Genome Analysis: Basic sequence classification

Dr. Jared Simpson
Ontario Institute for Cancer Research
&
Department of Computer Science
University of Toronto

Picking up signals

So far, we've focused on how to stitch fragments of evidence into longer units, i.e. genomes

Once the genome is assembled, we can ask more questions:

Where are the genes?

Where/what is the *functional* DNA?

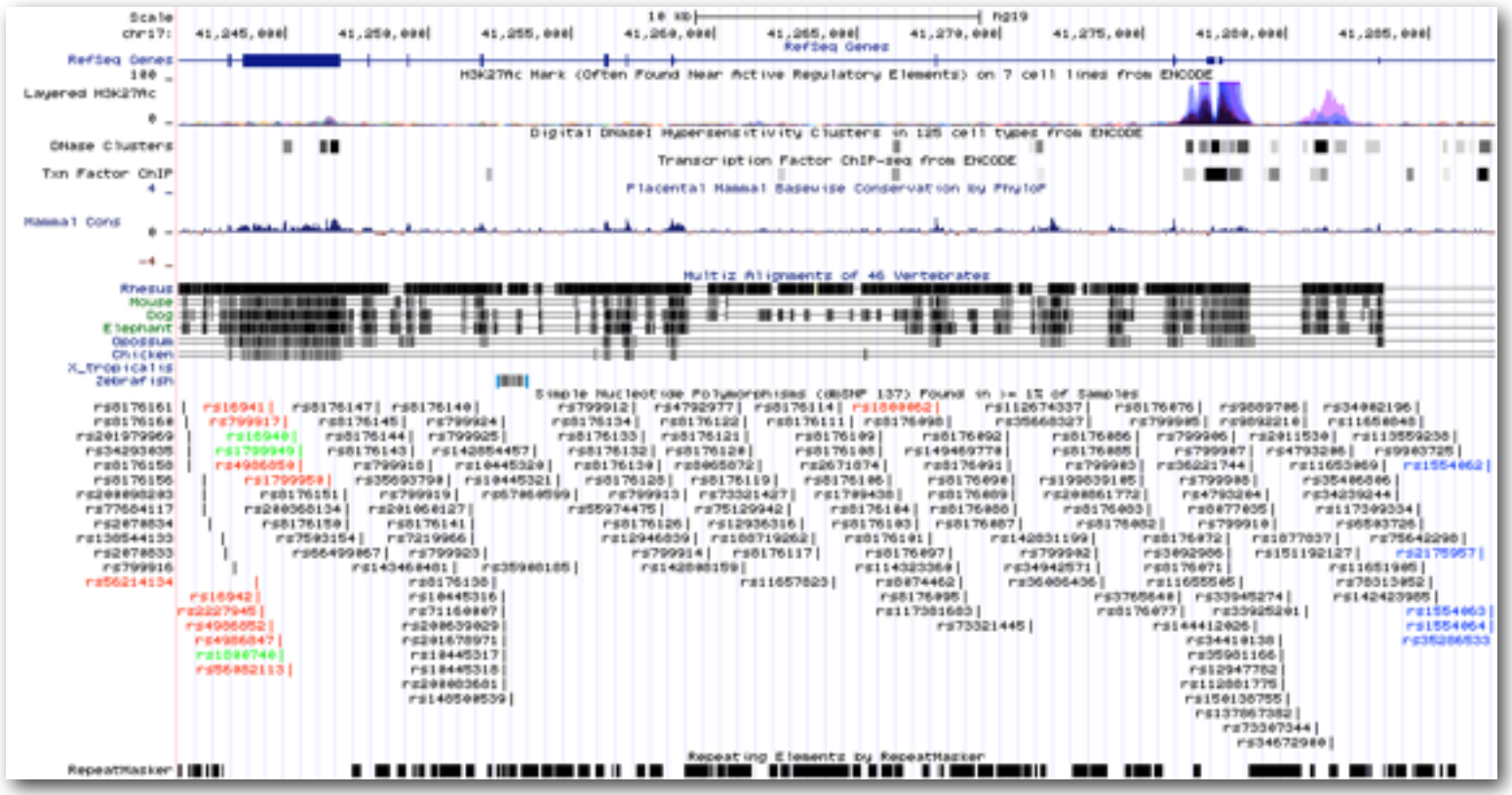
What's different about the DNA in different tissues?

In what abundance do we find various molecules?

What differences exist between individuals?

Picking up signals

Through many experiments, we know much more about the genome than just its DNA sequence:



40 K nt region of chromosome 17

http://genome.ucsc.edu/cgi-bin/hgTracks

Experimentally observed products, e.g. messenger RNAs

Epigenetic marks

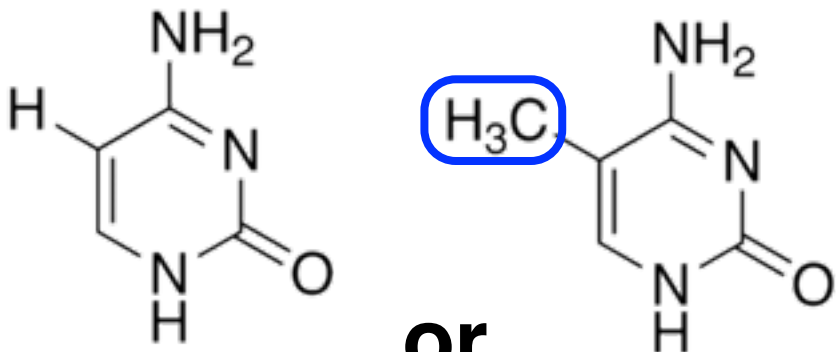
Sequence conservation among related species

Sites that vary across individuals

CpG Islands

A signal we can discern from genome sequence alone: CpG islands

Dinucleotide “CG” (AKA “CpG”) is special because the C can possibly have a *methyl group* attached



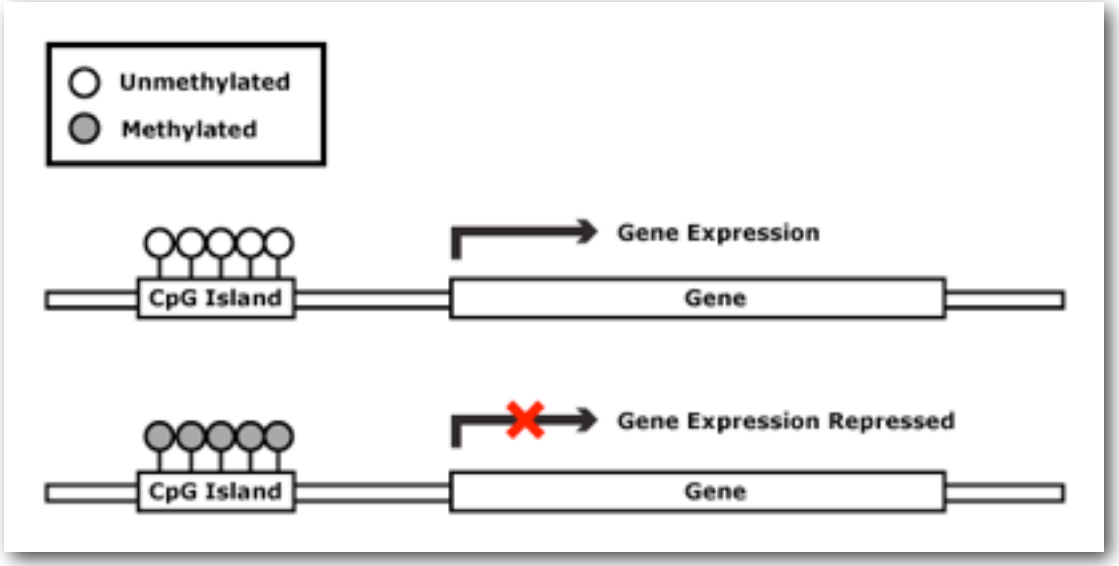
Unmethylated Methylated

Proteins involved in gene expression can be repelled or attracted by the methyl group

CpG Islands

CpG island: part of the genome where CG occurs particularly frequently

CpG islands usually regulate expression of nearby genes

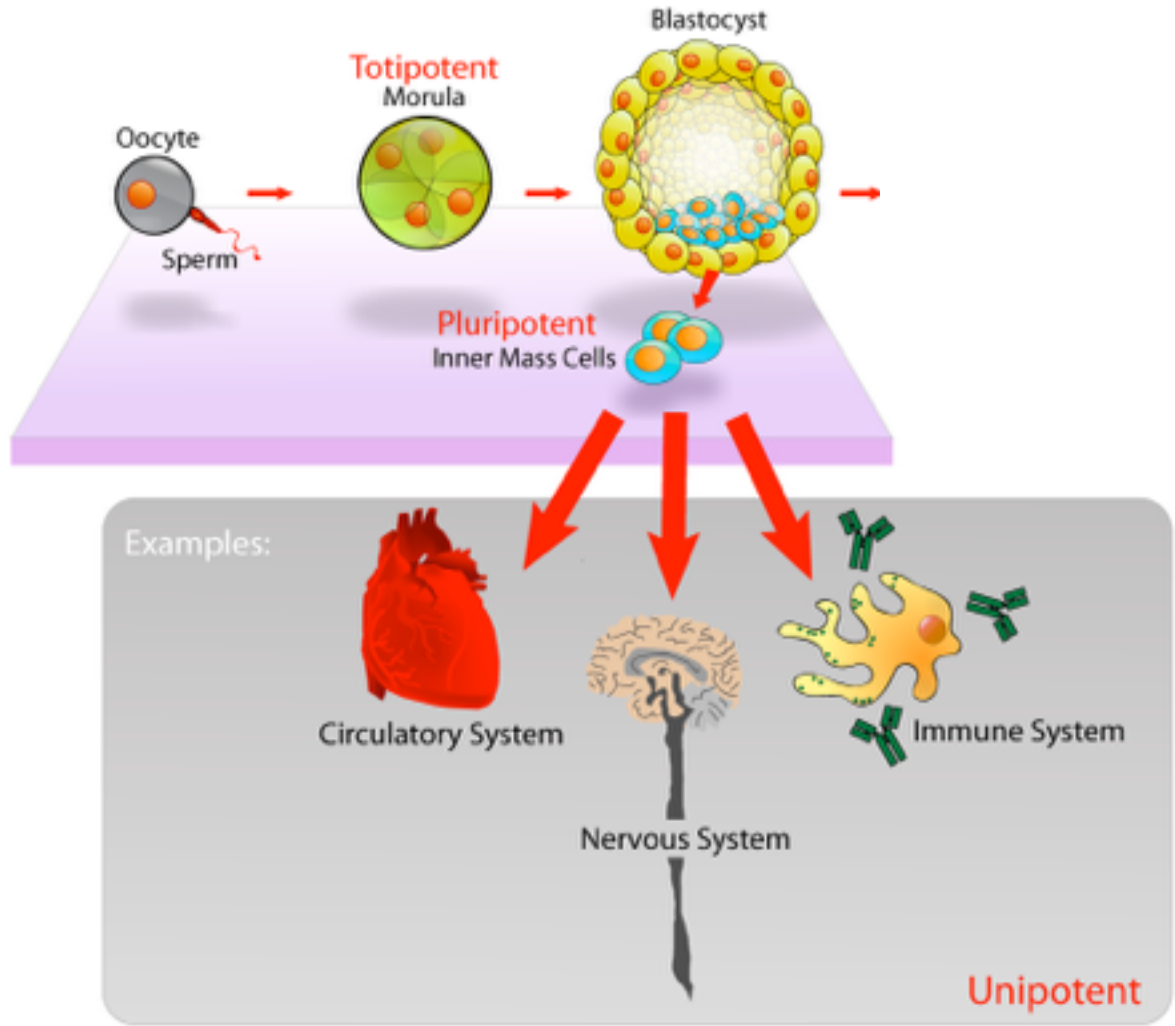


http://missinglink.ucsf.edu/lm/genes_and_genomes/methylation.html

Cells from different tissues have different patterns of CpG methylation, in turn giving them different gene expression profiles

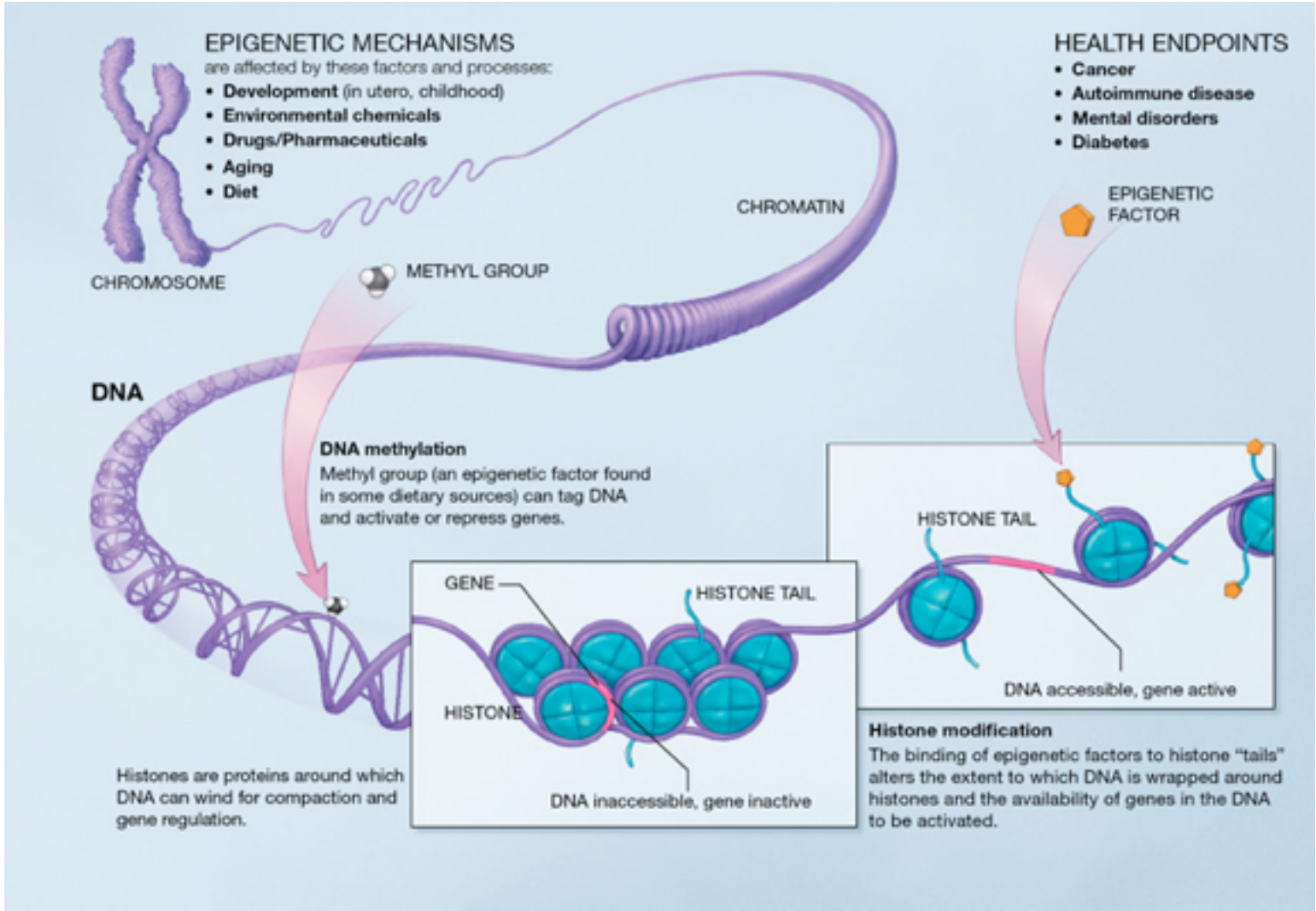
Key *epigenetic* phenomenon

Background: Epigenetics



http://en.wikipedia.org/wiki/File:Stem_cells_diagram.png

Background: Epigenetics



http://upload.wikimedia.org/wikipedia/commons/d/dd/Epigenetic_mechanisms.jpg

Background: Epigenetics

Study of how characteristics are inherited across generations *without* changes to the DNA sequence itself

How does a heart cell know it's a heart cell?

How does a calico cat get its splotches?

Epigenetic changes are important in various diseases: Fragile X, Rett, and Angelman syndromes, cancer

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

CpG Islands

Task: design a method that, given a candidate string, scores it according to how confident we are it came from inside a CpG island

Ideally, scores should be *probabilities*

Scores that aren't probabilities are still useful, mainly for ranking

Probabilities are more interpretable, capturing how likely we are to be right or wrong.

Probability review

Sample space (Ω) is set of all possible outcomes

E.g. Ω = { all possible rolls of 2 dice }

An *event* (*A*, *B*, *C*, ...) is a subset of Ω

A = { rolls where first die is odd }, *B* = { rolls where second die is even }

We're often concerned with assigning a probability to an event

P(*A*): fraction of all possible outcomes that are in *A*

P(*A*) = | *A* | / | Ω | = 18 / 36 = 0.5

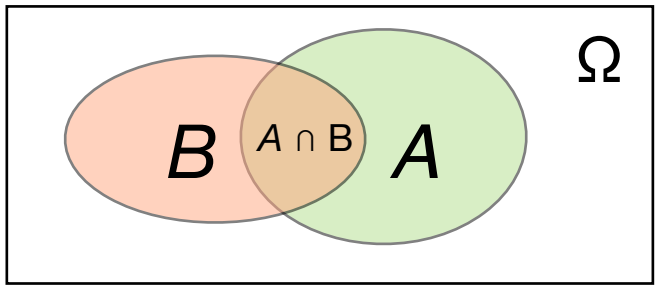
Probability review

P(*A, B*): fraction of all possible outcomes that are in both *A* and *B*

P(*A, B*) = | A ∩ B | / | Ω | = 9 / 36 = 0.25

Sometimes written P(*A* ∩ *B*) or P(*AB*)

Joint probability of *A* and *B*

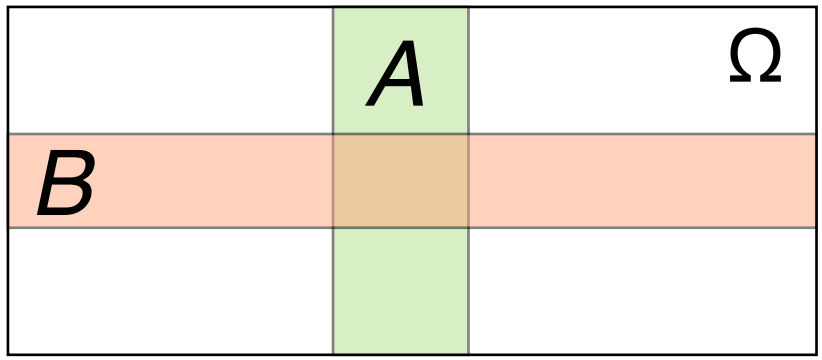


P(*A* | *B*): fraction of outcomes in *B* that are also in *A*

P(*A* | *B*) = | A ∩ B | / | B | = 9 / 18 = 0.5

P(*A* | *B*) can be rewritten P(*A*, *B*) / P(*B*)

Probability review



Events A and B are independent if $P(A \mid B) = P(A)$

So $P(A, B) = P(B)\, P(A \mid B) = P(A)\, P(B)$

Probability review

Random variable is a variable whose possible values are outcomes of a random phenomenon

E.g. random variable X represents the outcome of a flip of a fair coin: $p(X = \text{heads}) = p(X = \text{tails}) = 0.5$.

Sequence models

Sequence model is a *probabilistic model* that associates probabilities with sequences

Useful for weighing the relative likelihood of seeing certain strings under certain circumstances

What *k*-mers am I likely to see inside versus outside of a CpG island?

Given a genome, where are the genes?

What's the probability of next character being A if previous characters were GATTAC?

Right: a model for eukaryotic gene finding

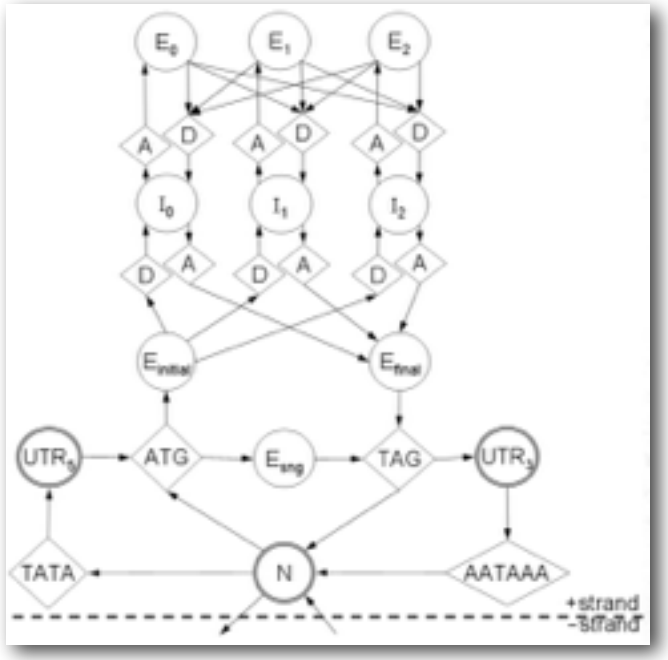


Image: Bill Majoros, http://www.genezilla.org/design.html

Sequence models

Sequence models learn from examples

Say we have sampled 100K 5-mers from inside CpG islands and 100K 5-mers from outside

Can we guess whether CGCGC came from a CpG island?

# CGCGC inside	315
# CGCGC outside	12

p(inside) = 315/(315 + 12)= 0.963

Python example: http://nbviewer.ipython.org/7413873

Sequence models

Let P(x) be the probability of sequence x as assigned by the model

$$P(x) = \underbrace{P(x_k, x_{k-1}, \ldots x_1)}$$

Joint probability of all sequence items appearing as they do

P(x) could be probability that DNA string x is part of a CpG island

To estimate P(x), count # times x appears in the training set labeled *inside* divided by *total #* times x appears in training set

But for sufficiently long k, we might not see *any* occurrences of x, or very few. Joint probabilities for rare events are hard to estimate well.

Sequence models

$$P(x) = P(x_k, x_{k-1}, \ldots x_1)$$

Re-write with conditional probability:

$$= P(x_k \mid x_{k-1}, \ldots x_1)\, P(x_{k-1}, \ldots x_1)$$

$$= P(x_k \mid x_{k-1}, \ldots x_1)\, P(x_{k-1} \mid x_{k-2}, \ldots x_1)\, P(x_{k-2}, \ldots x_1)$$

(etc)

Add a simplifying assumption: to know the probability of having a particular item x_k, *we only have to know the previous item*: x_{k-1}

Formally: random variable x_k is *conditionally independent* of $x_1 \ldots x_{k-2}$ given x_{k-1}

Informally: ”the future is independent of the past given the present”

Sequence models

Add a simplifying assumption: to know the probability of having a particular item x_k, *we only have to know the previous item*: x_{k-1}

$$P(x) = P(x_k, x_{k-1}, \ldots x_1)$$

$$= P(x_k \mid x_{k-1}, \ldots x_1)\, P(x_{k-1}, \ldots x_1)$$

$$= P(x_k \mid x_{k-1}, \ldots x_1)\, P(x_{k-1} \mid x_{k-2}, \ldots x_1)\, P(x_{k-2}, \ldots x_1)$$

(etc)

drop drop (bunch more drops once this is expanded)

$$\approx P(x_k \mid x_{k-1})\, P(x_{k-1} \mid x_{k-2}) \ldots P(x_2 \mid x_1)\, P(x_1)$$

Markov property / Markov assumption

It's a big assumption, but it's often reasonable and it makes the model much easier to work with

Markov chain

Assigning a probability to a sequence using Markov property:

$$P(x) \underset{\text{Markov property}}{\approx} P(x_k \mid x_{k-1})\, P(x_{k-1} \mid x_{k-2}) \ldots P(x_2 \mid x_1)\, P(x_1)$$

Say x is a nucleotide k-mer

$P(x_i \mid x_{i-1})$ probability of seeing nucleotide x_i in i^{th} position given that previous nucleotide is x_{i-1}

Shorthand: P(G I C) = probability of G given previous is C

Markov chain

Say someone gives us the sequences of several CpG islands. How do we estimate, say, P(G | C)?

P(G | C) = # times CG occurs / # times C occurs

Markov chain

Given CpG island sequences from human chromosome 1, count nucleotide and dinucleotide occurrences and estimate all 16 possible $P(x_i \mid x_{i-1})$:

$P(A \mid A)$ = # times AA occurs / # times A occurs

$P(C \mid A)$ = # times AC occurs / # times A occurs

$P(G \mid A)$ = # times AG occurs / # times A occurs

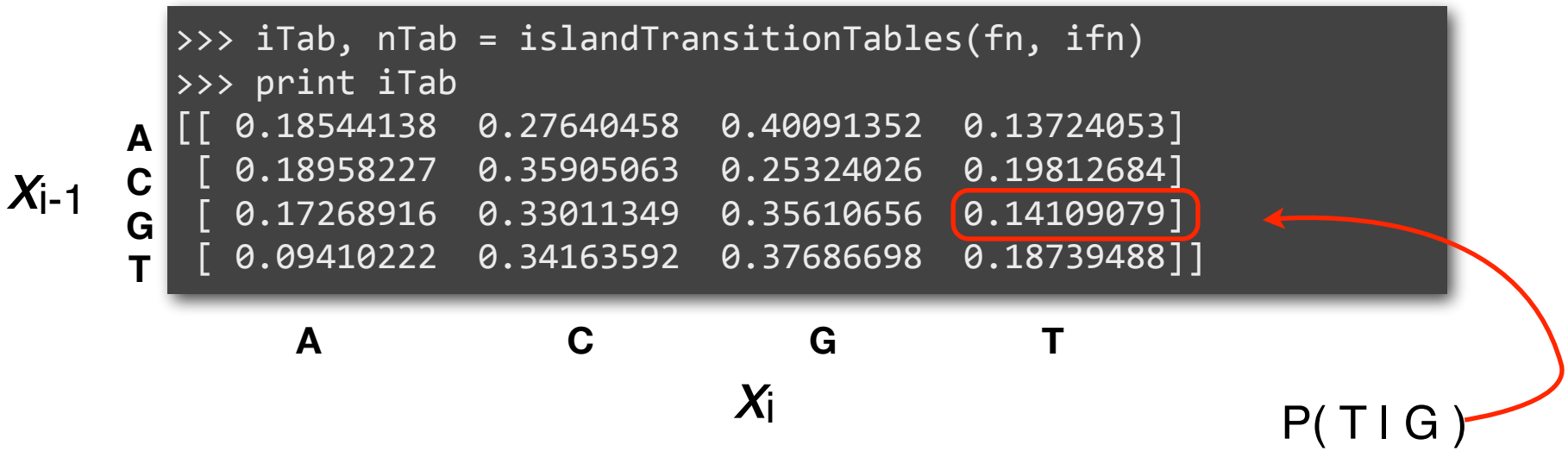
$P(T \mid A)$ = # times AT occurs / # times A occurs

$P(A \mid C)$ = # times CA occurs / # times C occurs

(etc)

Markov chain

Given CpG island sequences from human chromosome 1, count nucleotide and dinucleotide occurrences and estimate all 16 possible $P(x_i \mid x_{i-1})$:



Rows sum to 1

Python example: http://nbviewer.ipython.org/7413873

Markov chain

We can do the same for dinucleotides *outside* of CpG islands

```
>>> iTab, nTab = islandTransitionTables(fn, ifn)
>>> print iTab
[[ 0.18544138  0.27640458  0.40091352  0.13724053]
 [ 0.18958227  0.35905063  0.25324026  0.19812684]
 [ 0.17268916  0.33011349  0.35610656  0.14109079]
 [ 0.09410222  0.34163592  0.37686698  0.18739488]]
>>> print nTab
[[ 0.2948135   0.19467897  0.28696205  0.22354548]
 [ 0.32681187  0.29415529  0.06172587  0.31730697]
 [ 0.25713351  0.23354071  0.29423494  0.21509084]
 [ 0.17956538  0.23250026  0.29462341  0.29331096]]
```

Inside: A C G T

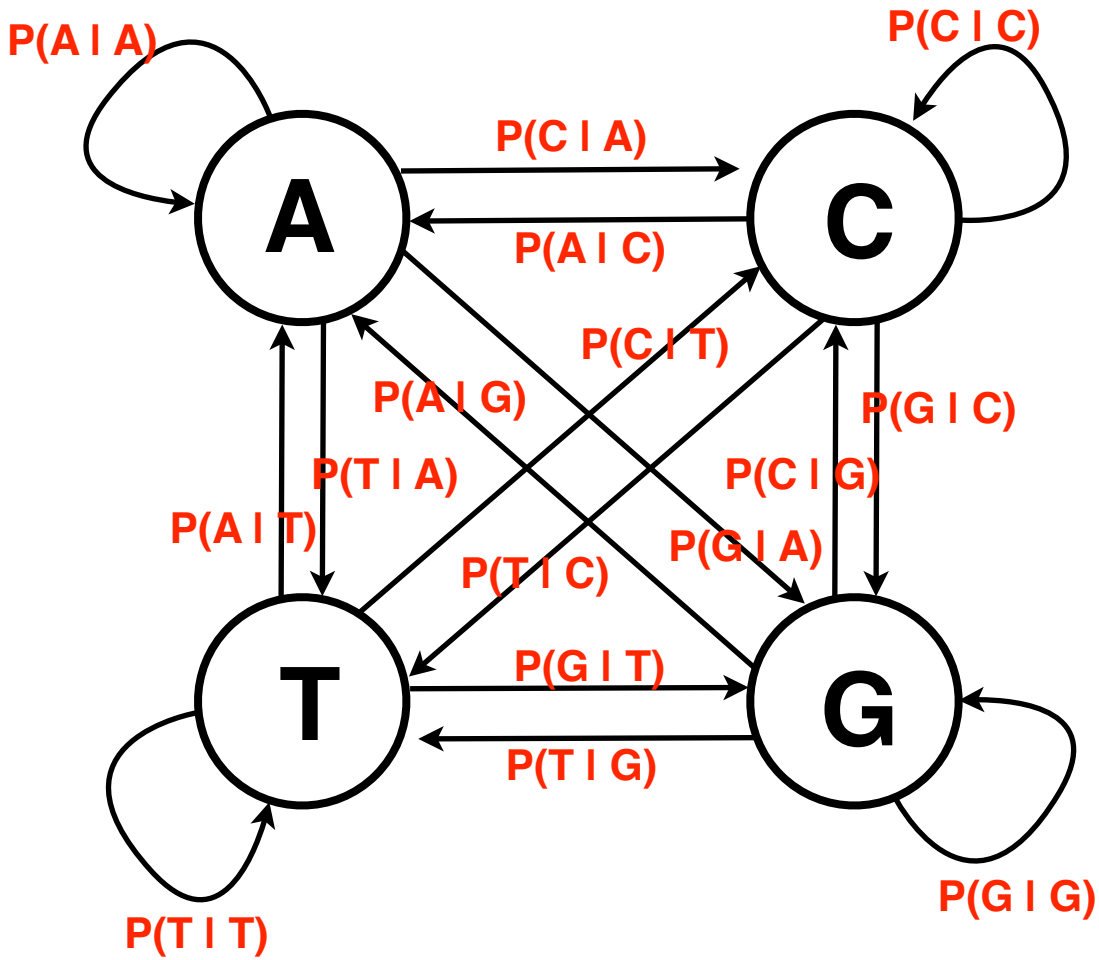
Outside: A C G T

A C G T

Notice anything interesting about the outside conditional probabilities?

P(G | C) is low, matching our expectation that there are few CpGs outside islands

Markov chain



Markov chain is a probabilistic automaton

Each edge has a *transition probability*: probability that edge's destination is the next node visited after edge's source

Here, nodes labels are symbols and transition labels are conditional probabilities

Markov chain

Recall how we assign a probability to a single string

$$P(x) \underset{\text{Markov property}}{\approx} P(x_k \mid x_{k-1})\, P(x_{k-1} \mid x_{k-2}) \ldots P(x_2 \mid x_1)\, P(x_1)$$

For simplicity, drop $P(x_1)$

$$P(x_k \mid x_{k-1})\, P(x_{k-1} \mid x_{k-2}) \ldots P(x_2 \mid x_1)\, \cancel{P(x_1)}$$

$$P(x) \approx P(x_k \mid x_{k-1})\, P(x_{k-1} \mid x_{k-2}) \ldots P(x_2 \mid x_1)$$

$P(x)$ now equals product of all the Markov chain edge weights on our string-driven walk through the chain

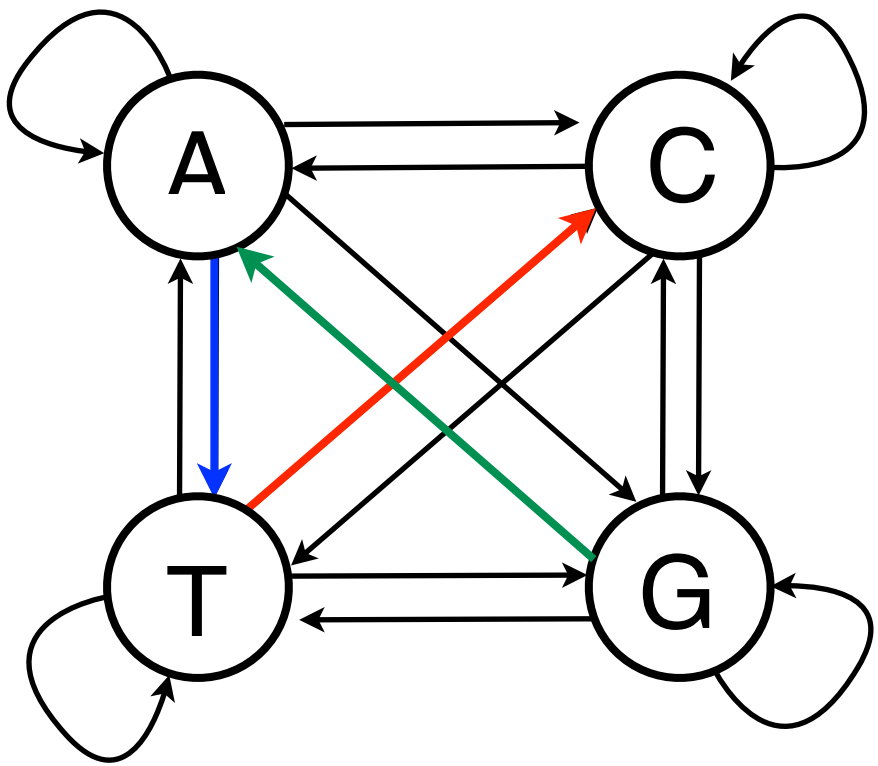
Markov chain

```
>>> iTab, nTab = islandTransitionTables(fn, ifn)
>>> print iTab
[[ 0.18544138  0.27640458  0.40091352  0.13724053]
 [ 0.18958227  0.35905063  0.25324026  0.19812684]
 [ 0.17268916  0.33011349  0.35610656  0.14109079]
 [ 0.09410222  0.34163592  0.37686698  0.18739488]]
```

X_{i-1} A C G T

A C G T

X_i



x = GATC

$P(x) = P(x_4 \mid x_3)\, P(x_3 \mid x_2)\, P(x_2 \mid x_1)$

$P(x) = P(C \mid T)\, P(T \mid A)\, P(A \mid G)$

= 0.34163592 *
0.13724053 *
0.17268916

= 0.00809675

Markov chain

To avoid repeated multiplies yielding small numbers, we switch to log domain

$$\log P(x) \approx \log [\, P(x_k \mid x_{k-1})\, P(x_{k-1} \mid x_{k-2}) \ldots P(x_2 \mid x_1)\,]$$

$$= \log P(x_k \mid x_{k-1}) + \log P(x_{k-1} \mid x_{k-2}) + \ldots + \log P(x_2 \mid x_1)$$

$$= \sum_{i=2}^{k} \log P(x_i \mid x_{i-1})$$

Switching to logs, multiplies become adds

We'll use base-2 logs

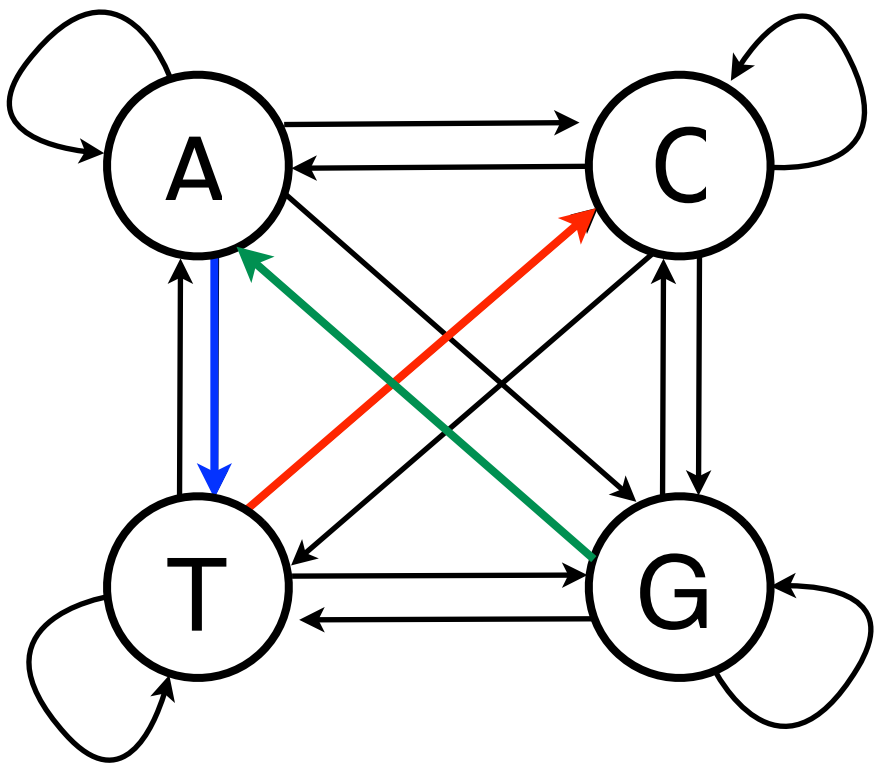
Markov chain

```
>>> iTab, nTab = islandTransitionTables(fn, ifn)
>>> print numpy.log2(iTab)
[[-2.43096492 -1.85514658 -1.31863704 -2.86522151]
 [-2.39910406 -1.4777408  -1.98142131 -2.33550376]
 [-2.53375061 -1.59896599 -1.48961909 -2.82530423]
 [-3.40962748 -1.54946844 -1.40787269 -2.41584653]]
```

X_{i-1} A C G T

A C G T

X_i



x = GATC

$$\log P(x) = \sum_{i=2}^{4} \log P(x_i \mid x_{i-1})$$

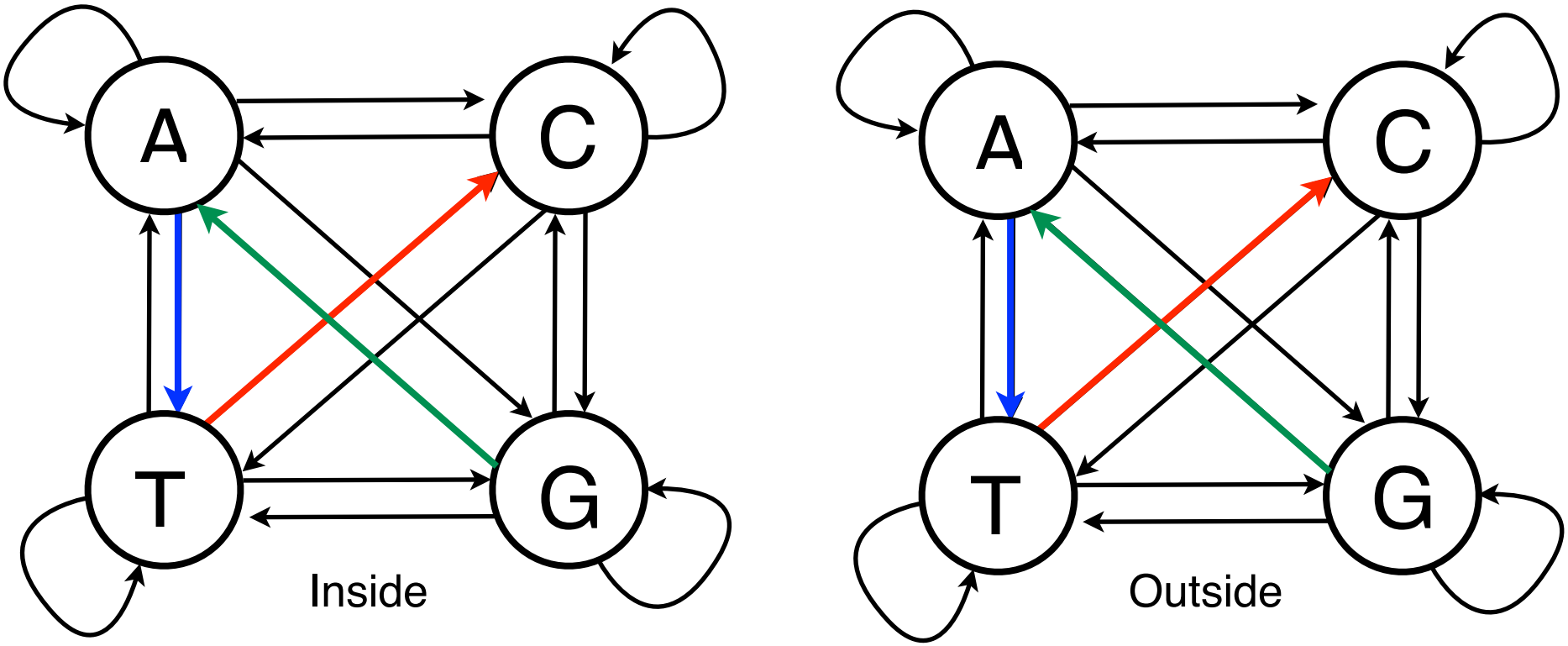
$$= -1.54946844 + -2.86522151 + -2.53375061$$

$$= -7.30174249$$

Markov chain

P(x) given the inside-CpG model is helpful, but we really want to know which model is better, inside CpG or outside CpG?

Use *ratio:* $\frac{\text{P}(x) \text{ from inside model}}{\text{P}(x) \text{ from outside model}}$



Markov chain

Taking log, we get *log ratio*: $S(x) = \log \frac{P(x) \text{ inside CpG}}{P(x) \text{ outside CpG}}$

If inside more probable than outside, fraction is > 1 and log ratio is > 0. Otherwise, fraction is ≤ 1 and log ratio is ≤ 0.

$$S(x) = \log \frac{P(x) \text{ inside CpG}}{P(x) \text{ outside CpG}}$$

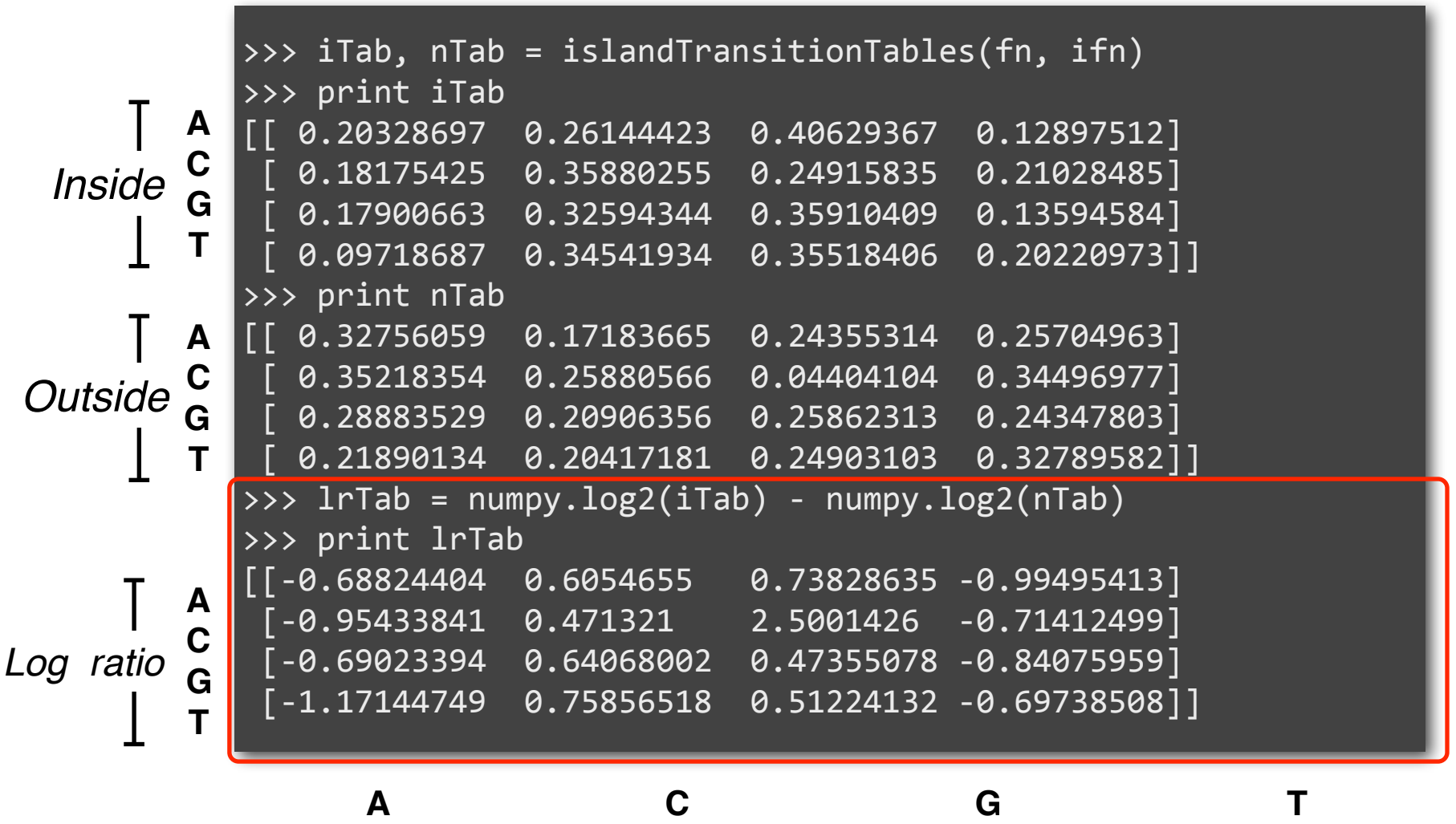
$$= \log [\, P(x) \text{ inside CpG} \,] - \log [\, P(x) \text{ outside CpG} \,]$$

$$= \sum_{i=2}^{k} (\log [\, P(x_i \mid x_{i-1}) \text{ inside CpG} \,]) - \sum_{i=2}^{k} \log ([\, P(x_i \mid x_{i-1}) \text{ outside CpG} \,])$$

$$= \sum_{i=2}^{k} \left(\log \left[P(x_i \mid x_{i-1}) \text{ inside CpG} \right] - \log \left[P(x_i \mid x_{i-1}) \text{ outside CpG} \right] \right)$$

New table: take elementwise log of the inside/outside tables, subtract outside from inside

Markov chain



Markov chain

Now, given a string x, we can easiy assign it a log ratio "score" $S(x)$:

$$S(x) = \log \frac{\text{P}(x) \text{ inside CpG}}{\text{P}(x) \text{ outside CpG}}$$

$$\approx \sum_{i=2}^{k} \left(\log \left[\text{P}(x_i \mid x_{i-1}) \text{ inside CpG} \right] - \log \left[\text{P}(x_i \mid x_{i-1}) \text{ outside CpG} \right] \right)$$

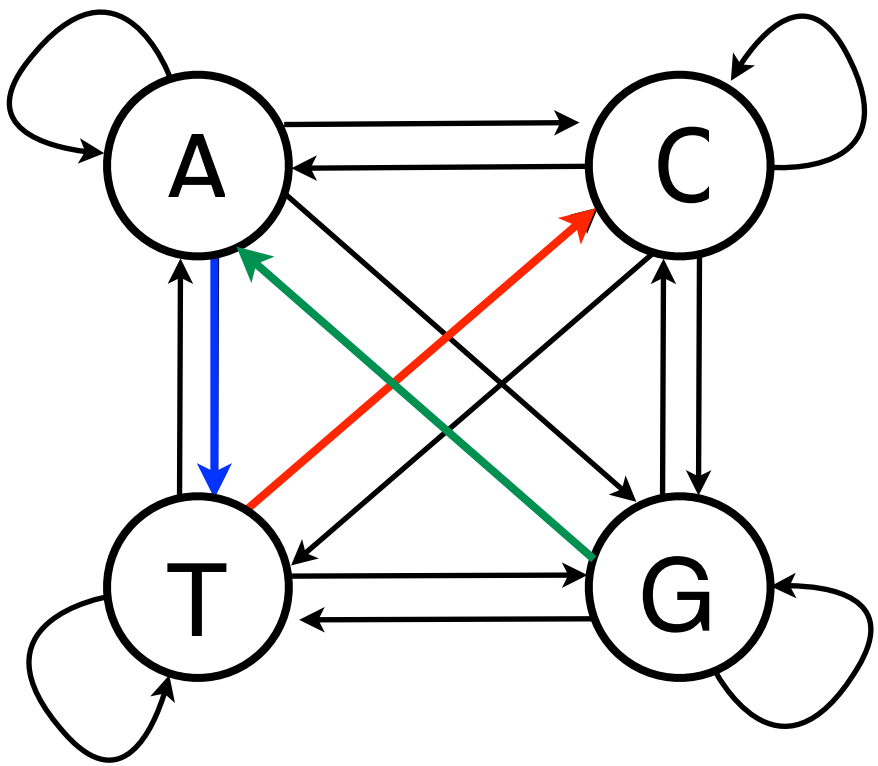
Markov chain

```
>>> iTab, nTab = islandTransitionTables(fn, ifn)
>>> lrTab = numpy.log2(iTab) - numpy.log2(nTab)
>>> print lrTab
[[-0.66883939  0.50568449  0.48243108 -0.70386181]
 [-0.78563635  0.28760934  2.03655959 -0.67945489]
 [-0.57434013  0.49928806  0.27534041 -0.60832223]
 [-0.9322086   0.55522735  0.35518335 -0.6463494 ]]
```

X_{i-1} A C G T

A C G T

X_i



x = GATC

S(x) = 0.55522735 +
-0.70386181 +
-0.57434013

= -0.72297459

Negative, so probability with *outside* model is greater

Markov chain

$$S(x) = \log \frac{P(x) \text{ inside CpG}}{P(x) \text{ outside CpG}}$$

S(CGCGCGCGCGCGCGCGCGCGCGCG) = 32.246609048

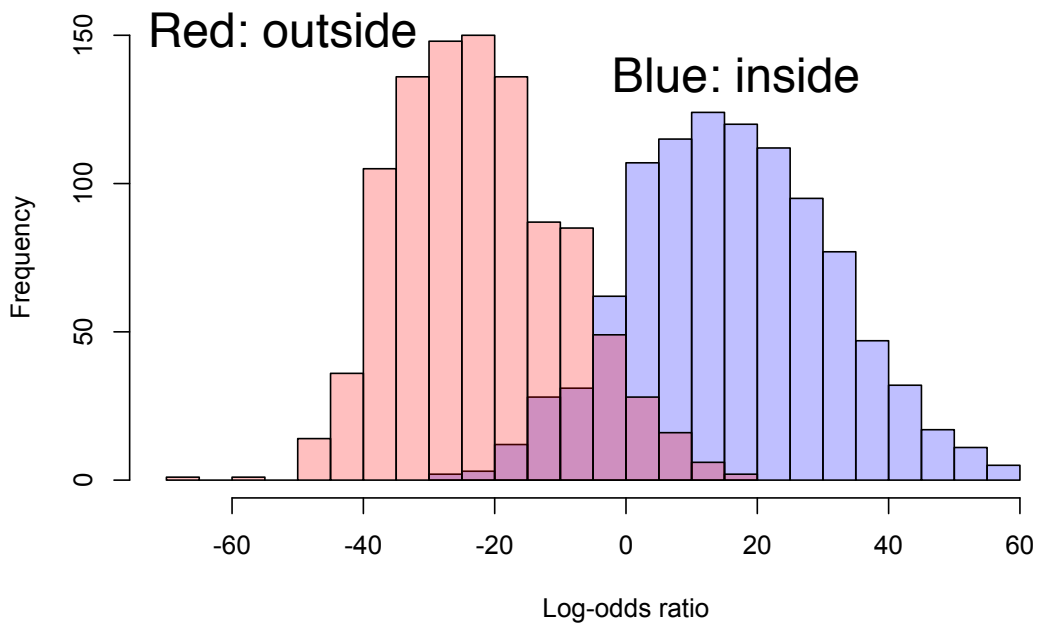
S(ATTCTACTATCATCTATCTATCTTCT) = -9.501209765

Python example: http://nbviewer.ipython.org/7413873

Markov chain: experiment

Drew 1,000 100-mers from inside CpG islands on chromosome 1, and another 1,000 from outside, and calculated log ratios for all

Trained markov chain on dinucleotides from chromosome 22



Python example: http://nbviewer.ipython.org/7413873

Markov chain

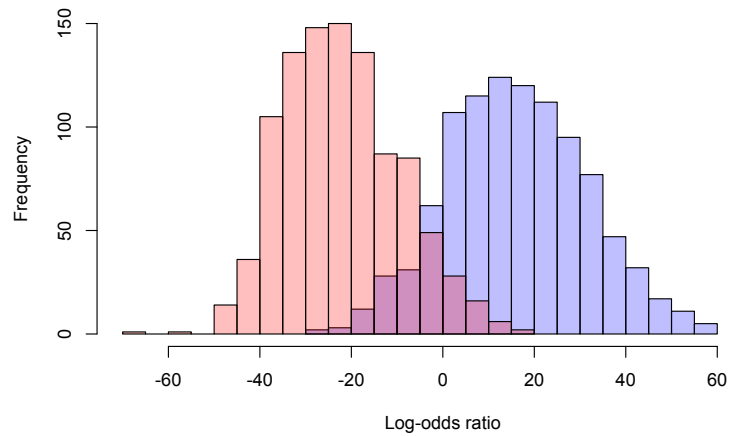
Markov property made our problem very tractable

P(x_i | x_{i-1}) estimated in single, simple pass through training data

Transition probability tables have $|\sum|^2$ cells; fine for DNA & protein

Calculating S(x) is O($|x|$); just lookups and additions

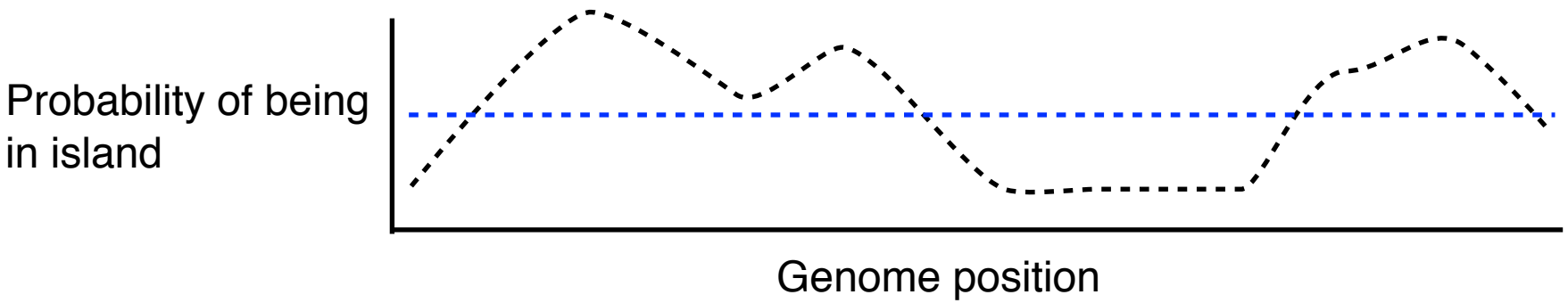
... and discriminates well between inside & outside examples in CpG island example



Sequence models

Can we use Markov chains to pick out CpG islands from the rest of the genome?

Markov chain assigns a score to a string; doesn't naturally give a "running" score across a long sequence

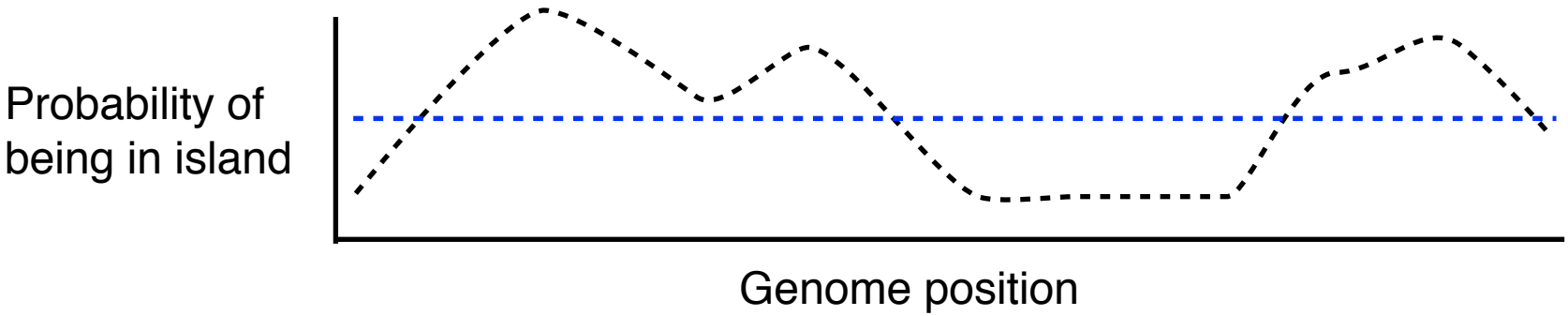


We could use a *sliding window*

(a) Pick window size *w*, (b) score every *w*-mer using Markov chains, (c) use a cutoff to find islands

Smoothing before (c) might also be a good idea

Sequence models



Choosing *w* involves an assumption about how long the islands are

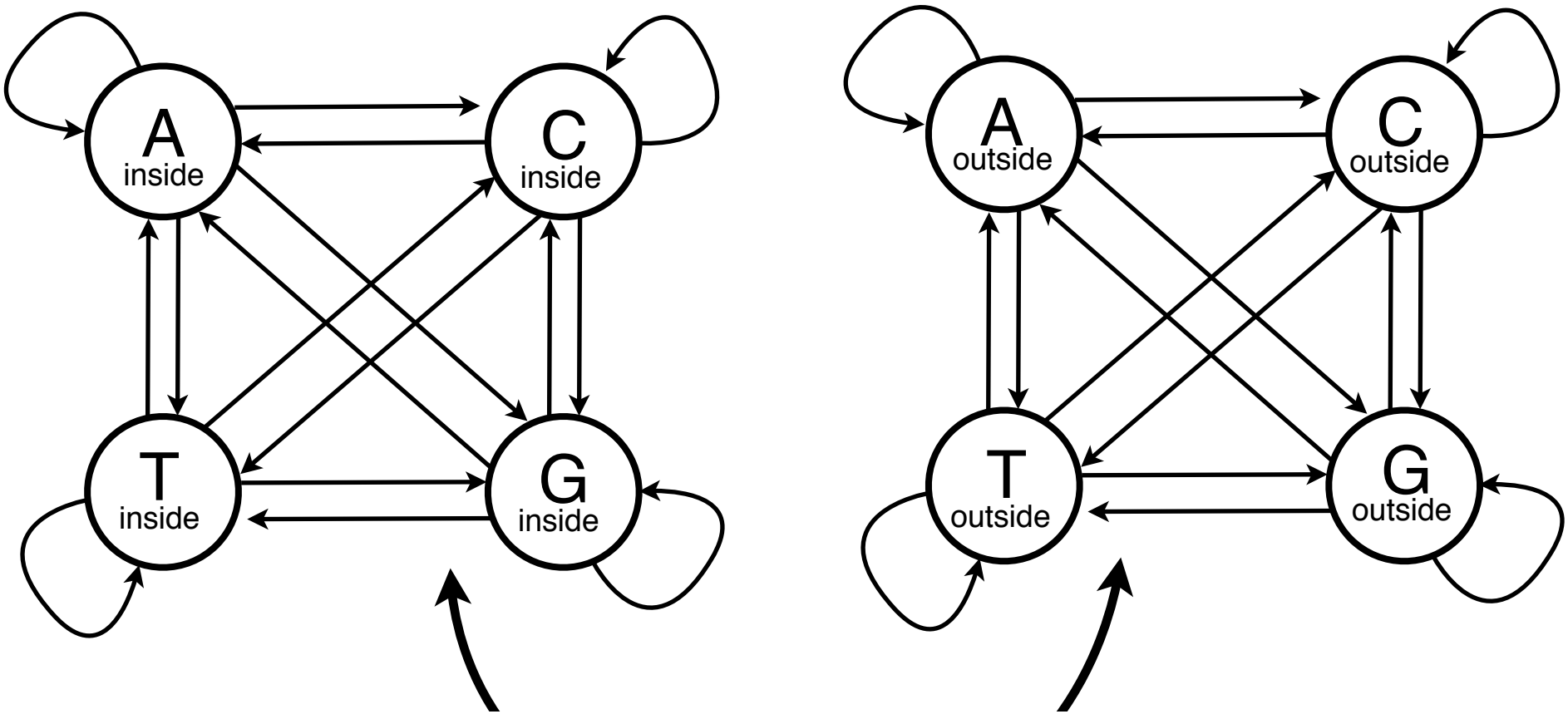
If *w* is too large, we'll miss small islands

If *w* is too small, we'll get many small islands where perhaps we should see fewer larger ones

In a sense, we want to switch *between Markov chains* when entering or exiting a CpG island

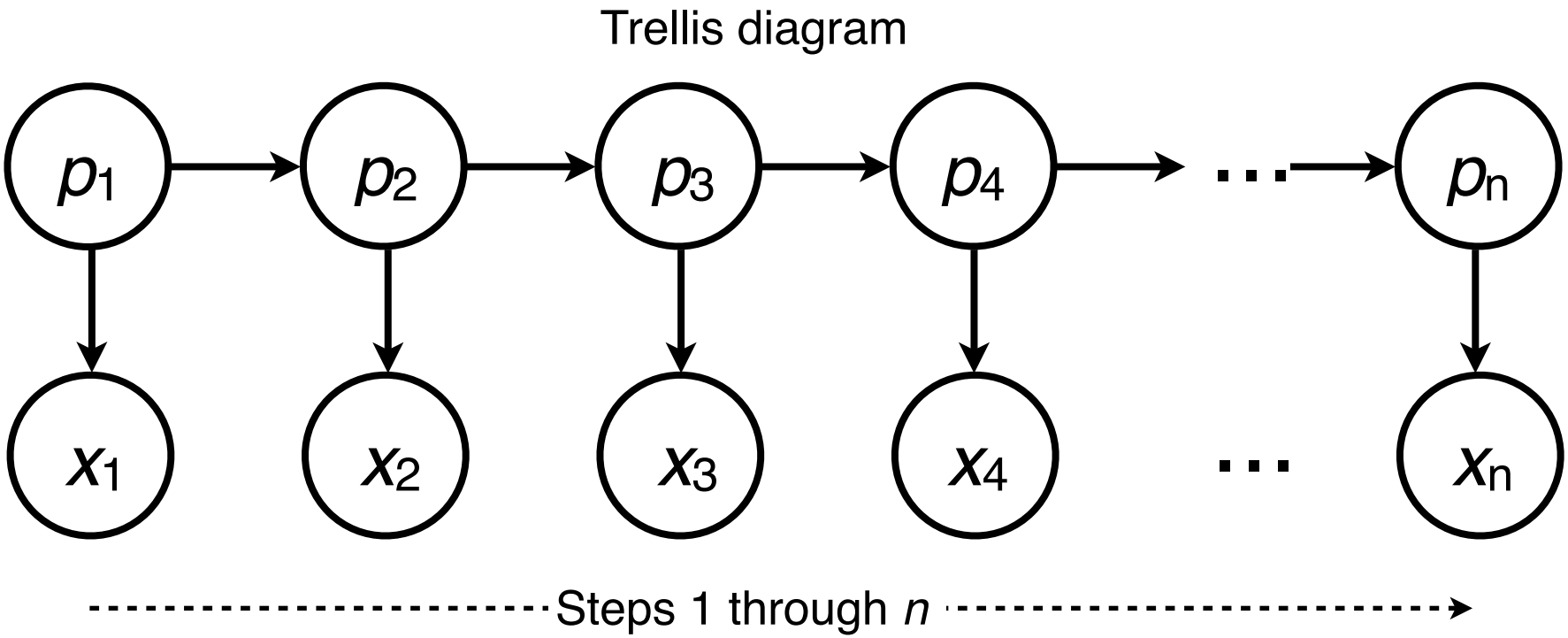
Sequence models

Something like this:



All inside-outside edges

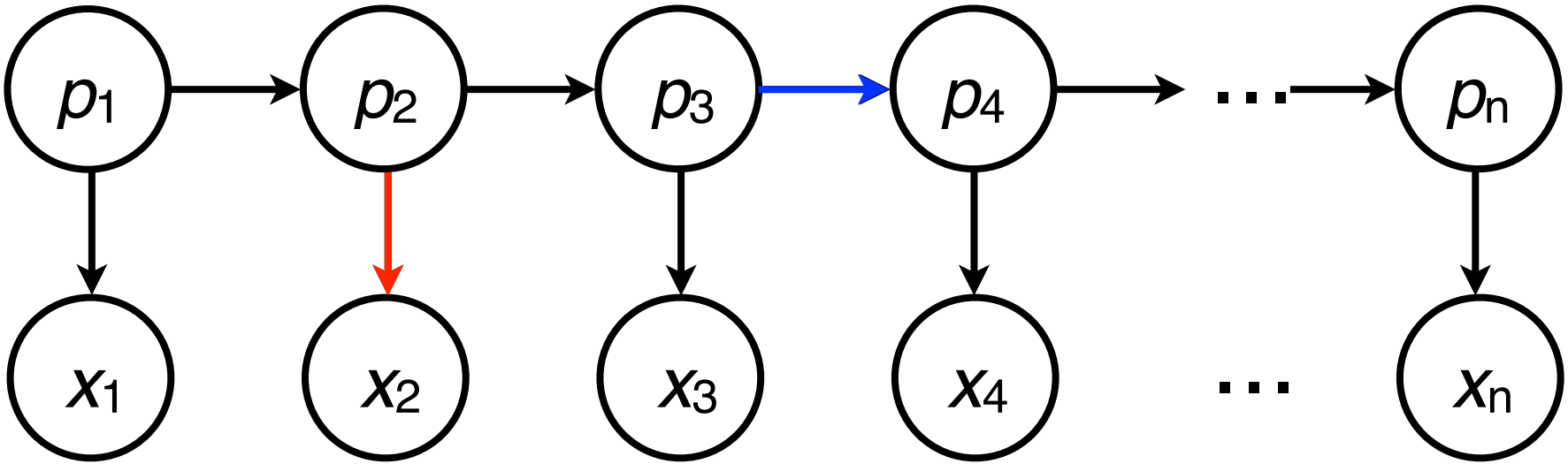
Hidden Markov Model



$p = \{ p_1, p_2, ..., p_n \}$ is a sequence of *states* (AKA a *path*). Each p_i takes a value from set Q. We ***do not*** observe p.

$x = \{ x_1, x_2, ..., x_n \}$ is a sequence of *emissions.* Each x_i takes a value from set $\sum$. We ***do*** observe x.

Hidden Markov Model



Like for Markov chains, edges capture conditional independence:

x_2 is conditionally independent of everything else given p_2

p_4 is conditionally independent of everything else given p_3

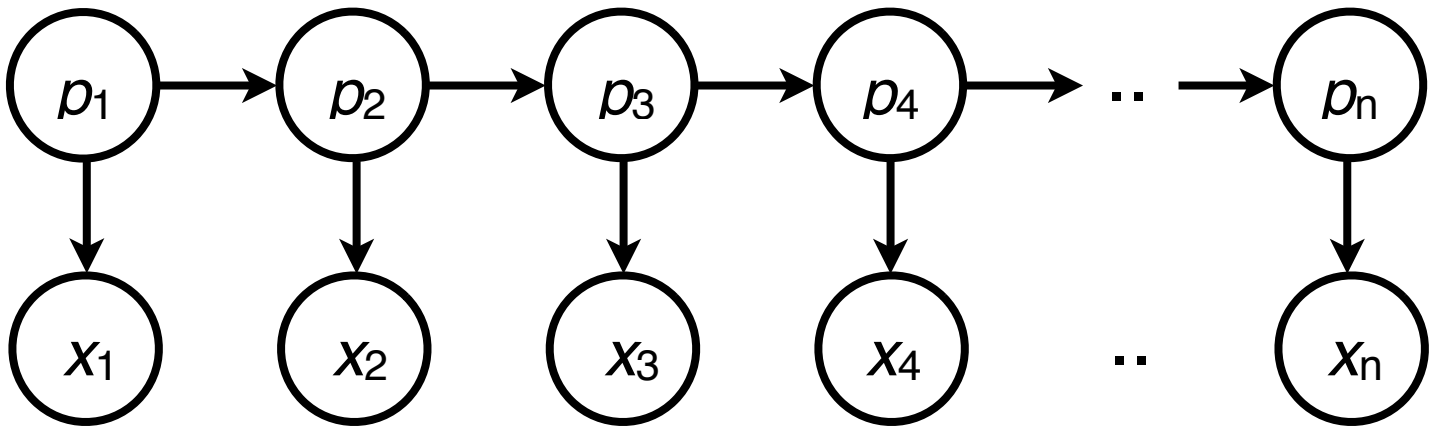
Probability of being in a particular state at step *i* is known once we know what state we were in at step *i*-1. Probability of seeing a particular emission at step *i* is known once we know what state we were in at step *i*.

Hidden Markov Model

Example: occasionally dishonest casino

Dealer repeatedly flips a coin. Sometimes the coin is *fair*, with P(heads) = 0.5, sometimes it's *loaded*, with P(heads) = 0.8. Dealer occasionally switches coins, invisibly to you.

How does this map to an HMM?

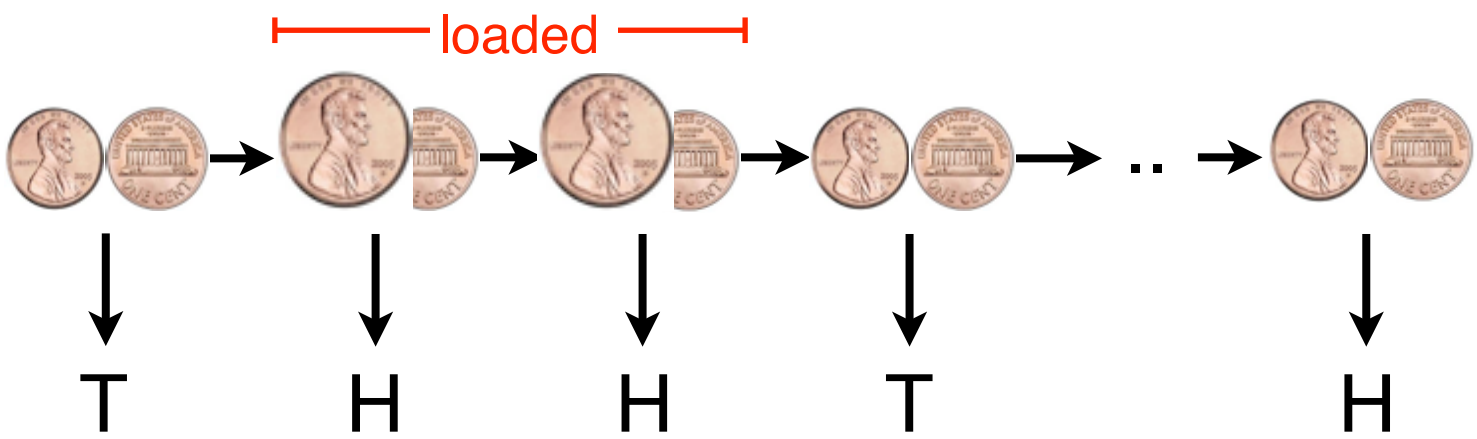


Hidden Markov Model

Example: occasionally dishonest casino

Dealer repeatedly flips a coin. Sometimes the coin is *fair*, with P(heads) = 0.5, sometimes it's *loaded*, with P(heads) = 0.8. Dealer occasionally switches coins, invisibly to you.

How does this map to an HMM?



Emissions encode flip outcomes (observed), states encode loadedness (hidden)

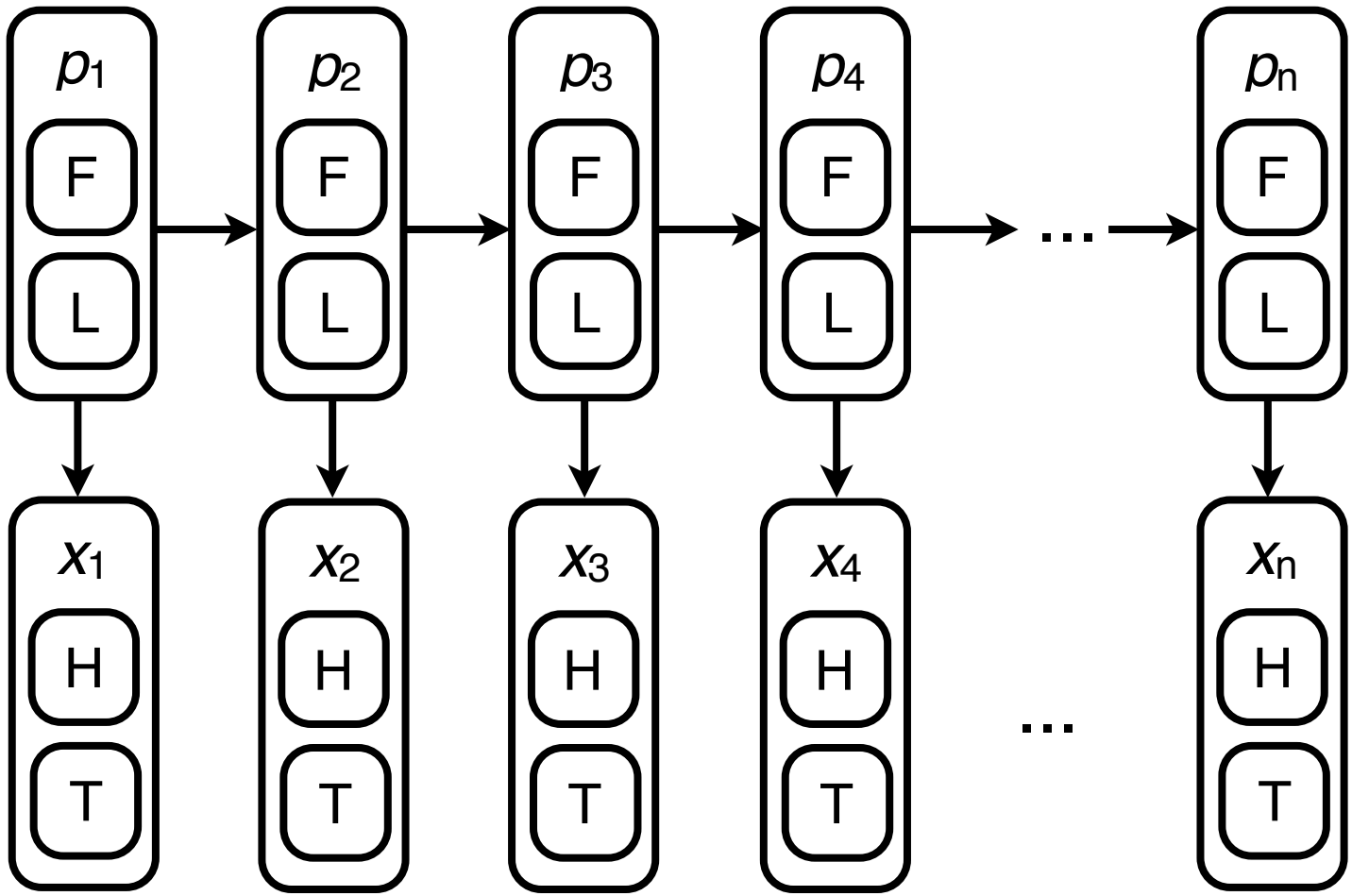
Hidden Markov Model

States encode which coin is used

F = fair
L = loaded

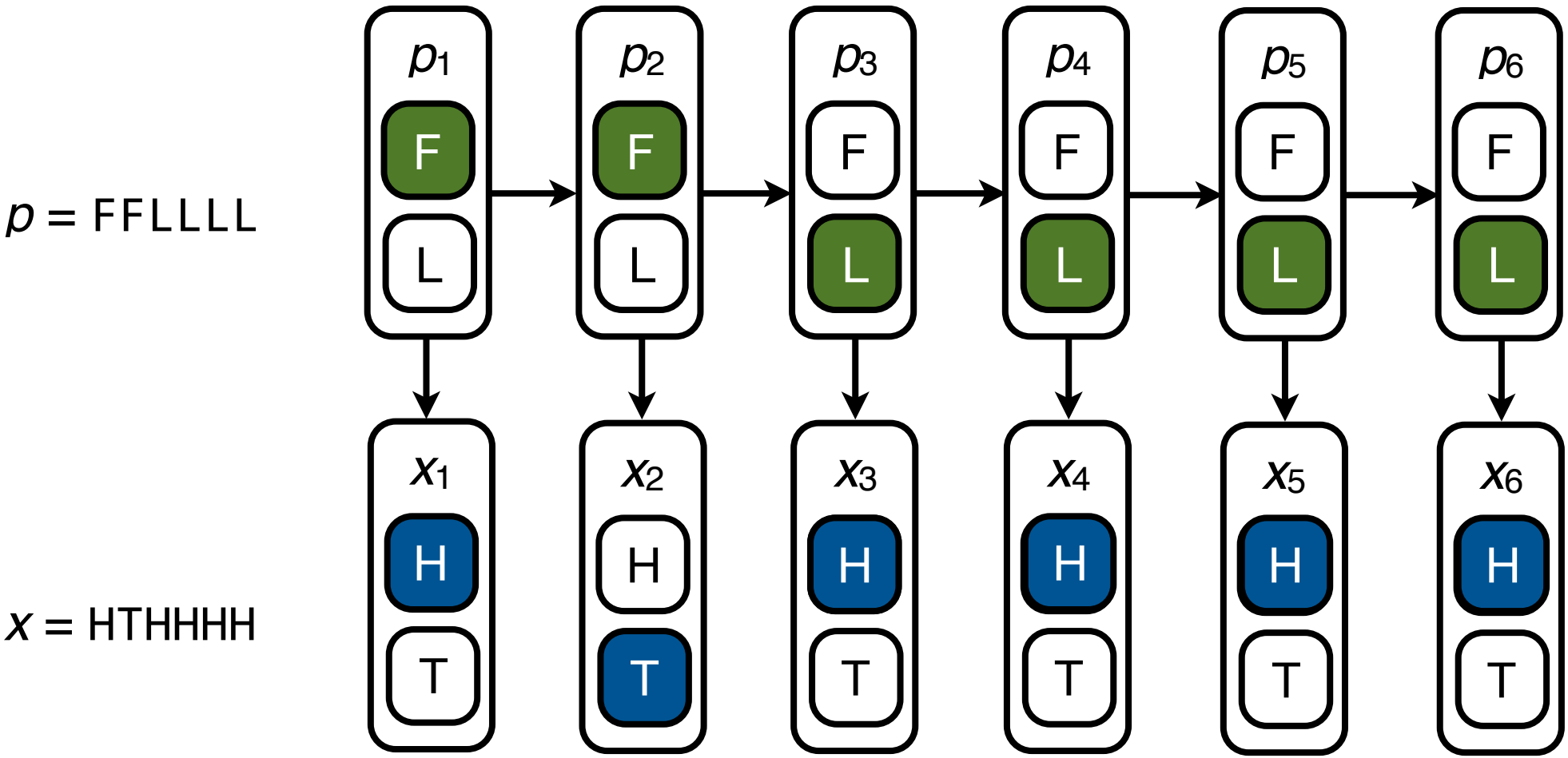
Emissions encode flip outcomes

H = heads
T = tails

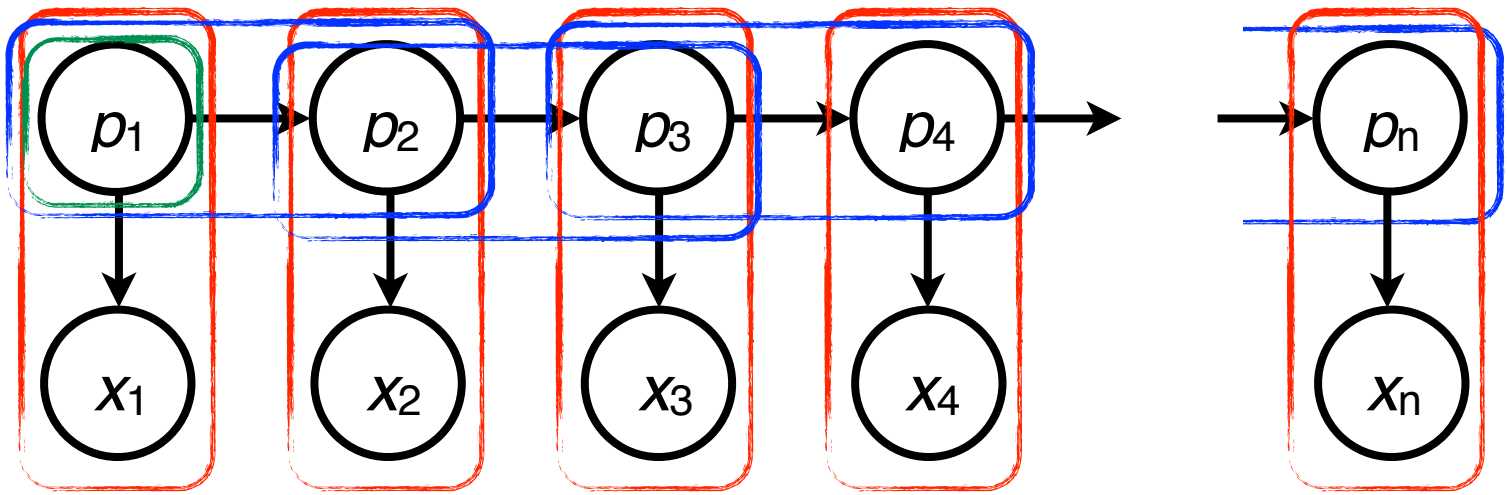


Hidden Markov Model

Example with six coin flips:



Hidden Markov Model



$$P(p_1, p_2, \ldots, p_n, x_1, x_2, \ldots, x_n) = \prod_{k=1}^{n} P(x_k \mid p_k) \prod_{k=2}^{n} P(p_k \mid p_{k-1}) P(p_1)$$

| Q | × | ∑ | emission matrix E encodes $P(x_i \mid p_i)$s — $E[p_i, x_i] = P(x_i \mid p_i)$

| Q | × | Q | transition matrix A encodes $P(p_i \mid p_{i-1})$s — $A[p_{i-1}, p_i] = P(p_i \mid p_{i-1})$

| Q | array I encodes initial probabilities of each state — $I[p_i] = P(p_1)$

Hidden Markov Model

Dealer repeatedly flips a coin. Coin is sometimes *fair*, with P(heads) = 0.5, sometimes *loaded*, with P(heads) = 0.8. Dealer occasionally switches coins, invisibly to you.

After each flip, dealer switches coins with probability 0.4

A:

	F	L
F	0.6	0.4
L	0.4	0.6

E:

	H	T
F	0.5	0.5
L	0.8	0.2

| Q | × | ∑ | emission matrix *E* encodes P(x_i | p_i)s $E\,[p_i, x_i] = P(x_i \mid p_i)$

| Q | × | Q | transition matrix *A* encodes P(p_i | p_{i-1})s $A\,[p_{i-1}, p_i] = P(p_i \mid p_{i-1})$

Hidden Markov Model

Given *A* & *E* (right), what is the joint probability of ***p*** & ***x***?

A	F	L
F	0.6	0.4
L	0.4	0.6

E	H	T
F	0.5	0.5
L	0.8	0.2

p	**F**	**F**	**F**	**L**	**L**	**L**	**F**	**F**	**F**	**F**	**F**
x	**T**	**H**	**T**	**H**	**H**	**H**	**T**	**H**	**T**	**T**	**H**
P(x_i | p_i)	0.5	0.5	0.5	0.8	0.8	0.8	0.5	0.5	0.5	0.5	0.5
P(p_i | p_{i-1})	-	0.6	0.6	0.4	0.6	0.6	0.4	0.6	0.6	0.6	0.6

If $P(p_1 = F) = 0.5$, then joint probability = $0.5^9\ 0.8^3\ 0.6^8\ 0.4^2 = 0.0000026874$

Hidden Markov Model

Given flips, can we say when the dealer was using the loaded coin?

We want to find p^*, the most likely path given the emissions.

$$p^* = \underset{p}{\text{argmax}}\ P(\ p \mid x\ ) = \underset{p}{\text{argmax}}\ P(\ p, x\ )$$

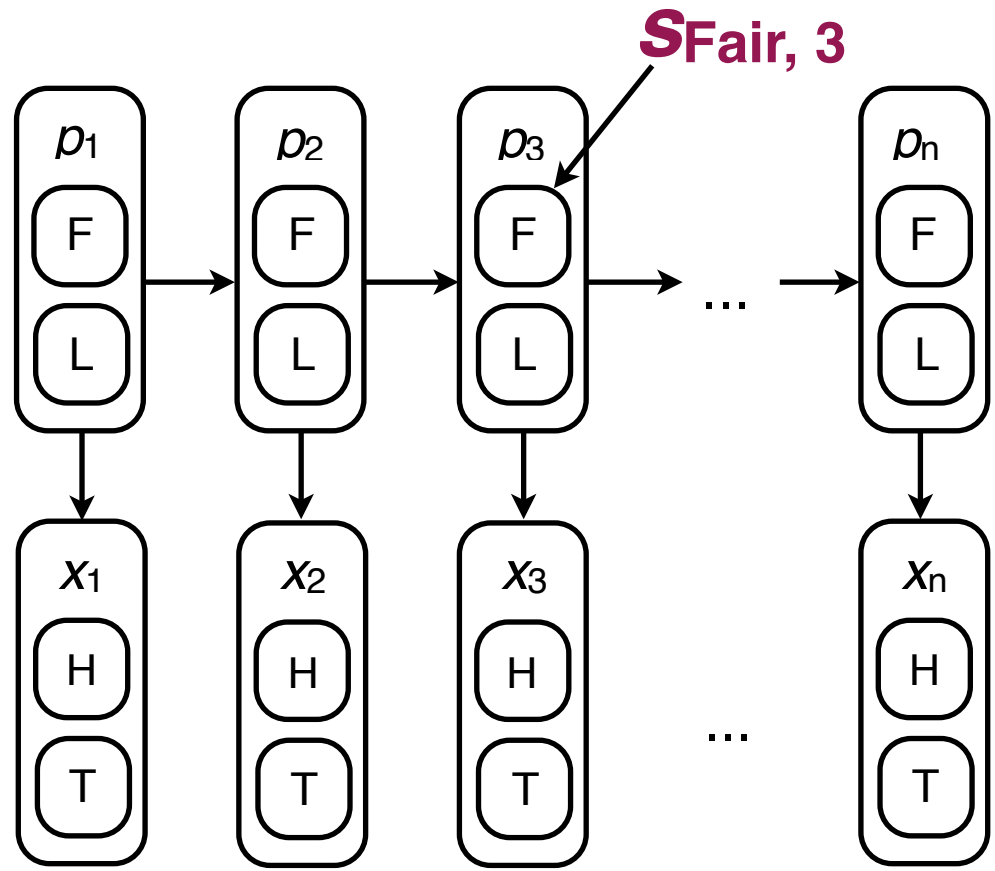
This is *decoding*. *Viterbi* is a common decoding algorithm.

Hidden Markov Model: Viterbi algorithm

Bottom-up dynamic programming

$S_{k,i}$ = score of the most likely path up to step i with $p_i = k$

Start at step 1, calculate successively longer $S_{k,i}$'s



Hidden Markov Model: Viterbi algorithm

Given transition matrix *A* and emission matrix *E* (right), what is the most probable path *p* for the following *x*?

Initial probabilities of F/L are 0.5

A	F	L
F	0.6	0.4
L	0.4	0.6

E	H	T
F	0.5	0.5
L	0.8	0.2

p	?	?	?	?	?	?	?	?	?	?	?
x	T	H	T	H	H	H	T	H	T	T	H
$s_{\text{Fair}, i}$	0.25	?	?	?	?	?	?	?	?	?	?
$s_{\text{Loaded}, i}$	0.1	?	?	?	?	?	?	?	?	?	?

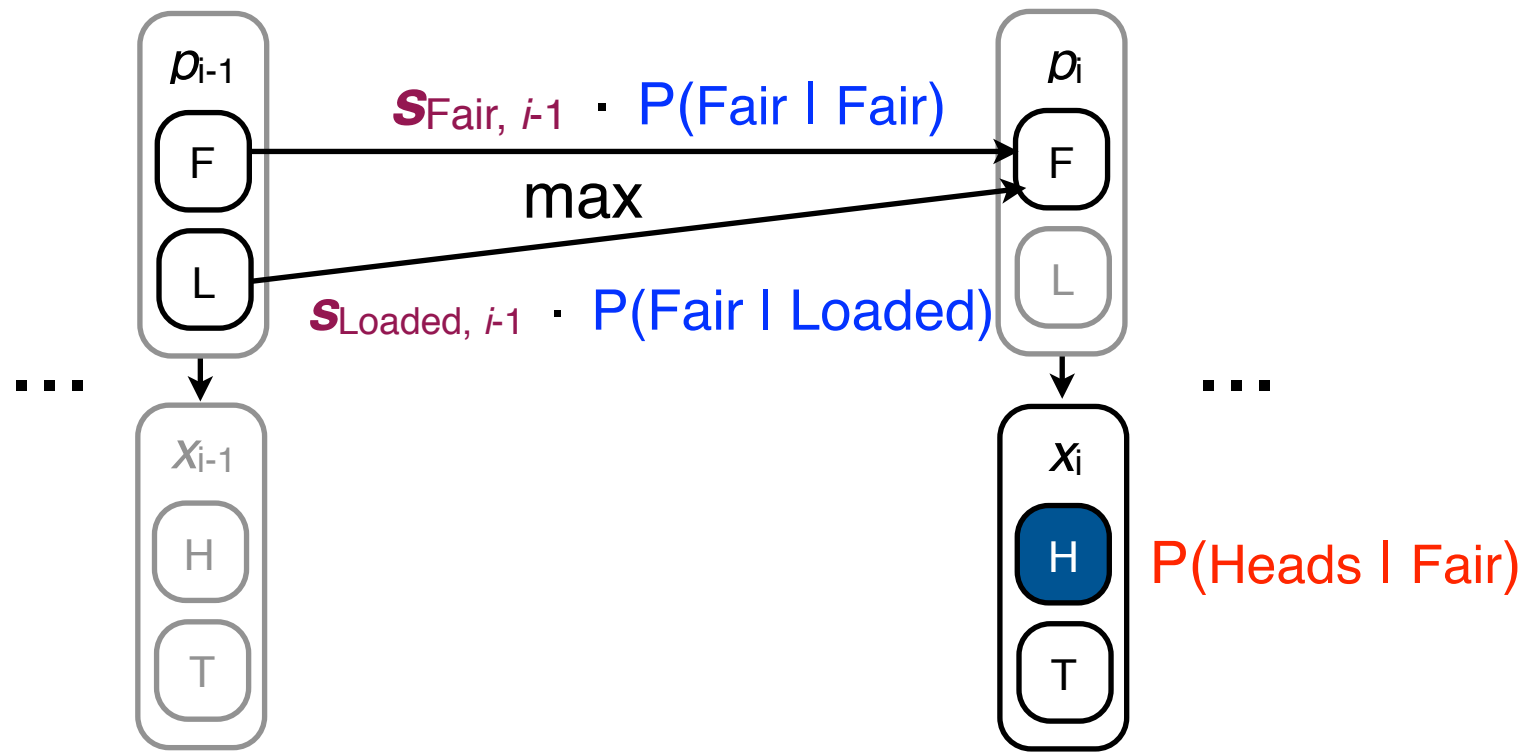
Viterbi fills in all the question marks

Hidden Markov Model: Viterbi algorithm

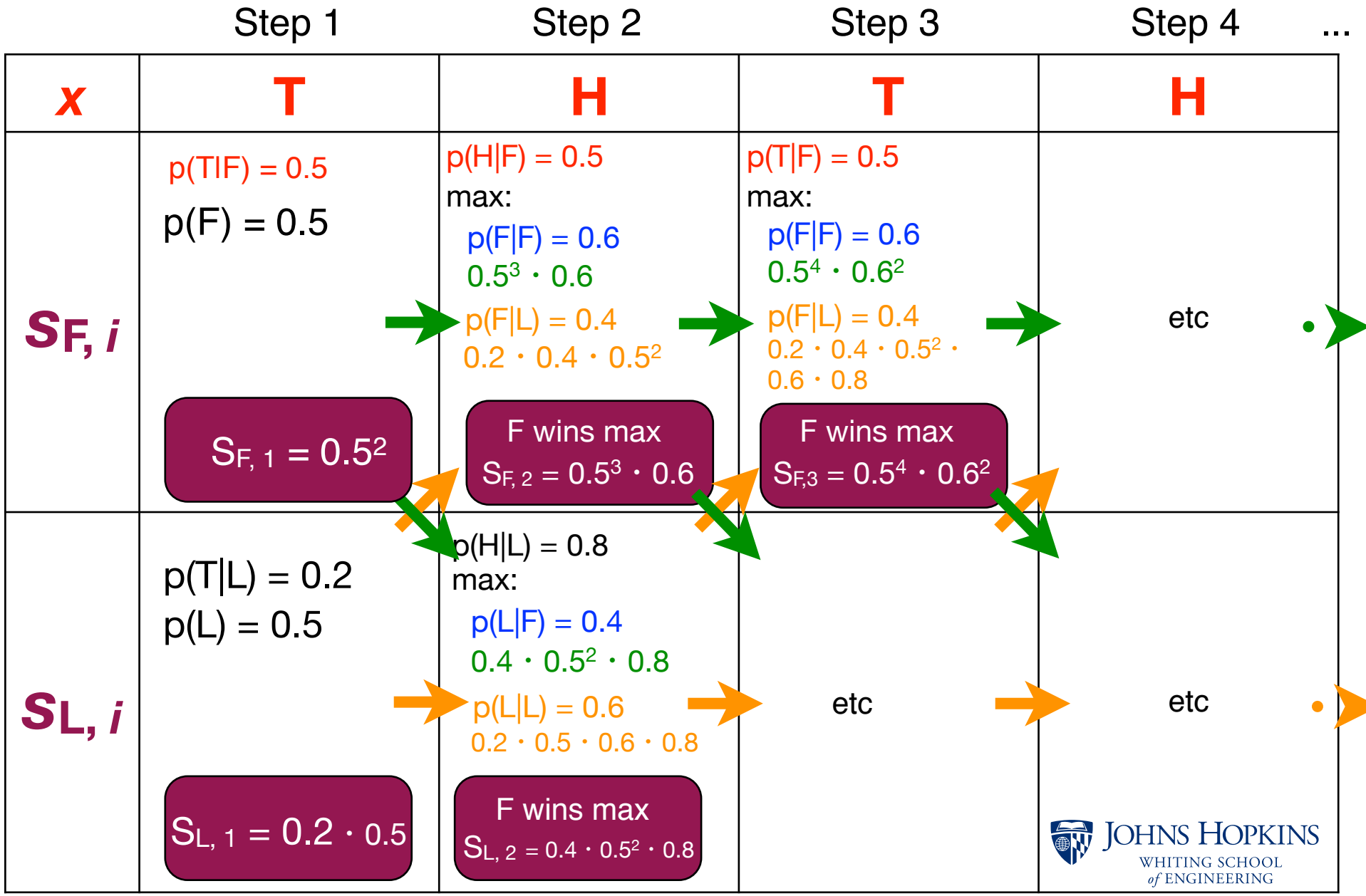
$$s_{\text{Fair},i} = P(\text{Heads} \mid \text{Fair}) \cdot \max_{k \in \{\text{Fair, Loaded}\}} \{ s_{k,i-1} \cdot P(\text{Fair} \mid k) \}$$

Emission prob

Transition prob



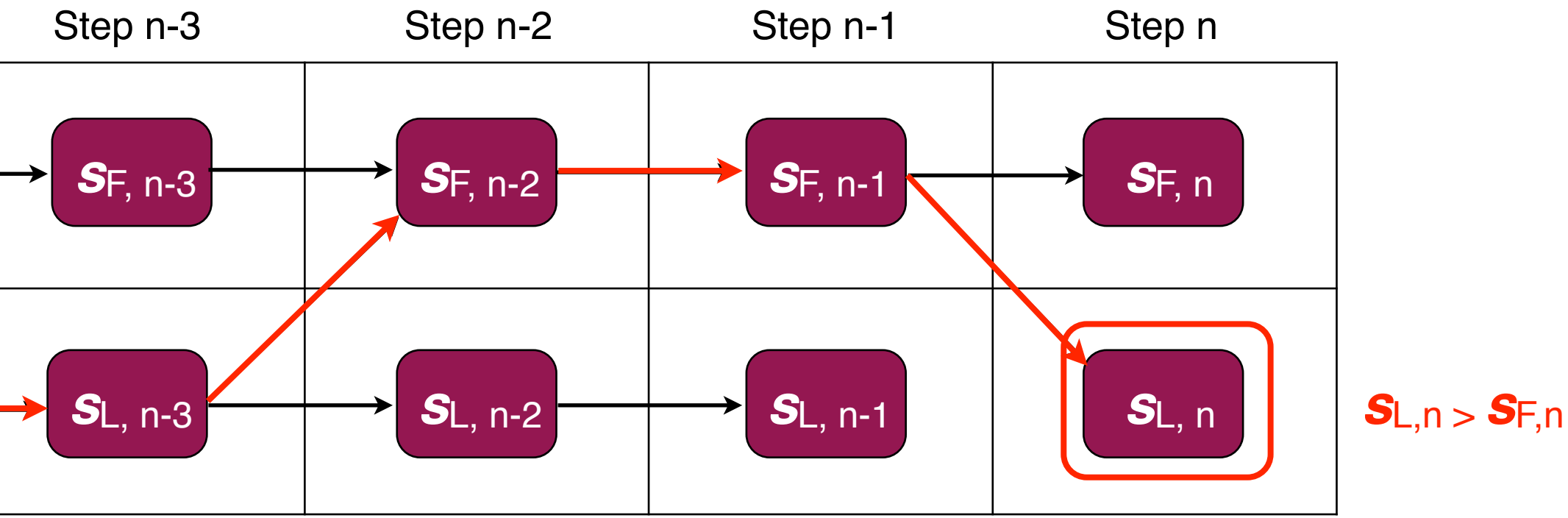
Hidden Markov Model: Viterbi algorithm



Hidden Markov Model: Viterbi algorithm

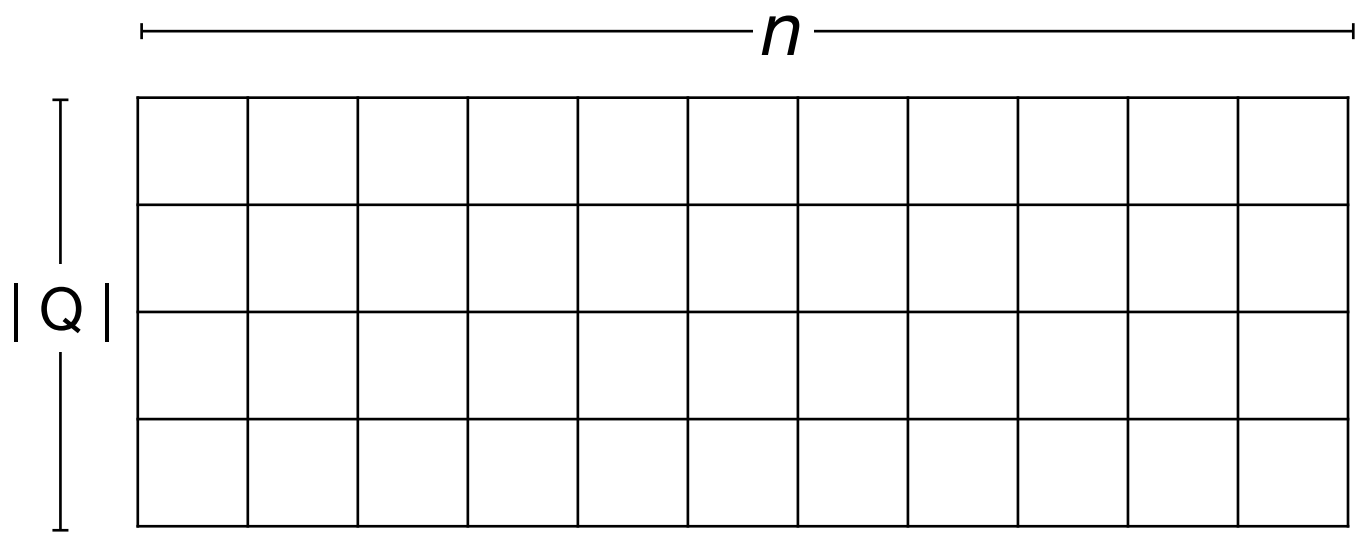
Pick state in step *n* with highest score; *backtrace* for most likely path

Backtrace according to which state *k* "won" the max in:



Hidden Markov Model: Viterbi algorithm

How much work did we do, given Q is the set of states and n is the length of the sequence?



$S_{k,i}$ values to calculate = $n \cdot |Q|$, each involves max over $|Q|$ products

$O(n \cdot |Q|^2)$

Matrix A has $|Q|^2$ elements, E has $|Q||\sum|$ elements, I has $|Q|$ elements

Hidden Markov Model: Implementation

```
def viterbi(self, x):
    ''' Given sequence of emissions, return the most probable path
        along with its joint probability. '''
    x = map(self.smap.get, x) # turn emission characters into ids
    nrow, ncol = len(self.Q), len(x)
    mat   = numpy.zeros(shape=(nrow, ncol), dtype=float) # prob
    matTb = numpy.zeros(shape=(nrow, ncol), dtype=int)   # backtrace
    # Fill in first column
    for i in xrange(0, nrow):
        mat[i, 0] = self.E[i, x[0]] * self.I[i]
    # Fill in rest of prob and Tb tables
    for j in xrange(1, ncol):
        for i in xrange(0, nrow):
            ep = self.E[i, x[j]]
            mx, mxi = mat[0, j-1] * self.A[0, i] * ep, 0
            for i2 in xrange(1, nrow):
                pr = mat[i2, j-1] * self.A[i2, i] * ep
                if pr > mx:
                    mx, mxi = pr, i2
            mat[i, j], matTb[i, j] = mx, mxi
    # Find final state with maximal probability
    omx, omxi = mat[0, ncol-1], 0
    for i in xrange(1, nrow):
        if mat[i, ncol-1] > omx:
            omx, omxi = mat[i, ncol-1], i
    # Backtrace
    i, p = omxi, [omxi]
    for j in xrange(ncol-1, 0, -1):
        i = matTb[i, j]
        p.append(i)
    p = ''.join(map(lambda x: self.Q[x], p[::-1]))
    return omx, p # Return probability and path
```

`mat` holds the $S_{k,i}$'s

`matTb` holds traceback info

`self.E` holds emission probs

`self.A` holds transition probs

`self.I` holds initial probs

Calculate $S_{k,i}$'s

Find maximal $S_{k,n}$

Backtrace

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Hidden Markov Model: Viterbi algorithm

```
>>> hmm = HMM({"FF":0.6, "FL":0.4, "LF":0.4, "LL":0.6},
...           {"FH":0.5, "FT":0.5, "LH":0.8, "LT":0.2},
...           {"F":0.5, "L":0.5})
>>> prob, _ = hmm.viterbi("THTHHHTHTTH")
>>> print prob
2.86654464e-06
>>> prob, _ = hmm.viterbi("THTHHHTHTTH" * 100)
>>> print prob
0.0
```

Repeat string 100 times

Occasionally dishonest casino setup

What happened? Underflow!

Python example: http://nbviewer.ipython.org/7460513

Hidden Markov Model: Viterbi algorithm

```
>>> hmm = HMM({"FF":0.6, "FL":0.4, "LF":0.4, "LL":0.6},
...           {"FH":0.5, "FT":0.5, "LH":0.8, "LT":0.2},
...           {"F":0.5, "L":0.5})
>>> prob, _ = hmm.viterbi("THTHHHTHTTH")
>>> print prob
2.86654464e-06
>>> prob, _ = hmm.viterbi("THTHHHTHTTH" * 100)
>>> print prob
0.0
>>> logprob, _ = hmm.viterbiL("THTHHHTHTTH" * 100)
>>> print logprob
-1824.4030071946879
```

Repeat string 100 times

log-space Viterbi

When multiplying many numbers in (0, 1], we quickly approach the smallest number representable in a machine word. Past that we have *underflow* and processor rounds down to 0.

Switch to log space. Multiplies become adds.

Python example: http://nbviewer.ipython.org/7460513

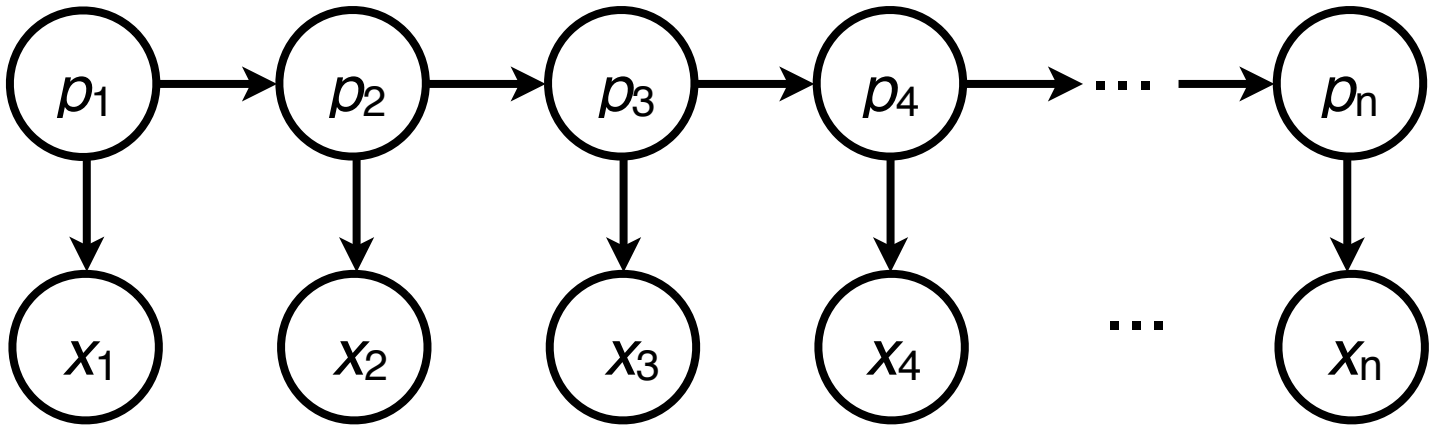
Hidden Markov Model: Implementation

```
def viterbiL(self, x):
    ''' Given sequence of emissions, return the most probable path
        along with log2 of its joint probability. '''
    x = map(self.smap.get, x) # turn emission characters into ids
    nrow, ncol = len(self.Q), len(x)
    mat   = numpy.zeros(shape=(nrow, ncol), dtype=float) # prob
    matTb = numpy.zeros(shape=(nrow, ncol), dtype=int)   # backtrace
    # Fill in first column
    for i in xrange(0, nrow):
        mat[i, 0] = self.Elog[i, x[0]] + self.Ilog[i]
    # Fill in rest of log prob and Tb tables
    for j in xrange(1, ncol):
        for i in xrange(0, nrow):
            ep = self.Elog[i, x[j]]
            mx, mxi = mat[0, j-1] + self.Alog[0, i] + ep, 0
            for i2 in xrange(1, nrow):
                pr = mat[i2, j-1] + self.Alog[i2, i] + ep
                if pr > mx:
                    mx, mxi = pr, i2
            mat[i, j], matTb[i, j] = mx, mxi
    # Find final state with maximal log probability
    omx, omxi = mat[0, ncol-1], 0
    for i in xrange(1, nrow):
        if mat[i, ncol-1] > omx:
            omx, omxi = mat[i, ncol-1], i
    # Backtrace
    i, p = omxi, [omxi]
    for j in xrange(ncol-1, 0, -1):
        i = matTb[i, j]
        p.append(i)
    p = ''.join(map(lambda x: self.Q[x], p[::-1]))
    return omx, p # Return log probability and path
```

log-space version

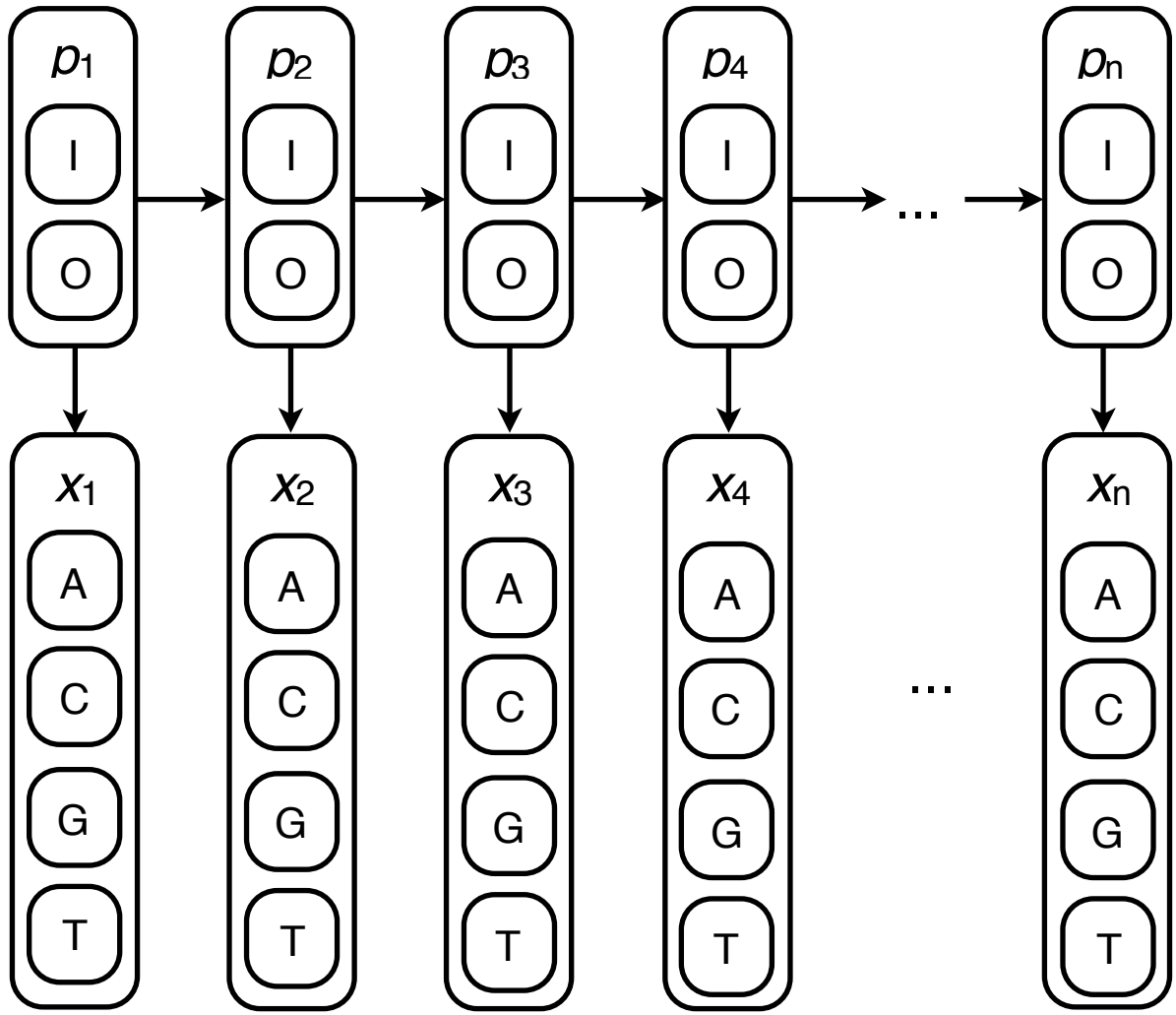
Hidden Markov Model

Task: design an HMM for finding CpG islands?



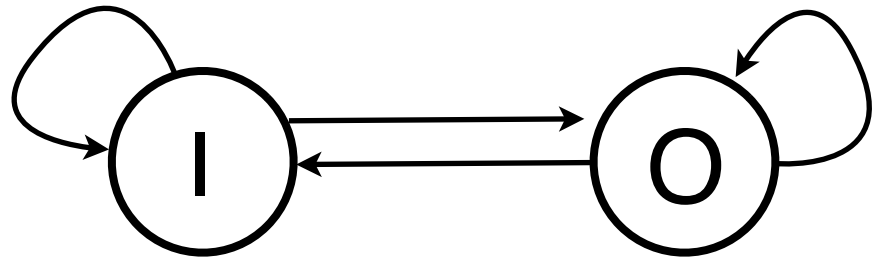
Hidden Markov Model

Idea 1: Q = { inside, outside }, ∑ = { A, C, G, T }



Hidden Markov Model

Idea 1: Q = { inside, outside }, ∑ = { A, C, G, T }



inside outside

A	I	O
I		
O		

Transition matrix

E	A	C	G	T
I				
O				

Emission matrix

Estimate as fraction of positions where we transition from inside to outside

Estimate as fraction of nucleotides inside islands that are C

Hidden Markov Model

Example 1 using HMM idea 1:

A	I	O
I	0.8	0.2
O	0.2	0.8

E	A	C	G	T
I	0.1	0.4	0.4	0.1
O	0.25	0.25	0.25	0.25

x: ATATATACGCGCGCGCGCGATATATATATATA

p: OOOOOOIIIIIIIIIIIIIOOOOOOOOOOO

(from Viterbi)

Python example: http://nbviewer.ipython.org/7460513

Hidden Markov Model

Example 2 using HMM idea 1:

A	I	O
I	0.8	0.2
O	0.2	0.8

E	A	C	G	T
I	0.1	0.4	0.4	0.1
O	0.25	0.25	0.25	0.25

x: ATATCGCGCGCGATATATCGCGCGCGATATATAT

p: OOOOIIIIIIIIOOOOOOIIIIIIIIOOOOOOOO

(from Viterbi)

Python example: http://nbviewer.ipython.org/7460513

Hidden Markov Model

Example 3 using HMM idea 1:

A	I	O
I	0.8	0.2
O	0.2	0.8

E	A	C	G	T
I	0.1	0.4	0.4	0.1
O	0.25	0.25	0.25	0.25

x: ATATATACCCCCCCCCCCCCCATATATATATATA

p: OOOOOOIIIIIIIIIIIIIIOOOOOOOOOOOO

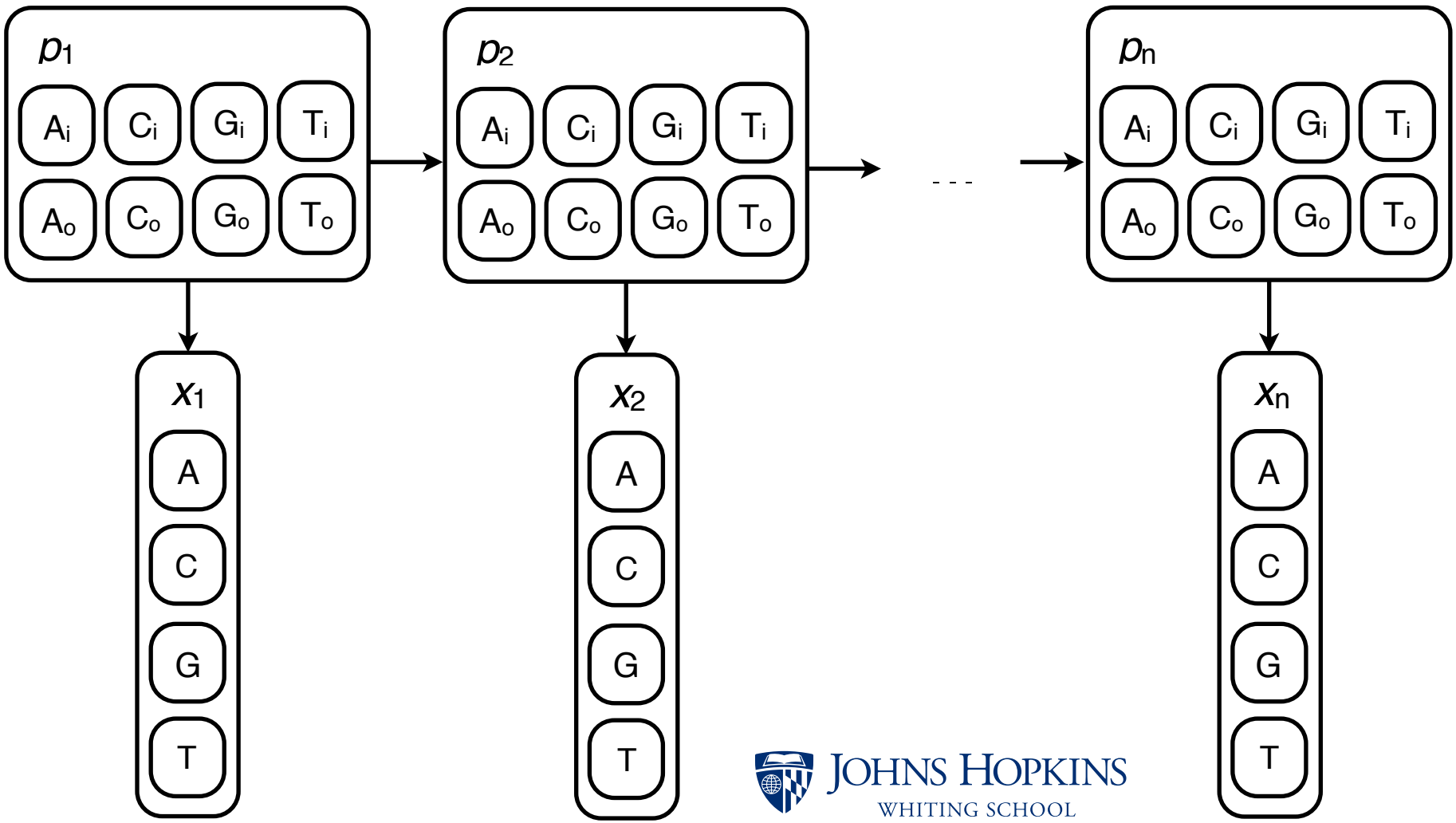
(from Viterbi)

Oops - not a CpG island!

Python example: http://nbviewer.ipython.org/7460513

Hidden Markov Model

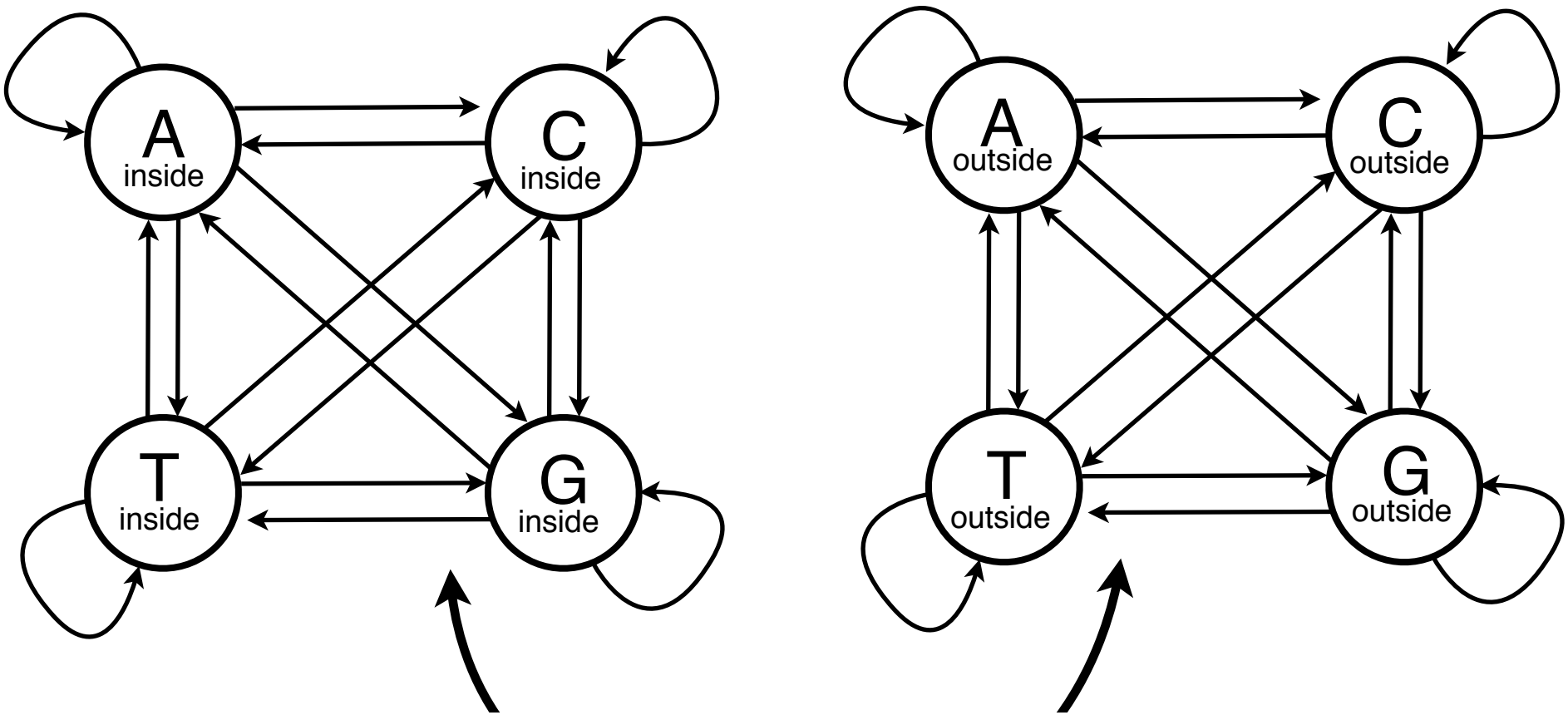
Idea 2: $Q = \{ A_i, C_i, G_i, T_i, A_o, C_o, G_o, T_o \}$, $\Sigma = \{ A, C, G, T \}$



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Hidden Markov Model

Idea 2: $Q = \{ A_i, C_i, G_i, T_i, A_o, C_o, G_o, T_o \}$, $\Sigma = \{ A, C, G, T \}$



All inside-outside edges

Hidden Markov Model

Idea 2: $Q = \{ A_i, C_i, G_i, T_i, A_o, C_o, G_o, T_o \}$, $\Sigma = \{ A, C, G, T \}$

A	A_i	C_i	G_i	T_i	A_o	C_o	G_o	T_o
A_i								
C_i								
G_i								
T_i								
A_o								
C_o								
G_o								
T_o								

Estimate $P(C_i \mid T_i)$ as fraction of all dinucleotides where first is an inside T, second is an inside C

E	A	C	G	T
A_i	1	0	0	0
C_i	0	1	0	0
G_i	0	0	1	0
T_i	0	0	0	1
A_o	1	0	0	0
C_o	0	1	0	0
G_o	0	0	1	0
T_o	0	0	0	1

Hidden Markov Model

Actual trained transition matrix A:

```
A:
[[  1.85152516e-01   2.75974026e-01   4.00289017e-01   1.37026750e-01
    3.19045117e-04   3.19045117e-04   6.38090233e-04   2.81510397e-04]
 [  1.89303979e-01   3.58523577e-01   2.52868527e-01   1.97836007e-01
    4.28792308e-04   5.72766368e-04   3.75584503e-05   4.28792308e-04]
 [  1.72369088e-01   3.29501650e-01   3.55446538e-01   1.40829292e-01
    3.39848138e-04   4.94038497e-04   7.64658311e-04   2.54886104e-04]
 [  9.38783432e-02   3.40823149e-01   3.75970400e-01   1.86949063e-01
    2.56686367e-04   5.57197235e-04   1.05804868e-03   5.07112091e-04]
 [  0.00000000e+00   3.78291020e-05   0.00000000e+00   0.00000000e+00
    2.94813496e-01   1.94641138e-01   2.86962055e-01   2.23545482e-01]
 [  0.00000000e+00   7.57154865e-05   0.00000000e+00   0.00000000e+00
    3.26811872e-01   2.94079570e-01   6.17258712e-02   3.17306971e-01]
 [  0.00000000e+00   5.73810399e-05   0.00000000e+00   0.00000000e+00
    2.57133507e-01   2.33483327e-01   2.94234944e-01   2.15090841e-01]
 [  0.00000000e+00   3.11417347e-05   0.00000000e+00   0.00000000e+00
    1.79565378e-01   2.32469115e-01   2.94623408e-01   2.93310958e-01]]
```

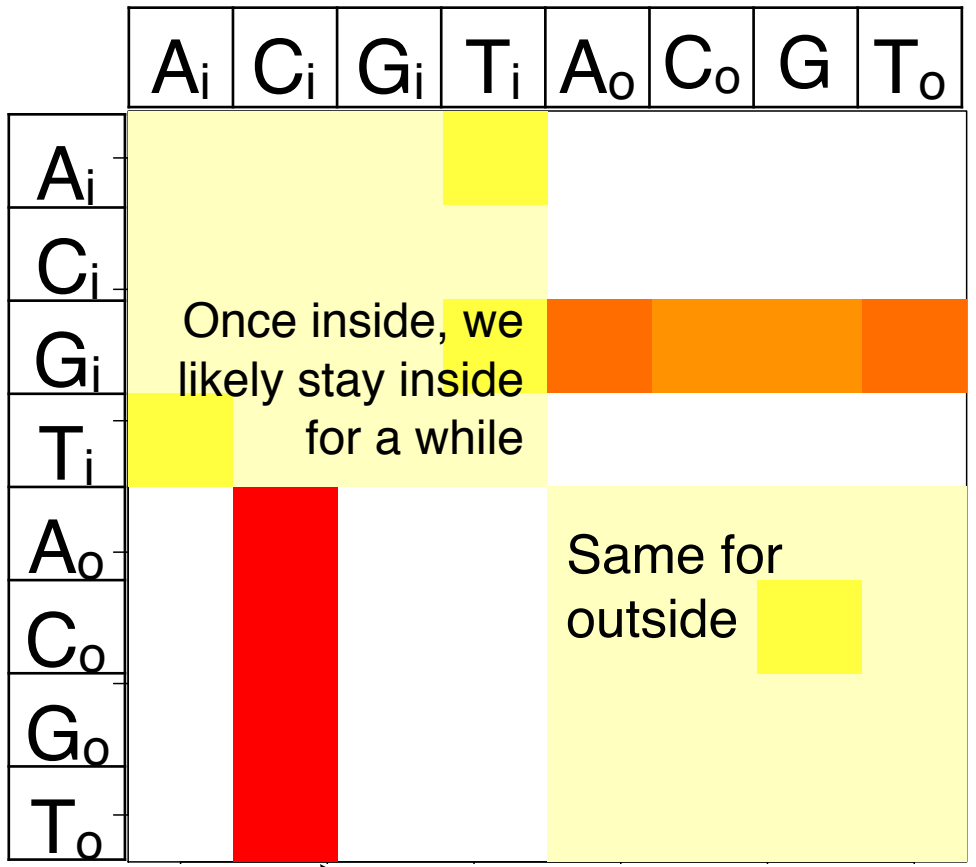
Hidden Markov Model

Actual trained transition matrix A:

Red & orange: low probability

Yellow: high probability

White: probability = 0



← CpG islands end with G (in fact, they end with CG)

↖ CpG islands start with C (in fact, they start with CG)

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Hidden Markov Model

Viterbi result; lowercase = *outside*, uppercase = *inside*:

```
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatCGCGCGCGCG
CGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCG
CGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
```

Hidden Markov Model

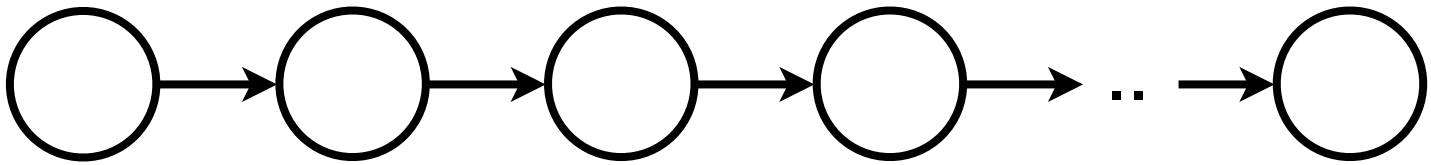
Viterbi result; lowercase = *outside*, uppercase = *inside*:

```
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatgggggggggggg
gggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
atatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatatat
```

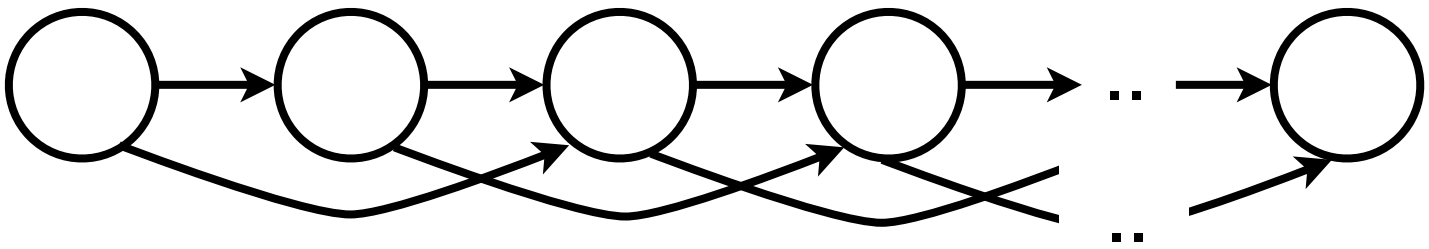
Hidden Markov Model

Many of the Markov chains and HMMs we've discussed are *first order,* but we can also design models of higher orders

First-order Markov chain:

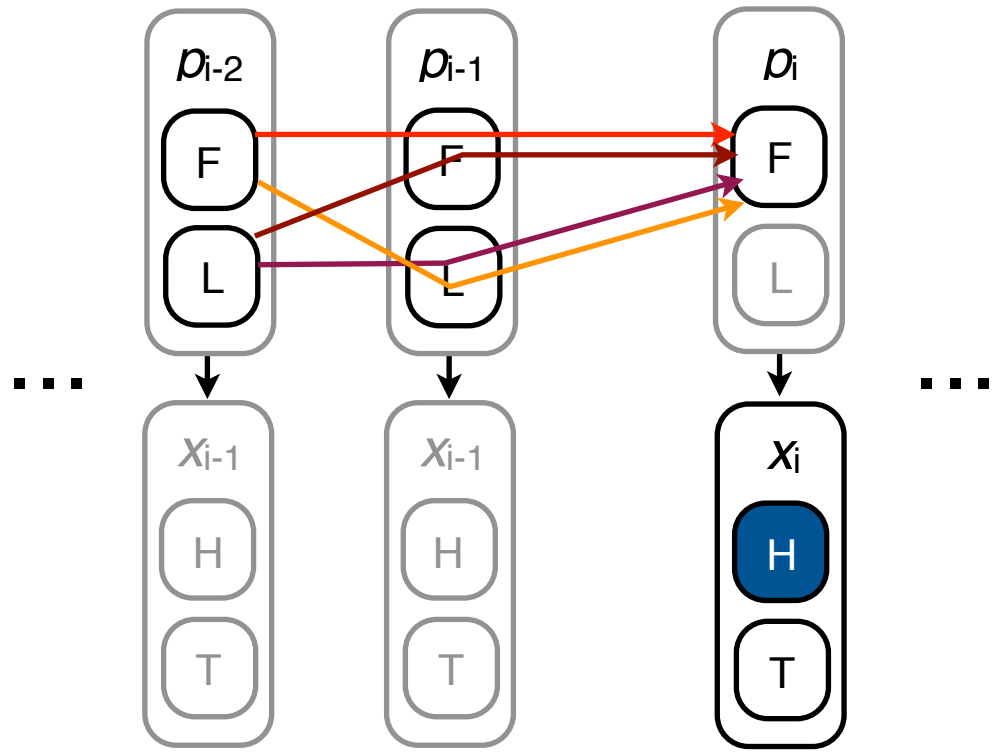


Second-order Markov chain:



Hidden Markov Model

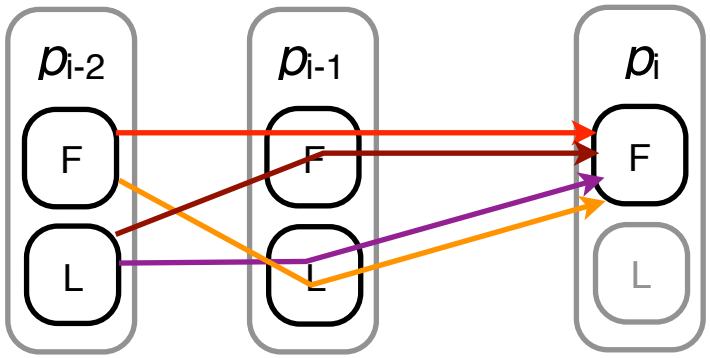
For higher-order HMMs, Viterbi $S_{k,\,i}$ no longer depends on just the previous state assignment

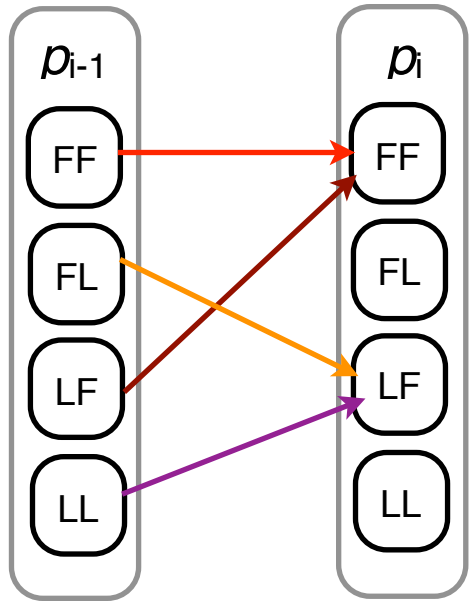


Can sidestep the issue by expanding the state space...

Hidden Markov Model

Now *one* state encodes the last *two* "loadedness"es of the coin





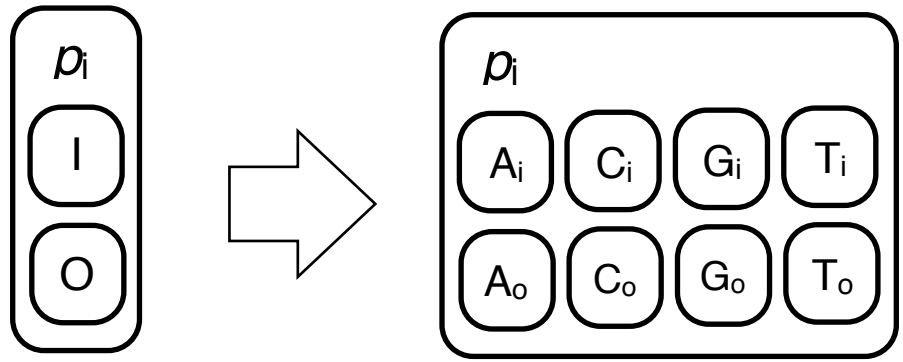
$Q = \{ F, L \}$

$Q = \{ F, L \} \times \{ F, L \}$

After expanding, usual Viterbi works fine.

Hidden Markov Model

We also expanded the state space here:



$Q = \{ \mathrm{I}, \mathrm{O} \}$ $\qquad$ $Q = \{ \mathrm{I}, \mathrm{O} \} \times \{ \mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T} \}$

Other types of HMMs in genomics

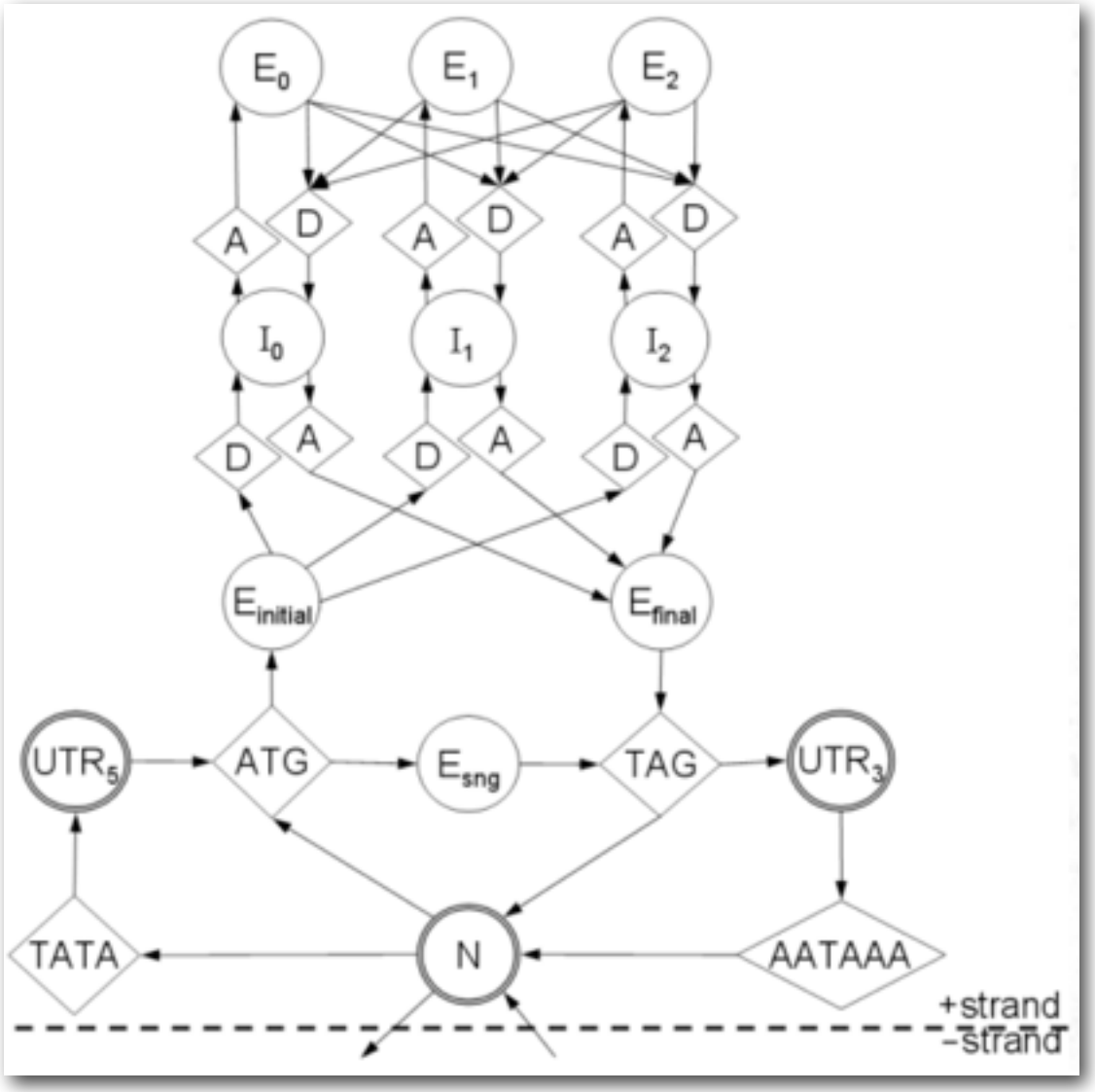


Image: Bill Majoros,
http://www.genezilla.org/design.html

Gene Structure

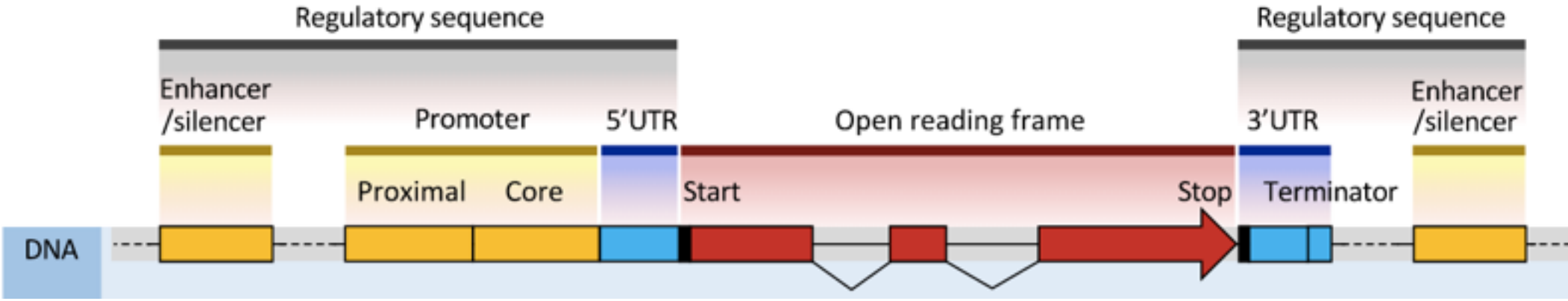


Image: wikipedia
https://en.wikipedia.org/wiki/gene

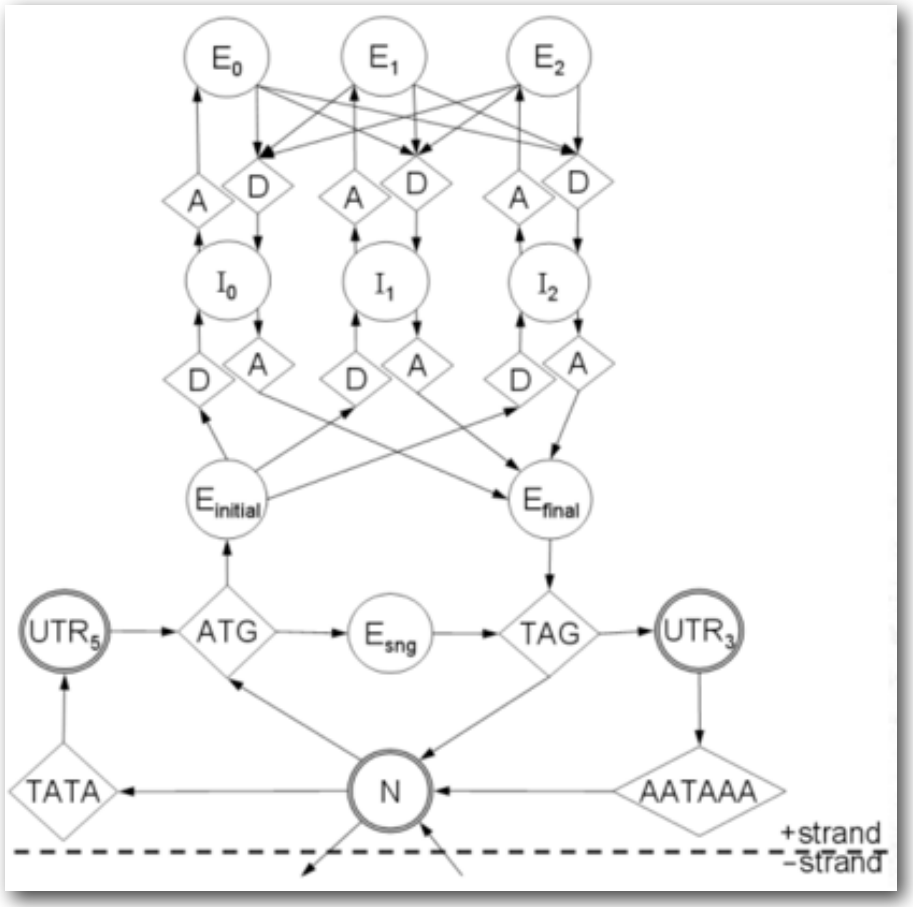


Image: Bill Majoros,
http://www.genezilla.org/design.html

Summary

- After sequencing a genome, we need to annotate genes and other functional elements
- Can try to infer functional elements from experiments or directly from primary sequence
- Hidden Markov Models are suited to many of these tasks (e.g. CpG Island and gene finding)
- Next week: guest lecture from Michael Hoffman (PMCC) on more advanced genome annotation