1 Introduction

In the last lecture we got familiar with the concept of discrete-time Markov chains and Hidden Markov Models (HMMs). A Markov chain is a discrete random process that abides by the Markov property, that the probability of the next state depends only on the current state and not the past. The Markov chain models how a state changes from step to step using transition probabilities. Therefore, a Markov Model (MM) is fully defined as:

- $\pi_i \in Q$, 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 $\Sigma$ of length $K$
- $a_{jk} \in A$, the state 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 $j$

$A$ is the stochastic matrix. The probabilities of the leaving the state must sum to one, $\sum_k a_{jk} = 1$, $\forall j$. The value a state can hold for example in the Dishonest Casino problem would be $\pi_i = \{\text{"fair die"}, \text{"loaded die"}\}$, or $\pi_i = \{\text{"background"}, \text{"promoter region"}\}$ for the GC-rich finding problem. The state transition probabilities describe the chance of a state “fair die” to change to “loaded die” versus staying “fair die”. This is were one models the likelihood of the switching to occur. Here the states are completely “visible”, as if there was a marker on the fair or loaded die and you can see it, or the GC-rich areas were already labeled from the background. Realistically, one only sees the number that the die shows or the A, T, G, C nucleotide sequence one observes from a gene. As a result, the value of the state is not seen and is therefore considered “hidden”. This led us to the Hidden Markov Models, a further generalization of the Markov Model.

An HMM adds the mechanics on top of the MM to model the discrete random process of the “observations” that are coupled with the states. For an $N$ long sequence, there are now also $N$ number of observations. To model this, each state emits a character from a given alphabet with a certain probability and the emitted characters are observed. Therefore, the two additional descriptors of an HMM on top of the MM as listed above is:

- $x_i \in X$, 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 $k$, $P(x_i = v_l | \pi_i = k)$

The emission probabilities must also sum to one, $\sum_l e_k(v_l) = 1$, $\forall k$. There are three things to consider about the HMM:

- $a_{jk}, e_k(v_l)$, and $a_{0j}$ that model the discrete random process
- $\pi_i$, the sequence of hidden states
- $x_i$, the sequence of emissions that are observed
Previously we have showed when given the full HMM \((Q, A, X, E, P)\) what is the likelihood that the discrete random process produced the given hidden states and emissions.

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

(1)

This is the total joint probability \(P(x, \pi)\). Usually the hidden states are not given and so we then asked the decoding problem, given the partial HMM \((A, X, E, P)\), infer the hidden states sequence \(Q\) that maximizes the total joint probability. This lead us to the Viterbi decoding algorithm. The algorithm leveraged the use of the principle of optimality and runs in \(O(K^2N)\) time and \(O(KN)\) space, where \(K\) is the number of states.

This lecture will discuss Posterior Decoding, which again will infer the hidden states sequence \(X\) that maximizes some other metric. It finds the most likely state at every position over all possible paths. The algorithm relies on both the forward and backward algorithm. The forward algorithm computes the total probability of a given emission sequence being generated by a particular HMM over all possible state paths that could have generated it, and was discussed in last lecture. The backward algorithm is closely related. Afterwards we will show how to encode “memory” in a Markov chain such as the search for di-nucleotide CpG islands. Then we will discuss learning by means of Maximum Likelihood, EM, and Viterbi learning and we’ll discuss different learning frameworks for estimating parameters where we may be given a labeled dataset or a genome completely unlabeled. From there we will infer annotations and parameters for learning.

However the question is what will constrain the model in the unlabelled case? And we will see that it is the framework/structure we are given apriori. That is, the data itself is going to tell where the states are because it contains patterns embedded in it.

The example we are going to discuss is high GC and low GC regions and we wish to fit some states to high GC and some to low GC regions.

2 Posterior Decoding

2.1 Forward Algorithm

This algorithm will allow us to compute \(P(x_1, \ldots, x_N)\), the probability of a sequence of emissions \(X\) given the parameters of the full HMM \((Q, A, X, E, P)\). Let’s rewrite this probability as the sum over all possible ways of generating the last emission in the sequence, \(x_N\):

\[
P(x_1, \ldots, x_N) = \sum_l P(x_1, \ldots, x_N, \pi_N = l)
\]  

(2)

Define the quantity \(f_l(i)\) as the joint probability of generating \(x_1, \ldots, x_i\) (the first \(i\) emissions of the sequence) from the HMM and ending up in state \(l\) at step \(i\):

\[
f_l(i) = P(x_1, \ldots, x_i, \pi_i = l)
\]  

(3)

Now rewrite \(P(x_1, \ldots, x_N)\) in terms of \(f_l(i)\):

\[
P(x_1, \ldots, x_N) = \sum_l f_l(n)
\]  

(4)

In last lecture, the recursion formula of \(f_l(i)\) was derived:

\[
f_l(i) = e_l(x_i) \sum_k f_k(i-1)a_{kl}
\]  

(5)

We can now compute \(f_l(i)\) based on \(f_l(i-1)\) summing over all probability in previous state and because these emissions are in fact identical in the sum we can extract the emissions. The forward algorithm is shown in Figure 1.
2.2 Application

We have computed the probability of a sequence summed over all paths. But why would we ever want $P(X)$? Suppose we’re given 2 HMMs that can model your genomes. In the first one Promoters are modeled so that only Cs and Gs matter while in the second one, the CpGs (where the p denoted that they’re on the same strand) matter. You can calculate the total probability of explaining the data given the model while you don’t know where the promoters are. Given all possible parses of the sequence you can still compute the probability and compare the two HMMs.

Markov Chains and HMMs are memoryless, but there is trick to add memory by increasing the number of states, (actually, square the num of states). Similarly, we can combine two models of:

- ‘+’ model: from CpG islands
- ‘-’ model: from non-islands

To give 4 ‘+’ states and 4 ‘-’ states which have different emission probabilities is shown in Figure 2.

The probability $P(\pi_i = k|x_i = G)$ is equal to the prior of being in state k at position i times the most likely emission. Now what if we know the entire sequence? Having observed the entire sequence, we want to know what is the probability by summing all possible paths which is what posterior decoding does. To do this efficiently, we implement the forward and backward algorithm Figure 3 and we already know how to compute the first part which is forward algorithm.

2.3 Backward Algorithm

Now to compute the backward algorithm, the probability of observing the entire sequence following a particular state given that we’re in state k. (prob of end of seq given current state) is defined as:

$$b_k(i) = \sum_l e_l(x_{i+1} a_{kl} b_l(i + 1))$$  \hspace{1cm} (6)

The only difference with the forward algorithm is that “l” now appears within the summation and therefore, the emission probabilities are different for every state, and cannot be extracted from the sum, giving the backward algorithm in Figure 4.
Figure 2: HMM for CpG Islands

- Build a single model that combines two such Markov chains:
  - '+' states: $A_+$, $C_+$, $G_+$, $T_+$
    - Emit symbols: $A$, $C$, $G$, $T$ in CpG Islands
  - '-' states: $A_-$, $C_-$, $G_-$, $T_-$
    - Emit symbols: $A$, $C$, $G$, $T$ in non-islands
- Emission probabilities distinct for the '+' and the '-' states
  - Infer most likely set of states, giving rise to observed emissions
  - 'Paint' the sequence with + and - states

Why we need so many states...
In our simple GC-content example, we only had 2 states ($P|B$)
Why do we need 8 states here: 4 CpG+ / 4 CpG- ?
$\Rightarrow$ Encode 'memory' of previous state: nucleotide transitions

\[
P(\pi_i = k, x) = P(x_1 \ldots x_k, \pi_i = k, x_{i+1} \ldots x_n)
= P(x_1 \ldots x_k, \pi_i = k) \cdot P(x_{i+1} \ldots x_n | x_1 \ldots x_k, \pi_i = k)
= P(x_1 \ldots x_k, \pi_i = k) \cdot P(x_{i+1} \ldots x_n | \pi_i = k)
\]

$\text{Forward, } f_k(i)$ \quad $\text{Backward, } b_k(i)$

Figure 3: Forward and Backward Algorithm
Putting them altogether, we now have $P(\pi_i = k | x)$ known as posterior decoding, which gives a new path that gives the most likely state at every position. The probability that $i^{th}$ state is $k$, given all emissions $x$.

$$P(k) = P(\pi_i = k | x) = \frac{f_k(i)b_k(i)}{P(x)} \tag{7}$$

For classification, based on the entire sequence, if we’re trying to find out if a region is in fact a promoter or not, the Posterior decoding method works better because it’s more informative than the Viterbi path with summing over all path. Even though the Viterbi path is the most likely path it may share only a tiny portion of the probability. We might also want to make a prediction which in that case we use the posterior path. However, it may give us an invalid sequence of states! For example it might give two consecutive states that transition between them is not possible at all. So the actual probability of that path is zero because of an illegal transition.

All in all, if your application is to decode the whole genome you may prefer the Viterbi path, but if you want high accuracy at every position, you may prefer the posterior decoding.

### 3 Learning

We saw how to score and decode a sequence in two different ways.

In order to train an HMM, we have to learn the transition and emission probabilities. If the right answer is known for example if we’re given a genomic region of say a million nucleotides, and we have good experimental annotation of CpG islands, we are going to do Supervised Learning.

If instead we don’t know the right answer and we don’t know how frequent the CpG islands are, neither do we know the composition and where they are, and we just know that there exist some CpG islands, we wish to update our parameters of the model to maximize $P(x | \theta)$ the sequence given the parameters $\theta = \{E_i, A_{ij}\}$.

#### 3.1 Supervised Learning

Estimating model parameters based on labeled data is actually trivial. Suppose that you are given a labeled sequence $x_1, \ldots, x_N$, meaning the true hidden state sequence $\pi_1, \ldots, \pi_N$ is known and want to estimate the
parameters. We first define $A_{kl}$ to be the number of times that a transition from state $k$ to state $l$ occurs in $\pi$, and $E_k(b)$ to be the number of times that state $k$ in $\pi$ emits $b$ in $x$. The parameters $\theta$ that maximize $P(x|\theta)$ is simply obtained by counting.

\[
a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}} \quad (8a)
\]
\[
e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)} \quad (8b)
\]

An example is shown in Figure 5. Now the problem is that in the above example, every time you have a T, you get a zero probability and this will bias interpretations which are due to over-fitting or small sample size. To deal with over-fitting, the solution is using pseudocounts. We can add pseudocounts representing our prior beliefs about the model parameters:

- $A_{kl} = \# \text{ of times a } k \rightarrow l \text{ transition occurs in } \pi + r_{kl}$
- $E_k(b) = \# \text{ of times state } k \text{ in } \pi \text{ emits } b \text{ in } x + r_k(b)$

Larger pseudocounts correspond to a strong prior belief. Consequently, small pseudocounts ($\epsilon < 1$) are used to avoid 0 probabilities.

### 3.2 Unsupervised Learning

Unsupervised learning is estimating parameters based on unlabeled data, not knowing $A_{kl}$ or $E_k(b)$. If we have some guesses to the probabilities $\theta$ for initialization, we can find the other parameters with parsing the data based on initial probabilities by Viterbi or Posterior decoding. Then we will have labeled data which can be used for supervised learning. Afterwards, the cycle repeats where we update our guess and so on. Now, where does the constraint come in into the learning? If we initialize differently do we end up with the same results? Yes because the data will bring down a false high probability that we might have guessed for a region. So the genome actually filters the data at each step. However, different initializations might lead to local maxima. There are two methods for learning:
(Sum over all $k \rightarrow l$ transitions, at any time step $i$)

So,

$$f_s(i) \ a_{kl} \ e_i(x_{i+1}) \ b_i(l+1)$$

$$A_{kl} = \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \sum_i \frac{P(\pi_i = k, \pi_{i+1} = l \mid x, \theta)}{P(x \mid \theta)}$$

Similarly,

$$E_{kl}(b) = [1/P(x)] \sum_i \{i \mid x_i = b\} \ f_s(i) \ b_k(i)$$

**Figure 6**: EM - New Parameters give probabilistic parse

**Viterbi training**: picking the best guess for the model and performing the Viterbi algorithm. It converges to a local maxima very rapidly.

**Baum-Welch Learning**: Learning over all paths by applying EM to HMMs.

The Viterbi training is a simple case. First pick the best-guess for model parameters and iterate the following steps until convergence.

1. Perform Viterbi to find $\pi^*$
2. Calculate $A_{kl}, E_k(b)$ according to $\pi^* + \text{pseudocounts}$
3. Calculate the new parameters $a_{kl}, e_k(b)$

**3.2.1 Expectation Maximization: The Baum-Welch Algorithm**

**E step**: Estimate the expected probability of hidden labels given the current latest parameters and the observed unchanging sequence.

**M step**: Choose new ML sets of parameters over the distribution of sequence and labels given the current probabilistic assignments.

For example to count the transition matrix in Figure 6. We’re effectively using the forward and backward algorithm to find the entire probability capturing that position. What we end up with is known as the Baum-Welch algorithm, shown in Figure 7.

The time complexity is the number of iterations multiplied by the cost $O(K^2N)$. It guarantees to converge locally but not necessarily globally.
The Baum-Welch Algorithm

**Initialization:**
Pick the best-guess for model parameters
(or arbitrary)

**Iteration:**
1. Forward
2. Backward
3. \( \rightarrow \) Calculate new log-likelihood \( P(x \mid \theta) \) (E step)
4. Calculate \( A_{ik}, E_k(b) \)
5. \( \rightarrow \) Calculate new model parameters \( a_{ik}, e_k(b) \) (M step)

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until \( P(x \mid \theta) \) does not change much

**Figure 7:** Baum-Welch Algorithm