

✓✓ Outline

- Markov Chain
- HMM (Hidden Markov Model)
- Hidden Markov Models in Bioinformatics
- Gene Finding
- Gene Finding Model
- Viterbi algorithm
- HMM Advantages
- HMM Disadvantages
- Conclusions

Markov Chain

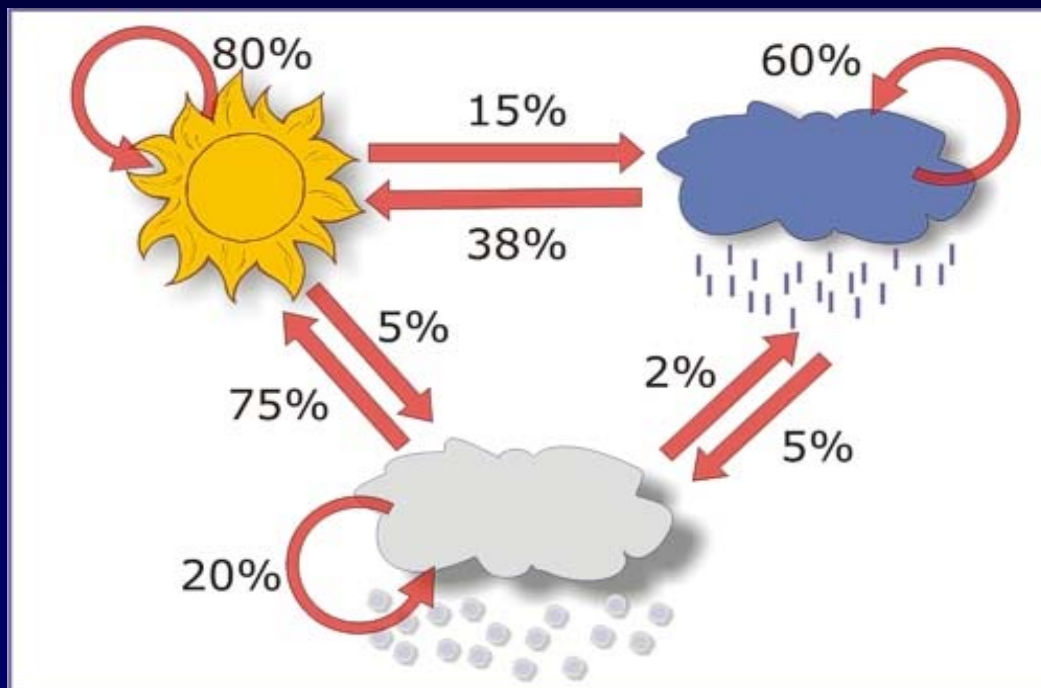
Definition: A **Markov chain** is a triplet $(Q, \{p(x_1 = s)\}, A)$, where:

- **Q** is a finite set of states. Each state corresponds to a symbol in the alphabet
- **p** is the initial state probabilities.
- **A** is the state transition probabilities, denoted by a_{st} for each $s, t \in Q$.
- For each $s, t \in Q$ the transition probability is:
$$a_{st} \equiv P(x_i = t | x_{i-1} = s)$$

Output: The output of the model is the set of states at each instant time => the set of states are observable

Property: The probability of each symbol x_i depends only on the value of the preceding symbol x_{i-1} : $P(x_i | x_{i-1}, \dots, x_1) = P(x_i | x_{i-1})$

Example of a Markov Model



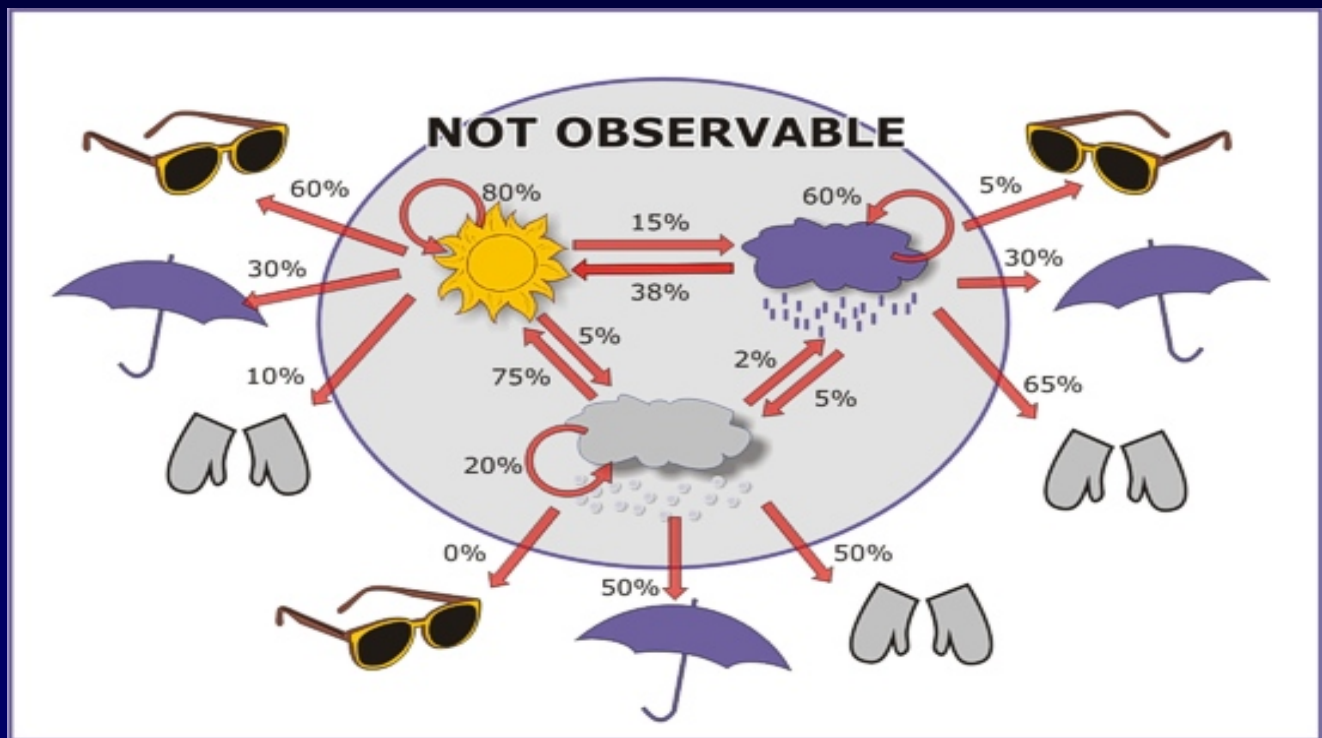
HMM (Hidden Markov Model)

Definition: An **HMM** is a 5-tuple (Q, V, p, A, E) , where:

- Q is a finite set of states, $|Q|=N$
- V is a finite set of observation symbols per state, $|V|=M$
- p is the initial state probabilities.
- A is the state transition probabilities, denoted by a_{st} for each $s, t \in Q$.
- For each $s, t \in Q$ the transition probability is:
 $a_{st} \equiv P(x_i = t | x_{i-1} = s)$
- E is a probability emission matrix, $e_{sk} \equiv P(v_k \text{ at time } t | q_t = s)$

Output: Only **emitted symbols** are observable by the system but not the underlying random walk between states \rightarrow "hidden"

Example of a Hidden Markov Model



The HMMs can be applied efficiently to well known biological problems. That's why HMMs gained popularity in bioinformatics, and are used for a variety of biological problems like:

✓ protein secondary structure recognition

• multiple sequence alignment

✓ gene finding

Handwritten red notes and scribbles on the right side of the slide, including the word "input" and various lines and loops.

What HMMs do?

A HMM is a statistical model for sequences of discrete symbols.
Hmms are used for many years in speech recognition.

HMMs are *perfect* for the gene finding task.

Categorizing nucleotids within a genomic sequence can be interpreted as a clasification problem with a set of ordered observations that posses hidden structure, that is a suitable problem for the application of hidden Markov models.

~~Hidden~~ Markov Models in Bioinformatics

The most challenging and interesting problems in computational biology at the moment is finding genes in DNA sequences. With so many genomes being sequenced so rapidly, it remains important to begin by identifying genes computationally.

Gene Finding

Gene finding refers to identifying stretches of nucleotide sequences in genomic DNA that are biologically functional.

Computational gene finding deals with algorithmically identifying protein-coding genes.

Gene finding is not an easy task, as gene structure can be very complex.

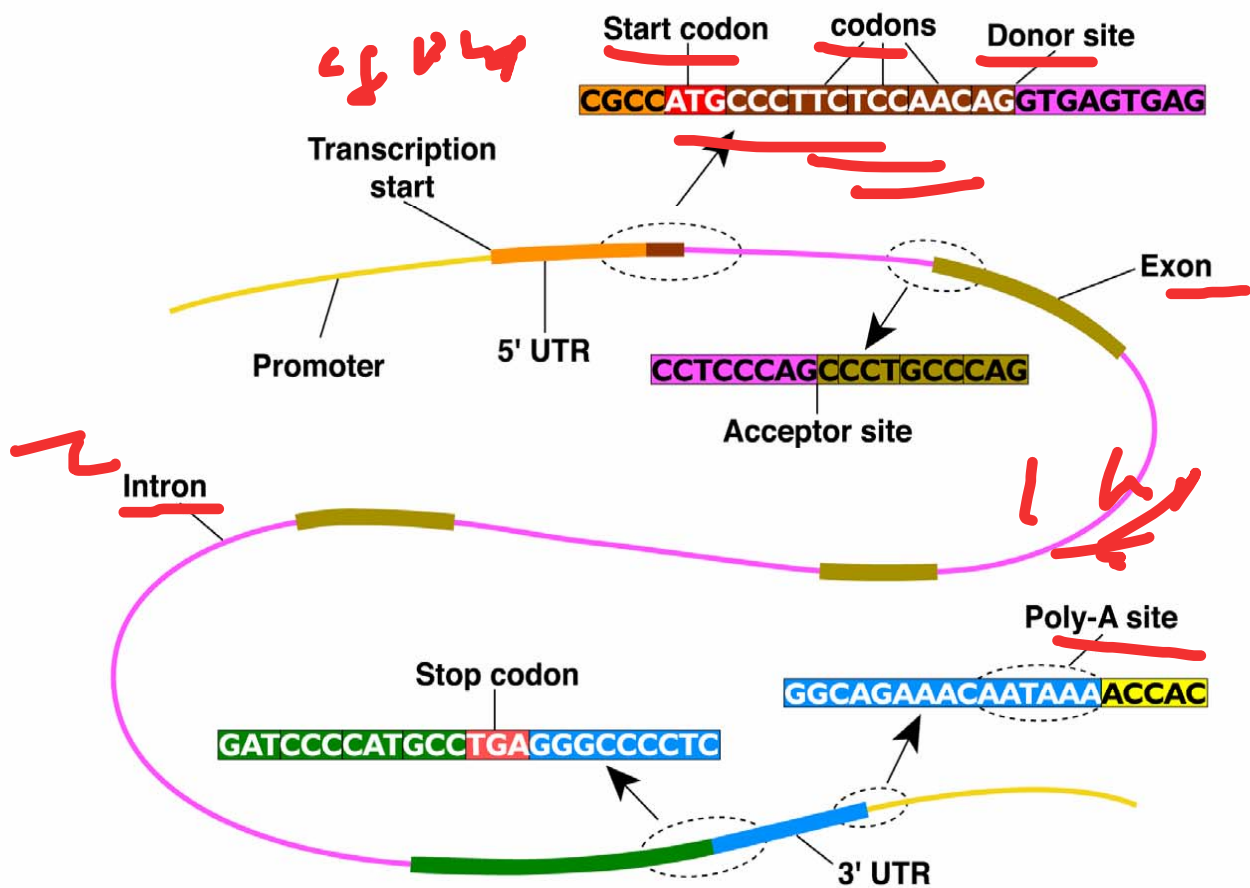
~~Objective:~~

1441
To find the coding and non-coding regions of an unlabeled string of DNA nucleotides

Motivation:

Assist in the annotation of genomic data produced by genome sequencing methods

Gain insight into the mechanisms involved in transcription, splicing and other processes



Structure of a gene

The ~~gene~~ is discontinuous, coding both:

- ♦ **exons** (a region that encodes a sequence of amino acids).
- ♦ **introns** (non-coding polynucleotide sequences that interrupts the coding sequences, the exons, of a gene) .

AAAGCATGCATTTAACGAGTGCATCAGGACTCCATACGTAATGCCG



Gene finder



AAAGC ATG CAT TTA ACG A GT GCATC AG GA CTC CAT ACG TAA TGCCG

we have
to double
the

In gene finding there are some important biological rules:

- ♦ ***Translation starts with a start codon (ATG).***
- ♦ ***Translation ends with a stop codon (TAG, TGA, TAA).***
- ♦ ***Exon can never follow an exon without an intron in between.***
- ♦ ***Complete genes can never end with an intron.***

Gene Finding Models

When using HMMs first we have to specify a model.

When choosing the model we have to take into consideration their complexity by:

- ♦ The number of states and allowed transitions.

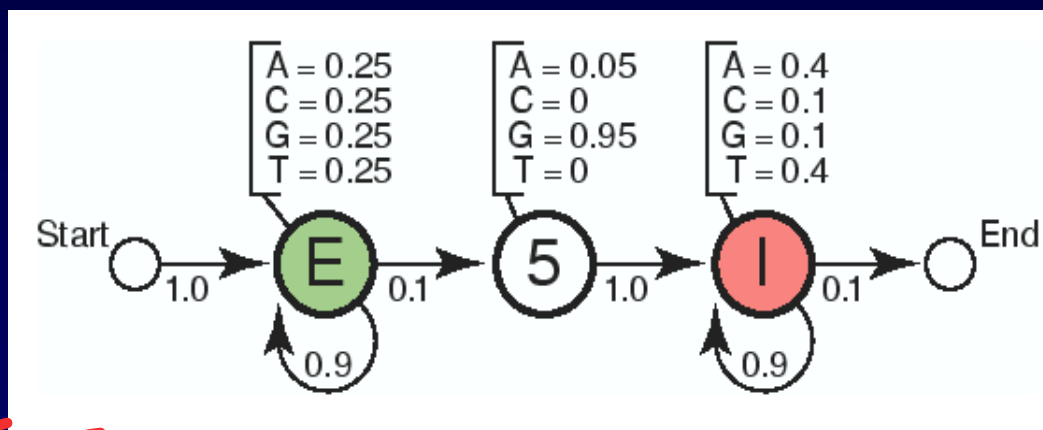
- ♦ How sophisticated the learning methods are.

- ♦ The learning time.

~~The~~ Model consists of a finite set of states, each of which can emit a symbol from a finite alphabet with a fixed probability distribution over those symbols, and a set of transitions between states, which allow the model to change state after each symbol is emitted.

The models can have different complexity, and different built in biological knowledge.

The model for the Viterbi algorithm



3-state

codon
max → 4

```
states = ('Begin', 'Exon', 'Donor', 'Intron')
```

```
observations = ('A', 'C', 'G', 'T')
```

The Model Probabilities

□ Transition probability:

transition_probability = {

'Begin' : {'Begin' : 0.0, 'Exon' : 1.0, 'Donor' : 0.0, 'Intron' : 0.0},

'Exon' : {'Begin' : 0.0, 'Exon' : 0.9, 'Donor' : 0.1, 'Intron' : 0.0},

'Donor' : {'Begin' : 0.0, 'Exon' : 0.0, 'Donor' : 0.0, 'Intron' : 1.0},

'Intron' : {'Begin' : 0.0, 'Exon' : 0.0, 'Donor' : 0.0, 'Intron' : 1.0}

}

A ✓✓

4x4
✓

□ Emission probability:

```
emission_probability = {  
'Begin' : {'A' :0.00 , 'C' :0.00, 'G' :0.00, 'T' :0.00},  
'Exon' : {'A' :0.25 , 'C' :0.25, 'G' :0.25, 'T' :0.25},  
'Donor' : {'A' :0.05 , 'C' :0.00, 'G' :0.95, 'T' :0.00},  
'Intron' : {'A' :0.40 , 'C' :0.10, 'G' :0.10, 'T' :0.40}  
}
```

5

3x4
7

Viterbi algorithm

Dynamic programming algorithm for finding the most likely sequence of hidden states.

The Vitebi algorithm finds the most probable path – called the Viterbi path .

$$\pi^* = \arg \max_{\pi} P(x, \pi),$$

~~The~~ main idea of the Viterbi algorithm is to find the most probable path for each intermediate state, until it reaches the end state.

At each time only the most likely path leading to each state survives.

The steps of the Viterbi algorithm

Initialisation ($i = 0$): $v_0(0) = 1, v_k(0) = 0$ for $k > 0$ $\xrightarrow{\log}$ $v_0(0) = 0, v_k(0) = -\text{Inf}$ for $k > 0$

Recursion ($i = 1 \dots L$): $v_l(i) = e_l(x_i) \max_k (v_k(i-1) a_{kl})$ $\xrightarrow{\log}$ $v_l(i) = e_l(x_i) + \max_k (v_k(i-1) + a_{kl})$
 $\text{ptr}_i(l) = \arg \max_k (v_k(i-1) a_{kl})$ $\xrightarrow{\log}$ $\text{ptr}_i(l) = \arg \max_k (v_k(i-1) + a_{kl})$

Termination: $P(x, \pi^*) = \max_k (v_k(L) a_{k0})$ $\xrightarrow{\log}$ $P(x, \pi^*) = \max_k (v_k(L) + a_{k0})$
 $\pi_L^* = \arg \max_k (v_k(L) a_{k0})$ $\xrightarrow{\log}$ $\pi_L^* = \arg \max_k (v_k(L) + a_{k0})$

Traceback ($i = L \dots 1$): $\pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$

The arguments of the Viterbi algorithm

```
viterbi(observations,  
       states,  
       start_probability,  
       transition_probability,  
       emission_probability)
```


The working of the Viterbi algorithm

The algorithm works on the mappings T and U.

The algorithm calculates **prob**, **v_path**, and **v_prob** where **prob** is the total probability of all paths from the start to the current state, **v_path** is the Viterbi path, and **v_prob** is the probability of the Viterbi path, and

The mapping T holds this information for a given point t in time, and the main loop constructs U, which holds similar information for time $t+1$.

The algorithm computes the triple $(\text{prob}, v_{\text{path}}, v_{\text{prob}})$ for each possible next state.

The total probability of a given next state, total is obtained by adding up the probabilities of all paths reaching that state. More precisely, the algorithm iterates over all possible source states.

For each source state, T holds the total probability of all paths to that state. This probability is then multiplied by the emission probability of the current observation and the transition probability from the source state to the next state.

The resulting probability prob is then added to total .

For each source state, the probability of the Viterbi path to that state is known.

This too is multiplied with the emission and transition probabilities and replaces valmax if it is greater than its current value.

The Viterbi path itself is computed as the corresponding argmax of that maximization, by extending the Viterbi path that leads to the current state with the next state.

The triple $(\text{prob}, \text{v_path}, \text{v_prob})$ computed in this fashion is stored in U and once U has been computed for all possible next states, it replaces T , thus ensuring that the loop invariant holds at the end of the iteration.

Example

Input DNA sequence:

CTTCATGTGAAAGCAGACGTAAGTCA

Result:

Total: $2.6339193049977711e-17$ – the sum of all the calculated probabilities

□ Viterbi Path:

['Exon', 'Exon', 'Exon', 'Exon', 'Exon', 'Exon', 'Exon',
'Exon', 'Exon', 'Exon', 'Exon', 'Exon', 'Exon', 'Exon',
'Exon', 'Exon', 'Exon', 'Exon', 'Donor', 'Intron', 'Intron',
'Intron', 'Intron', 'Intron', 'Intron', 'Intron', 'Intron']

□ Viterbi probability: 7.0825171238258092e-18

HMM Advantages

- **Statistics**

- ◆ *HMMs are very powerful modeling tools*
- ◆ *Statisticians are comfortable with the theory behind hidden Markov models*
- ◆ *Mathematical / theoretical analysis of the results and processes*

- **Modularity**

- ◆ *HMMs can be combined into larger HMMs*

- ***Transparency***

- ◆ *People can read the model and make sense of it*
- ◆ *The model itself can help increase understanding*

- ***Prior Knowledge***

- ◆ *Incorporate prior knowledge into the architecture*
- ◆ *Initialize the model close to something believed to be correct*
- ◆ *Use prior knowledge to constrain training process*

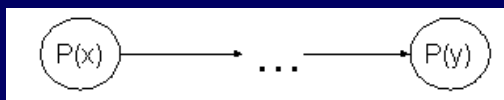
HMM Disadvantages

- *State independence*

- ◆ *States are supposed to be independent, $P(y)$ must be independent of $P(x)$, and vice versa. This usually isn't true*

- ◆ *Can get around it when relationships are local*

Not good for RNA folding problems



- **Over-fitting**

- ◆ *You're only as good as your training set*
- ◆ *More training is not always good*

- **Local maximums**

- ◆ *Model may not converge to a truly optimal parameter set for a given training set*

- **Speed**

- ◆ *Almost everything one does in an HMM involves:
“enumerating all possible paths through the model”*
- ◆ *Still slow in comparison to other methods*

Conclusions

- *HMMs have problems where they excel, and problems where they do not*
- *You should consider using one if:*
 - *The problem can be phrased as classification*
 - *The observations are ordered*
 - *The observations follow some sort of grammatical structure*
- *If an HMM does not fit, there's all sorts of other methods to try: Neural Networks, Decision Trees have all been applied to Bioinformatics*

Bibliography

Pierre Baldi, Soren Brunak: The machine learning approach

<http://www1.imim.es/courses/BioinformaticaUPF/Ttreballs/programming/donorsitemodel/index.html>

http://en.wikipedia.org/wiki/Viterbi_algorithm