Prof. Sergio A. Alvarez
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609, USA

web: http://www.cs.wpi.edu/~alvarez/
e-mail: alvarez@cs.wpi.edu
phone: (508) 831-5118
fax: (508) 831-5776

CS2223, D Term 2000 Extra Credit Problem (Optional; due Monday, May 1)

1 Crash course in optical text recognition

Basic problem. Given imperfect *images* of n handwritten alphabetical characters, presumably the letters of a word in English, one wishes to determine the sequence of n actual *letters* that most likely gave rise to the images. A translation of this statement into the language of probability theory is given below.

1.1 Probabilistic description

Let $I_1, \cdots I_n$ represent the n observed images, and let $L_1, \cdots L_n$ represent a *word hypothesis*, that is, a particular sequence of n letters whose likelihood of having generated the observed images is to be evaluated. Then using Bayes' theorem one may express the conditional probability that the given word hypothesis is correct given the observed images as follows:

$$P(L_1,\ L_2,\ \cdots L_n|I_1,\ I_2 \cdots I_n) = C \cdot P(I_1,\ I_2,\ \cdots I_n|L_1,\ L_2,\ \cdots L_n)P(L_1,\ L_2,\ \cdots L_n) \tag{1}$$

Here, C is a normalization constant (this just means that you want to make sure that all the probabilities add up to 1). The basic problem thus becomes a concrete optimization problem: *maximize the right-hand side of the above equation over all possible word hypotheses* $L_1, \cdots L_n$. The solution to this problem depends on the model that one assumes for image generation and letter co-occurrence as encoded in the two terms on the right-hand side above. Some comments on this statement follow.

1.2 Using only image data

If one assumes that all letters occur independently of one another and with the same probability, then the letter co-occurrence probability $P(L_1,\ L_2,\ \cdots L_n)$ is the same for all word hypotheses, and so that term effectively drops out of the maximization problem for Eq. 1. The expression to be maximized is now just the image generation probability:

$$P(I_1,\ I_2,\ \cdots I_n|L_1,\ L_2,\ \cdots L_n)$$

We will assume for simplicity that knowledge of the identity of a letter completely determines the probability that a given image will be generated for that letter, and that the images for different letters within a given word are generated independently of one another; image probabilities for various letters have to be learned by the system by examining multiple sample images for each letter. The above expression then simplifies:

$$P(I_1,\ I_2,\ \cdots I_n|L_1,\ L_2,\ \cdots L_n) = P(I_1|L_1)P(I_2|L_2)\cdots P(I_n|L_n)$$

It is now possible to break the maximization problem down into n subproblems: for each position in the sequence, one selects the letter that is most consistent with the image for that position; in other words, L_k should maximize the expression $P(I_k|L_k)$. The resulting sequence of letters is then the "winning" word.

Models such as those described above use knowledge about image generation, and in particular about the consistency between a given image and the hypothesis that a particular letter may have given rise to that image. However, such models completely ignore the letter-order restrictions that exist in English; for instance, the sequence "cmt" could beat "cat", despite one's intuitive sense that there should be a strong syntactical preference in favor of the latter.

1.3 Incorporating language restrictions

A slightly better model uses *transition probabilities* to represent letter-order restrictions. For each pair (x, y) of letters, a number $P(y|x)$ between 0 and 1 is given, corresponding to the fraction of words in which letter y is observed to follow immediately after letter x. We assume that the *prior probabilities* of all letters are also known; these are numbers between 0 and 1 that measure the relative frequencies of the letters in the given language (English, for example). These probabilities allow one to assess the likelihood of a given sequence of letters *in the absence of image data.* The likelihood of observing the sequence $L_1,\ L_2,\ \cdots L_n$ can be computed as a product:

$$P(L_1,\ L_2,\ \cdots L_n) = P(L_1|\text{blank})P(L_2|L_1)\cdots P(L_n|L_{n-1})P(L_n|\text{blank})$$

The language generation model encoded in the transition probabilities can now be combined with the image generation model implicit in the conditional probabilities $P(I_k|L_k)$ as described previously to quantify the likelihood of a given sequence of letters in the presence of specific image data. Indeed, Eq. 1 becomes:

$$P(L_1,\ \cdots L_n|I_1,\ \cdots I_n) = C \cdot P(I_1|L_1)\cdots P(I_n|L_n)P(L_1|\text{blank})P(L_2|L_1)\cdots P(L_n|L_{n-1})P(L_n|\text{blank})$$

The text recognition problem therefore reduces to the following optimization problem: *maximize the likelihood that appears on the right-hand side of the above equation over all letter sequences $L_1 \cdots L_n$.* The winning letter sequence is the one most likely to have led to the observed images, given the known transition probabilities between letters.

2 Formulation as a graph problem

The optimization problem stated above may be viewed as a special variant of the single-source shortest path problem in a graph. The underlying graph in this case has a source node and a sink node, corresponding to the blank characters that delimit the given sequence of n characters. The other nodes of the graph are arranged in columns. There is one column for each of the n characters in the word to be recognized. Each column contains m (typically 26) nodes, one for each letter of the alphabet. Each column is "fully connected" to the column immediately to the right, i.e. there is a directed arc from each node of a given column to each node of the next column. This reflects the desire to initially allow any possible sequence of n letters.

Word hypotheses correspond to paths in this graph from the source node to the sink node. Each path has an associated cost. In order to make the cost function additive, we simply take the logarithm of the likelihood described in the previous section and split up the result into parts corresponding to various arcs. Specifically, the weight of the arc from the node for letter L_i in column $c-1$ to the node for letter L_j in column c is

$$\text{cost}((c-1, i), (c, j)) = -\log P(L_j|L_i) - \log P(I_c|L_j)$$

Notice that we have changed the notation slightly here relative to the notation that was being used previously; namely, *L_j now represents the j-th letter of the alphabet, regardless of what position(s) within a word hypothesis it is being considered for.* The total likelihood over a given word hypothesis thus corresponds directly to the sum of the weights over the path in the graph that represents the given word hypothesis (you have to change the signs of the weights, then take an exponential at the end to recover the likelihood, but this is straightforward). The second term in the expression for the weights is perhaps more naturally associated with the terminal node of the arc in question rather than with the arc itself; nonetheless, the above description also leads to a correct formulation. In either case, the most likely word hypothesis will correspond to the path with the lowest total cost. In summary, one arrives at the following problem.

Trellis problem. Given a weighted directed graph G organized into $n+2$ columns numbered from 0 to $n+1$, with columns 0 and $n+1$ each consisting of a single node, columns $1...n$ each consisting of the same number m of nodes (for example, $m = 26$), and with column c fully connected to column $c+1$ for each $c = 0..n$, find the path of least weight from the source node in column 0 to the sink node in column $n+1$.

3 Your task, should you choose to accept it ...

Refer to the above description for an explanation of the context of this problem as well as a formulation of the trellis problem. Define $C(k, j)$ to be the minimum cost of a path from the source node to the node in column k corresponding to letter L_j. We will refer to C as the *value function* for the trellis problem.

Problem 1: Recursive description of the value function

Find a recurrence relation satisfied by the value function. This relation should express the value function for column k, namely the values $C(k, j)$ for $j = 1..m$, in terms of the values of the value function for the previous column, $C(k-1, \cdot)$, together with the weight values $\text{cost}((k-1, \cdot), (k, j))$ of the arcs from nodes in column $k-1$ to each given node in column k. Here, k ranges from 1 to $n+1$.

Problem 2: Dynamic programming approach to calculating the value function

Based on your work for the previous problem, write pseudocode for a dynamic programming algorithm that calculates the value function $C(n+1, \text{blank})$. Describe in words the order in which your algorithm processes the nodes of the word hypotheses graph.

Problem 3: Evaluation of your approach

Estimate the running time of your algorithm as a function of n and m. Explain your reasoning in detail. How does the efficiency of your algorithm compare to that of the obvious recursive algorithm based on your answer to problem 1 ?