O | \lambda) = \alpha_T(s_F) = \sum_{i=1}^{N} \alpha_T(i)a_{iF} \]
Viterbi Scores

• Recursively compute the probability of the most likely subsequence of states that accounts for the first $t$ observations and ends in state $s_j$.

$$v_t(j) = \max_{q_0, q_1, \ldots, q_{t-1}} P(q_0, q_1, \ldots, q_{t-1}, o_1, \ldots, o_t \mid s_j, \lambda)$$

• Also record “backpointers” that subsequently allow backtracing the most probable state sequence.
  • $bt_t(j)$ stores the state at time $t-1$ that maximizes the probability that system was in state $s_j$ at time $t$ (given the observed sequence).

Computing the Viterbi Scores

• Initialization

$$v_1(j) = a_{0j} b_j(o_1) \quad 1 \leq j \leq N$$

• Recursion

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) \quad 1 \leq j \leq N, \ 1 < t \leq T$$

• Termination

$$P^* = v_T(s_F) = \max_{i=1}^N v_T(i) a_{iF}$$

Analogous to Forward algorithm except take max instead of sum.

Computing the Viterbi Backpointers

• Initialization

$$bt_1(j) = s_0 \quad 1 \leq j \leq N$$

• Recursion

$$bt_t(j) = \arg\max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) \quad 1 \leq j \leq N, \ 1 \leq t \leq T$$

• Termination

$$q_T^* = bt_T(s_F) = \arg\max_{i=1}^N v_T(i) a_{iF}$$

Final state in the most probable state sequence. Follow backpointers to initial state to construct full sequence.

Supervised Parameter Estimation

• Estimate state transition probabilities based on tag bigram and unigram statistics in the labeled data.

$$a_{ij} = \frac{C(q_t = s_i, q_{t+1} = s_j)}{C(q_t = s_i)}$$

• Estimate the observation probabilities based on tag/word co-occurrence statistics in the labeled data.

$$b_j(k) = \frac{C(q_t = s_j, o_t = v_k)}{C(q_t = s_j)}$$

• Use appropriate smoothing if training data is sparse.
Context Free Grammars (CFG)

- \( N \) a set of \textit{non-terminal symbols} (or \textit{variables})
- \( \Sigma \) a set of \textit{terminal symbols} (disjoint from \( N \))
- \( R \) a set of \textit{productions} or \textit{rules} of the form \( A \rightarrow \beta \), where \( A \) is a non-terminal and \( \beta \) is a string of symbols from \((\Sigma \cup N)^*\)
- \( S \), a designated non-terminal called the \textit{start symbol}

Estimating Production Probabilities

- Set of production rules can be taken directly from the set of rewrites in the treebank.
- Parameters can be directly estimated from frequency counts in the treebank.

\[
P(\alpha \rightarrow \beta \mid \alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{count}(\alpha \rightarrow \gamma)} = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)}
\]