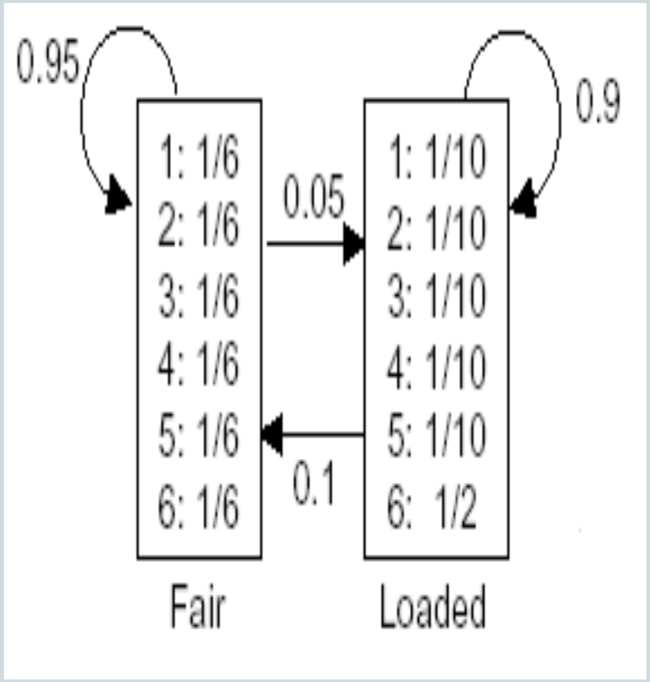
Lecture Mon 7.11.

HIDDEN MARKOV MODELS & GENE PREDICTION

Parameter estimation for HMMs

- So far we have assumed that we have knowledge of the transition probabilities and emission probabilities
- How to obtain these if we only know
  - the emitted sequence and HMM structure (here: Fair, Loaded)?
  - possibly the hidden state sequence



Parameter estimation when the state sequence is known

- Assume we have
  - a set of training sequences $x^{(1)},\ldots,x^{(n)}$ where $x^{(i)} = x_1^{(i)}\ldots x_{l(i)}^{(i)}$, e.g.
    - Sequences of rolls of dice: $x^{(1)} = 1,3,4,3,\ldots$, $x^{(2)} = 5,6,4,3,\ldots$
    - Nucleotide sequences $x^{(1)}$ = AGTCGT… $x^{(2)}$ = CTGTAT…,
  - The set of states and corresponding state sequences of HMM
    - Which die is being used: $y^{(1)}$ = FFFF…, $y^{(2)}$ = LLFF…
    - CpG-island / non-island: $y^{(1)}$ = NNNYYY…, $y^{(2)}$ = NNNNNN
- The goal is to optimize HMM parameters
  - Transition probabilities a_{kl}
  - Emission probabilities $e_k(x_i)$

Parameter estimation when the state sequence is known

- Transition probabilities,
  - we examine the given state sequences $y^{(1)},\ldots,y^{(n)}$
  - denote by A_{kl} the number of times transition k→l was taken among the sequences
  - Our estimate for the transition probability is

$$a_{kl} = \frac{A_{kl}+1}{\sum_{l'}(A_{kl'}+1)}$$

- Emission probabilities
  - Examine the emitted sequences $x^{(1)},\ldots,x^{(n)}$ and the state sequences $y^{(1)},\ldots,y^{(n)}$ together
  - Denote by $E_k(b)$ the number of times b was emitted while in state k
  - The estimate for emission probability is

$$e_k(b) = \frac{E_k(b)+1}{\sum_{b'}(E_k(b')+1)}$$

- '+1' is a pseudo-count to make all estimates non-zero

Pseudo-counts (A_{kl}+1, $E_k(b)$+1)

- Pseudo-counts are typically used to make the models less prone to overfitting due to insufficient data
- In HMMs, the pseudo-counts also correct a problem arising if some state k is not visited in the training data:
  - Related to ‘missing mass’ problem: need to allocate some probability to so far unseen events
- In general, the pseudo-counts can be any positive real numbers, however
  - too large numbers will override the training data
  - too small numbers will cause the parameters to overfit the training data (leads to poorer performance on new, yet unseen data)

Parameter estimation when the state sequence is unknown

- Depending on the application, assumption of the state sequence to be known may be valid
  - In many cases we have are training set that contains the states e.g. known coding regions in genes, known CpG islands, …
- In other applications, such an assumption is not valid
  - e.g. which die is used by the dishonest casino
  - Data from newly sequenced organisms where no annotation has not been done.

Parameter estimation when the state sequence is not known

- Assume we have
  - a set of training sequences $x^{(1)},\dots,x^{(n)}$, and the
  - set of states of the HMM
- The goal is to optimize HMM parameters
  - Transition probabilities a_{kl}
  - Emission probabilities $e_k(x_i)$
- Idea: choose the HMMs parameters so that the likelihood of the training data is maximized (in a certain sense)
- In the following, we present a training algoritm that uses path as a subroutine the Viterbi algorithm to find the most probable path

Viterbi training

1. Initialize the HMM parameters in some way, e.g. setting
   1. $e_k(x) = 1/|X|$ uniformly, where X is the set of possible symbols to emit
   2. $a_{kl} = 1/N(k)$ uniformly, where N(k) is the set of states that can follow k

- Alternatively, one can use a "best guess"
  - e.g. in the CpG island example, compute transition probabilities from dinucleotide frequencies

Viterbi training

2. Iterate the following, until parameters do not change:
    1. For each sequence $x^{(i)}$, using Viterbi algorithm, find the most probable state sequence $\pi^{*(i)}$, given the current HMM parameters $\theta=(a,e)$
    2. Count how many times each transition $k \rightarrow l$ was taken in the optimal paths $\pi^{*(1)},\ldots\pi^{*(n)}$, denote that number by A_{kl}
    3. Set the new transition probabilities as $$a_{kl} = \frac{A_{kl}+1}{\sum_{l'}(A_{kl}+1)}$$
    4. Count how many times each symbol s was emitted in each state k, denote that number by $E_k(s)$
    5. Set the new emission probabilities as $$e_k(b) = \frac{E_k(b)+1}{\sum_{b'}(E_k(b')+1)}$$

Viterbi training

- The above algorithm works in *batch mode*: it assumes all training data is already available
- The training can also work in *online mode*, where the model is re-estimated when new data arrives
- Also, the training can work just as well on a single long sequence as on a set of short sequences
- The casino example highlights this training mode

Viterbi training at the casino

- Let us enter the occasionaly dihonest casino, with our HMM, with initial guesses about the underlying model:

a	Fair	Loaded
Fair	.90	.10
Loaded	.10	.90

e	1	2	3	4	5	6
Fair	.167	.167	.167	.167	.167	.167
Loaded	.10	.10	.10	.10	.10	.50

- We observe a sequence of rolls: 3,4,6,4,6,6,2,6,3,4,1,5,3

Viterbi training at the casino

- We observe a sequence of rolls: 3,4,6,4,6,6,2,6,3,4,1,5,3
- With Viterbi estimation with the current model, we get: LLLLLLLLFFFFF
- Count transitions and emissions, add pseudo-counts

A+1	Fair	Loaded
Fair	4+1	0+1
Loaded	1+1	7+1

E+1	1	2	3	4	5	6
Fair	1+1	0+1	2+1	1+1	1+1	0+1
Loaded	0+1	1+1	1+1	2+1	0+1	4+1

Viterbi training at the casino

- Normalize to obtain estimated transition and emission probabilities

A+1	Fair	Loaded
Fair	4+1	0+1
Loaded	1+1	7+1

E+1	1	2	3	4	5	6
Fair	1+1	0+1	2+1	1+1	1+1	0+1
Loaded	0+1	1+1	1+1	2+1	0+1	4+1

a	Fair	Loaded
Fair	.83	.17
Loaded	.18	.82

e	1	2	3	4	5	6
Fair	.18	.09	.27	.18	.18	.09
Loaded	.07	.14	.14	.21	.07	.36

- We observe some more rolls: 5,3,4,2,1, 6,1,6,6,2,6,5

Viterbi training at the casino

- All rolls seen so far: 3,4,6,4,6,6,2,6,3,4,1,5,3,5,3,4,2,1, 6,1,6,6,2,6,5
- Viterbi estimation with the new model gives: LLLLLLLFFFFFFFFFFLLLLLLL
- Count transitions and emissions in all rolls seen so far, add pseudo-counts

A+1	Fair	Loaded
Fair	9+1	1+1
Loaded	1+1	13+1

E+1	1	2	3	4	5	6
Fair	2+1	1+1	3+1	2+1	2+1	0+1
Loaded	1+1	2+1	1+1	2+1	1+1	8+1

Viterbi training at the casino

- Normalize to obtain estimated transition and emission probabilities

A+1	Fair	Loaded
Fair	9+1	1+1
Loaded	1+1	13+1

E+1	1	2	3	4	5	6
Fair	2+1	1+1	3+1	2+1	2+1	0+1
Loaded	1+1	2+1	1+1	2+1	1+1	8+1

a	Fair	Loaded
Fair	.83	.17
Loaded	.125	.875

e	1	2	3	4	5	6
Fair	.187	.125	.25	.187	.187	.063
Loaded	.095	.14	.095	.14	.095	.43

- Casino closes, so we do not get more rolls, but we can continue training with the current data

Viterbi training at the casino

- All rolls seen so far: 3,4,6,4,6,6,2,6,3,4,1,5,3,5,3,4,2,1, 6,1,6,6,2,6,5
- Viterbi estimation with the new model gives: LLLLLLLFFFFFFFFFFLLLLLLL
- This turns out to be the same predicted sequence as in previous step, so our model stays the same

a	Fair	Loaded
Fair	.83	.17
Loaded	.125	.875

e	1	2	3	4	5	6
Fair	.187	.125	.25	.187	.187	.063
Loaded	.095	.14	.095	.14	.095	.43

- In general, with a longer sequence, more interations could be needed for convergence

Viterbi training: convergence

- **If no more data arises** Viterbi training algorithm will eventually converge (and stop)
- Each update of the parameters increase the probability of the most probable paths,
  - so the algorithm will never revisit a previous solution
- There are only finite (but large) number of Viterbi paths to consider,
  - so we will eventually run out of solutions that we have not considered

Accuracy of estimation depends on the amount of training data

True Model	Fair	Loaded
Fair	.95	.05
Loaded	.10	.90

True	1	2	3	4	5	6
Fair	.17	.17	.17	.17	.17	.17
Loaded	.10	.10	.10	.10	.10	.50

300 rolls	Fair	Loaded
Fair	.73	.27
Loaded	.29	.71

300 rolls	1	2	3	4	5	6
Fair	.19	.19	.23	.08	.23	.08
Loaded	.07	.10	.10	.17	.05	.52

30000 rolls	Fair	Loaded
Fair	.93	.07
Loaded	.12	.88

30000 rolls	1	2	3	4	5	6
Fair	.17	.17	.17	.17	.17	.15
Loaded	.10	.11	.10	.11	.10	.48

Other tasks and algorithms for HMMs

- Forward algorithm:
  - finds the probability of the sequence, given all the paths: $P(x) = \sum_{\pi} P(x,\pi)$
- Forward-backward algorithm: finding posterior state probabilities given the observed sequence

$$P(\pi_i = k \mid x)$$

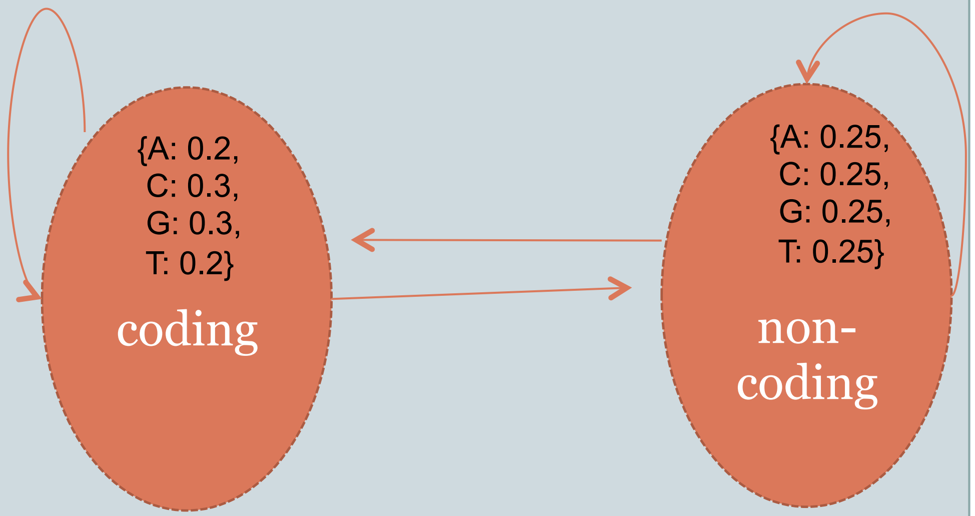
- Baum-Welch algorithm: another training algorithm for HMMs
  - Uses forward-backward algorithm as a subroutine
- All are dynamic programming methods operating along the sequence in forward and/or backward fashion

Part II

MARKOV METHODS FOR GENE PREDICTION

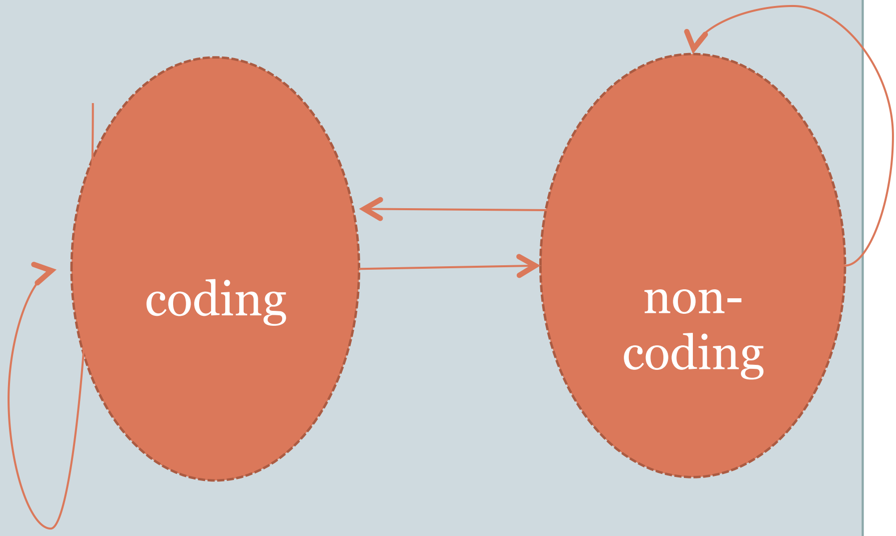
Gene prediction with HMMs: 1st try

- Could the HMM approach used at the occasionally dishonest casino directly mapped to gene prediction?
- Recognition of coding regions could be formulated as structurally equivalent HMM



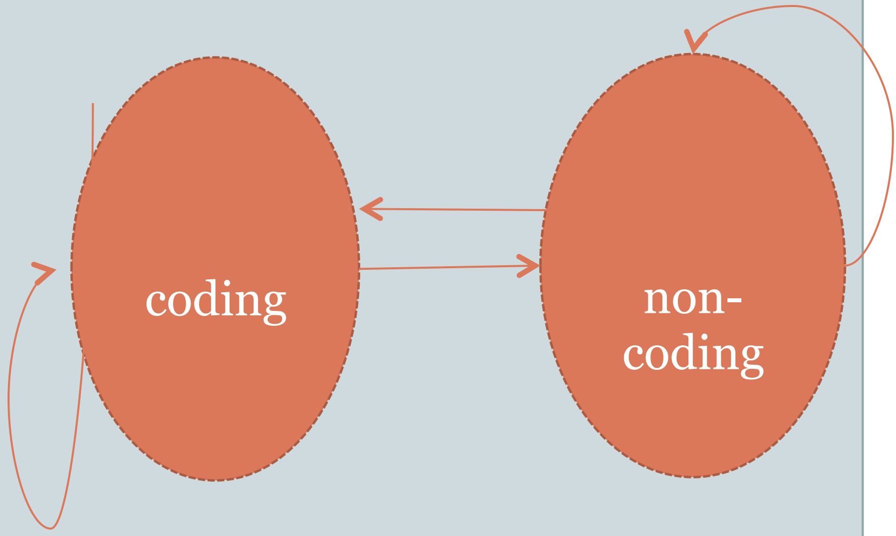
Gene prediction with HMMs: 1st try

- Two states: one for coding region, one for non-coding region
- Both states emit nucleotides according to their own distributions
- What can/cannot this HMM learn from the sequence data?



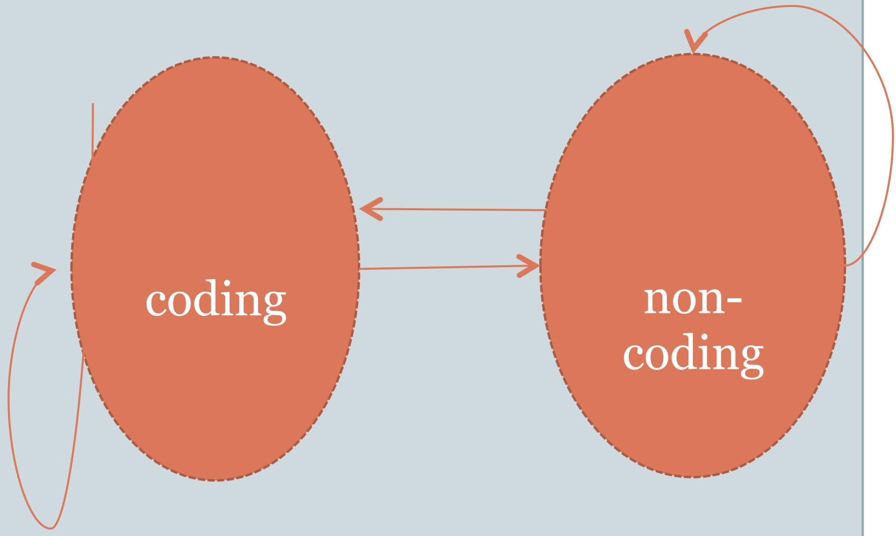
Gene prediction with HMMs: 1st try

- The HMM can learn
  - via the transition probabilities, statistics of the lengths of the respective regions
  - via the emission probabilities, the nucleotide distributions
- It cannot learn
  - Higher order statistics (dinucleotides, codons) within a region
- Not enough to recognize coding regions well



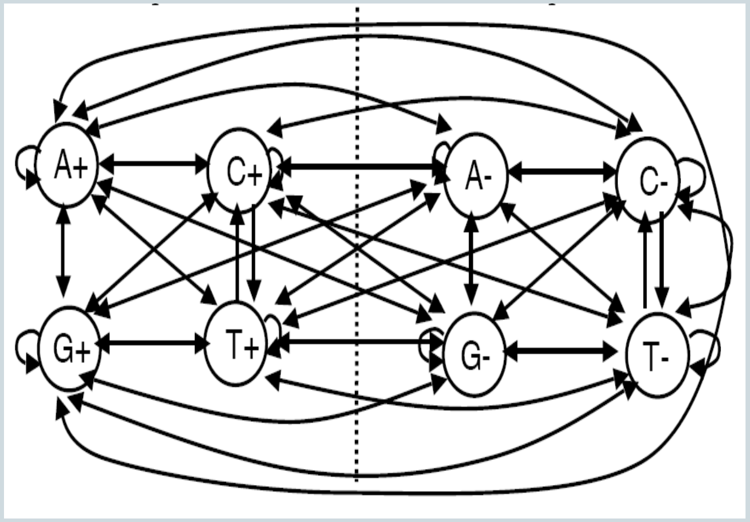
Gene prediction with HMMs: 1st try

- The HMM can learn
  - via the transition probabilities, statistics of the lengths of the respective regions
  - via the emission probabilities, the nucleotide distributions
- It cannot learn
  - Higher order statistics (dinucleotides, codons) within a region
- Not enough to recognize coding regions well



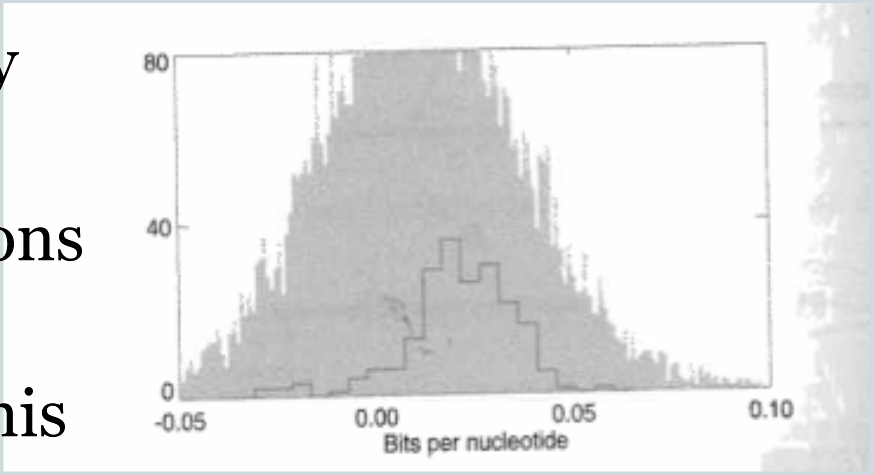
Gene prediction with HMMs: 2nd try

- What about borrowing the CpG model?
- 4 states for coding regions, 4 states for non-coding regions
- Can learn
  - Length statistics via the transition probabilities
  - Statistics of dinucleotides,
- Codons represented by chains of two transitions
  - Cannot represent the start and stop codons explicitly



Gene prediction with HMMs: 2nd try

- Log-odds scores of a 4-state Markov chain normalized by the length: S(x)/L
- Comparison model is one that assumes all nucleotides occurring independently
- Distributions from coding regions (black line) and non-coding regions (grey area) are shown
- Coding regions score slightly higher on average
- However, the two distributions overlap completely
- Cannot predict genes with this model

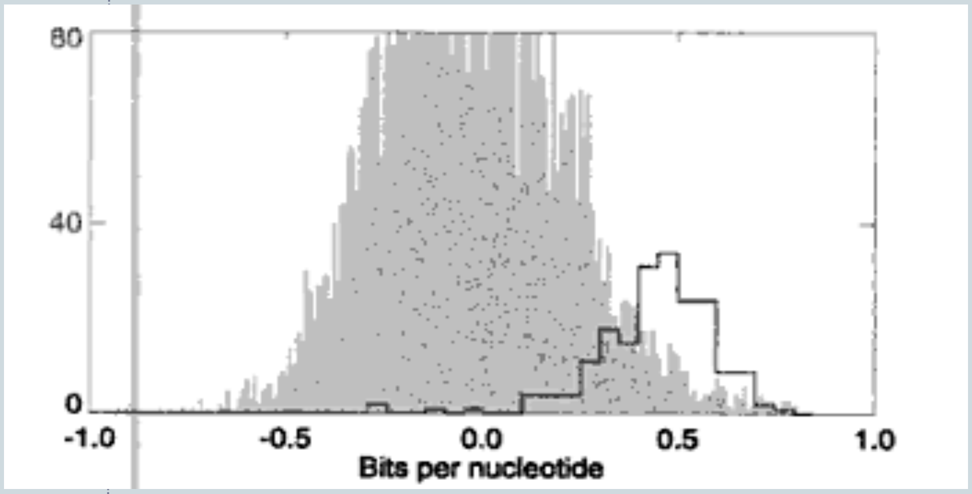


Modelling codon usage

- Try to model codons explicitly
- Transform the nucleotide sequences into a sequences of codons
  - Unique letter assigned to each of the $4^3 = 64$ different codons (AAA->s1,AAC,->s2,...TTT->s64)
  - Yields sequences that are 1/3 of the length of the original sequences
- We get a single 64-state first-order Markov chain
- Can represent distributions of codon usage
  - Known to be different in coding regions and non-coding regions

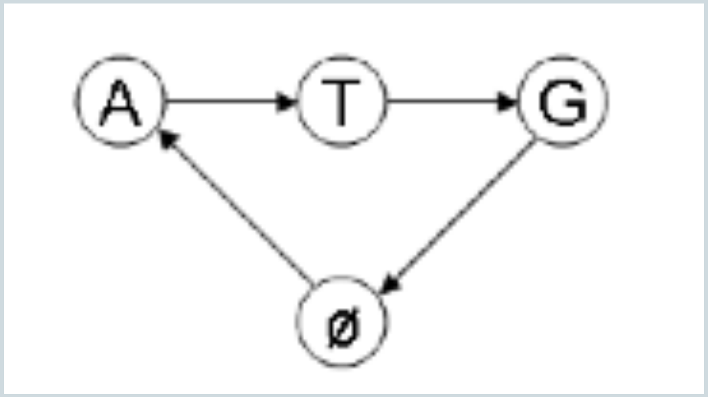
Modelling codon usage

- Log-odds scores (normalized by sequence length) between the coding (black line) and non-coding regions (grey histogram) are shown
- The Markov chain is able to score coding regions higher than the non-coding regions
- Separation is not perfect, so the model would make many prediction errors



Modelling start and stop codons explictly

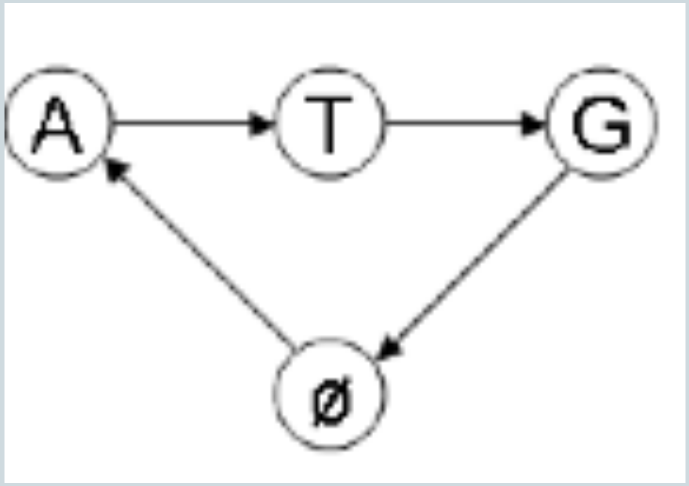
- The previous model treats start and stop codons just as the amino acid coding codons
- However, start and stop codons are distinct signals about the exact property that we are trying to learn here



Modelling start and stop codons explictly

- The previous model treats start and stop codons just as the amino acid coding codons
- However, start and stop codons are distinct signals about the exact property that we are trying to learn here

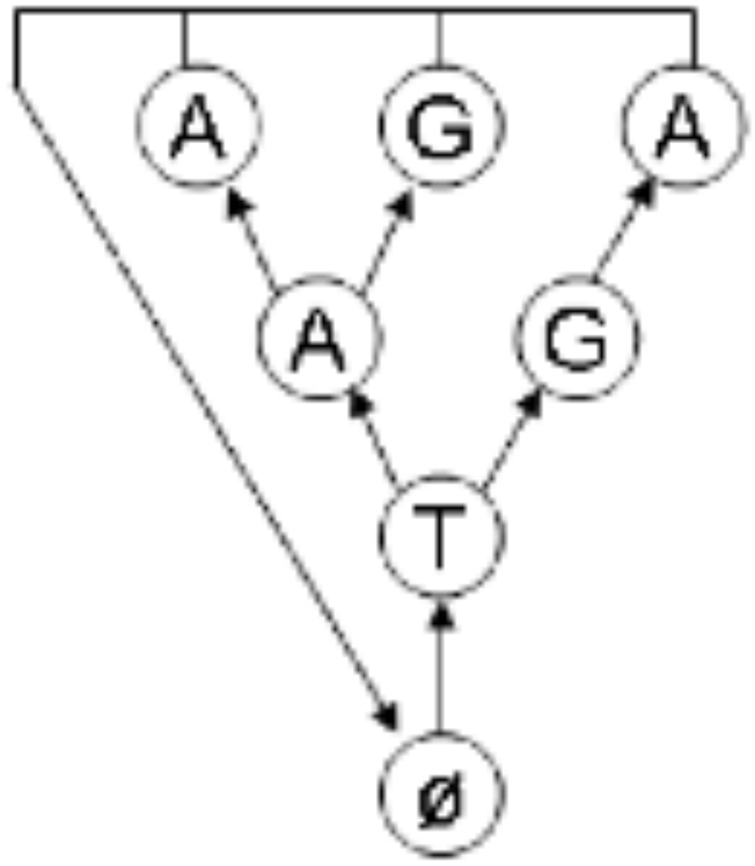
- The start codon is easily represented by a 3-state HMM-component


$$\begin{tikzcd}
A \arrow[r] & T \arrow[r] & G \arrow[dl] \\
& \emptyset \arrow[ul] &
\end{tikzcd}$$

Modelling start and stop codons explictly

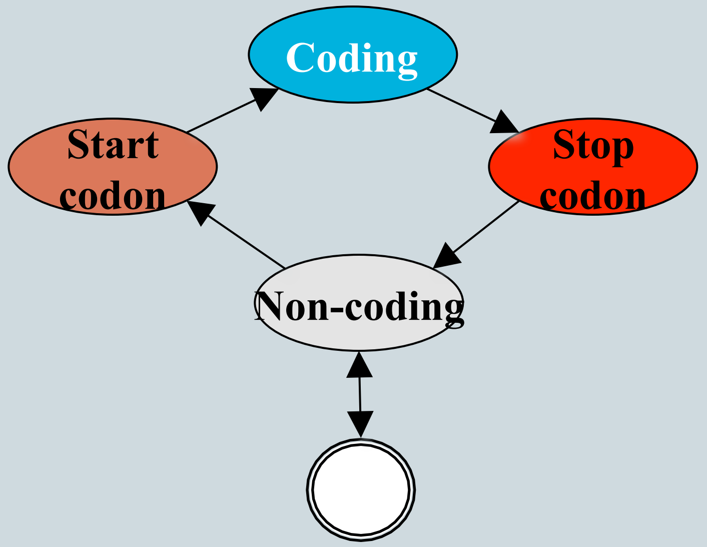
- The stop codons

(TAA, TAG, TGA) can be modeled as a 7-state HMM

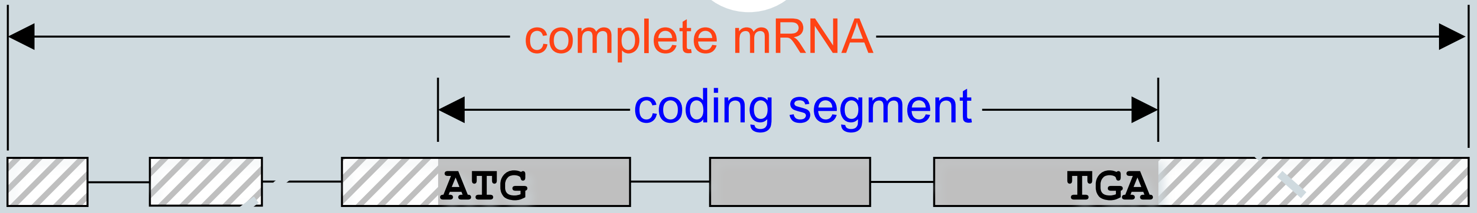


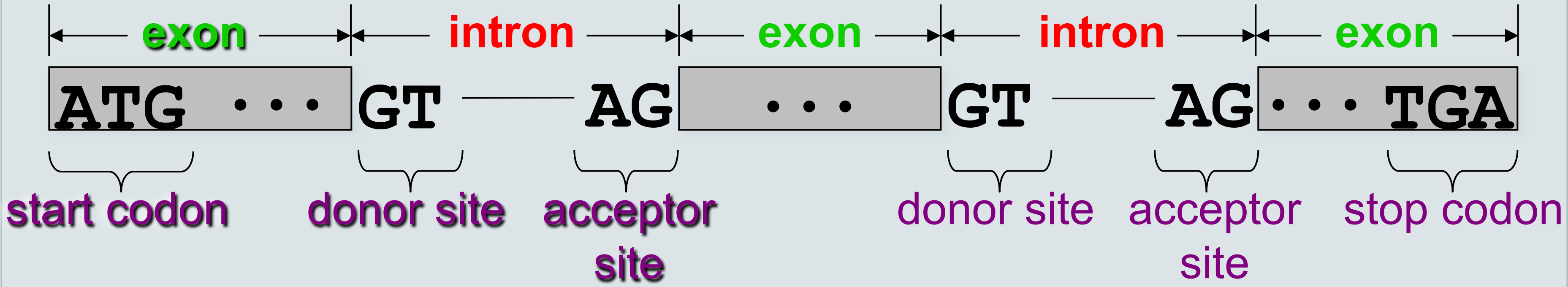
Overall architecture

- Overall architecture used in many prokaryotic gene finders consists of separate submodels for
  - Coding region (e.g. 61-state)
  - Non-coding region (at its simplest, just one state modelling the base dsitribution)
  - Start codon
  - Stop codon



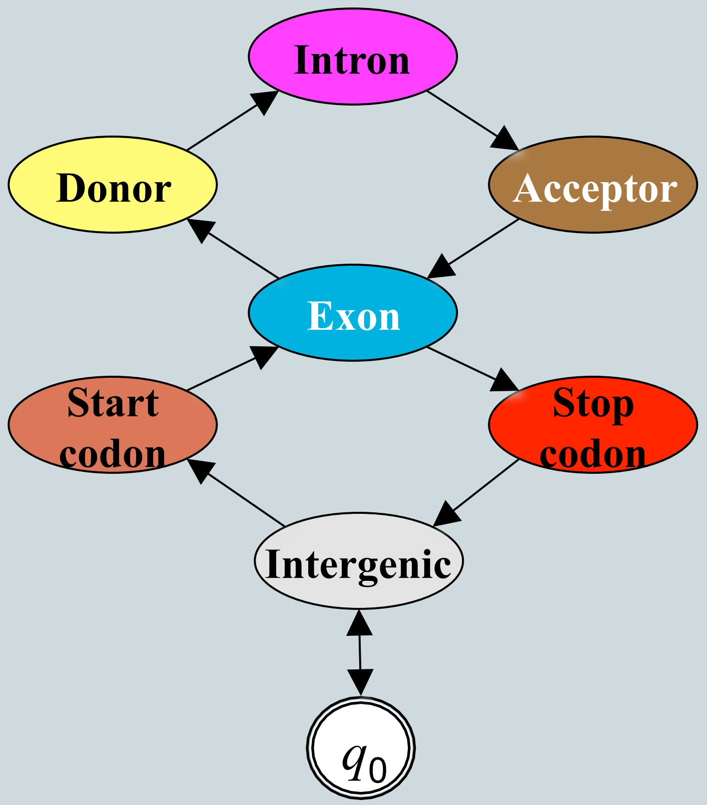
Eukaryotic Gene Structure





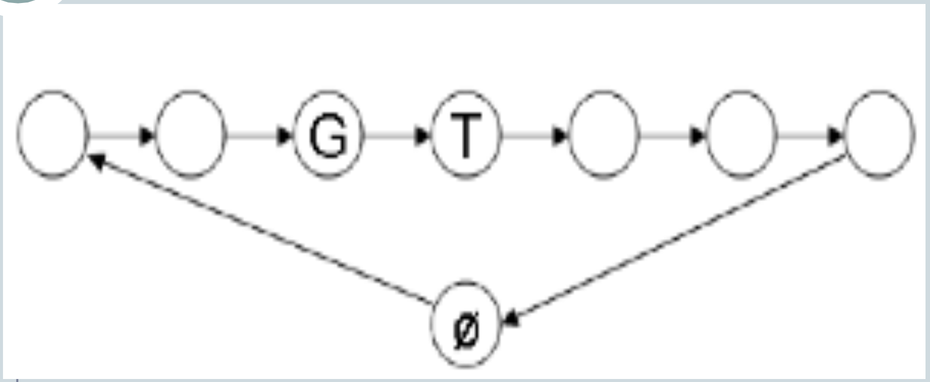
Eukaryotic Gene Prediction

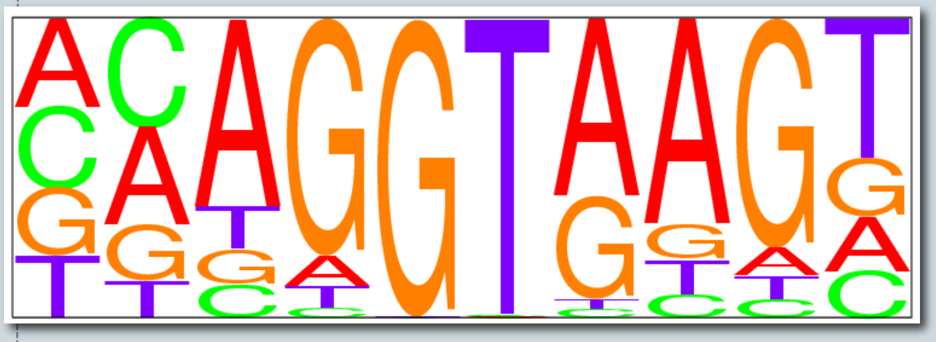
- Due to intro-exon structure, the overall structure of the HMMs is also more complex
- Separate states for introns and exons
- Donor and acceptor states model the transition between introns and exons explictly



Donor site submodel

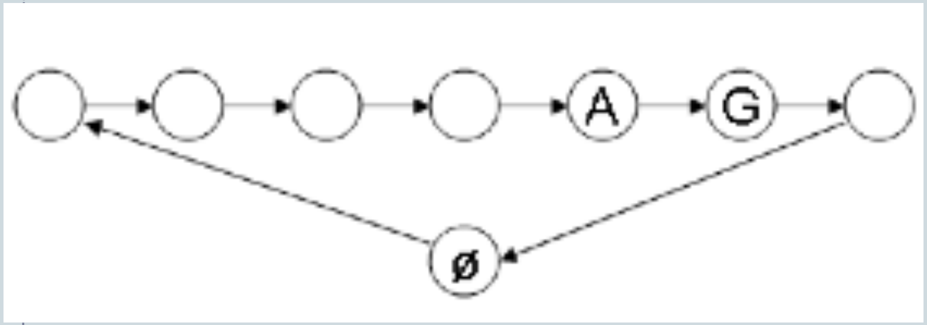
- Donor site is modelled by a HMM with two states exactly recognizing the 'GT' dinucleotide
- In addition, context before and after is modelled
- Right, a sequence logo representing donor site nucleotide frequencies is shown

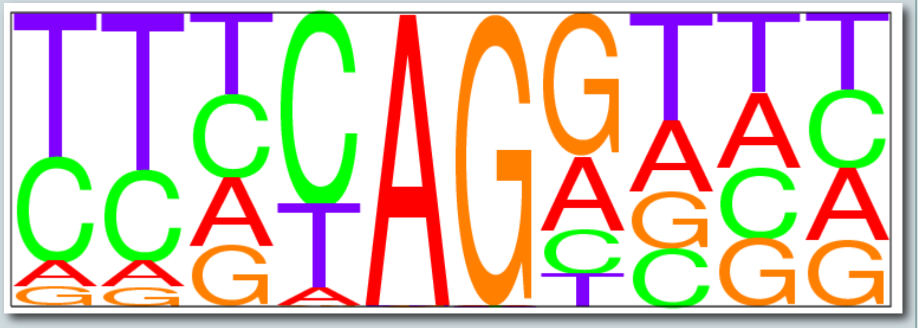




Acceptor site submodel

- Acceptor site is modelled by a HMM with two states exactly recognizing the 'AG' dinucleotide
- In addition, context before and after is modelled
- Right, a sequence logo representing acceptor site nucleotide frequencies is shown





Variants and extensions

- Many variants and generalizations of HMMs are in use in real world gene finders:
  - Higher-order HMMs whose emission probabilities also depend on previously emitted symbols
  - HMMs that emit more complex features, e.g. motifs
  - HMMs that allow variable length contexts (i.e. mixing HMMs with different order)
  - HMMs that allow modelling the duration of staying in a state more explicitly