Biochemistry 324

Bioinformatics

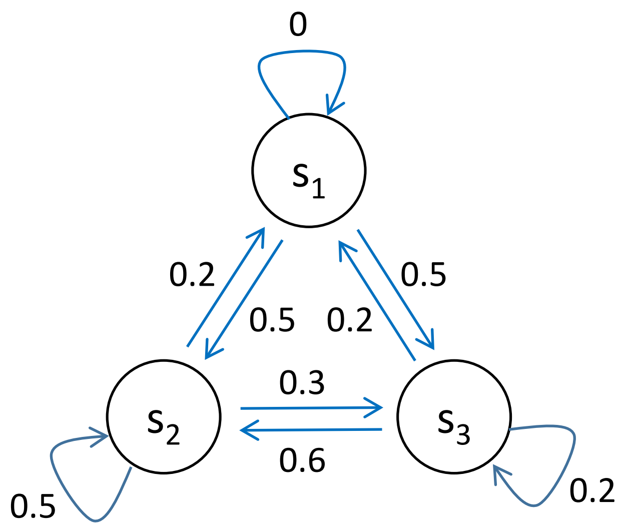
Hidden Markov Models (HMMs)

Find the hidden tiger in the image…

https://www.moillusions.com/hidden-tiger-illusion/

Markov Chain

- A Markov chain a system represented by **N states, $s_1,s_2,s_3,\ldots,s_N$** which **can be seen**
- There are discrete times t=0, t=1, … during which the system is in state $s_1,s_2,\ldots$
- At time step *t* the system is in state q_t where $\mathbf{q_t \in \{s_1,s_2,s_3,\ldots,s_N\}}$
- The system can make a transition between states at consecutive time points with certain probabilities, i.e. $\mathbf{p(q_{t+1}=s_1|q_t=s_2) = 0.5}$. [...$q_{t+1}=s_1$ **given that** $q_t=s_2$...]
- Moving from state $\mathbf{q_t}$ to state $\mathbf{q_{t+1}}$ **depends only on $\mathbf{q_t}$**, not q_{t-1}, q_{t-2} etc.
- This is known as a **first order** Markov chain
- In the general case, the transition probability $\mathbf{a_{ij}=p(q_{t+1}=s_j|q_t=s_i)}$ going from s_i to s_j
- The chance to **start** with s_1,s_2 or s_3 is π = {0.5,0.3,0.2}

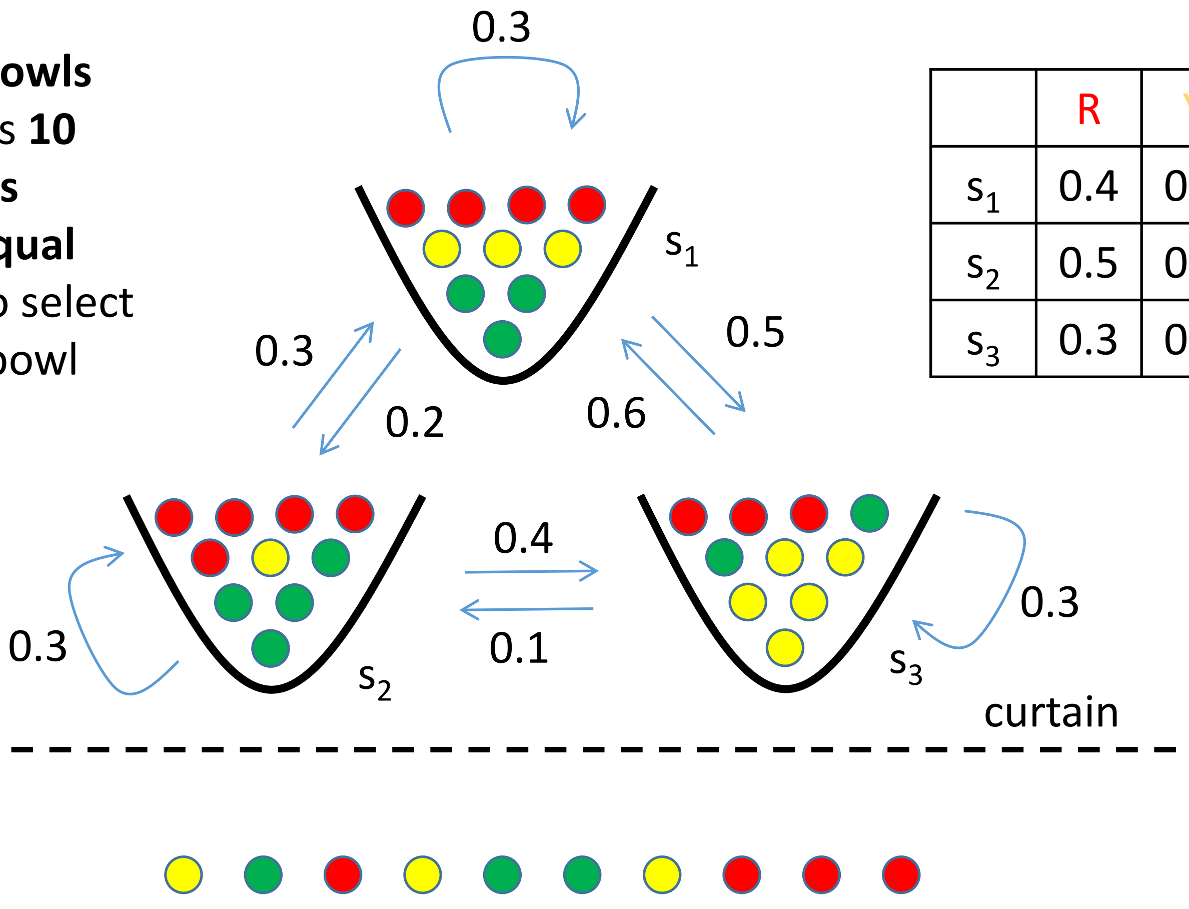


	End state			
Start state		**s₁**	**s₂**	**s₃**
	s₁	0	0.5	0.5
	s₂	0.2	0.5	0.3
	s₃	0.2	0.6	0.2

Thus, the chance of observing the sequence s_1,s_3,s_3,s_2,s_1,s_3 = $0.5 \times 0.5 \times 0.2 \times 0.6 \times 0.2 \times 0.5 = 0.003$

Hidden Markov Model (HMM)

- There are **3 bowls**
- Each bowl has **10 coloured balls**
- There is an **equal probability** to select any ball in a bowl



	R	Y	G
s_1	0.4	0.3	0.3
s_2	0.5	0.1	0.4
s_3	0.3	0.5	0.2

- You only **observe** the **series of coloured balls** on this side of the curtain
- Did the person choosing the balls, **pick them** from the 3 bowl **according to the transition probabilitie**s?

Formal description of a HMM

T = length of observation sequence
N = number of states (bowls)
M = number of observation symbols (coloured balls)
Q = $\{q_1,q_2,\ldots,q_N\}$ series of states
V = $\{v_1,v_2,\ldots,v_N\}$ set of possible observation symbols

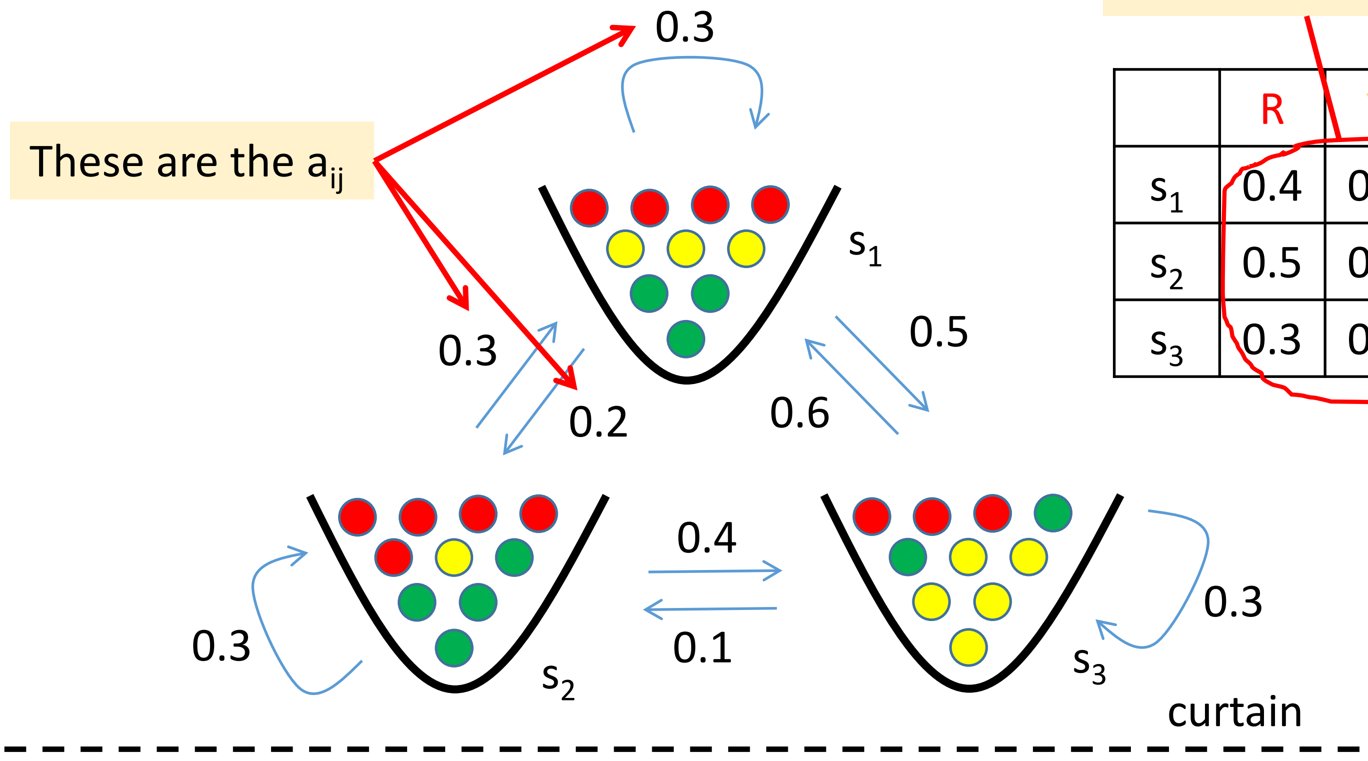
A HMM λ is described by

- A = $\{a_{ij}\}$ where $a_{ij} = p(q_j \text{ at } t+1 | q_i \text{ at } t)$ the state transition probabilities
- B = $\{b_j(k)\}$ where $b_j(k) = p(v_k \text{ at } t | q_i \text{ at } t)$
- $\pi = \{\pi_i\}$ where $\pi_I = p(q_i \text{ at } t=1)$ initial state distribution
- The model λ is written as $\lambda = (A,B,\pi)$

An observation sequence O = $O_1,O_2,\ldots,O_N$ is generated as follows:

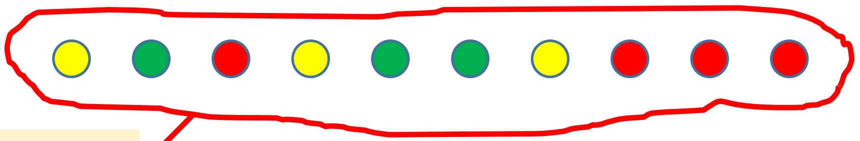
1. Choose an initial state q_1 according to the initial state distribution π
2. Set t = 1
3. Choose O_t according to $b_{1t}(k)$, the symbol probability distribution of state q_1
4. Choose a state q_2 according to $\{a_{ij}\}$ for
5. Set t = t+1
6. Return to 3 if t < T

Demystified



These are the $b_j(k)$

	R	Y	G
s_1	0.4	0.3	0.3
s_2	0.5	0.1	0.4
s_3	0.3	0.5	0.2



This is O

If we could start with any of the 3 bowls, then $\pi = \{0.33, 0.33, 0.33\}$

The 3 problems to solve for a HMM

Is this a TF binding site?

Problem 1 – *What is the chance that a pattern was generated by a HMM*
Given observation sequence O = $O_1,O_2,\dots,O_N$ and the model $\lambda = (A,B,\pi)$
How do we compute $p(O|\lambda)$, i.e., how do we compute the probability of the observation sequence O given the model λ?

Forward/backward algorithm

Problem 2 – *What is the most likely series of states to have produced a pattern*
Given observation sequence O = $O_1,O_2,\dots,O_N$ and the model $\lambda = (A,B,\pi)$
How do we compute a series of states $Q = \{q_1,q_2,\dots,q_N\}$ that is likely to have produced O?

Viterbi algorithm

Is this a non-coding region?

Problem 3 – *Can the HMM parameters be adjusted to better describe a pattern*
How can we adjust the model parameters $\lambda = (A,B,\pi)$ to maximize $p(O|\lambda)$?

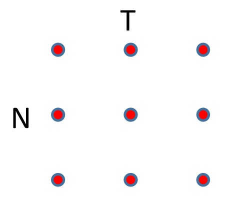
Baum-Welch algorithm

What HMM λ best represents this?

Problem 1 – What is the chance that a pattern was generated by a HMM

We are given an output series O = $\{O_1, O_2, \ldots, O_T\}$ representing T observations
This must have been produced by T states (not necessarily different states)
Say we observe 3 balls R, Y and G (T=3)
Let us *assume*, also this was produced by the state series Q =$\{s_1, s_2, s_3\}$
The probability of this series is $\mathbf{A = \pi_1 * a_{12} * a_{23}}$ = 0.33*0.2*0.4
The probability of the R, Y and G output series from this *specific* state series is
$\mathbf{B = b_1(1) * b_2(2) * b_3(3)}$ = 0.4*0.1*0.2 (see $b_j(k)$ table on previous slide)
Thus the probability of getting the observed series O from A and B,
$\mathbf{p(O|A,B) = \pi_1 * a_{12} * a_{23} * b_1(1) * b_2(2) * b_3(3)}$ = 0.33*0.2*0.4*0.4*0.1*0.2 = 0.0002
But this is only one possible path. We can also choose A = $\pi_2 * a_{22} * a_{21}$
$p(O|A,B) = \pi_2 * a_{22} * a_{21} * b_2(1) * b_2(2) * b_1(3)$ = 0.33* 0.3*0.3*0.5*0.1*0.3 = 0.0004
The **probability of O** = R, Y and G is the **sum of all** the independent, individual **paths** (remember independent, mutually exclusive probabilities add: a chance that you flip a head OR a tail is 0.5+0.5 = 1)

But there are 3*3*3 = 27 possible paths!

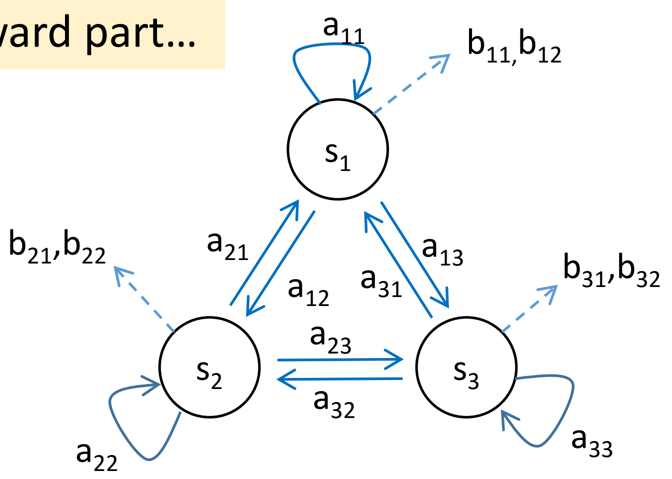


$\mathbf{O(N^T)}$ for 20 states with 50 samples (50 residue peptide): $20^{50} = 10^{34}$ years to calculate at 1 calculation/nanosecond

We need an algorithm!

The Forward/backward algorithm

First the forward part…

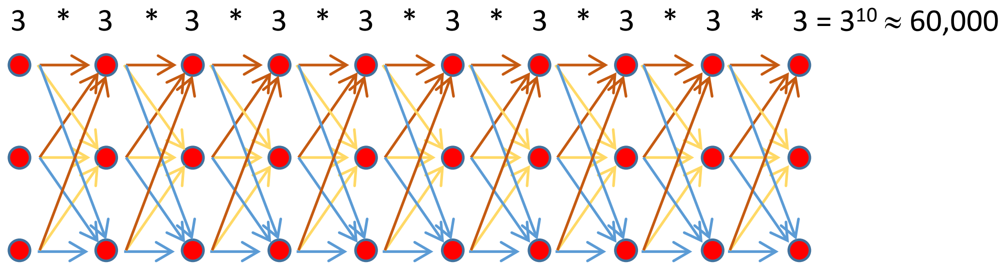


Imagine the are **three states** s1, s2 and s_3

Each state has 2 outputs $\mathbf{b_{11}, b_{12}, b_{21}, b_{22}, b_{31}}$ **and** $\mathbf{b_{32}}$

If we have a pattern of 10 symbols (T = 10)

There are thus $\mathbf{3^{10}}$ **(~60,000) paths** to produce 10 symbols

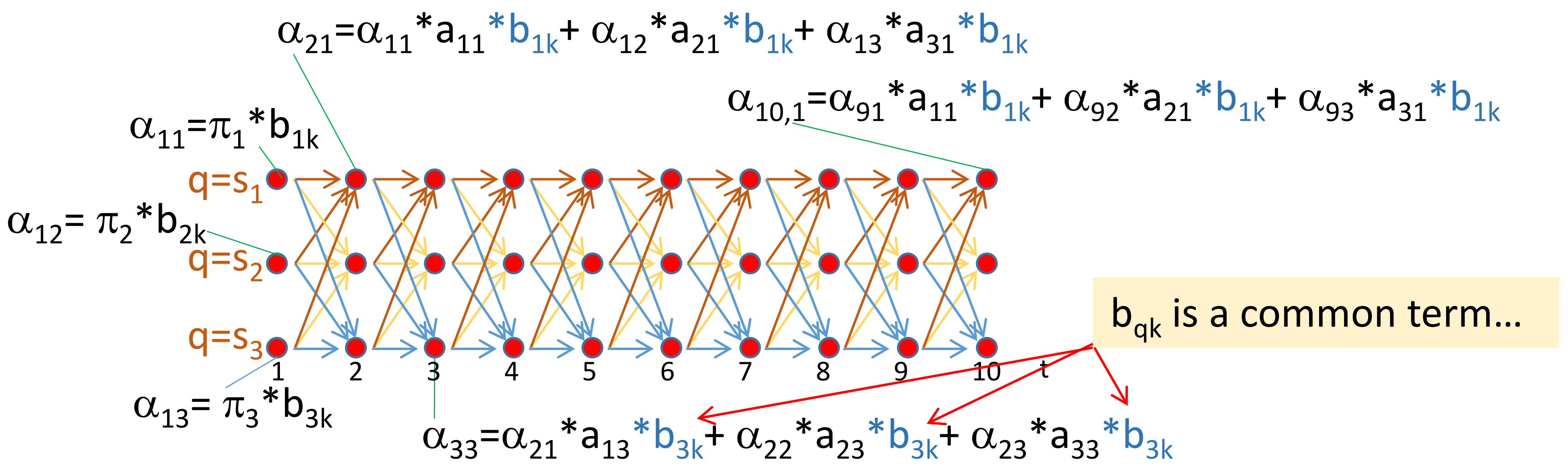


$3 * 3 * 3 * 3 * 3 * 3 * 3 * 3 * 3 * 3 = 3^{10} \approx 60{,}000$

What if we **store the answer** at each t?

The Forward algorithm – implementation

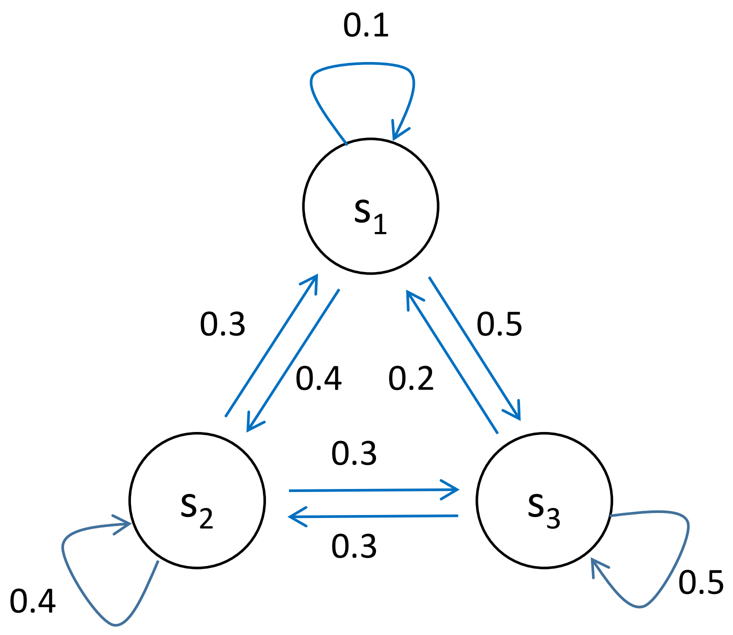
- Lets write α, the **sum of the probabilities** to produce output b_{qk} at state q_t at time t as $\boldsymbol{\alpha_{tq}}$



$\alpha_{21}=\alpha_{11}*a_{11}*b_{1k}+\alpha_{12}*a_{21}*b_{1k}+\alpha_{13}*a_{31}*b_{1k}$
$\alpha_{10,1}=\alpha_{91}*a_{11}*b_{1k}+\alpha_{92}*a_{21}*b_{1k}+\alpha_{93}*a_{31}*b_{1k}$
$\alpha_{11}=\pi_1*b_{1k}$
$\alpha_{12}=\pi_2*b_{2k}$
$q=s_1$
$q=s_2$
$q=s_3$
$\alpha_{13}=\pi_3*b_{3k}$
1
2
3
4
5
6
7
8
9
10
t
b_{qk} is a common term…
$\alpha_{33}=\alpha_{21}*a_{13}*b_{3k}+\alpha_{22}*a_{23}*b_{3k}+\alpha_{23}*a_{33}*b_{3k}$

- So, at any time t+1, the probability to arrive at a state q_{t+1} is **the sum of the probabilities to arrive from states q_t**
  - $\alpha_{t+1}(j) = \left[\sum_{i=1}^{T} \alpha_t(i)a_{ij}\right]b_{j(k)}$ **eqn 1**
- Thus, starting at t=1, **calculate α_t(i)** for each state, remember it, and use it to calculate each **α_{t+1}(i) at t=t+1**, etc.
- Thus, for this example you will perform 3^2*10 calculations, i.e. **O(N^2T)**
- You finally **add the $\alpha_{10,q}$ values** to get the **overall probability** to observe pattern O

An example HMM for the Forward algorithm



a_{ij}

	1	**2**	**3**
1	0.1	0.4	0.5
2	0.3	0.4	0.3
3	0.2	0.3	0.5

$b_j(k)$

	1	**2**
1	0.5	0.5
2	0.5	0.5
3	0.5	0.5

O={0,0,0,0,0,1,1,1,1,1}

Forward algorithm code

```
pi_matrix = np.array([0.4,0.3,0.3],float)
a_matrix = np.array([[0.1,0.4,0.5],[0.3,0.4,0.3],[0.2,0.3,0.5]],float)
b_matrix = np.array([[0.5,0.5],[0.5,0.5],[0.5,0.5]],float)
pattern_list = [0,0,0,0,0,1,1,1,1,1]
```

```
def forward(pi_matrix,a_matrix,b_matrix,pattern_list):
    number_of_states = len(a_matrix)
    length = len(pattern_list)
    alpha_matrix = np.zeros(number_of_states,dtype = float)
    temp_alpha_matrix = np.zeros(number_of_states,dtype = float)
    alpha_matrix = np.copy(pi_matrix)
    alpha_results = np.zeros((number_of_states,length),dtype = float)
    for i in range(length):
        for j in range(number_of_states):
            if(i==0):
                temp_alpha_matrix[j] =
                alpha_matrix[j]*b_matrix[j,pattern_list[i]]
            else:
                temp_alpha_matrix[j] =
                np.dot(alpha_matrix,a_matrix[:,j])*
                b_matrix[j,pattern_list[i]]
            alpha_results[j,i] = temp_alpha_matrix[j]
        alpha_matrix = np.copy(temp_alpha_matrix)
    return(np.sum(alpha_matrix))
```

Forward algorithm code output

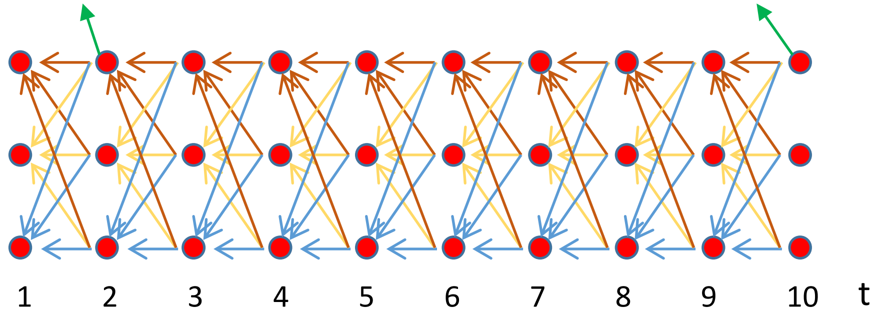
```
alpha 0 0  =  0.2
alpha 1 0  =  0.15
alpha 2 0  =  0.15
alpha 0 1  =  0.0475
alpha 1 1  =  0.0925
alpha 2 1  =  0.11
alpha 0 2  =  0.02725
alpha 1 2  =  0.0445
alpha 2 2  =  0.05325
alpha 0 3  =  0.0133625
alpha 1 3  =  0.0223375
alpha 2 3  =  0.0268
alpha 0 4  =  0.00669875
alpha 1 4  =  0.01116
alpha 2 4  =  0.01339125
alpha 0 5  =  0.0033480625
alpha 1 5  =  0.0055804375
alpha 2 5  =  0.0066965
alpha 0 6  =  0.00167411875
alpha 1 6  =  0.002790175
alpha 2 6  =  0.00334820625
alpha 0 7  =  0.0008370528125
alpha 1 7  =  0.0013950896875
alpha 2 7  =  0.0016741075
alpha 0 8  =  0.00041852684375
alpha 1 8  =  0.000697544625
alpha 2 8  =  0.00083705353125
alpha 0 9  =  0.000209263389062
alpha 1 9  =  0.000348772323437
alpha 2 9  =  0.0004185267875
Probability =  0.0009765625
```

- Danger of **underflow**
- **Add logarithms**

The Backward algorithm

The Backward algorithm is the **reverse of the Forward algorithm**
Use either, not both!

$\beta_{12} = (a_{11}\beta_{13} + a_{12}\beta_{23} + a_{13}\beta_{33})b_{1k}$ $\beta_{1,10} = 1*b_{1k} + 1*b_{2k} + 1*b_{3k}$



We must be at t=10, because we have 10 symbols

$\beta_T = 1$

$\beta_i(t-1) = \left(\sum_{j=1}^{N} a_{ij}\beta_j(t)\right)b_j(k)$

O(N²T)

Calculate $\beta_i(t-1)$ for every t from t=T to t=1
Finally max$\left[\left(\sum_{j=1}^{N} \pi_i\beta_j(t)\right)b_j(k)\right]$ is calculated

Accounts for the starting π-distribution

Backwards algorithm code

```
pi_matrix = np.array([0.4,0.3,0.3],float)
a_matrix = np.array([[0.1,0.4,0.5],[0.3,0.4,0.3],[0.2,0.3,0.5]],float)
b_matrix = np.array([[0.5,0.5],[0.5,0.5],[0.5,0.5]],float)
pattern_list = [0,0,0,0,0,1,1,1,1,1]
```

```
def backward(pi_matrix,a_matrix,b_matrix,pattern_list):
    number_of_states = len(a_matrix)
    length = len(pattern_list)
    beta_matrix = np.ones((number_of_states,1),dtype=float)
    temp_beta_matrix = np.zeros((number_of_states,1),dtype = float)
    beta_results = np.ones((number_of_states,length),dtype = float)
    for i in range(length-1,-1,-1): #N-1 to 0, backwards
        for j in range(number_of_states):
            temp_beta_matrix[j,0] =
           np.dot(a_matrix[j,:],beta_matrix[:,0])*
           b_matrix[j,pattern_list[i]]
            beta_results[j,i] = temp_beta_matrix[j,0]
        beta_matrix = np.copy(temp_beta_matrix)
    return(np.dot(pi_matrix,beta_matrix))
```

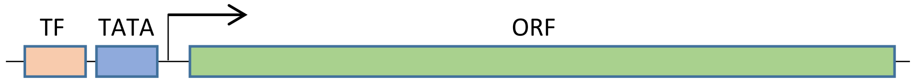
Backward algorithm code output

```
beta 0 0 = 0.0009765625
beta 1 0 = 0.0009765625
beta 2 0 = 0.0009765625
beta 0 1 = 0.001953125
beta 1 1 = 0.001953125
beta 2 1 = 0.001953125
beta 0 2 = 0.00390625
beta 1 2 = 0.00390625
beta 2 2 = 0.00390625
beta 0 3 = 0.0078125
beta 1 3 = 0.0078125
beta 2 3 = 0.0078125
beta 0 4 = 0.015625
beta 1 4 = 0.015625
beta 2 4 = 0.015625
beta 0 5 = 0.03125
beta 1 5 = 0.03125
beta 2 5 = 0.03125
beta 0 6 = 0.0625
beta 1 6 = 0.0625
beta 2 6 = 0.0625
beta 0 7 = 0.125
beta 1 7 = 0.125
beta 2 7 = 0.125
beta 0 8 = 0.25
beta 1 8 = 0.25
beta 2 8 = 0.25
beta 0 9 = 0.5
beta 1 9 = 0.5
beta 2 9 = 0.5
Probability =  0.0009765625
```

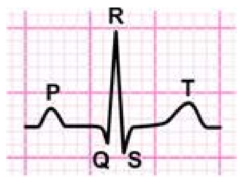
Same *p* as with the Forward algorithm

Applications of Problem 1 – *What is the chance that a pattern was generated by a HMM*

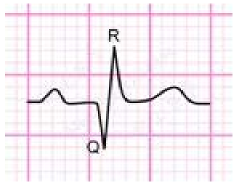
- Compare a sequence to a trained HMM for functional sequences such as TATA boxes, transcription factor binding sites, replication origins, centromeres, etc.



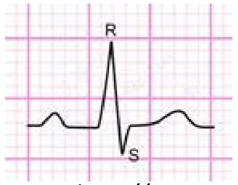
- A normal EKG is composed of three wave segments: the P, the QRS complex and the T



normal



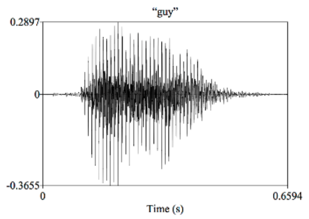
QR deflection

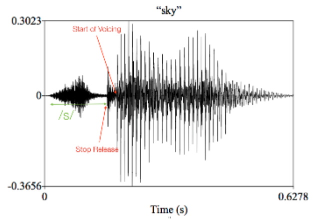


RS deflection

http://www.medicine-on-line.com

- The measured EKG can be compared to normal and abnormal HMM to detect cardiac problems
- Word and image recognition





http://linguisticmystic.com

Excuse me while I kiss the sky

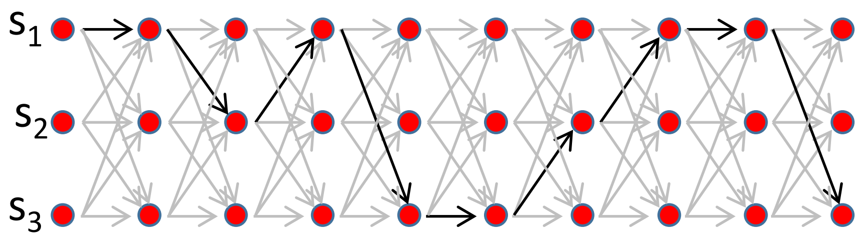
vs

Excuse me while I kiss this guy

Jimmy Hendrix – Purple Haze

Problem 2 – *What is the most likely series of states to have produced a pattern*

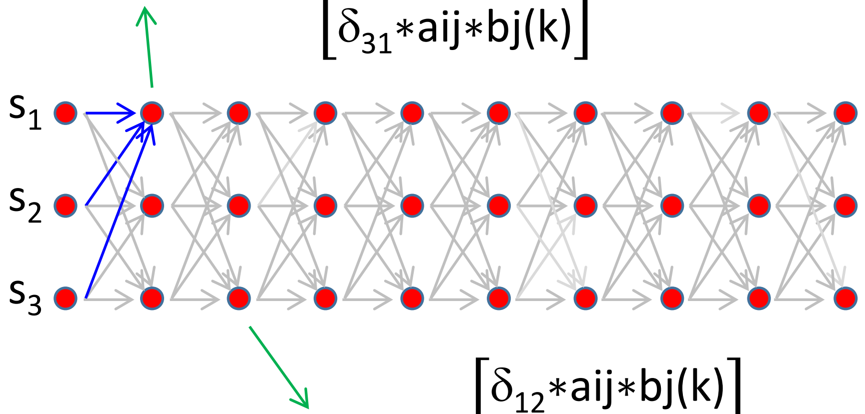
Given **observation sequence $O = O_1,O_2,…,O_N$** and the model **$\lambda = (A,B,\pi)$**
How do we compute a **series of states $Q = \{q_1,q_2,…,q_N\}$** that is **likely** to have **produced O**?



N^T possible paths (light grey arrows), i.e. **$O(N^T)$** – unfeasible calculation
The **Viterbi algorithm** finds a **path** that results in the **largest cumulative probability** of the output pattern O (illustrated by the black arrows)
Viterbi is **related** to the **Forward algorithm**, but records the **maximum probability** for the transitions to a state q_i, as **opposed to the sum of all probabilities** for the q_{i-1} to q_i transition
Viterbi algorithm complexity: **$O(N^2T)$**
Dynamic programming type algorithm

Viterbi algorithm

$$\delta_{12} = \max\begin{bmatrix}\delta_{11} * aij * bj(k)\\ \delta_{21} * aij * bj(k)\\ \delta_{31} * aij * bj(k)\end{bmatrix}$$



$$\delta_{33} = \max\begin{bmatrix}\delta_{12} * aij * bj(k)\\ \delta_{22} * aij * bj(k)\\ \delta_{32} * aij * bj(k)\end{bmatrix}$$

$\delta_{i1} = \max[\pi_i * b_i(k)] \quad t = 1$

$\delta_{it} = \max[\delta_{it-1} * a_{ij} * b_j(k)] \quad 2 \leq t \leq N$

For the maximum δ_{it} for every state *i* at every time *t*, **record the δ_{it-1}** that **resulted** in the **current max δ_{it}** in **matrix Ψ_t**

At t = T, choose the **maximum δ_{it}**, and **trace the path** that resulted in that maximum using the **Ψ_t matrix** back to t=1

Viterbi algorithm code

```
pi_matrix = np.array([0.4,0.3,0.3],float)
a_matrix = np.array([[0.1,0.4,0.5],[0.3,0.4,0.3],[0.2,0.3,0.5]],float)
b_matrix = np.array([[0.5,0.5],[0.2,0.2],[0.1,0.1]],float)
pattern_list = [0,0,0,0,0,1,1,1,1,1]
```

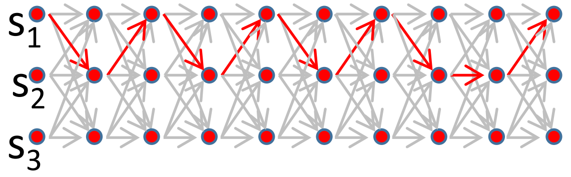
Note the emission probability of state 3 is low

```
def viterbi(pi_matrix,a_matrix,b_matrix,pattern_list):
        number_of_states = len(a_matrix)
        length = len(pattern_list)
        delta_matrix = np.zeros((number_of_states,length),dtype = float)
        temp_delta_matrix = np.zeros(number_of_states,dtype = float)
        phi_matrix = np.zeros((number_of_states,length), dtype=int)
        path_matrix = np.zeros((length), dtype=int)
        for position_in_pattern in range(length):
            for current_state in range(number_of_states):
                for previous_state in range(number_of_states):
                    if(position_in_pattern == 0): #handle t=1 use pi_matrix
                        temp_delta_matrix[previous_state] = pi_matrix[previous_state]*
                      b_matrix[current_state,pattern_list[position_in_pattern]]
                    else:
                        temp_delta_matrix[previous_state] = delta_matrix[previous_state,
                      position_in_pattern-1]*a_matrix[previous_state,current_state]*
                      b_matrix[current_state,pattern_list[position_in_pattern]]
                delta_matrix[current_state,position_in_pattern] =
                np.max(temp_delta_matrix)
                phi_matrix[current_state,position_in_pattern] =
                np.argmax(temp_delta_matrix)
        path_matrix[length-1]=np.argmax(delta_matrix[:,length-1])
        for position in range(length-1,0,-1):
            path_matrix[position-1] = phi_matrix[path_matrix[position],position]
        return(path_matrix)
```

Viterbi algorithm output

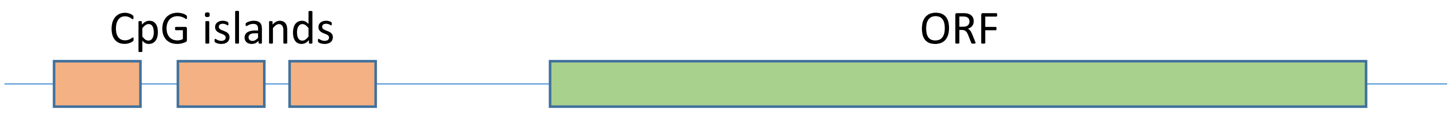
```
t = 1 delta 0 = 0.2     delta 1 = 0.15    delta 2 = 0.15    max = 0.2
t = 1 delta 0 = 0.08    delta 1 = 0.06    delta 2 = 0.06    max = 0.08
t = 1 delta 0 = 0.04    delta 1 = 0.03    delta 2 = 0.03    max = 0.04
t = 2 delta 0 = 0.01    delta 1 = 0.012   delta 2 = 0.004   max = 0.012
t = 2 delta 0 = 0.016   delta 1 = 0.0064  delta 2 = 0.0024  max = 0.016
t = 2 delta 0 = 0.01    delta 1 = 0.0024  delta 2 = 0.002   max = 0.01
t = 3 delta 0 = 0.0006  delta 1 = 0.0024  delta 2 = 0.001   max = 0.0024
t = 3 delta 0 = 0.00096 delta 1 = 0.00128 delta 2 = 0.0006  max = 0.00128
t = 3 delta 0 = 0.0006  delta 1 = 0.00048 delta 2 = 0.0005  max = 0.0006
t = 4 delta 0 = 0.00012 delta 1 = 0.000192      delta 2 = 6e-05   max = 0.000192
t = 4 delta 0 = 0.000192      delta 1 = 0.0001024     delta 2 = 3.6e-05 max = 0.000192
t = 4 delta 0 = 0.00012 delta 1 = 3.84e-05      delta 2 = 3e-05   max = 0.00012
t = 5 delta 0 = 9.6e-06 delta 1 = 2.88e-05      delta 2 = 1.2e-05 max = 2.88e-05
t = 5 delta 0 = 1.536e-05     delta 1 = 1.536e-05     delta 2 = 7.2e-06 max = 1.536e-05
t = 5 delta 0 = 9.6e-06 delta 1 = 5.76e-06      delta 2 = 6e-06   max = 9.6e-06
t = 6 delta 0 = 1.44e-06      delta 1 = 2.304e-06     delta 2 = 9.6e-07 max = 2.304e-06
t = 6 delta 0 = 2.304e-06     delta 1 = 1.2288e-06    delta 2 = 5.76e-07      max = 2.304e-06
t = 6 delta 0 = 1.44e-06      delta 1 = 4.608e-07     delta 2 = 4.8e-07 max = 1.44e-06
t = 7 delta 0 = 1.152e-07     delta 1 = 3.456e-07     delta 2 = 1.44e-07      max = 3.456e-07
t = 7 delta 0 = 1.8432e-07    delta 1 = 1.8432e-07    delta 2 = 8.64e-08      max = 1.8432e-07
t = 7 delta 0 = 1.152e-07     delta 1 = 6.912e-08     delta 2 = 7.2e-08 max = 1.152e-07
t = 8 delta 0 = 1.728e-08     delta 1 = 2.7648e-08    delta 2 = 1.152e-08     max = 2.7648e-08
t = 8 delta 0 = 2.7648e-08    delta 1 = 1.47456e-08   delta 2 = 6.912e-09     max = 2.7648e-08
t = 8 delta 0 = 1.728e-08     delta 1 = 5.5296e-09    delta 2 = 5.76e-09      max = 1.728e-08
t = 9 delta 0 = 1.3824e-09    delta 1 = 4.1472e-09    delta 2 = 1.728e-09     max = 4.1472e-09
t = 9 delta 0 = 2.21184e-09   delta 1 = 2.21184e-09   delta 2 = 1.0368e-09    max = 2.21184e-09
t = 9 delta 0 = 1.3824e-09    delta 1 = 8.2944e-10    delta 2 = 8.64e-10      max = 1.3824e-09
t = 10      delta 0 = 2.0736e-10    delta 1 = 3.31776e-10   delta 2 = 1.3824e-10    max = 3.31776e-10
t = 10      delta 0 = 3.31776e-10   delta 1 = 1.769472e-10  delta 2 = 8.2944e-11    max = 3.31776e-10
t = 10      delta 0 = 2.0736e-10    delta 1 = 6.63552e-11   delta 2 = 6.912e-11     max = 2.0736e-10
```

optimum path = [0 1 0 1 0 1 0 1 1 0]

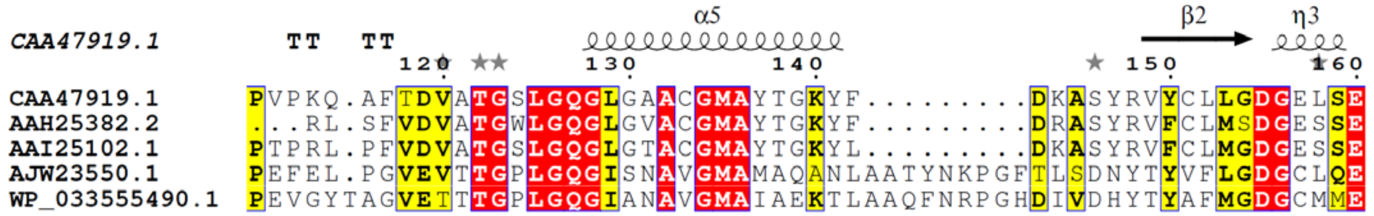


Applications of Problem 2 – What is the most likely series of states to have produced a pattern?

Identifying ORFs, intergenic regions, CpG islands etc. by base composition



Multiple sequence alignments



```
CAA47919.1              TT   TT                α5                                   β2      η3
                              120  ★★        130                140               ★  150            160

CAA47919.1      PVPKQ.AFTDVATGSLGQGLGAACGMAYTGKYF.........DKASYRVYCLLGDGELSE
AAH25382.2      ...RL.SFVDVATGWLGQGLGVACGMAYTGKYF.........DRASYRVFCLMSDGESSE
AAI25102.1      PTPRL.PFVDVATGSLGQGLGTACGMAYTGKYL.........DKASYRVFCLMGDGESSE
AJW23550.1      PEFEL.PGVEVTTGPLGQGISNAVGMAMAQANLAATYNKPGFTLSDNYTYVFLGDGCLQE
WP_033555490.1  PEVGYTAGVETTTGPLGQGIANAVGMAIAEKTLAAQFNRPGHDIVDHYTYAFMGDGCMME
```

Matching to protein profiles and domains