

CS 262 : Computational Genomics

Lecture Notes : May 23, 2002

Prepared by : Sanket B. Malde (sanket@stanford.edu)

Introduction:

In the last lecture, we looked at *Stochastic Context Free Grammars (SCFGs)*.

The three computational problems associated with SCFGs are :

1. EVALUATION,
2. LEARNING, and
3. DECODING.

In the last lecture, we studied the EVALUATION problem, in which we looked at the ‘Inside’ and ‘Outside’ Algorithms. In the Inside Algorithm, we saw how to compute $a(i, j, V)$ which is the probability of deriving the string X_i, \dots, X_j from the non-terminal V . In the Outside Algorithm, we saw how to compute $b(i, j, V)$ which is the probability of deriving the string $X_1, \dots, X_{i-1}, V, X_{j+1}, \dots, X_n$ from the start state S . These algorithms together help us solve the problem of EVALUATION, which is to find $P(X | \theta)$.

LEARNING:

We shall now see how we can solve the LEARNING problem.

Suppose we are given the entire parse tree for a particular RNA sequence $X = X_1, \dots, X_n$, that is, we know exactly which rules were used in the derivation of X from S . Then we can count the number of times every transition is used, that is for every rule $V \rightarrow YZ$, we can maintain the count $C(V \rightarrow YZ)$. Similarly, we can count $C(V \rightarrow a)$ and $C(V)$ which is the number of times a non-terminal appears in the derivation. Then we can train the parameters as follows :

$$t_V(Y, Z) = \frac{C(V \rightarrow YZ)}{C(V)}$$

$$e_V(b) = \frac{C(V \rightarrow b)}{C(V)}$$

But if we do not know the parse tree, then we can train the model using the $a(i, j, V)$ and $b(i, j, V)$ values from the forward and backward algorithms as follows:

$$C(V) = \frac{1}{P(X | \theta)} \sum_{i=1}^n \sum_{j=i}^n (a(i, j, V) b(i, j, V))$$

Here, $a(i, j, V)b(i, j, V)$ is the probability that V appears in the derivation and derives X_i, \dots, X_j . We then sum these over all i, j . We divide by $P(X | \theta)$ to get a number as opposed to the probability.

Similarly,

$$C(V \rightarrow YZ) = \frac{1}{P(X | \theta)} \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^{j-1} (b(i, j, V) a(i, k, Y) a(k+1, j, Z) t_V(Y, Z))$$

Here, $b(i, j, V)$ is the probability of deriving the string $X_1, \dots, X_{i-1}, V, X_{j+1}, \dots, X_n$ from the start state S , $t_V(Y, Z)$ is the probability of the transition $V \rightarrow YZ$, $a(i, k, Y)$ is the probability of deriving X_i, \dots, X_k from Y , and $a(k+1, j, Z)$ is the probability of deriving X_{k+1}, \dots, X_j from Z .

Similarly,

$$C(V \rightarrow a) = \frac{1}{P(X | \theta)} \sum_{\text{all } i \text{ such that } X_i=a} (b(i, i, V) e_V(a))$$

DECODING:

The DECODING problem is defined as follows: Given the string X , find the most probable parse tree that can derive X from S . We now look at the CYK Algorithm for DECODING.

Define:

$$\gamma(i, j, V) = \log P(V \text{ derives } X_i, \dots, X_j, \pi_{i \rightarrow j}^* | \theta)$$

Hence,

$$\gamma(1, n, 1) = \log P(X, \pi^* | \theta)$$

where $S = 1$ is the start state.

CYK Algorithm:

Initialization:

For $i = 1, \dots, n$ and $V = 1, \dots, M$:

$$\gamma(i, i, V) = \log e_V(X_i)$$

$$\tau(i, i, V) = (0, 0, 0)$$

where $\tau(i, j, V)$ are the traceback pointers.

Inductive Step:

For $i = 1, \dots, n - 1$ and $j = i + 1, \dots, n$ and $V = 1, \dots, M$:

$$\gamma(i, j, V) = \max_Y \max_Z \max_{k=i, \dots, j-1} (\gamma(i, k, Y) + \gamma(k + 1, j, Z) + \log t_V(Y, Z))$$

$$\tau(i, j, V) = \arg \max_{(Y, Z, k)} (\gamma(i, k, Y) + \gamma(k + 1, j, Z) + \log t_V(Y, Z))$$

Termination:

As said above, $\gamma(1, n, 1)$ gives the maximum log probability of a parse, and, the traceback starting from $\tau(1, n, 1)$ gives us the parse tree in a top down fashion.

Running Time:

The running time is $O(M^3 n^3)$ where M is the number of non-terminals and n is the length of the string.

Exploiting covariance in RNA strands:

Let X^1, \dots, X^l be l RNA strands, each of length n . Define a *column* X_k as the column formed by aligning these l strings and looking at X_k^1, \dots, X_k^l . If two columns X_i and X_j are covarying, that is, they have complimentary bases, then we can link them using hydrogen bonds. Define the *mutual*

information between i and j , M_{ij} , as the extent to which columns X_i and X_j covary:

$$M_{ij} = \sum_{X_i, X_j} f_{X_i X_j} \log_2 \frac{f_{X_i X_j}}{f_{X_i} f_{X_j}}$$

Here, f_{X_i} is the frequency of one of the four bases observed in column i . $f_{X_i X_j}$ is the joint (pairwise) frequency of one of the sixteen possible base pairs observed in columns i and j . M_{ij} measures how much the joint frequency distribution deviates from the distribution that is expected if the two columns vary independently.

Nussinov Folding Algorithm:

The Nussinov Folding Algorithm is a dynamic programming algorithm for finding the secondary structure with most base pairs. The algorithm is recursive. The key idea is that there are only four possible ways of getting the best structure for i, j from the best structures of the smaller subsequences:

1. Add unpaired position i onto best structure for subsequence $i + 1, j$.
2. Add unpaired position j onto best structure for subsequence $i, j - 1$.
3. Add i, j pair onto best structure found for subsequence $i + 1, j - 1$.
4. Combine two optimal substructures i, k and $k + 1, j$.

Formally, let $X = X_1, \dots, X_n$ be a sequence of length n . Let $\delta(i, j) = 1$ if X_i and X_j are a complimentary base pair, else $\delta(i, j) = 0$. Let $\gamma(i, j)$ be the maximal number of base pairs that can be formed for subsequence X_i, \dots, X_j .

Nussinov Algorithm:

Initialization:

For $i = 1, \dots, n$:

$$\gamma(i, i) = 0$$

For $i = 2, \dots, n$:

$$\gamma(i, i - 1) = 0$$

Recursion:

Starting with all subsequences of length 2 to length n :

$$\gamma(i, j) = \max \begin{cases} \gamma(i + 1, j) \\ \gamma(i, j - 1) \\ \gamma(i + 1, j - 1) + \delta(i, j) \\ \max_{i < k < j} [\gamma(i, k) + \gamma(k + 1, j)] \end{cases}$$

Running Time:

The running time is $O(n^3)$ as we are filling n^2 cells, and filling each cell takes time $O(n)$.

SCFG for generic RNA structure:

Consider the following production rules of a simple RNA folding SCFG :

$$\begin{aligned} S &\rightarrow aS \mid cS \mid gS \mid uS && (i \text{ unpaired}) \\ S &\rightarrow Sa \mid Sc \mid Sg \mid Su && (j \text{ unpaired}) \\ S &\rightarrow aSu \mid uSa \mid cSg \mid gSc && (i, j \text{ paired}) \\ S &\rightarrow SS && (\text{bifurcation}) \\ S &\rightarrow \epsilon && (\text{termination}) \end{aligned}$$

Assume that we know the probability of each of these 14 production rules, denoted by $p(aS)$, $p(aSu)$, etc. The algorithm to find the best possible parse of a sequence is as follows:

Initialization:

$$\gamma(i, i - 1) = -\infty \quad \text{for } i = 2 \text{ to } n$$

$$\gamma(i, i) = \max \begin{cases} \log p(X_i S) \\ \log p(S X_i) \end{cases}$$

for $i = 1, \dots, n$.

Recursion:

For $i = 1, \dots, n - 1, j = i + 1, \dots, n$:

$$\gamma(i, j) = \max \begin{cases} \gamma(i + 1, j) + \log p(X_i S) \\ \gamma(i, j - 1) + \log p(SX_i) \\ \gamma(i + 1, j - 1) + \log p(X_i SX_j) \\ \max_{i < k < j} [\gamma(i, k) + \gamma(k + 1, j) + \log p(SS)] \end{cases}$$

Termination:

At the end, $\gamma(1, n) = \log P(X, \pi^* | \theta)$ is the log likelihood of the optimal parse given the model θ , as required.