15.1 MMIE Training (continued)

In the previous lecture we discussed using gradient descent as a method for optimizing the parameters for our Hidden Markov Model (HMM). While this method does work it had several drawbacks. First, there is no guarantee of convergence. This is a serious problem by itself. But the gradient descent method is also computationally expensive and only looks at the first order terms. So we want to find a better way.

To do this we look to the Extended Baum-Welch update equations developed by Gopalakrishnan. This is a theorem about polynomials defined over simplexes so we will establish some notation about these below.

$P(\Lambda)$ is used to denote a function of polynomials on $\Lambda$

\[ P(\Lambda) = P([\lambda_{ij}]) i = 1...p, j = 1...q_i \]  

(15.1)

The polynomials are defined over a set of multiple simplexes $D$.

\[ \Lambda \in D = [\lambda_{ij} : \lambda_{ij} \geq 0, \sum_{j=1}^{q_i} \lambda_{ij} = 1] \]  

(15.2)

A ratio of polynomials $S_1$ and $S_2$ is expressed as follows.

\[ R(\Lambda) = \frac{S_1(\Lambda)}{S_2(\Lambda)} \]  

(15.3)

A simplex is a tetrahedral region that has been generalized to n-dimensions. For the purposes here it is probably easiest to think of it as a subspace over which the polynomials are defined. Some examples of simplexes are shown below.

Now that we have some notation theorem will make more sense. The Baum-Welch theorem states that if we have a homogenous polynomial with nonnegative coefficients of degree $d$ defined on $D$ and it obeys the following restriction

\[ \sum_{j=1}^{q_i} \lambda_{ij} \frac{\partial P}{\partial \lambda_{ij}}(\lambda_{ij}) \neq 0, \forall i \]  

(15.4)
If these conditions are met than we can define a growth transform on $\xi = T(\lambda)$ which is defined as follows.

$$
\xi_{ij} = \lambda_{ij} \frac{\partial P}{\partial \Lambda_{ij}} (\lambda_{ij}) \sum_{j=1}^{p} \lambda_{ij} \frac{\partial P}{\partial \Lambda_{ij}} (\lambda_{ij})
$$

(15.5)

From this equation we can see the complex looking constraint placed on the polynomial is there merely to prevent dividing by zero in the transformation.

In order to use this theorem we have to put our data in a usable form. Namely we have to turn the ratio of polynomials we get from the mutual information and convert it into a standard homogenous polynomial with nonnegative coefficients. Luckily we have a nice three step procedure to do just that.

The first step is to move from a ratio of polynomials to a standard polynomial. So we start with

$$
P_{\lambda}(\Lambda) \equiv S_{1}(\Lambda) - R_{\lambda}(\lambda) S_{2}(\lambda)
$$

(15.6)

This is a bit tricky since $P_{\lambda}(\lambda)$ is identically zero for all $\lambda$. But if $P_{\lambda}(\xi) > 0$ then $R(\xi) > R(\lambda)$ Here is the proof.

$$
P_{\lambda}(\xi) = S_{1}(\xi) - R(\lambda) S_{2}(\xi) > 0
$$

(15.7)

$$
R(\lambda) < \frac{S_{1}(\xi)}{S_{2}(\xi)}
$$

(15.8)

$$
R(\lambda) < R(\xi)
$$

(15.9)

So if we can find a growth transform on $P_{\lambda}(\lambda)$ we will automatically have one for $R(\lambda)$ because of the relationship we just derived.

The second step is to guarantee that the polynomial has all nonnegative coefficients. To do this we will again modify the polynomial, this time by adding a constant. The new polynomial is defined as follows.

$$
P'(\Lambda) \equiv P(\Lambda) + C(\Lambda)
$$

(15.10)

And $C(\Lambda)$ is defined as follows.

$$
C(\Lambda) = -a\left(\sum_{i=1}^{p} \sum_{j=1}^{p} \Lambda_{ij} + 1\right)^{d} = -a(p + 1)^{d}
$$

(15.11)

Here $a$ is the minimal negative coefficient of $P$ (a=0 if there are none) and $d$ is the degree of the polynomial. The value of $C$ is a constant because all of the sums of the rows in the simplex sum to one. Raising all the coefficients to the $d$th power will generate every possible polynomial of order $d$ or less from the simplex.

By adding a constant to the polynomial we achieve our two goal. We are now guaranteed to have a polynomial with nonnegative coefficients and it will also leave the growth transform unaffected.

The final step is making a homogenous polynomial. A homogenous polynomial is one where all the terms have the same order. So $x^{2} + x + 1$ is not homogenous but $x^{2} + xy + y^{2}$ is homogenous. Turning a nonhomogeneous polynomial into a homogenous one is actually rather simple. It involves a simple variable substitution. The new polynomial is as follows.

$$
P''(\psi) = \psi_{p+1,1}^{d} P'(\psi_{ij}/\psi_{p+1,1})
$$

(15.12)

All we have done is replace $\Lambda_{ij}$ with $\psi_{ij}/\psi_{p+1,1}$. The trick is that $\psi_{p+1,1} = 1$. We also have to describe the new set of constrained simplexes over which it is defined.

$$
\psi \in D' = \{\psi_{ij} : \psi_{ij} \geq 0, \sum_{j=1}^{q} \psi_{ij} = 1\}
$$

(15.13)
Essentially all we have done is create a new variable $\psi$ that is identically equal to 1 and multiplied all the lower order terms by it enough times so that every term is now of the same order. For example, we can turn $x^2 + x + 1$ into $x^2 + x\psi + \psi^2$. Since $\psi = 1$ we have not changed the equation at all.

More technically we can see from the definitions of $D$ and $D'$ that they are isomorphic and that there is a bijection between $\psi$ and $\lambda$. Thus any growth function in $D'$ for $P''$ will also be a growth function in $D$ for $P'$.

Now that we have a usable polynomial we can use the growth function developed by Gopalakrishnan. According to the theorem if we have $R(\Lambda)$ as a rational function of polynomials in $\Lambda_{ij}$ then there exists $a_R$ such that for $C \geq a_R$ the following function $T^C()$ is a growth function in $D$ for $R$.

$$\langle T^C(\lambda) \rangle_{ij} = \frac{\lambda_{ij} \left( \frac{\partial P}{\partial \Lambda_{ij}}(\lambda) + C \right)}{\sum_{j=1}^{q_i} \lambda_{ij} \left( \frac{\partial P}{\partial \Lambda_{ij}}(\lambda) + C \right)}$$

(15.14)

Here $a_R = ad(p + 1)^{d-1}$ and $a = max_{\lambda} a_\lambda$ and $a_\lambda$ is the minimal negative coefficient for all $\lambda$.

Applying this to our MMIE training we an update equation for optimizing our parameters.

$$a_{ij}^{t+1} = \frac{a_{ij}^t \left( \frac{\partial \log Z_{\lambda}}{\partial a_{ij}}(\lambda) + C(\lambda) \right)}{\sum_{j=1}^{q_i} a_{ij}^t \left( \frac{\partial \log Z_{\lambda}}{\partial a_{ij}}(\lambda) + C(\lambda) \right)}$$

(15.15)

recall that

$$Z_\lambda = 2^{f_{i}(m;x)} = \frac{p(x|m,\lambda)}{p(x|\lambda)} = \frac{p(x|m,\lambda)}{\sum_{m'} p(x|m',\lambda)}$$

(15.16)

For this formula there is a lower bound on $C$. So choosing a high value of $C$ would seem like a good practice to ensure convergence. However, the larger $C$ is the longer it takes to converge. So it is desirable to choose as small a $C$ as possible. Since it is not always possible to determine $a_R$ the typical heuristic is to estimate the lower bound an $C$ and double that.

### 15.2 Minimum Classification Error Training

Maximum mutual information estimation has been around since 1986 but it has not really improved results until recently. The lack of success with this technique might prompt us to question some of our assumptions. A very important question to ask is whether the posterior is really the most important thing to optimize.

According to Bayes decision theory you can achieve the minimum error by using the posterior probability. So it is a sufficient condition for minimal error but not necessarily a necessary condition. In fact a slight change in the posterior probability can achieve the same results as the true posterior. So instead of looking for the true posterior probability why not look for the easiest posterior to find that will give us the same results. This is the idea behind Minimum Classification Error (MCE) training.

In MCE training we seek to minimize the classification error function directly. The error, however, is a discrete function. It counts how many samples were misclassified. But most of our maximization and minimization techniques like gradient descent require continuous functions. So in order to use the nice continuous minimizations we will need find a continuous function to approximate our discrete error function. An example of the continuous approximation to the max function is shown below.

$$\max g_i = \lim_{n \to \infty} \log \left( \frac{1}{N} \sum_j e^{g_j/n} \right)^{1/n}$$

(15.17)
The equation for the precise misclassification measure is as follows.

\[ d_i(X) = -g_i(X|\Lambda) + \log\left[ \frac{1}{N-1} \sum_{j \neq i} e^{g_j(X|\Lambda)n} \right]^{1/n} \] (15.18)

The loss function which measures the amount of misclassification is as follows.

\[ l(d) = \frac{1}{1 + e^{-\gamma d + \theta}}, \gamma \geq 1 \] (15.19)

This is the sigmoid function which approximates a unit step function. The \( \theta \) parameter simply models the shift and the \( \gamma \) controls how closely it models the step function.

The final performance criterion for \( X \) of class \( i \) is as follows.

\[ l(X|\Lambda) = \sum_{i=1}^{M} l_i(X|\Lambda)1(X \in C_i) \] (15.20)

Over training set

\[ L(\Lambda) = E_X \{ l(X|\Lambda) \} \] (15.21)

For a large enough \( n \) and \( \gamma \) this function does a good job of modeling our discrete error. However, if \( n \) or \( \gamma \) is too large it can do too good of a job. If the function models the discrete function too closely then the higher order Taylor series terms start to become very important so the function is not "nice." Because the function is changing so rapidly now computing the gradient is not as informative and it is harder to find a local maximum. Therefore training suffers. So a trade off has to be selected between accurate error and "nice" functions to find a gradient on.

### 15.3 The lighter side of HMMs

The following section highlights some of the quirks about working with Hidden Markov Models in speech recognition. So in no particular order here are some some fun facts about HMMs.

The observation densities come in three flavors. The continuous observations

\[ b_j(x_t) = \sum_{k=1}^{K} c_{jk} N(x|\mu_{jk}, \Sigma_{jk}) \] (15.22)

The discrete

\[ b_j(x_t) = \prod_{k=1}^{K} \rho_{tj}^{L_{x_t=x(j)}} \] (15.23)

and the semi-continuous

\[ b_j(x_t) = \sum_{k=1}^{K} c_{jk} N(x|\mu_{m(jk)}, \Sigma_{a(jk)}) \] (15.24)

Some states have no observations associated with them. Examples of these types of states would be start and stop states. These states are indicated by concentric circles and specify where the markov chain begins and where it ends or stops processing.
There are also two different ways to represent HMMs using a finite state automata approach. There is the Moore method for representing finite state machines where outputs or observations are associated only with the current state of the machine. Then there is the Mealy method where the output is a function both of the current state and the current input. This essentially means that the output is associated with the transitions rather than the states. An example of a Moore machine is in figure 15.3. A Mealy version is presented in figure 15.4.

The differences between a Moore and Mealy machine, however, are only superficial because they are representationally equivalent. Any Mealy machine with \( n \) states can be represented as a Moore machine with at most \( n^2 \) states. This is because there are at most \( n^2 \) transitions between \( n \) states.

When using HMMs for speech recognition it is most common to use a strictly left-to-right chain. In this model you can either stay in your current state or move to the next one. No other transitions are possible. This means the transition probability matrix will be an upper triangular matrix since you cannot go backwards. An ever stronger statement is
that only the diagonals and the first off diagonal will have nonzero values. Since you have to go forward if you don’t stay in the same state. This is illustrated in figure 15.5.

Figure 15.5: Left-to-right Markov chain for the word me

This is a fairly rigid structure that we are forcing a word to comply to. Just imagine all the ways there are to pronounce the or and in every sort of dialect. It would seem to make more sense to allow flexibility in the pronunciation of a word like the following Markov chain for the word and.

Figure 15.6: A flexible Markov chain for the word and.

Flexible chains like this do increase the likelihood of catching the word and. However, they also increase the likelihood and classifying a different word as and. For the classifier to work properly there has to be a balance between how flexible we are to raise the probability of finding true occurrences and how rigid we are to reduce the number of false positives. The strict left-to-right chain has been found to work reasonably well.

In addition, we have shown states corresponding to individual phones. However, a phone will most likely be broken down into several states. The reasoning behind this is to more accurately model the duration of a phone. In the previous model where one state corresponds to one phone, the amount of time spent in each state is geometrically distributed.

\[ p(D = d) = p^{d-1}(1 - p), d \geq 1 \]  

(15.25)

Where \( D \) is the duration and \( p \) is the probability of staying in the same state. But this does not match the real distribution of durations. In reality we want a distribution a little more gaussian but without any probability for negative durations. By breaking a state up into several states we now have a sum of geometric distributions for our duration. This creates a negative binomial distribution which is more more inline with what we want.

\[ p(Z = z) = \binom{z + n + 1}{n-1} p^n (1-p)^z \]  

(15.26)
Where \( n \) is the number of states and \( Z \) is the duration beyond \( n \). Obviously the duration cannot be less than \( n \) because \( n \) moves are required to get through the Markov chain. You can see the differences between the two distributions below.

![Geometric and Negative Binomial Distributions](image)

This opens up an important subfield of speech recognition which is pronunciation modeling. The next lecture will begin by discussing more advanced methods of modeling pronunciation of words.