# Today's Lecture

- Forward & forward/backward algorithms
- Baum-Welch training

# More algorithms

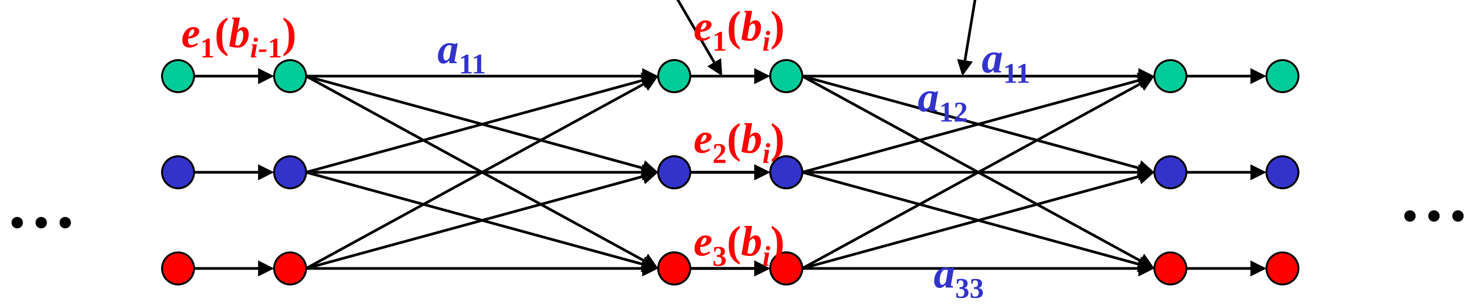
- Can also use dynamic programming to find
  - sum of all product path weights
    - = "forward algorithm" for probability of observed sequence
  - sum of all product path weights through particular node or particular edge
    - = "forward/backward algorithm" to find posterior probabilities
- Now must use product weights and non-log-transformed probabilities
  - because need to *add* probabilities

- In each case, compute successively for each node (by increasing depth: left to right)
  - the sum of the weights of all paths ending at that node
  - N.B. paths are constrained to begin at the begin node!
- In forward/backward algorithm,
  - work through all nodes a second time, in opposite direction
    - i.e. in reverse graph – constraining paths to start in rightmost column of nodes

# WDAG for 3-state HMM, length $n$ sequence

weights are emission probabilities $e_k(b_i)$ for $i^{th}$ residue $b_i$

weights are transition probabilities $a_{kl}$



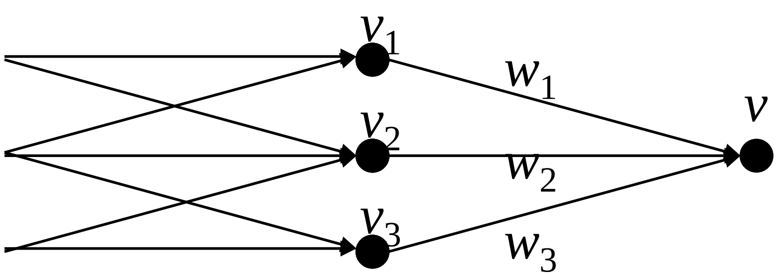

$b_{i-1}$
position $i$-1

$b_i$
position $i$

$b_{i+1}$
position $i$+1

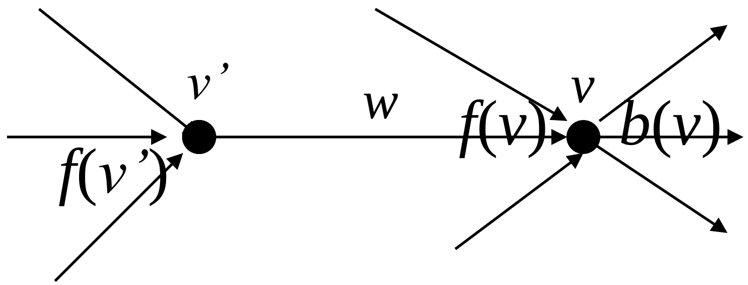
For each vertex $v$, let $f(v) = \sum_{\text{paths } p \text{ ending at } v} \text{weight}(p)$, where weight($p$) = *product* of edge weights in $p$. Only consider paths starting at 'begin' node.

Compute $f(v)$ by dynam. prog: $f(v) = \sum_i w_i \, f(v_i)$, where
$v_i$ ranges over the parents of $v$, and
$w_i$ = weight of the edge from $v_i$ to $v$.




Similarly for $b(v) = \sum_{p \text{ beginning at } v} \text{weight}(p)$

The paths *beginning* at $v$ are the ones *ending* at $v$ in the *reverse* *(or* *inverted)* *graph*




$f(v)b(v)$ = sum of the path weights of all paths *through v*.

$f(v')wb(v)$ = sum of the path weights of all paths *through the edge (v',v)*

- Numerical issues: multiplying many small values can cause underflow. Remedies:
  - *Scale* weights to be close to 1 (affects all paths by same constant factor – which can be multiplied back later); or
  - (where possible) use log weights, so can add instead of multiplying.
  - see Rabiner & Tobias Mann links on web page
    - & will discuss further in discussion section

# Forward/backward algorithm

- Work through graph in forward direction:
  - compute and store $f(v)$
- Then work through graph in backward direction:
  - compute $b(v)$
  - compute $f(v)\ b(v)$ and $f(v)wb(v)$ as above, store in appropriate cumulative sums
  - only need to store $b(v)$ until have computed $b$'s at next position
- Posterior probability of being in state $s$ at position $i$ is $f(v)\ b(v)$ / *total sequence prob*
  - where $v$ is the corresponding graph node

# Baum-Welch training

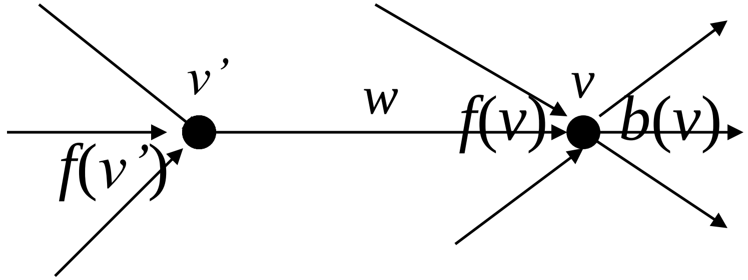
- Special case of EM (‘expectation-maximization’) algorithm
- like Viterbi training, but
  - uses *all* paths, each weighted by its probability rather than just highest probability path.
- sometimes give significantly better results than Viterbi
  - e.g. for PFAM

# Implementing Baum-Welch

– An edge in the WDAG contributes *fractional* (or *weighted*) *counts* given by its posterior probability:

– (*): $\left(\sum_{\text{all paths } p \text{ through edge } e} \text{weight}(p)\right) / \left(\sum_{\text{all paths } p} \text{weight}(p)\right)$

(Fractional counts are computed using forward-backward algorithm)




$f(v)b(v)$ = sum of the path weights of all paths *through v*.

$f(v')wb(v)$ = sum of the path weights of all paths *through the edge (v',v)*

– Compute new param estimates

- $e_k(b)$^ = (frac. # times symbol *b* emitted by state *k*) / (frac. # times state *k* occurs)
- $a_{kl}$ ^ = (frac. # times state *k* followed by state *l*) / (frac. # times state *k* occurs)
  - (In denom,, omit frac counts at last position of sequence)

where "frac. # times" is given by (*) for appropriate edge type (emission or transition)

– New Baum-Welch parameter estimates have higher likelihood
  - general property of EM algorithm
  - not true in general for Viterbi training

– Iterate: get series of estimates converging to a *local* maximum on likelihood surface

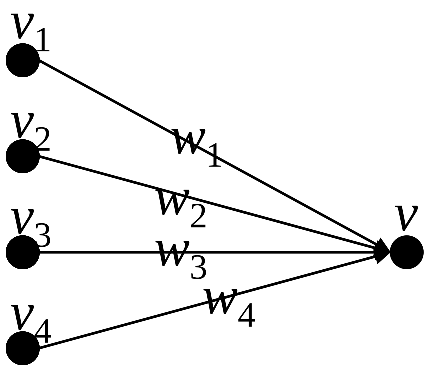
# Search of parameter space

- ML estimates correspond by definition to *global* maximum;
- but there may be many *local* maxima, and EM or Viterbi search can get "trapped" in one
- remedies:
  - Consider multiple starts (multiple choices for starting parameters)
  - use "reasonable values" to start search (e.g. unlikely transitions should be given small initial probabilities)

- Allow search to “jump” out of local maxima:
  - Add “noise” to counts at each iteration; gradually reduce the amount of noise
  - Use Viterbi-like training, but
    - sample paths probabilistically
      - (in retracing Viterbi path, choose edge at random according to its prob, rather than taking highest prob parent);
    - use “temperature” T to adjust probabilities;
      - initially with large T making all probs approximately equal;
      - then gradually reduce T
    - similar to Gibbs sampler

# Probabilistic Viterbi Backtracking

reset all weights $w$ to $w^{1/T}$. For large T ( >> 1), this makes distinct $w$'s relatively close; for small T (<< 1), relatively far apart




choose parent $v_i$ with probability $w_i\, f(v_i) \,/\, f(v)$. For large T, all parents almost equally likely to be chosen; for small T, strongly favor largest (max) $w_i\, f(v_i)$

given choice of paths, re-estimate weights; iterate