Motivation:
- Both DNA and protein sequences can be considered as sentences in a “language”.
- We know the “language” is not random
- What’s the grammar of the language?

- “The linguistics of DNA” – David Searls, American Scientist, 80(19920579).
Chomsky hierarchy of transformational grammars (1956)
• Regular expression (no recursive structure, cannot handle palindromes)
  – \( W \rightarrow aW \), or \( W \rightarrow a \)
• Context-free
  – \( W \rightarrow \beta \)
• Context-sensitive
  – \( \alpha_1 W \alpha_2 \rightarrow \alpha_1 \beta \alpha_2 \)
• Unrestricted grammars
  – \( \alpha_1 W \alpha_2 \rightarrow \gamma \)

Notations:
\( a \): any terminals;
\( \alpha, \gamma \): any string of non-terminals and/or terminals, including the null string;
\( \beta \): any string of non-terminals and/or terminals, not including the null string.

Prosite patterns
For example, a RNA binding motif is expressed as
\([RK]-G-[EDRKHPGC]-[AGSCI]-[FY]-[LIVA]-X-[FYM]\)

In regular expression, the motif is defined as
\[
\begin{align*}
S & \rightarrow r \ W_1 \mid kW_1 \\
W_1 & \rightarrow g \ W_2 \\
W_2 & \rightarrow [\text{afilmnqstvwy}] \ W_3 \\
W_3 & \rightarrow [\text{agsci}] \ W_4 \\
W_4 & \rightarrow f \ W_5 \mid y \ W_5 \\
W_5 & \rightarrow l \ W_6 \mid I \ W_6 \mid v \ W_6 \mid a \ W_6 \\
W_6 & \rightarrow [\text{a..y}] \ W_7 \\
W_7 & \rightarrow f \mid y \mid m
\end{align*}
\]

Note:
1. \([ac]W\) is an alternative way to say \( aW \mid cW \).
2. Non-terminals \( W_1, \ldots, W_7 \) correspond to the 7 positions in the motif.
RNA stem loop

hybridization pairing: A-U, and C-G

\[
\begin{array}{cccc}
A & A & C & A \\
G & A & G & A \\
G & C & U & G \\
A & U & C & G \\
C & G & G & A \\
\end{array}
\]

where “-” stands for a pairing, and “x” for no pairing.

In the above example, seq1 and seq2 fold into a similar structure, whereas seq3 does not.

Pairwise alignments disregarding such structural restrictions may be misleading;
seq2 and seq3 have 70% sequence identity, seq1 and seq3 have 60%, whereas
seq1 and seq2 have only 30%.

To capture the palindromic structure, a context-free grammar

\[
S \rightarrow aW_1u \mid cW_1g \mid gW_1c \mid uW_1a \\
W_1 \rightarrow aW_2u \mid cW_2g \mid gW_2c \mid uW_2a \\
W_2 \rightarrow aW_3u \mid cW_3g \mid gW_3c \mid uW_3a \\
W_3 \rightarrow gaaa \mid gcga
\]
Stochastic grammars

Motivations:

– Irregularities (or exceptions to the grammar) in languages
– Such irregularities can be taken care of by “growing” your grammar; introducing new non-terminals and new production rules.
– It is useful to differentiate productions that account for a large portion of the language from those that are rare exceptions.
– This can be achieved by assigning probabilities to various productions.
  • For any non-terminals, the probabilities of all its possible productions must add to 1.
  e.g.,
  • Given a sentence $x$, a stochastic grammar $\theta$ parsing the sentence will assign a probability $P(x|\theta)$ that the sentence belongs to the language specified by the grammar, whereas, a conventional grammar will just give a yer-or-no answer.

  \[ \sum_{x \in \text{language}} P(x|\theta) = 1 \]

Stochastic CFG for sequence modeling

• Calculate optimal alignment of a sequence to a parameterized SCFG. [CYK algorithm]
• Calculate probability for a given sequence to belong to the language specified by SCFG. [inside algorithm, and inside-outside algorithm]
• Given a set of example sequences/structures, estimate optimal probability parameters for a SCFG.

Note:

– The issues addressed above by SCFGs are quite similar to those for HMMs; actually it can be proved that HMMs are equivalent to stochastic regular grammars.
– Chomsky normal form: any CFG can be written such that all productions have the following forms:

  $\begin{align*}
  W_i & \rightarrow W_i W_i \\
  W_i & \rightarrow w
  \end{align*}$
Inside algorithm

- Non-terminals: \( W_1, \ldots, W_M \), where \( W_1 \) is the start.
- \( t_v(y,z) \): probability for a production like \( W_v \rightarrow W_y W_z \)
- \( e_v(a) \): probability for production like \( W_v \rightarrow a \)
- \( x = x_1, \ldots, x_L \) is a sequence of length \( L \).
- \( \alpha(i,j,v) \): probability of a parse subtree rooted at non-terminal \( W_v \) for subsequence \( x_i, \ldots, x_j \).

Initialization: for \( i = 1 \) to \( L \), \( v = 1 \) to \( M \)
\[
\alpha(i,i,v) = e_v(x_i)
\]

Iteration: for \( i = 1 \) to \( L-1 \), \( j = i+1 \) to \( L \), \( v = 1 \) to \( M \)
\[
\alpha(i,j,v) = \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=i}^{j-1} \alpha(i,k,y) \alpha(k+1,j,z) t_v(y,z)
\]

Termination:
\[
P(x|\theta) = \alpha(1,L,1)
\]

---

CYK (Cocke-Younger-Kasami) algorithm

Initialization: for \( i = 1 \) to \( L \), \( v = 1 \) to \( M \)
\[
\gamma(i,i,v) = \log e_v(x_i)
\]
\[
\tau(i,i,v) = (0,0,0)
\]

Iteration: for \( i = 1 \) to \( L-1 \), \( j = i+1 \) to \( L \), \( v = 1 \) to \( M \)
\[
\gamma(i,j,v) = \max_{y,z} \max_{k=i}^{j-1} \{ \gamma(i,k,y) + \gamma(k+1,j,z) + \log t_v(y,z) \}
\]
\[
\tau(i,j,v) = \arg\max_{y,z,k} \{ \gamma(i,k,y) + \gamma(k+1,j,z) + \log t_v(y,z) \}
\]

Termination:
\[
\gamma(1,L,1) = \log P(x, \pi^*|\theta) \quad \text{// where } \pi^* \text{ is the most probable parse tree.}
\]

Traceback
- push \((1,L,1)\) on the stack
- Iteration:
  - pop \((i,j,v)\)
  - \((y,z,k) = \tau(i,j,v)\)
  - if \(\tau(i,j,v) = (0,0,0)\), implying \(i = j\)
    - attach \(x_i\) as the child of \(v\)
  - else
    - attach \(y, z\) to parse tree as children of \(v\)
    - push \((k+1,j,z)\)
    - push \((i,k,y)\).