6.047/6.878 Lecture 08: HMMs II -Posterior Decoding and Learning

Charalampos Mavroforakis and Chidube Ezeozue (2012)
Thomas Willems (2011)
Amer Fejzic (2010)
Elham Azizi (2009)

December 13, 2012

Contents

List of Figures

1 Review of previous lecture

1.1 Introduction to Hidden Markov Models

In the last lecture, we familiarized ourselves with the concept of discrete-time Markov chains and Hidden Markov Models (HMMs). In particular, a Markov chain is a discrete random process that abides by the Markov property, i.e. that the probability of the next state depends only on the current state. To model how states change from step to step, the Markov chain uses a matrix of transition probabilities. In addition, it is characterized by a one-to-one correspondence between the states and observed symbols. More formally, a Markov chain is fully defined by the following variables:

- $\pi_i \in Q$, the state at the i^{th} step in a sequence of finite states Q of length N that can hold a value from a finite alphabet Σ of length K
- a_{jk}, the transition probability of moving from state j to state k, $P(\pi_i = k|\pi_{i-1} = j)$, for each j, k in Q
- $a_{0j} \in P$, the probability that the initial state will be j

Examples of Markov chains are abundant in everyday life. In the last lecture, we considered the canonical example of a weather system in which each state is either rain, snow, sun or clouds and the observables of the system correspond exactly to the underlying state. Suppose, however, that the weather results from some unknown underlying seasonal states. In this situation, the states are considered "hidden" and no longer share a one-to-one correspondence with the observables. These types of situations require a generalization of Markov chains known as Hidden Markov Models (HMMs).

Did You Know?
Markov Chains may be thought of as WYSIWYG - What You See Is What You Get

HMMs incorporate additional elements to model the disconnect between the observables of a system and the hidden states. For a sequence of length N, each observable state is instead replaced by a hidden state and a character emitted from that state. It is important to note that characters from each state are emitted according to a series of emission probabilities. More formally, the two additional descriptors of an HMM are:

- $x_i \in X$, the emission at the i^{th} step in a sequence of finite characters X of length N that can hold a character from a finite set of observation symbols $v_l \in V$
- $e_k(v_l) \in E$, the emission probability of emitting character v_l when the state is k, $P(x_i = v_l|\pi_i = k)$

In summary, an HMM is defined by the following variables:

- a_{jk}, $e_k(v_l)$, and a_{0j} that model the discrete random process
- π_i, the sequence of hidden states
- x_i, the sequence of observed emissions

1.2 Genomic Applications of HMMs

The figure below shows some genomic applications of HMMs

Application	Detection of GC-rich region	Detection of Conserved region	Detection of Protein coding exons	Detection of Protein coding conservation	Detection of Protein coding gene structures	Detection of chromatin states
Topology / Transitions	2 states, different nucleotide composition	2 states, difference conservation levels	2 states, different tri-nucleotide composition	2 states, different evolutionary signatures	~20 states, different composition / conservation, specific structure	40 states, different chromatin mark combinations
Hidden States / Annotation	GC-rich / AT-rich	Conserved/ non-Conserved	Coding (exon) / non-Coding (intron or intergenic)	Coding (exon) / non-Coding (intron or intergenic)	First / last / middle coding exon, UTRs, intron 1/2/3, intergenic, *(+,-) strand	Enhancer / Promoter / Transcribed / Repressed / Repetitive
Emissions / Observations	Nucleotides	Level of conservation	Triplets of nucleotides	64 x 64 matrix of codon substitution frequencies	Codons, nucleotides, splice sites, start/stop codons	Vector of chromatin mark frequencies

Figure 1: Genomic applications of HMMs

Niceties of some of the applications shown in figure 1 include:

- **Detection of Protein coding conservation**
  This is similar to the application of detecting protein coding exons because the emissions are also not nucleotides but different in the sense that, instead of emitting codons, substitution frequencies of the codons are emitted.

- **Detection of Protein coding gene structures**
  Here, it is important for different states to model first, last and middle exons because first exons go through a start codon and last exons go through a stop codon and that knowledge has to be encoded. This differs from the application of detecting protein coding exons because in the latter, the nature of the coding exons is unimportant.
  It is also important to differentiate between introns 1,2 and 3 so that the reading frame between one exon and the next exon can be remembered e.g. if one exon stops at the second codon position, the next one has to start at the third codon position. Therefore, the additional intron states encode the codon position.

- **Detection of chromatin states**
  Chromatin state models are dynamic and vary from cell type to cell type so every cell type will have its own annotation. They will be discussed in fuller detail in the genomics lecture including strategies for stacking/concatenating cell types.

1.3 Viterbi decoding

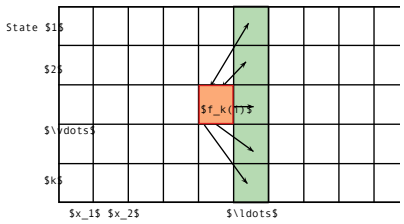
Previously, we demonstrated that when given a full HMM (Q, A, X, E, P), the likelihood that the discrete random process produced the provided series of hidden states and emissions is given by:

$$P(x_1, \ldots, x_N, \pi_1, \ldots, \pi_N) = a_{0\pi_i} \prod_i e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}} \tag{1}$$

This corresponds to the total joint probability, $P(x, \pi)$. Usually, however, the hidden states are not given and must be inferred. One solution to this *decoding* problem is known as the **Viterbi decoding** algorithm. Running in $O(K^2N)$ time and $O(KN)$ space, where K is the number of states, this algorithm determines the sequence of hidden states (the path π^*) that maximizes the joint probability of the observables and states, i.e. $P(x, \pi)$. Essentially, this algorithm defines $V_k(i)$ to be the probability of the most likely path through state $\pi_i = k$, and it recursively computes $V_k(i) = e_k(x_i) \times \max_j(V_j(i-1)a_{jk})$

1.4 Forward Algorithm

Another problem we touched was that, instead of computing the probability of a single path of hidden state emitting the observed sequence, we may want to calculate the probability of the sequence being produced by all possible paths. In order to do that, we proposed the **Forward algorithm**, which is described in Figure 2



Input: $x = x_1 \ldots x_N$

Initialization:
$f_0(0) = 1, f_k(0) = 0, \quad$ for all $k > 0$

Iteration:
$f_k(i) = e_k(x_i) \times \sum_j a_{jk} f_j(i-1)$

Termination:
$P(x, \pi^*) = \sum_k f_k(N)$

Figure 2: The Forward Algorithm

The forward algorithm first calculates the joint probability of observing the first t emitted characters and being in state k at time t. More formally,

$$f_k(t) = P(\pi_t = k, x_1, \ldots, x_t) \tag{2}$$

Given that the number of paths is exponential in t, dynamic programming must be employed to solve this problem. We can develop a simple recursion for the forward algorithm by employing the Markov property as follows:

$$f_k(t) = \sum_l P(x_1, \ldots, x_t, \pi_t = k, \pi_{t-1} = l) = \sum_l P(x_1, \ldots, x_{t-1}, \pi_{t-1} = l) * P(x_t, \pi_t | \pi_{t-1}) \tag{3}$$

Recognizing that the first term corresponds to $f_l(t-1)$ and that the second term can be expressed in terms of transition and emission probabilities, this leads to the final recursion:

$$f_k(t) = e_k(x_t) \sum_l f_l(t-1) * a_{lk} \tag{4}$$

Intuitively, one can understand this recursion as follows: Any path that is in state k at time t must have come from a path that was in state l at time $t-1$. The contribution of each of these sets of paths is then weighted by the cost of transitioning from state l to state k. It is also interesting to note that the Viterbi algorithm and forward algorithm largely share the same recursion. The only difference between the two algorithms lies in the fact that the Viterbi algorithm uses the maximum function whereas the forward algorithm uses a sum.

We can now compute $f_k(t)$ based on a weighted sum of all the forward algorithm results tabulated during the previous time step. As shown in Figure 2, the forward algorithm can be easily implemented in a KxN dynamic programming table. The first column of the table is initialized according to the initial state probabilities a_{i0} and the algorithm then proceeds to process each column from left to right. Because there are KN entries and each entry examines a total of K other entries, this leads to $O(K^2N)$ time complexity and $O(KN)$ space.

In order now to calculate the total probability of a sequence of observed characters under the current HMM, we need to express this probability in terms of the forward algorithm gives in the following way:

$$P(x_1, \ldots, x_n) = \sum_l P(x_1, \ldots, x_n, \pi_N = l) = \sum_l f_l(N) \tag{5}$$

Hence, the sum of the elements in the last column of the dynamic programming table provides the total probability of an observed sequence of characters. In practice, given a sufficiently long sequence of emitted characters, the forward probabilities decrease very rapidly. To circumvent issues associated with storing small floating point numbers, logs-probabilities are used in the calculations instead of the probabilities themselves. This alteration requires a slight adjustment to the algorithm and the use of a Taylor series expansion for the exponential function.

1.5 This lecture

- This lecture will discuss **posterior decoding**, an algorithm which again will infer the hidden state sequence π that maximizes a different metric. In particular, it finds the most likely state at every position over all possible paths and does so using both the **forward** and **backward** algorithm.
- Afterwards, we will show how to encode "memory" in a Markov chain by adding more states to search a genome for dinucleotide CpG islands.
- We will then discuss how to use Maximum Likelihood parameter estimation for supervised learning with a labelled dataset
- We will also briefly see how to use Viterbi learning for unsupervised estimation of the parameters of an unlabelled dataset
- Finally, we will learn how to use Expectation Maximization (EM) for unsupervised estimation of parameters of an unlabelled dataset where the specific algorithm for HMMs is known as the Baum-Welch algorithm.

2 Posterior Decoding

2.1 Motivation

Although the **Viterbi decoding** algorithm provides one means of estimating the hidden states underlying a sequence of observed characters, another valid means of inference is provided by **posterior decoding**.

Posterior decoding provides the most likely state at any point in time. To gain some intuition for posterior decoding, let's see how it applies to the situation in which a dishonest casino alternates between a fair and loaded die. Suppose we enter the casino knowing that the unfair die is used 60 percent of the time. With this knowledge and no die rolls, our best guess for the current die is obviously the loaded one. After one roll, the probability that the loaded die was used is given by

$$P(die = loaded|roll = k) = \frac{P(die = loaded) * P(roll = k|die = loaded)}{P(roll = k)}. \tag{6}$$

If we instead observed a sequence of N die rolls, how do perform a similar sort of inference? By allowing information to flow between the N rolls and influence the probability of each state, posterior decoding is a natural extension of the above inference to a sequence of arbitrary length. More formally, instead of identifying a single path of maximum likelihood, posterior decoding considers the probability of any path lying in state k at time t given all of the observed characters, i.e. $P(\pi_t = k|x_1, \ldots, x_n)$. The state that maximizes this probability for a given time is then considered as the most likely state at that point.

It is important to note that in addition to information flowing forward to determine the most likely state at a point, information may also flow backward from the end of the sequence to that state to augment or reduce the likelihood of each state at that point. This is partly a natural consequence of the reversibility of Bayes' rule. To elucidate this, imagine the casino example again. As stated earlier, without observing any rolls, the $state_0$ is most likely to be unfair. If the first roll is a 6, our belief that $state_1$ is unfair is reinforced (if rolling sixes is more likely in an unfair die). If a 6 is rolled again, information flow backwards from the second die roll and reinforces our $state_1$ belief of an unfair die even more. The more rolls we have, the more information that flows backwards and reinforces or contrasts our beliefs about the state thus illustrating the way information flows backward and forward to affect our belief about the states in Posterior Decoding.

Using some elementary manipulations, we can rearrange this probability into the following form using Bayes' rule:

$$\pi_t^* = argmax_{\pi_t} P(\pi_t = k|x_1, \ldots, x_n) = argmax_{\pi_t} \frac{P(\pi_t = k, x_1, \ldots, x_n)}{P(x_1, \ldots, x_n)} \tag{7}$$

Because $P(x)$ is a constant, we can neglect it when maximizing the function. Therefore,

$$\pi_t^* = argmax_{\pi_t} P(\pi_t = k, x_1, \ldots, x_t) * P(x_{t+1}, \ldots, x_n|\pi_t = k, x_1, \ldots, x_t) \tag{8}$$

Using the Markov property, we can simply write this expression as follows:

$$\pi_t^* = argmax_{\pi_t} P(\pi_t = k, x_1, \ldots, x_t) * P(x_{t+1}, \ldots, x_n|\pi_t = k) = argmax_{\pi_t} f_k(t) * b_k(t) \tag{9}$$

Here, we've defined $f_k(t) = P(\pi_t = k, x_1, \ldots, x_t)$ and $b_k(t) = P(x_{t+1}, \ldots, x_n|\pi_t = k)$. As we will shortly see, these parameters are calculated using the **forward algorithm** and the **backward algorithm** respectively. To solve the posterior decoding problem, we merely need to solve each of these subproblems. The forward algorithm has been illustrated in the previous chapter and in the review at the start of this chapter and the backward algorithm will be explained in the next section.

Before explaining the backward algorithm, one may ask if Posterior Decoding can provide a path like the Viterbi algorithm would. The answer is yes; since Posterior Decoding can provide the most likely state for every point, assembling these states provides the path. Later on, we would compare the pros and cons of the paths generated using Viterbi algorithm and Posterior Decoding.

2.2 Backward Algorithm

As previously described, the backward algorithm is used to calculate the following probability:

$$b_k(t) = P(x_{t+1}, \ldots, x_n|\pi_t = k) \tag{10}$$

We can begin to develop a recursion n by expanding into the following form:

$$b_k(t) = \sum_l P(x_{t+1}, \ldots, x_n, \pi_{t+1} = l | \pi_t = k) \tag{11}$$

From the Markov property, we then obtain:

$$b_k(t) = \sum_l P(x_{t+2}, \ldots, x_n | \pi_{t+1} = l) * P(\pi_{t+1} = l | \pi_t = k) * P(x_{t+1} | \pi_{t+1} = k) \tag{12}$$

The first term merely corresponds to $b_l(t+1)$. Expressing in terms of emission and transition probabilities gives the final recursion:

$$b_k(t) = \sum_l b_l(i+1) * a_{kl} * e_l(x_{t+1}) \tag{13}$$

Comparison of the forward and backward recursions leads to some interesting insight. Whereas the forward algorithm uses the results at $t-1$ to calculate the result for t, the backward algorithm uses the results from $t+1$, leading naturally to their respective names. Another significant difference lies in the emission probabilities; while the emissions for the forward algorithm occur from the current state and can therefore be excluded from the summation, the emissions for the backward algorithm occur at time $t+1$ and therefore must be included within the summation.

Given their similarities, it is not surprising that the backward algorithm is also implemented using a KxN dynamic programming table. The algorithm, as depicted in Figure 3, begins by initializing the rightmost column of the table to unity. Proceeding from right to left, each column is then calculated by taking a weighted sum of the values in the column to the right according to the recursion outlined above. After calculating the leftmost column, all of the backward probabilities have been calculated and the algorithm terminates. Because there are KN entries and each entry examines a total of K other entries, this leads to $O(K^2N)$ time complexity and $O(KN)$ space, bounds identical to those of the forward algorithm.

Just as P(X) was calculated by summing the rightmost column of the forward algorithm's DP table, P(X) can also be calculated from the sum of the leftmost column of the backward algorithm's DP table. Therefore, these methods are virtually interchangeable for this particular calculation.

Figure 3: The Backward Algorithm

Did You Know?
Note that even when executing the backward algorithm, forward transition probabilities are used i.e if moving in the backward direction involves a transition from state B $\rightarrow$ A, the probability of transitioning from state A $\rightarrow$ B is used

2.3 The Big Picture

Given that we can calculate both $f_k(t)$ and $b_k(t)$ in $\theta(K^2N)$ time and $\theta(KN)$ space for all $t = 1 \dots n$, we can use posterior decoding to determine the most likely state π_t^* for $t = 1 \dots n$. The relevant expression is given by

$$\pi_t^* = argmax_k P(\pi_i = k|x) = \frac{f_k(i) * b_k(i)}{P(x)} \tag{14}$$

With two methods (Viterbi and posterior) to decode, which is more appropriate? When trying to classify each hidden state, the Posterior decoding method is more informative because it takes into account all possible paths when determining the most likely state. In contrast, the Viterbi method only takes into account one path, which may end up representing a minimal fraction of the total probability. At the same time, however, posterior decoding may give an invalid sequence of states! For example, the states identified at time points t and $t + 1$ might have zero transition probability between them. As a result, selecting a decoding method is highly dependent on the application of interest.

FAQ

Q: What does it imply when the Viterbi algorithm and Posterior decoding disagree on the path?

A: In a sense, it is simply a reminder that our model of the world can never be correct. In the genomic context, it might be a result of some 'funky' biology; alternative splicing, for instance. In some cases, the Viterbi algorithm will be close to the Posterior decoding while in some others they may disagree.

3 Encoding Memory in a HMM: Detection of CpG islands

CpG islands are defined as regions within a genome that are enriched with pairs of C and G nucleotides on the same strand. Typically, when this dinucleotide is present within a genome, it becomes methylated and cytosine mutates into a thymine, thereby rendering CpG dinucleotides relatively rare. Because the methylation can occur on either strand, CpG's usually mutate into a TpG or a CpA. However, when situated within an active promoter, methylation is suppressed and CpG dinucleotides are able to persist. Similarly, CpGs in regions important to cell function are conserved due to evolutionary pressure. As a result, detecting CpG islands can highlight promoter regions and other important regions within a genome.

Did You Know?
CpG stands for [C]ytosine - [p]hosphate backbone - [G]uanine. The 'p' implies that we are referring to the same strand of the double helix

Given their biological significance, CpG islands are prime candidates for modelling. Initially, one may attempt to identify these islands by scanning the genome for fixed intervals rich in GC. This approach's efficacy is undermined by the selection of an appropriate window size; while too small of a window may not capture all of a particular CpG island, too large of a window would result in the dilution of the CpG island's effects. Examining the genome on a per codon basis also leads to difficulties because CpG pairs do not necessarily lie within one reading frame. Instead, HMMs are much better suited to modelling this scenario because, as we shall shortly see in unsupervised learning, HMMs can adapt their underlying parameters to maximize their likelihood.

Not all HMMs, however, are well suited to this particular task. An HMM model that only considers the single nucleotide frequencies of C's and G's will fail to capture the nature of CpG islands. Consider one such HMM with the two following hidden states :

- '+' state representing CpG islands
- '-' state: representing non-islands

Each of these two states then emits A, C, G and T bases with a certain probability. Although the CpG islands in this model can be enriched with C's and G's by increasing their respective emission probabilities, this model will fail to capture the fact that the C's and G's predominantly occur in pairs.

Because of the Markov property that governs HMM's, the only information available at each time step must be contained within the current state. Therefore, to encode memory within a Markov chain, we need to augment the state space. To do so, the individual '+' and '-' states can be replaced with 4 '+' states and 4 '-' states: A+, C+, G+, T+, A-, C-, G-, T- (Figure 4). Specifically, there are 2 ways to model this, and this choice will result in different emission probabilities:

- One model suggests that the state A+, for instance, implies that we are currently in a CpG island and the *previous* character was an A. The emission probabilities here will carry most of the information and the transitions will be fairly degenerate.
- Another model suggests that the state A+, for instance, implies that we are currently in a CpG island and the *current* character is an A. The emission probability here will be 1 for A and 0 for all other letters and the transition probabilities will bear most of the information in the model and the emissions will be fairly degenerate. We will assume this model from now on.

Did You Know?

The number of transitions is the square of the number of states. This gives a rough idea of how increasing HMM "memory" (and hence states) scale.

The memory of this system derives from the fact that each state can only emit one character and therefore "remembers" its emitted character. Furthermore, the dinucleotide nature of the CpG islands is incorporated within the transition matrices. In particular, the transition frequency from C_+ to G_+ states is significantly higher than from C_- to a G_- states, demonstrating that these pairs occur more often within the islands.

FAQ

Q: Since each state emits only one character, can we then say this reduces to a Markov Chain instead of a HMM?

A: No. Even though the emissions indicate the letter of the hidden state, they do not indicate if the state is a CpG island or not.

FAQ

Q: How do we incorporate our knowledge about the system while training HMM models eg. some emission probabilities of 0 in the CpG island detection case?

A: We could either force our knowledge on the model by setting some parameters and leaving others to vary or we could let the HMM loose on the model and let it discover those relationships. As a matter of fact, there are even methods that simplify the model by forcing a subset of parameters to be 0 but allowing the HMM to choose which subset.

Given the above framework, we can use posterior decoding to analyze each base within a genome and determine whether it is most likely a constituent of a CpG island or not. But having constructed the expanded HMM model, how can we verify that it is in fact better than the single nucleotide model? We previously demonstrated that the forward or backward algorithm can be used to calculate $P(x)$ for a given

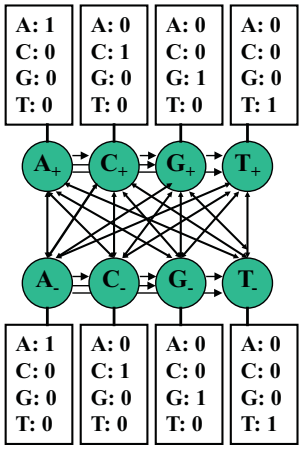


Figure 4: HMM for CpG Islands

model. If the likelihood of our dataset is higher given the second model than the first model, it most likely captures the underlying behavior more effectively.

FAQ

Q: Are there other ways to encode the memory for CpG island detection?

A: Other ideas that may be experimented with include

- Emit dinucleotides and figure out a way to deal with overlap.
- Add a special state that goes from C to G.

4 Learning

We saw how to score and decode an HMM-generated sequence in two different ways. However, these methods assumed that we already knew the emission and transition probabilities. Fortunately, the HMM framework enables the learning of these probabilities when provided a set of training data.

When the training data is labelled, estimation of the probabilities is a form of supervised learning. One such instance would occur if we were given a DNA sequence of one million nucleotides in which the CpG islands had all been experimentally annotated and were asked to use this to estimate our model parameters

In contrast, when the training data is unlabelled, the estimation problem is a form of unsupervised learning. Continuing with the CpG island example, this situation would occur if the provided DNA sequence contained no island annotation whatsoever and we needed to both estimate model parameters and identify the islands.

4.1 Supervised Learning

When provided with labelled data, estimating model parameters is actually trivial. Suppose that you are given a labelled sequence $x_1, \ldots, x_N$ as well as the true hidden state sequence $\pi_1, \ldots, \pi_N$. Intuitively, one might expect that the probabilities that maximize the data's likelihood are the actual probabilities that one observes within the data. This is indeed the case and can be formalized by defining A_{kl} to be the number of times hidden state k transitions to l and $E_k(b)$ to be the number of times b is emitted from hidden state k.

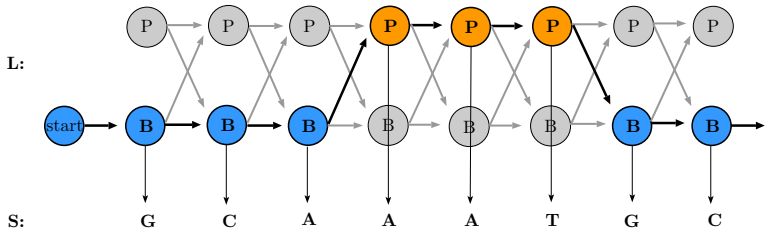


Figure 5: Supervised Learning of CpG islands

The parameters θ that maximize $P(x|\theta)$ are simply obtained by counting as follows:

$$a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}} \tag{15a}$$

$$e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)} \tag{15b}$$

One example training set is shown in Figure 5. In this example, it is obvious that the probability of transitioning from B to P is $\frac{1}{3+1} = \frac{1}{4}$ (there are 3 B to B transitions and 1 B to P transitions) and the probability of emitting a G from the B state is $\frac{2}{2+2+1} = \frac{2}{5}$ (there are 2 G's emitted from the B state, 2 C's and 1 A)

Notice, however, that in the above example the emission probability of character T from state B is 0 because no such emissions were encountered in the training set. A zero probability, either for transitioning or emitting, is particularly problematic because it leads to an infinite log penalty. In reality, however, the zero probability may merely have arisen due to over-fitting or a small sample size. To rectify this issue and maintain flexibility within our model, we use pseudocounts instead of absolute counts.

- $A^*_{kl} = A_{kl} + r_{kl}$
- $E_k(b)^* = E_k(b) + r_k(b)$

Larger pseudocount parameters correspond to a strong prior belief whereas small pseudocount parameters ($r << 1$) are used to avoid 0 probabilities.

4.2 Unsupervised Learning

Unsupervised learning involves estimating parameters based on unlabelled data. An iterative approach is typically most suitable to solving these types of problems. Suppose we have some sort of prior belief about what each emission and transition probability should be. Given these parameters, we can use a decoding method to infer the hidden states underlying the provided data sequence. Using this particular decoding parse, we can then re-estimate the transition and emission counts and probabilities in a process similar to that used for supervised learning. If we repeat this procedure until the improvement in the data's likelihood remains relatively stable, the data sequence should ultimately drive the parameters to their appropriate values.

FAQ

Q: Why does unsupervised learning even work? Or is it magic?

A: Unsupervised learning works because we have the sequence (input data) and this guides every step of the iteration; to go from a labelled sequence to a set of parameters, the later are guided by the input and its annotation, while to annotate the input data, the parameters and the sequence guide the procedure.

For HMM's in particular, two main methods of unsupervised learning are useful.

4.2.1 Expectation Maximization using Viterbi training

The first method, **Viterbi training**, is relatively simple but not entirely rigorous. After picking some initial best-guess model parameters, it proceeds as follows:

E step: Perform Viterbi decoding to find $\pi^\star$
Calculate A^*_{kl}, $E_k(b)^*$ using pseudocounts based on the transitions and emissions observed in $\pi^\star$ states given the latest parameters and observed sequence (Expectation step)

M step: Calculate the new parameters a_{kl}, $e_k(b)$ using the simple counting formalism in supervised learning (Maximization step)

Iteration: Repeat the E and M steps until the likelihood $P(x|\theta)$ converges

Although Viterbi training converges rapidly, its resulting parameter estimations are usually inferior to those of the Baum-Welch Algorithm. This result stems from the fact that Viterbi training only considers the most probable hidden path instead of the collection of all possible hidden paths.

4.2.2 Expectation Maximization: The Baum-Welch Algorithm

The more rigorous approach to unsupervised learning involves an application of Expectation Maximization to HMM's. In general, EM proceeds in the following manner:

Init: Initialize the parameters to some best-guess state

E step: Estimate the expected probability of hidden states given the latest parameters and observed sequence (Expectation step)

M step: Choose new maximum likelihood parameters using the probability distribution of hidden states (Maximization step)

Iteration: Repeat the E and M steps until the likelihood of the data given the parameters converges

The power of EM lies in the fact that $P(x|\theta)$ is guaranteed to increase with each iteration of the algorithm. Therefore, when this probability converges, a local maximum has been reached. As a result, if we utilize a variety of initialization states, we will most likely be able to identify the global maximum, i.e. the best parameters θ. The Baum-Welch algorithm generalizes EM to HMM's. In particular, it uses the forward and backward algorithms to calculate $P(x|\theta)$ and to estimate A_{kl} and $E_k(b)$. The algorithm proceeds as follows:

Initialization

1. Initialize the parameters to some best-guess state

Iteration

1. Run the forward algorithm
2. Run the backward algorithm
3. Calculate the new log-likelihood $P(x|\theta)$
4. Calculate A_{kl} and $E_k(b)$
5. Calculate a_{kl} and $e_k(b)$ using the pseudocount formulas
6. Repeat until $P(x|\theta)$ converges

Previously, we discussed how to compute $P(x|\theta)$ using either the forward or backward algorithm's final results. But how do we estimate A_{kl} and $E_k(b)$? Let's consider the expected number of transitions from state k to state l given a current set of parameters θ. We can express this expectation as

$$A_{kl} = \sum_t P(\pi_t = k, \pi_{t+1} = l|x, \theta) = \sum_t \frac{P(\pi_t = k, \pi_{t+1} = l, x|\theta)}{P(x|\theta)} \tag{16}$$

Exploiting the Markov property and the definitions of the emission and transition probabilities leads to the following derivation:

$$A_{kl} = \sum_t \frac{P(x_1 \dots x_t, \pi_t = k, \pi_{t+1} = l, x_{t+1} \dots x_N | \theta)}{P(x|\theta)} \tag{17a}$$

$$= \sum_t \frac{P(x_1 \dots x_t, \pi_t = k) * P(\pi_{t+1} = l, x_{t+1} \dots x_N | \pi_t, \theta)}{P(x|\theta)} \tag{17b}$$

$$= \sum_t \frac{f_k(t) * P(\pi_{t+1} = l | \pi_t = k) * P(x_{t+1} | \pi_{t+1} = l) * P(x_{t+2} \dots x_N | \pi_{t+1} = l, \theta)}{P(x|\theta)} \tag{17c}$$

$$\Rightarrow A_{kl} = \sum_t \frac{f_k(t) * a_{kl} * e_l(x_{t+1}) * b_l(t+1)}{P(x|\theta)} \tag{17d}$$

A similar derivation leads to the following expression for $E_k(b)$:

$$E_k(b) = \sum_{i|x_i=b} \frac{f_k(t) * b_k(t)}{P(x|\theta)} \tag{18}$$

Therefore, by running the forward and backward algorithms, we have all of the information necessary to calculate $P(x|\theta)$ and to update the emission and transition probabilities during each iteration. Because these updates are constant time operations once $P(x|\theta)$, $f_k(t)$ and $b_k(t)$ have been computed, the total time complexity for this version of unsupervised learning is $\theta(K^2NS)$, where S is the total number of iterations.

FAQ

Q: How do you encode your prior beliefs when learning with Baum-Welch?

A: Those prior beliefs are encoded in the initializations of the forward and backward algorithms

5 Current Research Directions

- HMM's have been used extensively in various fields of computational biology. One of the first such applications was in a gene-finding algorithm known as GENSCAN written by Chris Burge and Samuel Karlin [1]. Because the geometric length distribution of HMM's does not model exonic regions well, Burge et. al used an adaptation of HMM's known as hidden semi-Markov models (HSMM's). These types of models differ in that whenever a hidden state is reached, the length of duration of that state (d_i) is chosen from a distribution and the state then emits exactly d_i characters. The transition from this hidden state to the next is then analogous to the HMM procedure except that $a_{kk} = 0$ for all k, thereby preventing self-transitioning. Many of the same algorithms that were previously developed for HMM's can be modified for HSMM's. Although the details won't be discussed here, the forward and backward algorithms can be modified to run in $O(K^2N^3)$ time, where N is the number of observed characters. This time complexity assumes that there is no upper bound on the length of a state's duration, but imposing such a bound reduces the complexity to $O(K^2ND^2)$, where D is the maximum possible duration of a state.

  The basic state diagram underlying Burge's model is depicted in Figure 6. The included diagram only lists the states on the forward strand of DNA, but in reality a mirror image of these states is also included for the reverse strand, resulting in a total of 27 hidden states. As the diagram illustrates, the model incorporates many of the major functional units of genes, including exons, introns, promoters, UTR's and poly-A tails. In addition, three different intronic and exonic states are used to ensure that the total length of all exons in a gene is a multiple of three. Similar to the CpG island example, this expanded state-space enabled the encoding of memory within the model.

- An interesting subject that may be explored also concerns the agreement of Viterbi and Posterior decoding paths; not just for CpG island detection but even for chromatin state detection. One may look at multiple paths by sampling, asking questions such as:
  - What is the maximum a posteriori vs viterbi path? Where do they differ?
  - Can complete but maximally disjoint (from Viterbi) paths be found?

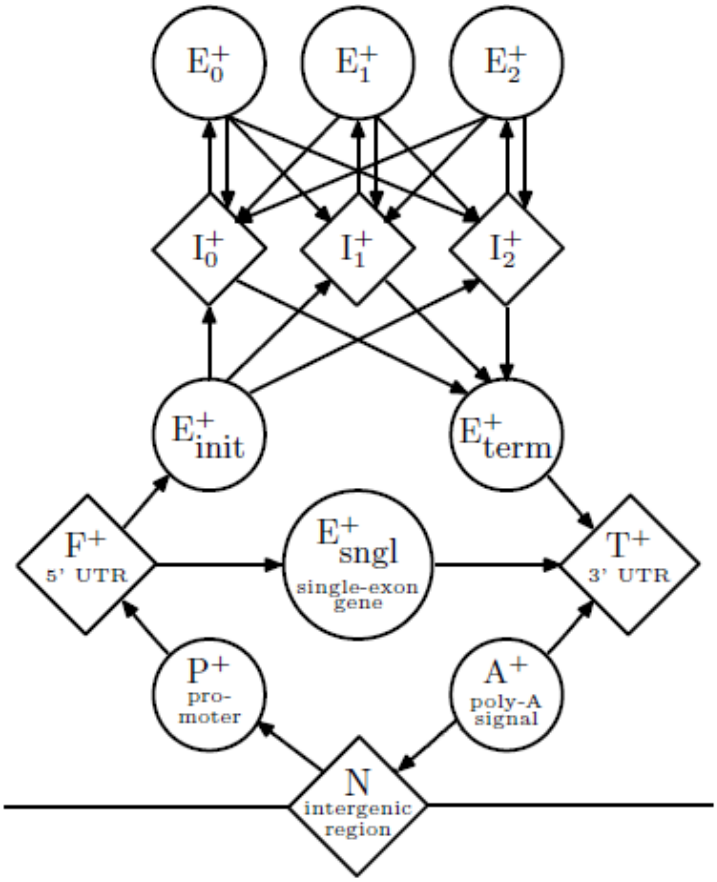


Figure 6: State Space Diagram used in GENSCAN

6 Further Reading

7 Tools and Techniques

8 What Have We Learned?

Using the basic computational framework provided by Hidden Markov Models, we've learned how to infer the most likely set of hidden states underlying a sequence of observed characters. In particular, a combination of the forward and backward algorithms enabled one form of this inference, i.e. posterior decoding, in $O(KN^2)$ time. We also learned how either unsupervised or supervised learning can be used to identify the best parameters for an HMM when provided with an unlabelled or labelled dataset. The combination of these decoding and parameter estimation methods enable the application of HMM's to a wide variety of problems in computational biology, of which CpG island and gene identification form a small subset. Given the flexibility and analytical power provided by HMM's, these methods will play an important role in computational biology for the foreseeable future.

References

[1] Christopher B Burge and Samuel Karlin. Finding the genes in genomic dna. *Current Opinion in Structural Biology*, 8(3):346 – 354, 1998.