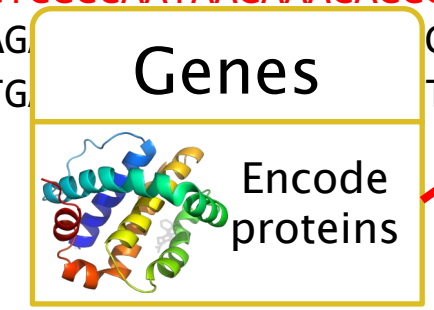
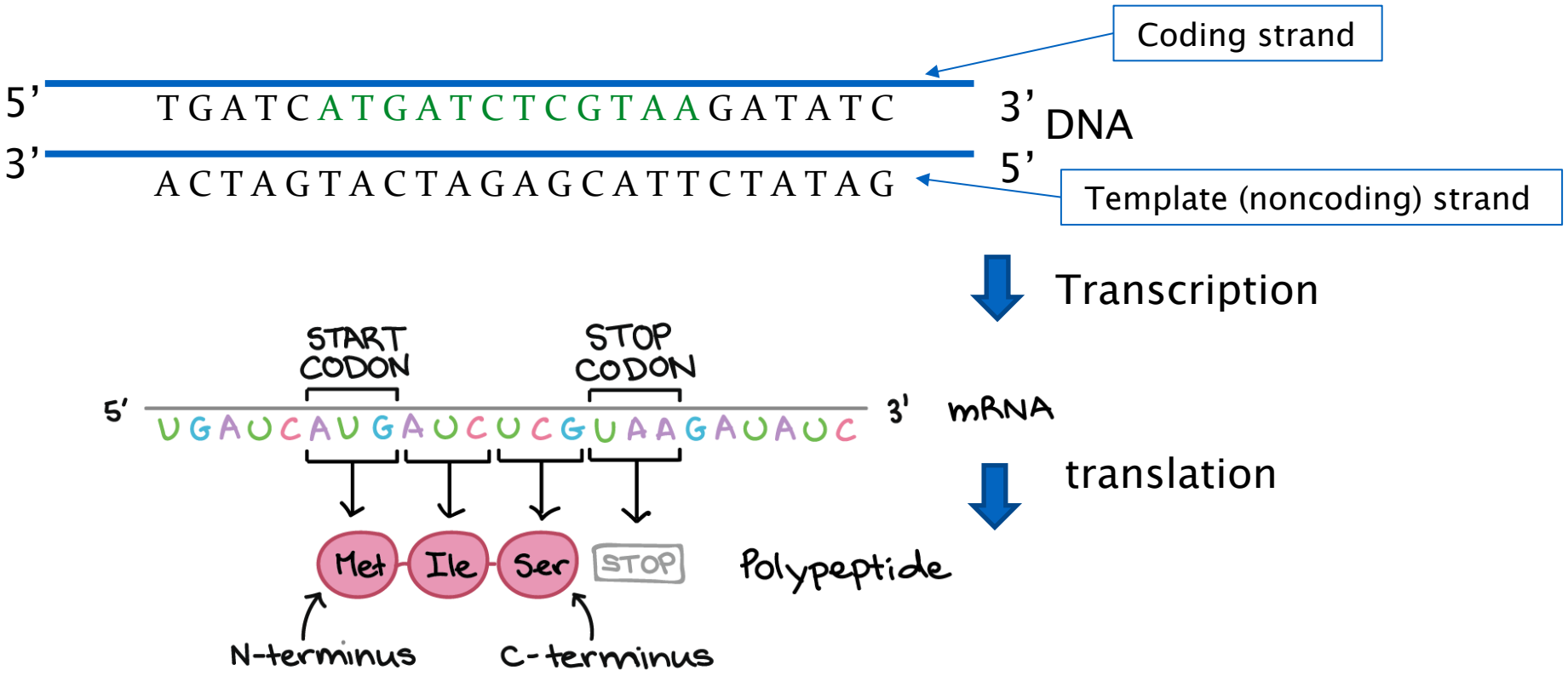
Gene Prediction

Gene Prediction

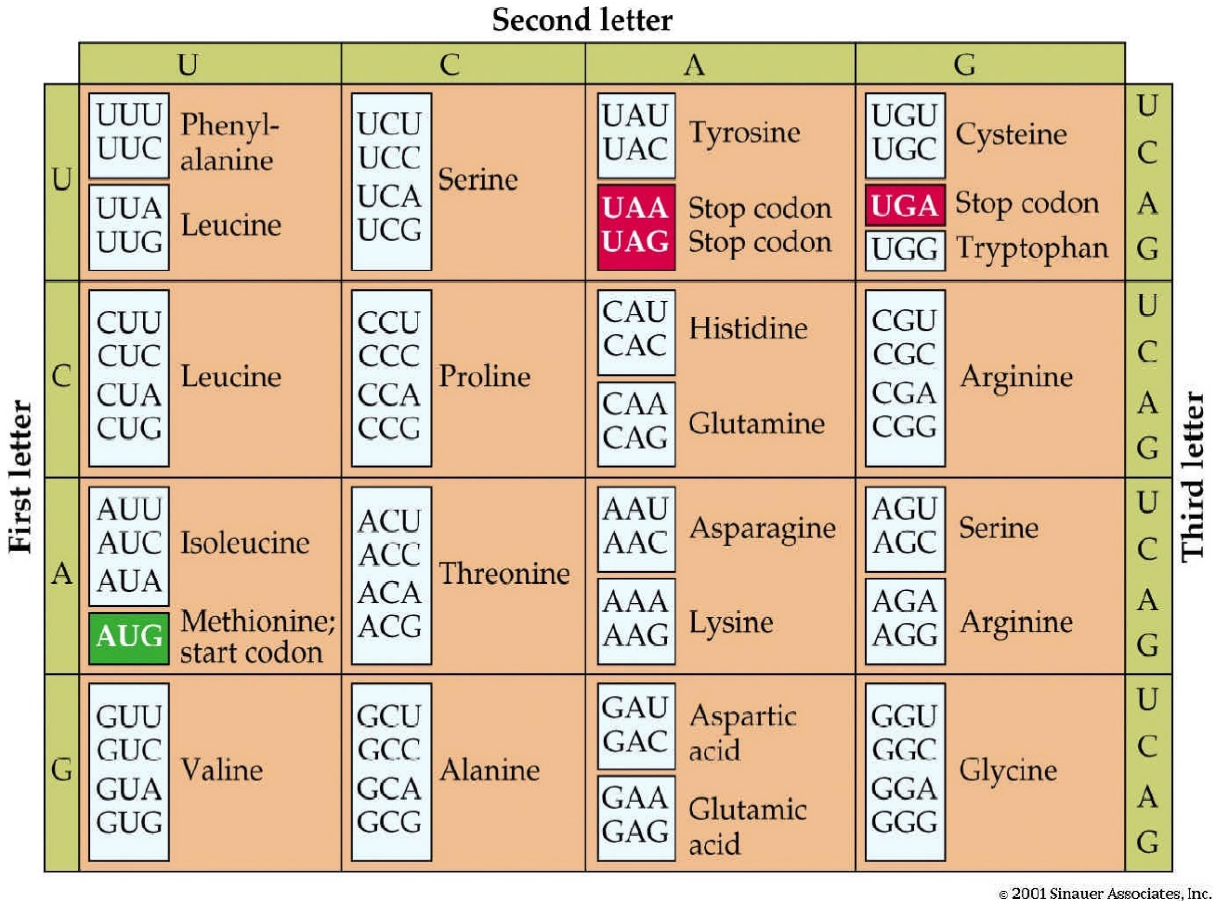
TTATATTGAATTTTCAAAAATTCTTACTTTTTTTTTGGATGGACGCAAAGAAGTTTAATAATCATATTACATGGCATTACCACCATATACATATCCATATCTAATCT
TACTTATATGTTGTGGAAATGTAAAGAGCCCCATTATCTTAGCCTAAAAAAACCTTCTCTTTGGAACTTTCAGTAATACGCTTAACTGCTCATTGCTATATTGAAGT
ACGGATTAGAAGCCGCCGAGCGGGCGACAGCCCTCCGACGGAAGACTCTCCTCCGTGCGTCCTCGTCTTCACCGGTCGCGTTCCTGAAACGCAGATGTGCCTCGCGC
CGCACTGCTCCGAACAATAAAGATTCTACAATACTAGCTTTTATGGTTATGAAGAGGAAAAATTGGCAGTAACCTGGCCCCACAAACCTTCAAATTAACGAATCAAA
TTAACAACCATAGGATGATAATGCGATTAGTTTTTTAGCCTTATTTCTGGGGTAATTAATCAGCGAAGCGATGATTTTTGATCTATTAACAGATATATAAATGGAAA
AGCTGCATAACCACTTTAACTAATACTTTCAACATTTTCAGTTTGTATTACTTCTTATTCAAATGTCATAAAAGTATCAACAAAAAATTGTTAATATACCTCTATAC
TTTAACGTCAAGGAGAAAAAACTATAATGACTAAATCTCATTCAGAAGAAGTGATTGTACCTGAGTTCAATTCTAGCGCAAAGGAATTACCAAGACCATTGGCCGAA
AAGTGCCCGAGCATAATTAAGAAATTTATAAGCGCTTATGATGCTAAACCGGATTTTGTTGCTAGATCGCCTGGTAGAGTCAATCTAATTGGTGAACATATTGATTA
TTGTGACTTCTCGGTTTTACCTTTAGCTATTGATTTTGATATGCTTTGCGCCGTCAAAGTTTTGAACGAGAAAAATCCATCCATTACCTTAATAAATGCTGATCCCA
AATTTGCTCAAAGGAAGTTCGATTTGCCGTTGGACGGTTCTTATGTCACAATTGATCCTTCTGTGTCGGACTGGTCTAATTACTTTAAATGTGGTCTCCATGTTGCT
CACTCTTTTCTAAAGAAACTTGCACCGGAAAGGTTTGCCAGTGCTCCTCTGGCCGGGCTGCAAGTCTTCTGTGAGGGTGATGTACCAACTGGCAGTGGATTGTCTTC
TTCGGCCGCATTCATTTGTGCCGTTGCTTTAGCTGTTGTTAAAGCGAATATGGGCCCTGGTTATCATATGTCCAAGCAAAATTTAATGCGTATTACGGTCGTTGCAG
AACATTATGTTGGTGTTAACAATGGCGGTATGGATCAGGCTGCCTCTGTTTGCGGTGAGGAAGATCATGCTCTATACGTTGAGTTCAAACCGCAGTTGAAGGCTACT
CCGTTTAAATTTCCGCAATTAAAAAACCATGAAATTAGCTTTGTTATTGCGAACACCCTTGTTGTATCTAACAAGTTTGAAACCGCCCCAACCAACTATAATTTAAG
AGTGGTAGAAGTCACTACAGCTGCAAATGTTTTAGCTGCCACGTACGGTGTTGTTTTACTTTCTGGAAAAGAAGGATCGAGCACGAATAAAGGTAATCTAAGAGATT
TCATGAACGTTTATTATGCCAGATATCACAACATTTCCACACCCTGGAACGGCGATATTGAATCCGGCATCGAACGGTTAACAAAGATGCTAGTACTAGTTGAAGAG
TCTCTCGCCAATAAGAAACAGGGCTTTAGTGTTGACGATGTCGCACGCATTGGGCAGCTGTCTATATGAATTATAAAATCCTTGAATTGTTCTCGCGAAGAATTCAC
AAGAG[illegible]CCAGTGAGATTTCAAGTCTTAAAGCTATATCAGAGGGCTAAGCATGTGTATTCTGAATCTTTAAGAGTCTTGAAGGCTGTGAAAT
TAATG[illegible]TGCCGACGAAGACTTTTTCAAGCAATTTGGTGCCTTGATGAACGAGTCTCAAGC



From Gene to Protein (in *Prokaryotes*)



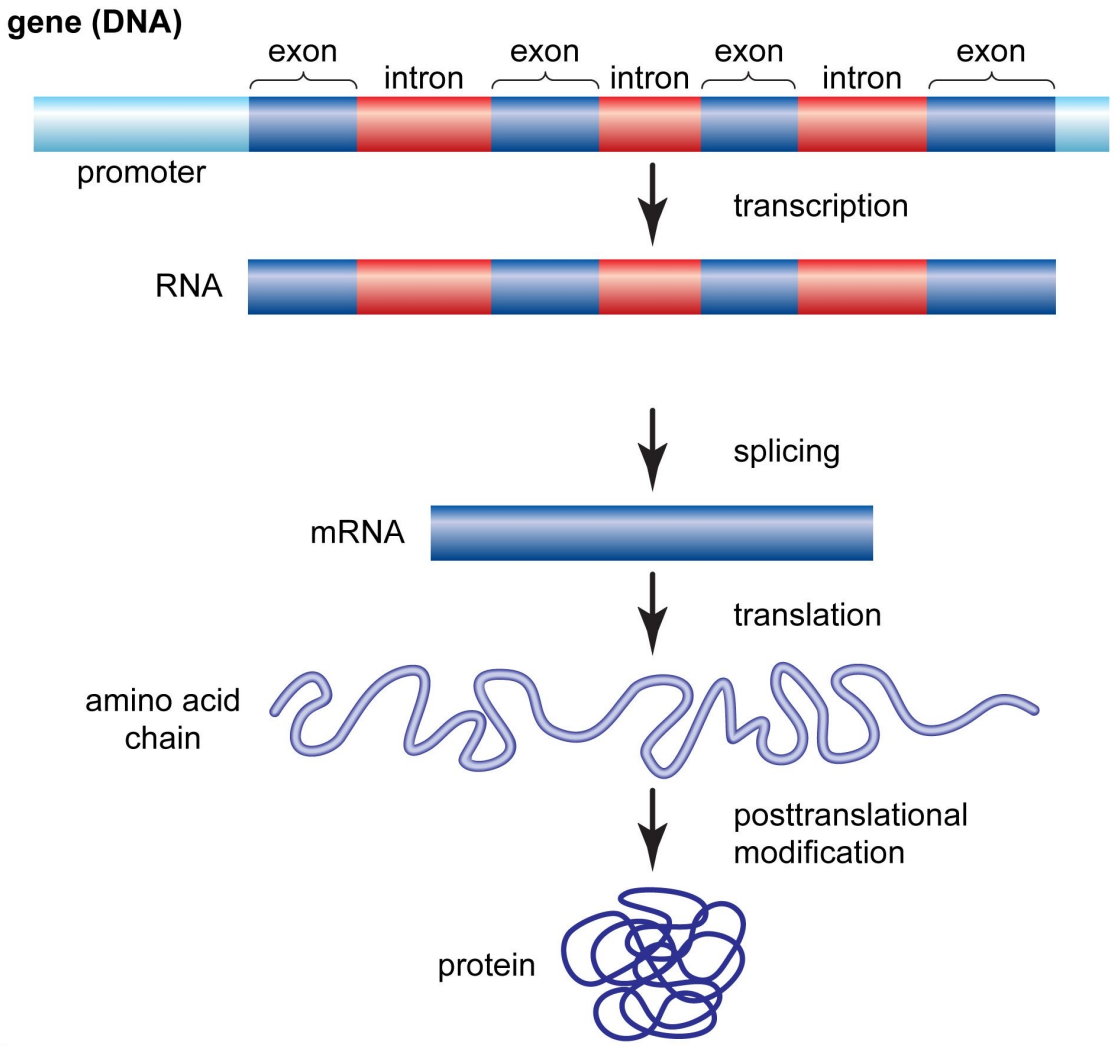
Genetic code



codons

ATT I
ATC
ATA
CAC H
CAA
CAG
AAA S
AAG
AGT
AGG G
GGG
GGA
GGG

From Gene to Protein (in *Eukaryotes*)

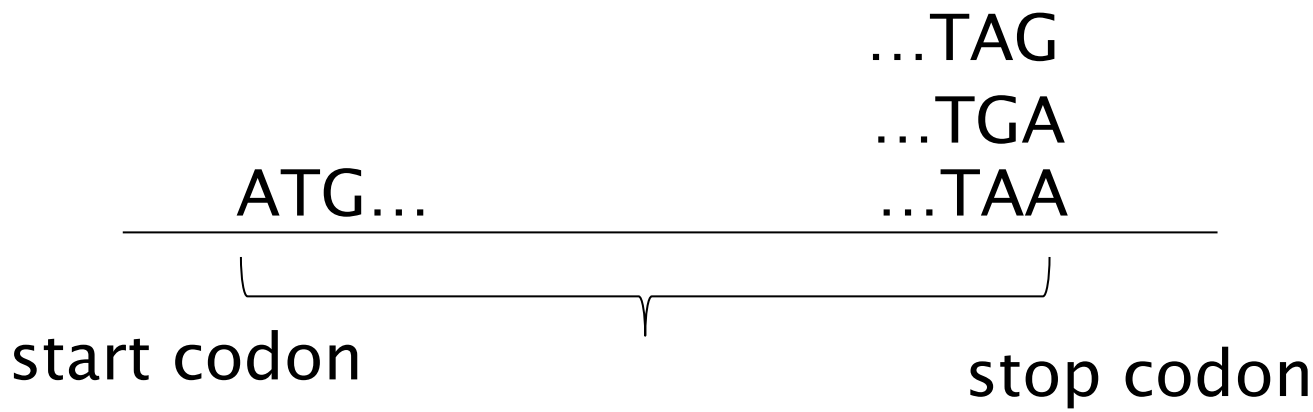


Eukaryotes vs Prokaryotes

	Prokaryotes	**Eukaryotes**
Nucleus	No membrane-bound nucleus (DNA in nucleoid)	Membrane-bound nucleus
Cellular organization	Almost always unicellular	Unicellular or multicellular
Genome form	Usually single circular chromosome	Multiple linear chromosomes
Genome size	~0.5–10 Mb typical	~10 Mb–100+ Gb
Gene density	High	Low (large intergenic regions common)
Introns	Rare (almost absent)	Common (especially in animals and plants)

A Trivial Gene Finder

- Open Reading Frame (ORF) is a substring that
  - starts with a start codon
  - ends with a stop codon
  - no stop codon in the middle
- If ORF is long, then likely it is a gene or a part of a gene.
- Why?



Translate Tool

- Try out the translate tool at: https://web.expasy.org/translate/
- With the sequence on slide 2.

Codon bias

- A codon XYZ occurs with different frequencies in coding regions and non-coding regions
  - different amino acids have different frequency.
  - Different codons for the same amino acid have different frequency.
  - In random regions approx. p(X)*p(Y)*p(Z)

Codon Bias Tables
(% of codons used
for each residue)

Amino Acid	Codon		
Gly	GGG	2	25
Gly	GGA	0	25
Gly	GGU	59	16
Gly	GGC	39	34

http://www.kazusa.or.jp/codon/

Escherichia coli O157:H7 EDL933 **[gbbct]: 5347 CDS's (1611503 codons)**

fields: [triplet] [frequency: **per thousand**] ([number])

UUU 22.2(35846)	UCU 8.7(14013)	UAU 16.5(26648)	UGU 5.2(8458)
UUC 15.9(25565)	UCC 8.9(14420)	UAC 12.3(19766)	UGC 6.4(10285)
UUA 13.8(22316)	UCA 8.1(13117)	UAA 2.0(3163)	UGA 1.1(1751)
UUG 13.0(20904)	UCG 8.8(14220)	UAG 0.3(435)	UGG 15.3(24656)
CUU 11.4(18366)	CCU 7.2(11657)	CAU 12.8(20631)	CGU 20.2(32590)
CUC 10.5(16869)	CCC 5.6(8961)	CAC 9.4(15116)	CGC 20.8(33547)
CUA 3.9(6257)	CCA 8.4(13507)	CAA 14.7(23703)	CGA 3.8(6166)
CUG 51.1(82300)	CCG 22.4(36178)	CAG 29.4(47324)	CGG 6.2(9955)
AUU 29.7(47838)	ACU 9.1(14639)	AAU 19.2(30864)	AGU 9.4(15123)
AUC 23.9(38504)	ACC 22.8(36724)	AAC 21.7(34907)	AGC 16.0(25800)
AUA 5.5(8835)	ACA 8.1(13030)	AAA 34.0(54723)	AGA 2.9(4656)
AUG 27.2(43846)	ACG 15.0(24122)	AAG 11.0(17729)	AGG 1.8(2915)
GUU 18.1(29200)	GCU 15.4(24855)	GAU 32.8(52914)	GGU 24.2(38983)
GUC 14.8(23870)	GCC 25.2(40571)	GAC 19.2(30953)	GGC 28.1(45226)
GUA 10.9(17561)	GCA 20.7(33343)	GAA 39.3(63339)	GGA 8.9(14286)
GUG 26.2(42261)	GCG 32.3(52091)	GAG 18.7(30158)	GGG 11.8(18947)

Coding GC 51.50% 1st letter GC 58.44% 2nd letter GC 40.88% 3rd letter GC 55.17%

A Better Gene Finder

- We can use the log likelihood ratio score to evaluate each ORF. Each codon XYZ contributes score
- $\log \frac{P(XYZ)}{P(X)P(Y)P(Z)}$
- An ORF is predicted as a gene if the sum of codon score is above a threshold.
- This is better. But it does not catch the correlation between adjacent codons.

HMM

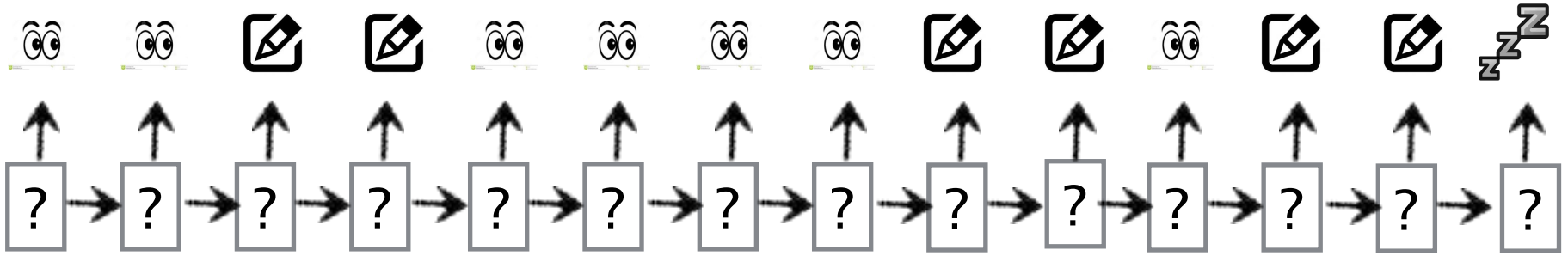
- Hidden Markov model was first invented in speech recognition. But are widely used in many other areas including bioinformatics.
- An automata that has “hidden states”. At each time point, it emits a symbol, and change a state with certain probability.
- We want to derive the hidden states by the emitted symbols.

Classroom example

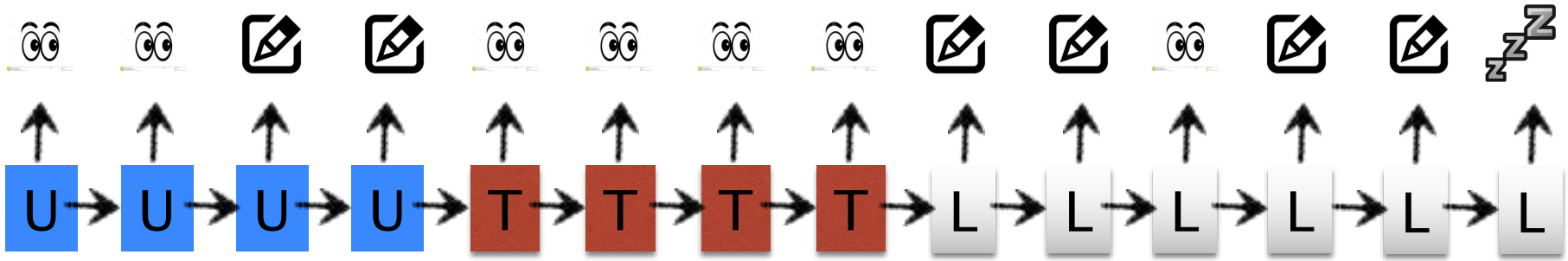
- Think of a student in classroom.
- At any minute, a student is in one of 3 hidden *states* that I try to figure out:
  - U: understands
  - T: does not understand but tries to understand
  - L: is lost completely and does not try to understand
- Meanwhile, the student emits one of 3 *symbols* that I can observe
  - Look at me
  - Write/Type
  - Sleep

Classroom Example

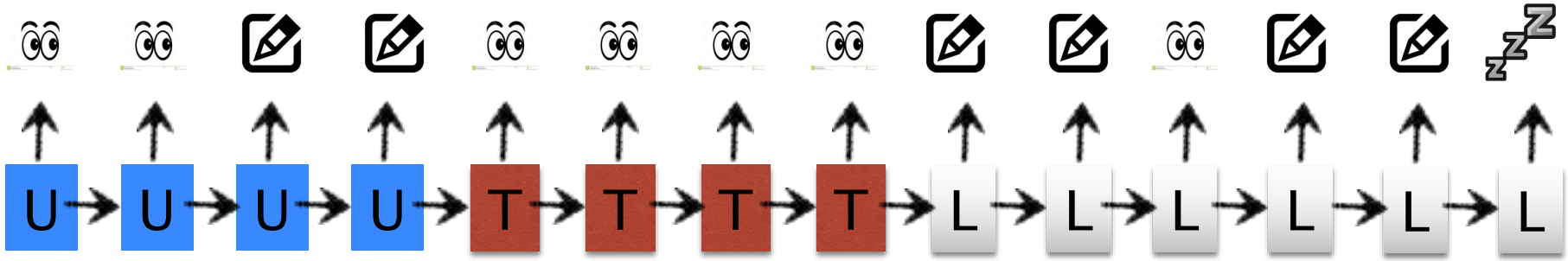
- Now suppose I see a student's behavior is the following in the past several minutes. What is his internal states at each minute?



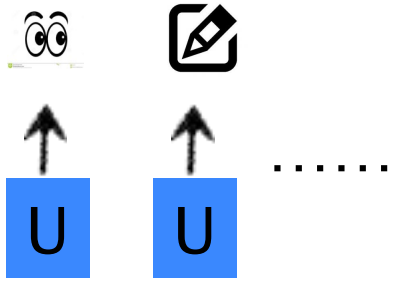
Classroom Example



Classroom Example

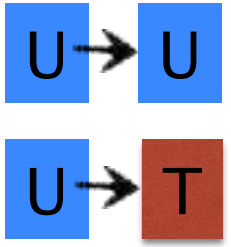


Emissions



.....

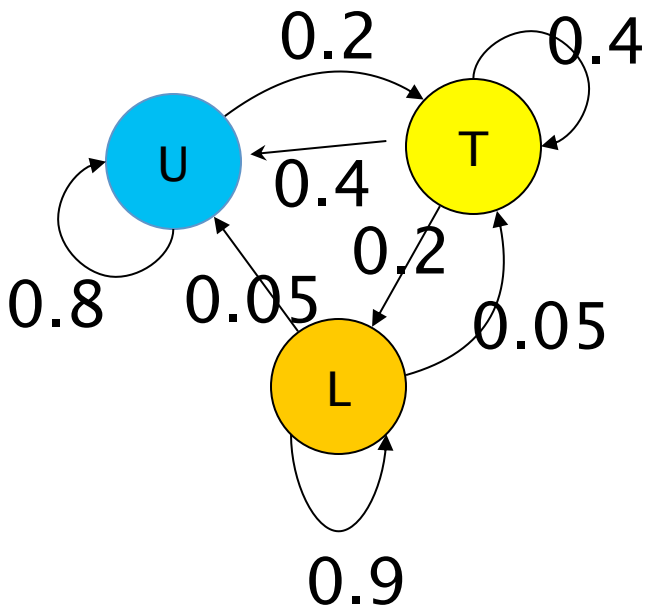
Transitions



.....

Typically, HMM assumes that emission probability depends only on current state; and current state only depends on previous state.
We want to find the most likely path of states given the symbols (observations).

Classroom Example



U: Understands
T: Tries to understand
L: Lost completely

(T) Transition matrix

	U	T	L
U	0.8	0.2	0
T	0.4	0.4	0.2
L	0.05	0.05	0.9

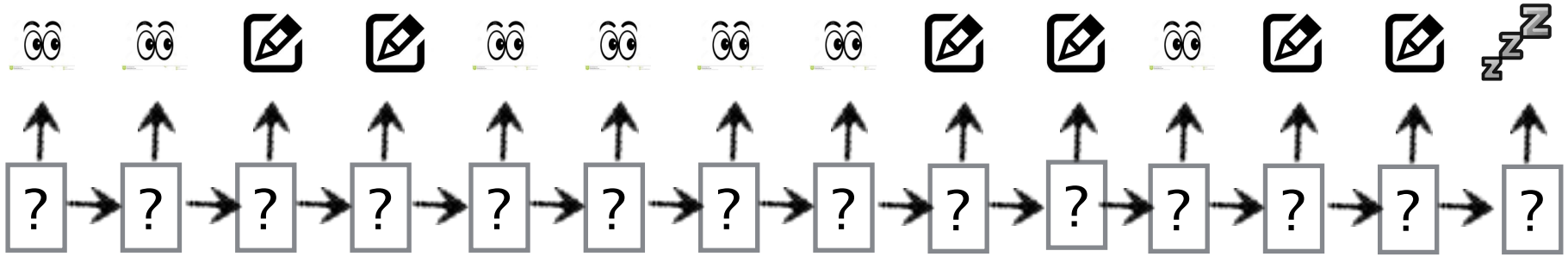
(E) Emission matrix

	Look	Write	Sleep
U	0.6	0.35	0.05
T	0.9	0.1	0
L	0.1	0.6	0.3

Classroom Example

- $S=S_1S_2\ldots S_n$: sequence of symbols;
- $P=P_1P_2\ldots P_n$: path of states.
- We want to maximize Pr(P|S) = Pr(P,S) / Pr(S).
- Therefore, we want to maximize

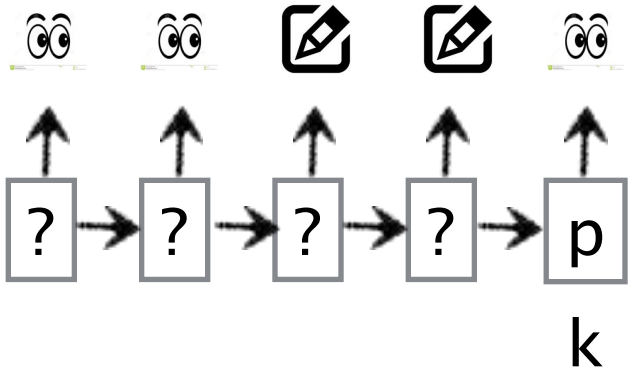
$$\Pr(P,S)=\prod_i \Pr(P_i \mid P_{i-1})\Pr(S_i \mid P_i)$$



* Note: To deal with the first state, we can define $\Pr(P_1|P_0)=1$ in above formula.

Solving HMM

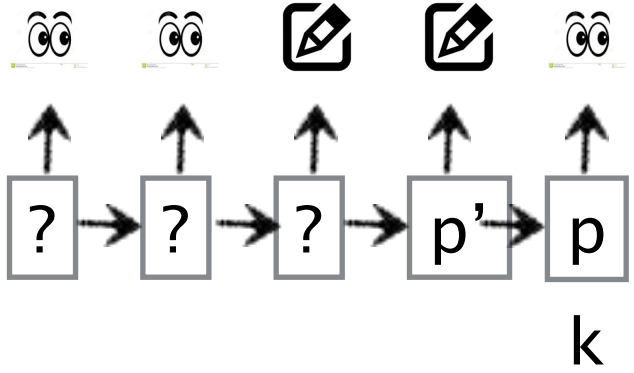
- We use dynamic programming again. Define D[k,p] be the maximum probability achieved by first k states given that the last state is p.



- Then $\max_p$ D[n,p] is the maximum probability achieved by the complete path, which is what we want to compute.
- It is not hard to obtain a recurrence Relation:

$$D[k,p] = \max_{P[1..k];P[k]=p} \prod_{1 \le i \le k} T[P_{i-1}, P_i] E[P_i, S_i]$$

Solving HMM



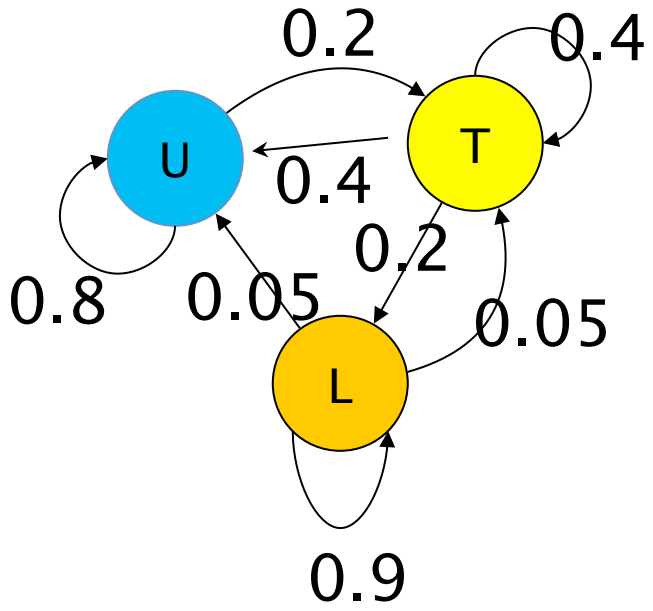
$$D[k,p] = \max_{p'} D[k-1,p'] \Pr(p|p') \Pr(S_i|p)$$

k

Solving HMM

- Viterbi algorithm:
- Input: $S = S_1 S_2 \ldots S_n$
- Output: $P = P_1 P_2 \ldots P_n$
- 1. for every state p, let $D[1,p] = \Pr(S_1|p)$.
- 2. for k from 2 to n,
- 2.1 for every state p,
- 2.1.1 let $D[k,p] = \max_{p'} D[k-1,p'] \Pr(p|p') \Pr(S_i|p)$
- 3. backtrace to compute the optimal path.

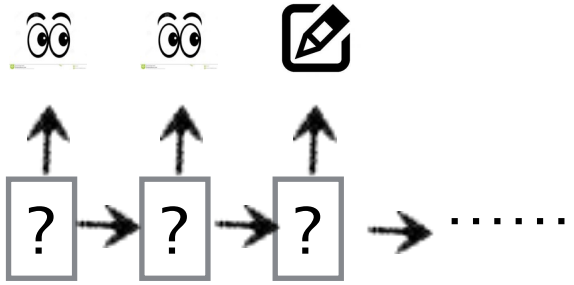
Example



(E) emission matrix

	Look	Write	Sleep
Understand	0.6	0.35	0.05
Try	0.9	0.1	0
Lost	0.1	0.6	0.3

$$D[k,p] = \max_{p'} D[k-1,p'] \Pr(p|p') \Pr(S_i|p)$$

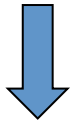


U	0.6		
T	0.9		
L	0.1		

Notes

- Do not multiply
  - because soon the numbers become so small that the double precision will give you value 0.
  - Do a logarithm and use additions instead.

$$D[k,p] = \max_{p'} D[k-1,p'] \Pr(p|p') \Pr(S_i|p)$$



$$\log D[k,p] = \max_{p'} (\log D[k-1,p'] + \log \Pr(p|p') + \log \Pr(S_i|p))$$

Parameter Estimation

- All of our computation depends on the transition probabilities and emission probabilities. How do we estimate these parameters?

Parameter Estimation

- If we have an annotated sequence with both symbols and states, then these can be trained by counting.
- If we do not, then we can start with a reasonable guess of the parameters and annotate the sequence.
- Then we use the annotation to train a new set of probabilities. Repeat until converge.
- There is some guarantee to the convergence. But does not guarantee this will converge to the right solution.

Pseudocounts

- If the training data include no cases of a particular emission from a particular state, then its probability will be 0 in this model.
- That's no good.
- So we add pseudocounts to make the probabilities not zero when an event should be able to happen.

Higher Order HMM

- Think again the classroom example:

U: Understands
T: Tries to understand
L: Lost

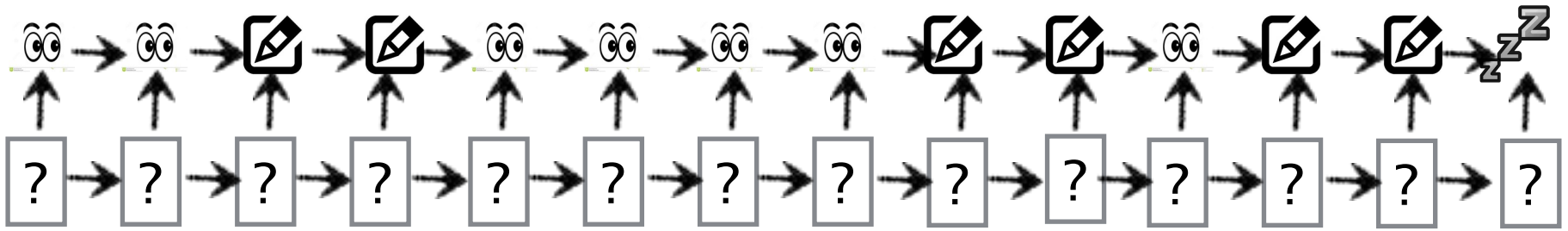
(E) Emission matrix

	Look	Write	Sleep
U	0.6	0.35	0.05
T	0.9	0.1	0
L	0.1	0.6	0.3

- The emission of a symbol should not only depend on current state, but sometimes also the previous symbol.
  - E.g. Sleeping at previous moment leads to a higher probability of sleeping now.

1st Order HMM

- To accommodate the correlation between the adjacent symbols, the emission matrix needs to be expanded.
- The emission matrix becomes $Pr(S_i | P_i, S_{i-1})$.



1st Order HMM

- Before

$$\Pr(P,S) = \prod_i \Pr(P_i \mid P_{i-1}) \Pr(S_i \mid P_i)$$

- Now

$$\Pr(P,S) = \prod_i \Pr(P_i \mid P_{i-1}) \Pr(S_i \mid P_i, S_{i-1})$$

- To find the path P to maximize, we let D[k,p] be the maximum probability obtained by the first k states ending at p. We can obtain the following recurrence relation similarly as before.

$$D[k,p] = \max_{p'} D[k-1,p'] \Pr(p|p') \Pr(S_i|p, S_{i-1})$$

- We can still do dynamic programming.

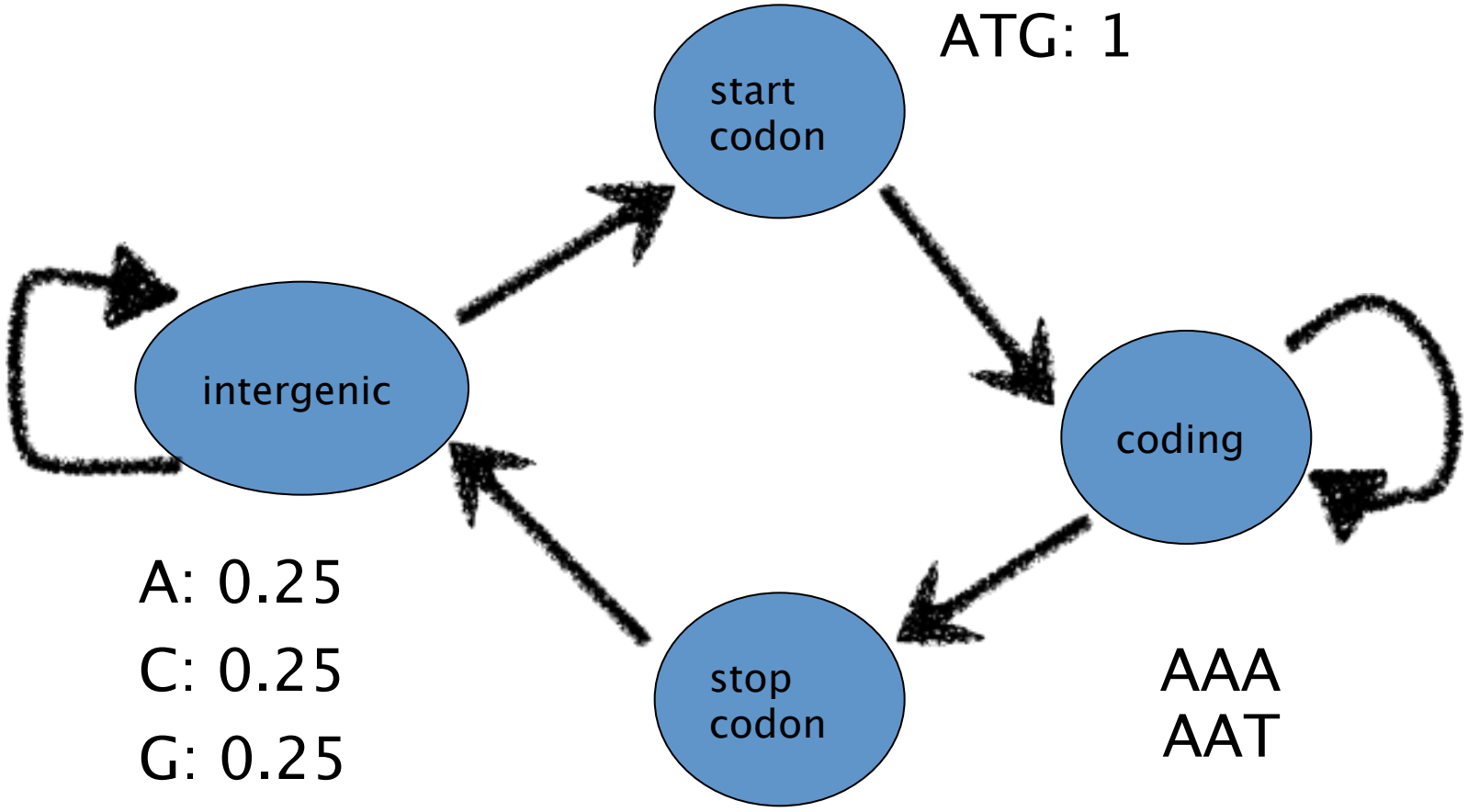
Higher Order HMM

- To generalize, we can let the current emission depend on the current state, and previous k symbols.
- Then this is called the k-th order HMM.
- Solving such a HMM is similar as before. Running time not changed.
- The only difficulty is the parameter training because the emission matrix has many more parameters for larger k.

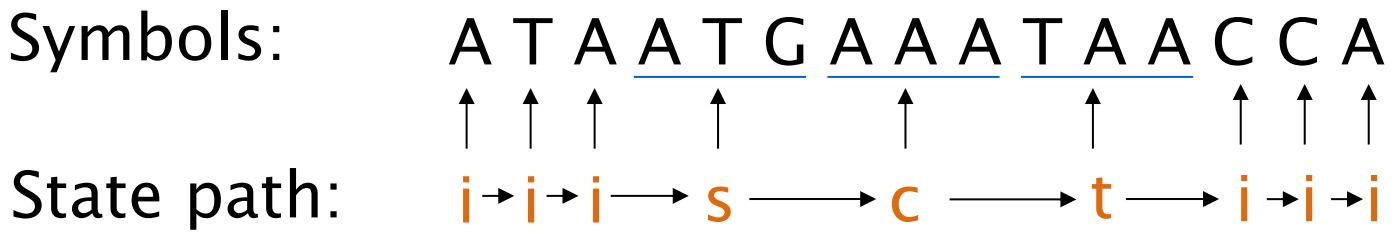
HMM

- We have used HMM in the classroom example to catch correlations between adjacent events.
- This can be used to model gene prediction.
- For example:
  - Symbols: Nucleotide bases.
  - States: start codon, stop codon, coding, non-coding (intergenic).

A Simple HMM for Prokaryote Gene Finding



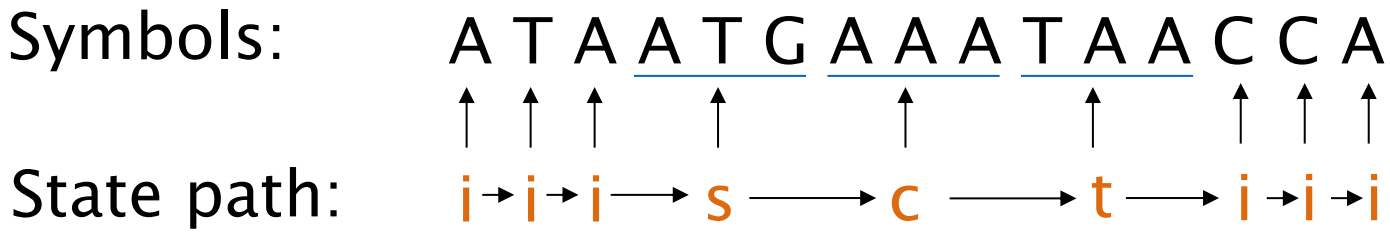
Gene Prediction as HMM



start codon
coding
intergenic
stop codon

- Annotated the sequence with most probable path of states. This provides a reasonable answer to gene prediction.
- A difference here: emission is not fixed length. But this does not forbid us from solving it with dynamic programming.

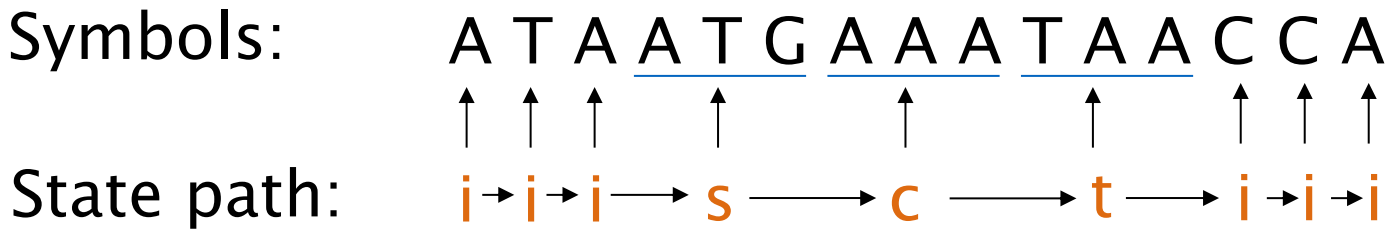
Dynamic Programming



start codon
coding
intergenic
stop codon

- Define D[k,p] be the max probability achieved by first k symbols for a path with the last state being p.

Recurrence Relation



start codon
coding
intergenic
stop codon

For p=intergenic

$$D[k,p] = D[k-1,p']\Pr(p|p')\Pr(S_k|p)$$

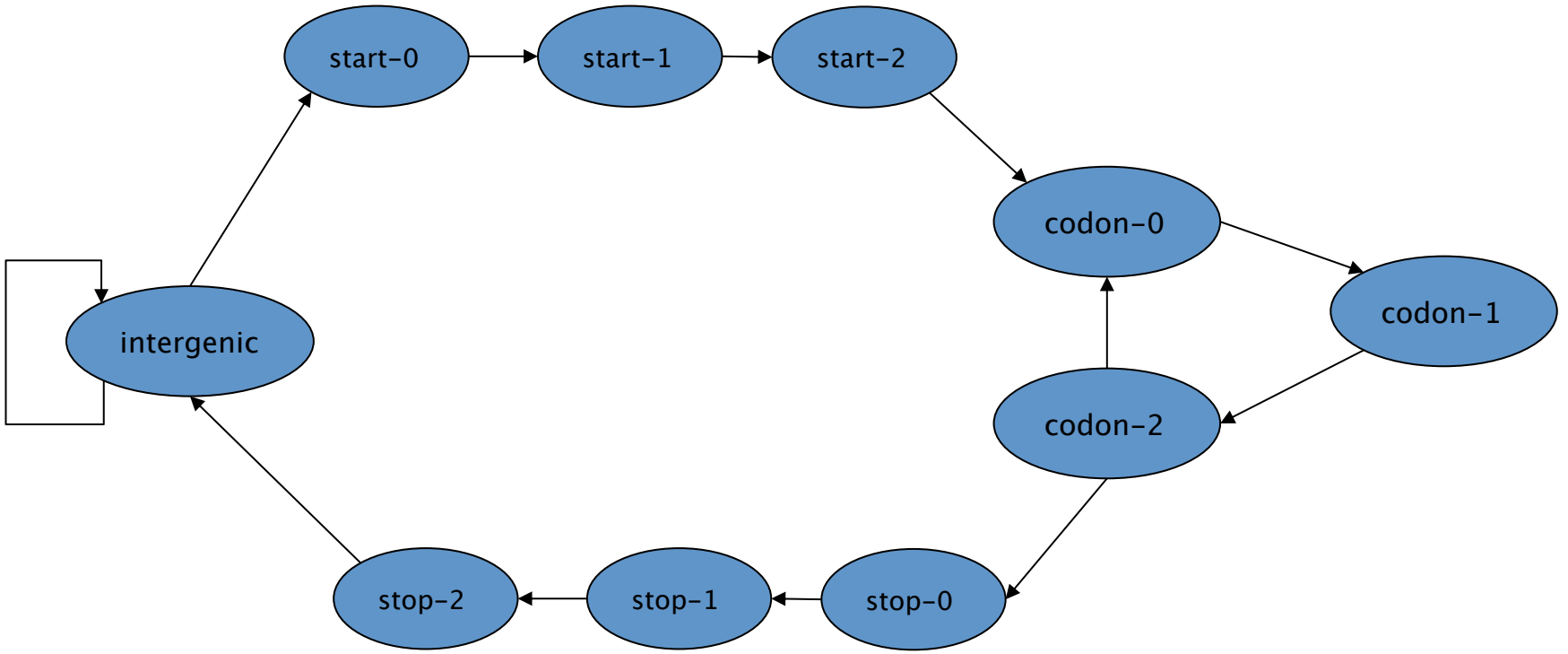
For p=start, coding, or stop

$$D[k,p] = D[k-3,p']\Pr(p|p')\Pr(S_{k-2}S_{k-1}S_k|p)$$

Dynamic Programming

- Once the recurrence relation is obtained. It is straightforward to work out a dynamic programming algorithm.
- This is very easy to implement.
- If desired, one can also use a higher-order HMM.
- Parameter training must be done carefully.

Another Way to Enforce the Codon Structure



Gene Prediction

- Besides the codon bias that can be captured by HMM, there are other signals in a gene structure that can be employed by a gene prediction program.
  - E.g. the promoter of a gene is a region of DNA sequence located near the start codon.

```
 <-- upstream                                                    downstream -->
5'-XXXXPPPPPPXXXXXXXXXPPPPPPXXXXGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGXXXX-3'
       -35            -10       Gene to be translated
```

Promoters

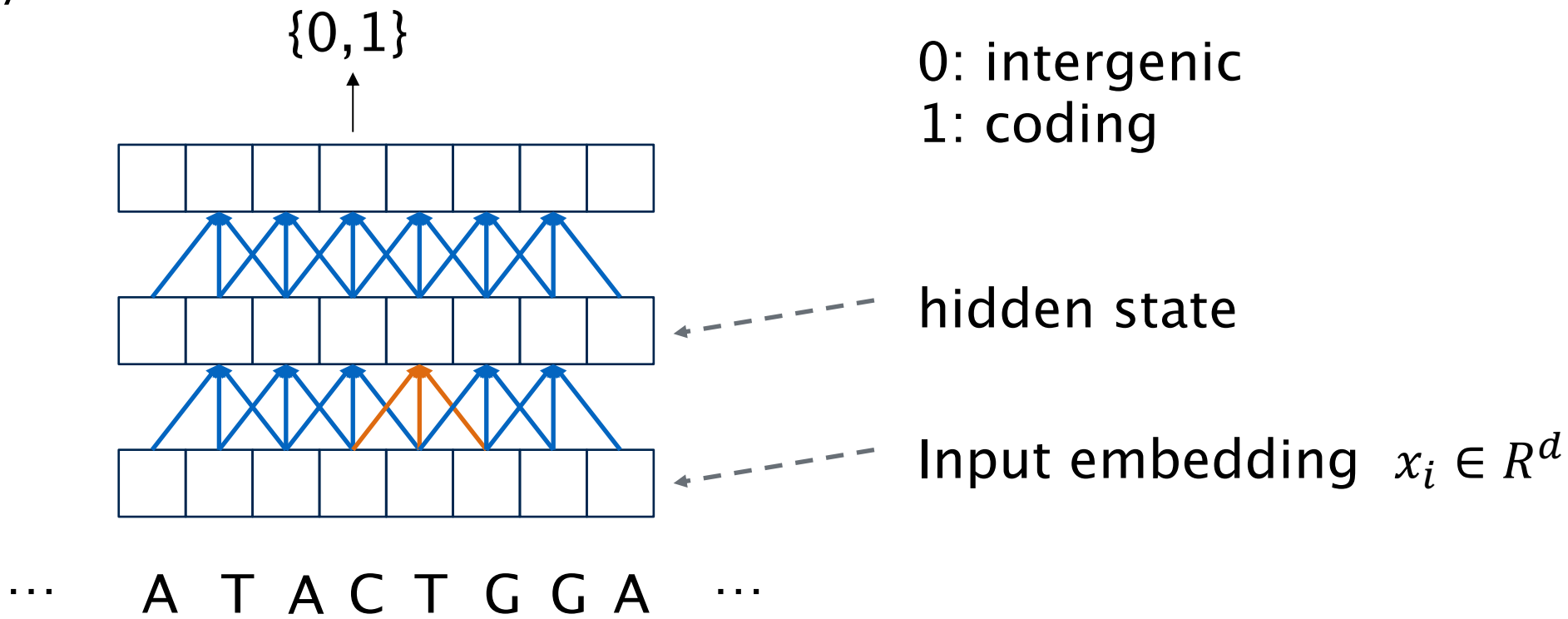
```
 <-- upstream                                                                    downstream -->
5'-XXXXPPPPPPXXXXXXXXXPPPPPPXXXXGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGXXXX-3'
       -35             -10       Gene to be translated
```

-10:	**T**	**A**	**T**	**A**	**A**	**T**
	77%	**76%**	**60%**	**61%**	**56%**	**82%**
-35:	**T**	**T**	**G**	**A**	**C**	**A**
	69%	**79%**	**61%**	**56%**	**54%**	**54%**

- These rules are only approximately correct.
- The presence of promoters allow a very high transcription rate.
- Exercise: How to assign a score to the promoter.

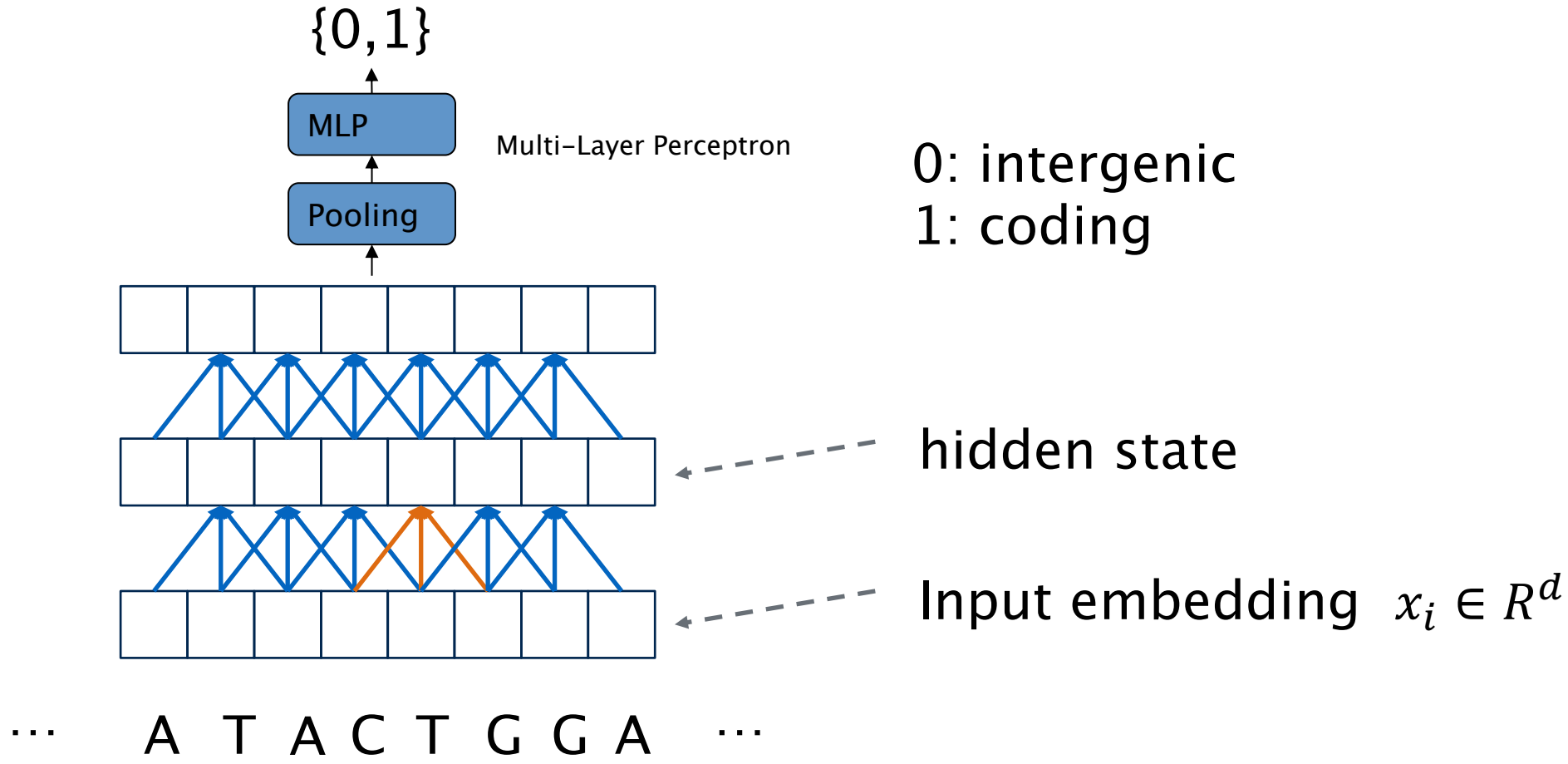
Gene Prediction with CNN

- A convolutional neural network use adjacent hidden states to compute next layer's hidden states. Parmeters are reused in small sliding windows
- Seems very suitable for coding/noncoding annotation.
- Learning of long range dependence requires additional efforts. E.g. downsampling the length of the sequence while increase dimensions in the middle layers.

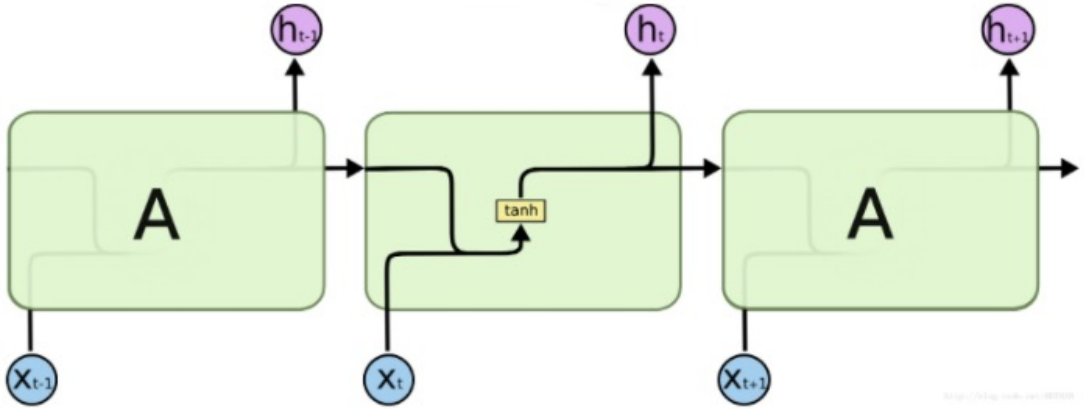


Gene Prediction with CNN

- If the task is to predict whether an ORF is a gene, just pool the last layer's hidden states (max, mean, concat etc.) and then connect to one or more dense layer before outout.
- This also deals with long range dependence naturally within the ORF.



Recurrence Neural Network

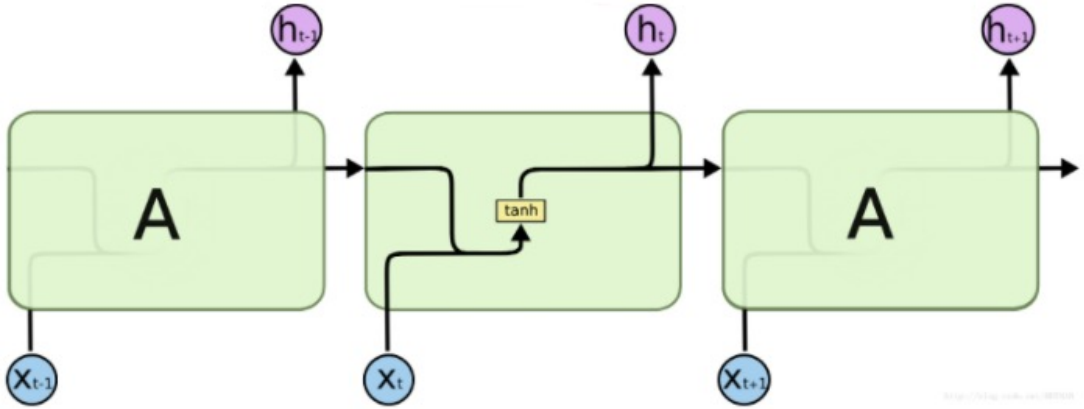


- RNN uses h_{t-1} (hidden state of time $t-1$) and x_t (the input embedding of time/position t) to compute h_t (the hidden state of time t).

$$h_t = g(h_{t-1}, x_t)$$

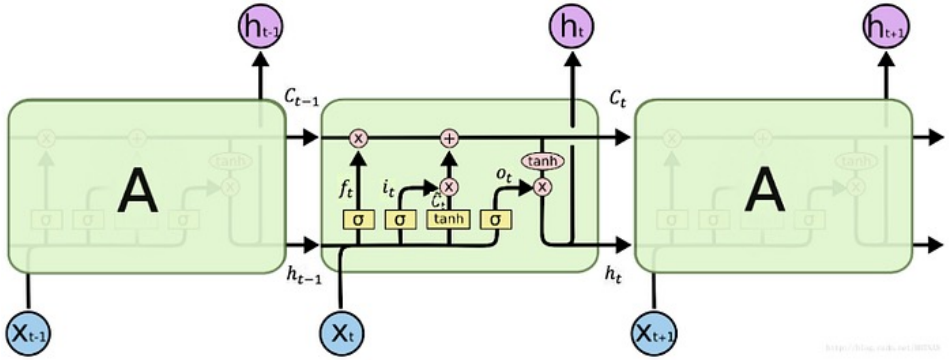
- The same parameters are shared over all positions.
- The figure shows its simplest form: a vanilla RNN.
- RNN deals with variable length sequences well
- This seems to be good for gene prediction/genome annotation…

Training of RNN

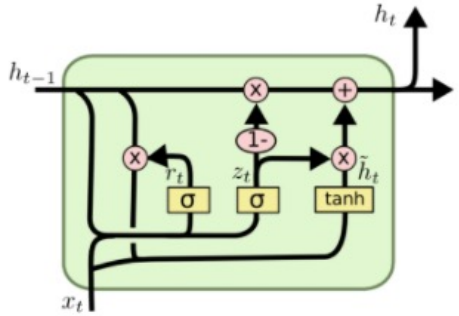


- RNN needs to be computed sequentially as h_t depends on h_{t-1}.
- During training, all intermediate activations of the forward computation need to be stored to facilitate the backpropagation of the gradient.
- This makes it harder to utilize the GPU's parallelism, and consumes much memory for extra long sequences.
- Additionally, the vanilla RNN's training has the vanishing gradient problem.

More Sophsiticated RNN



LSTM: Long Short-Term Memory



GRU: Gated Recurrent Unit

- LSTM and GRU are two more advanced forms of RNN, containing more wirings (more parameters) in each unit.
- They are built to enhance the learning of long-range dependencies.
- These are all standard modules in major deep learning frameworks, and can be used easily.

```
import torch.nn as nn

lstm = nn.LSTM(input_size=128, hidden_size=256, num_layers=2)
gru  = nn.GRU(input_size=128, hidden_size=256, num_layers=2)
```

Combine NN with HMM for Gene Prediction

- Lars Gabriel, Felix Becker, Katharina J Hoff, Mario Stanke, Tiberius: end-to-end deep learning with an HMM for gene prediction, *Bioinformatics*, Volume 40, Issue 12, December 2024, btae685, https://doi.org/10.1093/bioinformatics/btae685
- One of the best-performing ab initio gene prediction tool for eukaryotes.
- The method combines the advantage of NN and HMM. NN can learn complicated and long-range dependencies from raw data, but does not guarantee output structure is valid. HMM can control output structure, but does not learn complicated or long range dependencies.
- Suppose there are Q states of the HMM (corresponding to coding/noncoding etc.), it uses RNN to predict a state probability distribution at each time t. And treat this distribution as one of the missions of the HMM. Then it uses HMM to compute the final state transition path.

HMM Used by Tiberius

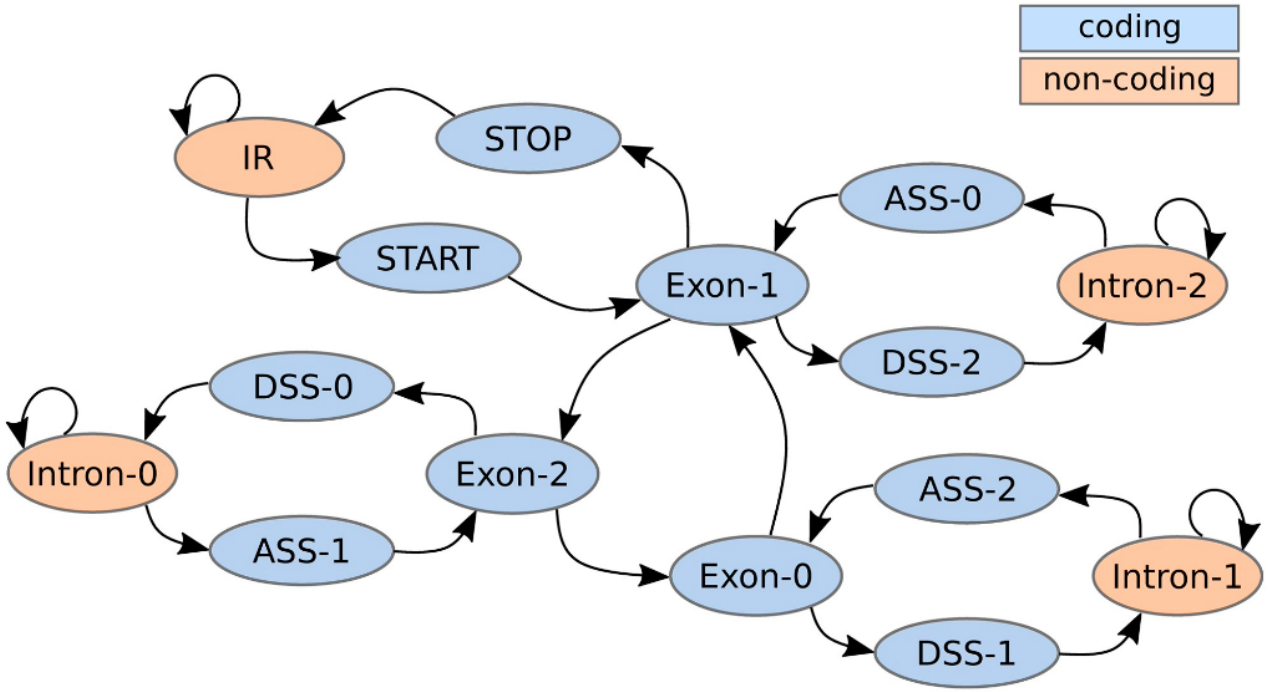
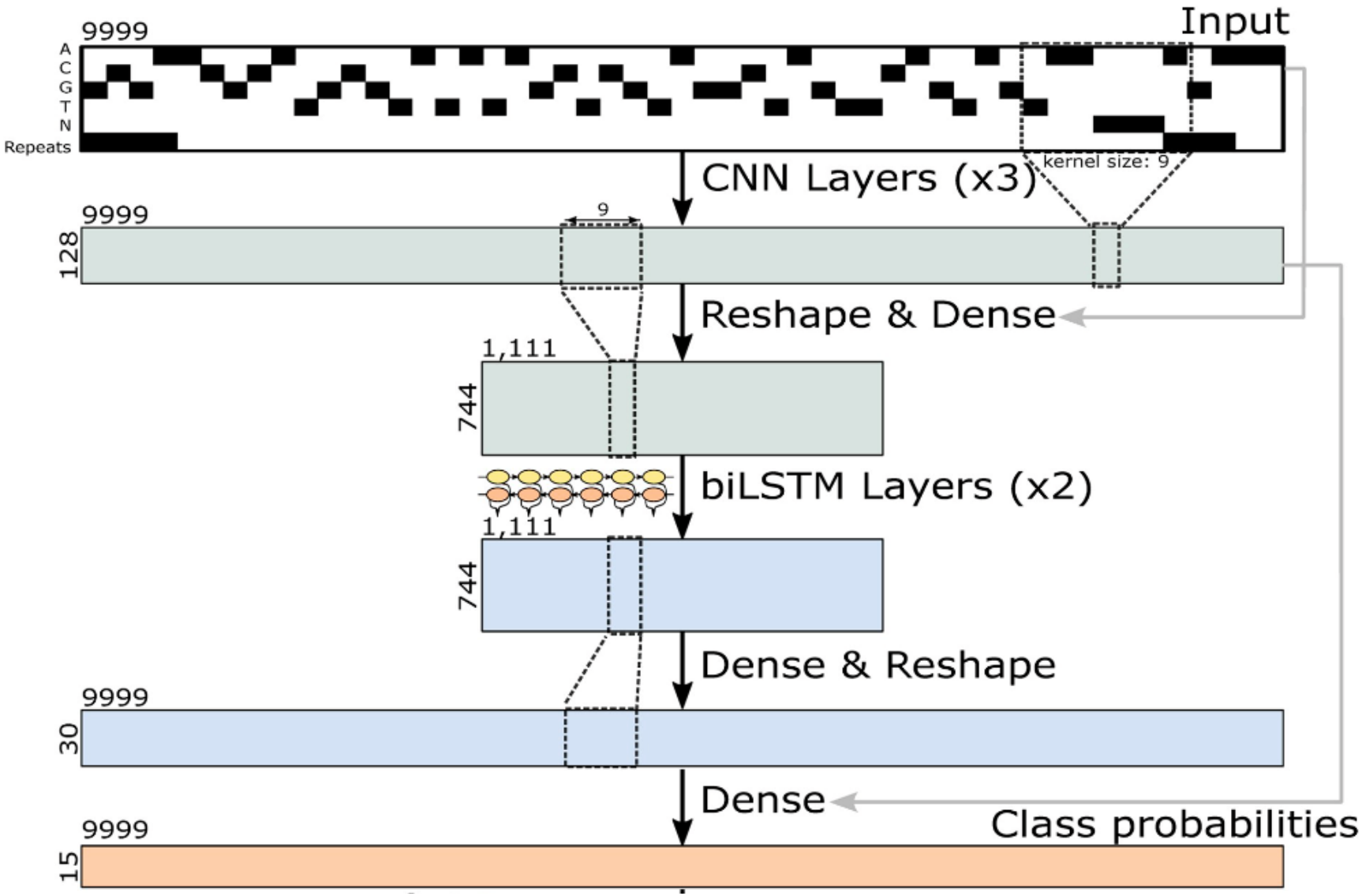


Figure 2. The states of the HMM used for inference with Tiberius and the transitions between them. The 11 coding-exon position states are subdivided by reading frame i: Exon-i represents non-border positions within an exon, while ASS-i (acceptor splice site) and DSS-i (donor splice site) states are the first and last position of an exon that starts and ends with reading frame *i*, respectively. The four non-coding position states are intergenic region (IR) or within an intron.

- In total 15 hidden states.
- Suppose each state emits 1 nucleotide.
- START, STOP, EXON-i,DSS-i and ASS-i all emit coding nucleotides.
- IR and Intro-i emits non-coding nucelotides.
- This ensures number of coding nucleotides on any path is a multiple of 3.

RNN Used by Tiberus



15 classes, corresponding to 15 hidden states

Combine NN with HMM for Gene Prediction

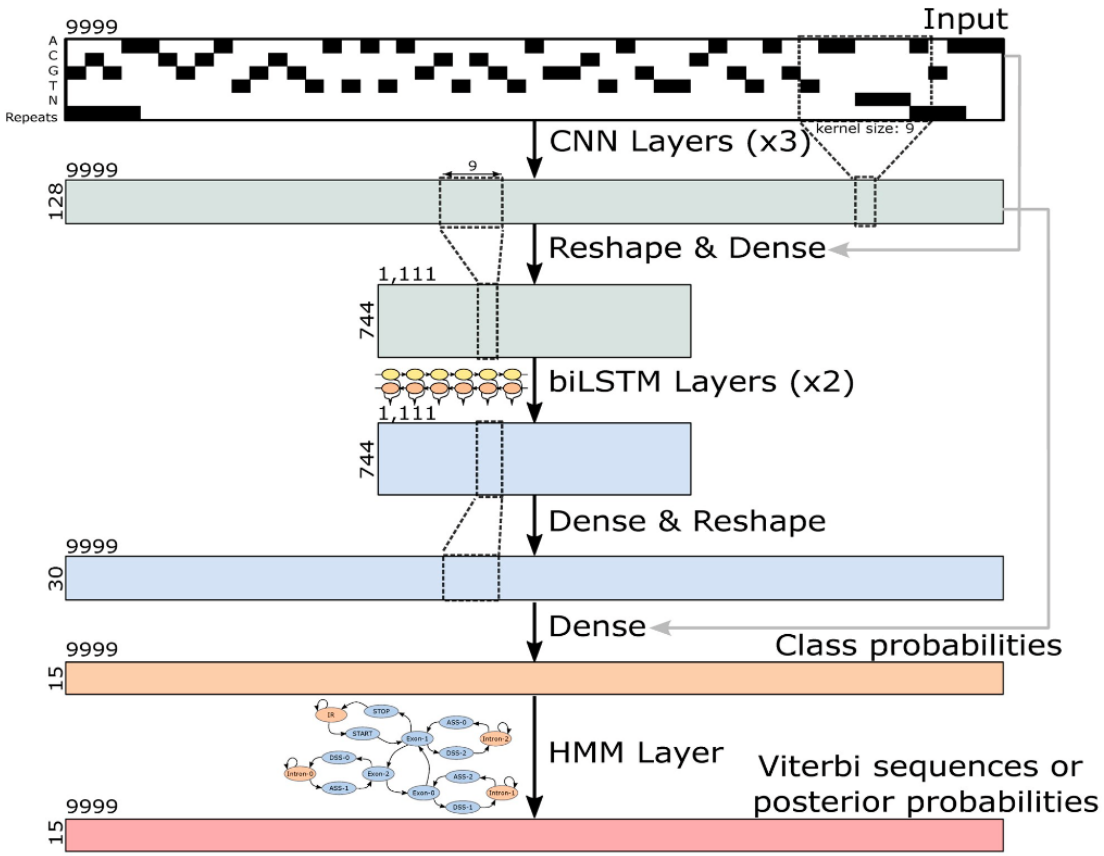


Figure 1. Illustration of the CNN-LSTM architecture of the Tiberius model for gene structure classification at each base position. The HMM layer computes posterior probabilities or complete gene structures (Viterbi sequences). The model has approximately 8 million trainable parameters, and it was trained with sequences of length $T = 9999$ and a length of $T = 500{,}004$ was used for inference.

- HMM emits three things concurrently at any time t:
  - 3-mer starting here
  - 3-mer ending here
  - y_t the class probabilities (length-Q vector) as predicted by the RNN
  - For each state q, emission probabilities of y_t is just $y_t[q]$, with adjustment by smoothing.
- Then the Viterbi algorithm can be used to find the path with highest joint probabilities.

Performance Comparison

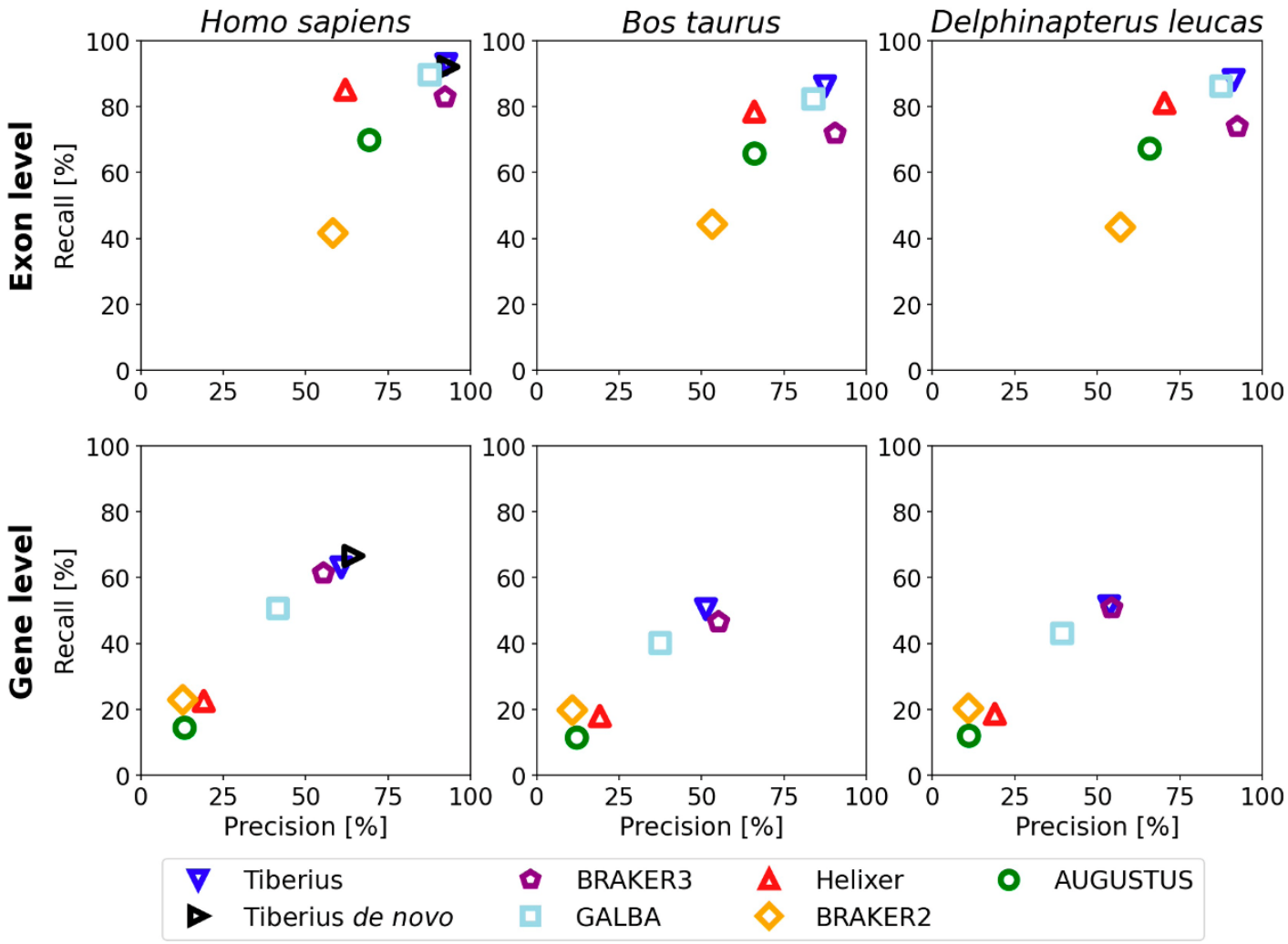


Figure 3. Gene and exon-level precision and recall for Tiberius, BRAKER3, GALBA, Helixer, BRAKER2, and AUGUSTUS. Tiberius, Helixer, and AUGUSTUS performed *ab initio* predictions while the other methods additionally incorporated extrinsic evidence: GALBA proteins from related species, BRAKER2 a large protein database, and BRAKER3 a large protein database and RNA-seq. For the human genome, Tiberius was also run *de novo*.

Summary

- HMM is a general model to predict some hidden states by examining emitted symbols.
- HMM can be used in gene prediction to harvest the codon bias and adjacent codon correlation.
- Gene prediction can use more information about the gene structure than codon bias.
- RNN is another commonly used deep learning architecture for biological sequence analysis.
- Deep learning and HMM can be combined together to make better gene prediction.