On-line Handwriting Recognition

Nikhil Lele, Ben Mildenhall

I. Introduction

We adapt a technique from C. Lawrence Zitnick's 2013 *SIGGRAPH* paper for comparing stroke data using curvature. Zitnick uses curvature-based stroke sampling to compare arbitrary "tokens" in handwritten data, then clusters similar tokens and morphs each one to a "token mean" in order to beautify the writing. However, Zitnick did not use these comparisons to try to recognize common symbols (or "glyphs") in the data. We extend the idea of comparing curvature to a supervised learning problem of identifying glyphs from a specific alphabet, like English, given a training data set.

II. Glyph data

We use the mouse input to a computer to collect data. Each data point is a single "glyph," which is a set of n points $\{t_i, x_i, y_i\}$ where t_i is elapsed time since the first point in the glyph and x_i, y_i are the two dimensional screen coordinates of the i-th point. Once the user indicates they are done inputting the glyph, this data is normalized so that $t_1 = 0$ and $t_n = 1$. In addition, the glyph is scaled and translated so that its centroid is at $(0.5, 0.5)$ and so that the point farthest from the centroid lies on the perimeter of the square ranging from $(0, 0)$ to $(1, 1)$.

We save this raw normalized data, but we run all comparison operations on smoothed versions of each glyph. Smoothing is a two-step process. First, we upsample the glyph. Each glyph has between 30-50 points when recorded, and we upsample the glyph at 300-500 evenly spaced time samples in the intervals $[0, 1]$. Given some time t we would like to sample a glyph G, we run through each point in G until we find point samples such that $t_{i-1} < t \leq t_i$. We then linearly interpolate the space coordinates (x_{i-1}, y_{i-1}) and (x_i, y_i) of those points using

$$t' = \frac{t - t_{i-1}}{t_i - t_{i-1}}$$

to get the sample for time t. We could also upsample the glyph using evenly spaced samples parametrized by the *distance* travelled along the glyph; we have not yet thoroughly explored this option.

After upsampling the glyph, we downsample to some lesser number of evenly spaced time samples (usually 100) by taking a weighted average of all the samples in the high-resolution version of the glyph. So our sample at t_i becomes

$$t_i = \sum_{j=1}^{m} P_j \cdot N(t_j - t_i)$$

assuming there are m samples in the upsampled glyph, where $P_j = (x_j, y_j)$ and t_j are the j-th point from the upsampled glyph and $N(t_j - t_i)$ is the p.d.f. of a normal distribution with mean 0 and variance σ^2. We have not yet determined which value of σ is most effective. Lower values preserve more detail from the raw sample, such as sharp turns and twists in more "pointy" letters, but higher values are more effective for smoothing out the noise in the raw data.

III. Least cost path comparison

In order to meaningfully compare time series of curvature or other data (possibly multidimensional), we adapt Zitnick's method of computing a "least cost path" to match the two series of data. Given two time series $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2 \ldots, b_n\}$ of length n and a cost function $C(a_i, b_j)$ that expresses some sort of "distance" between a_i and b_j, we create a cost matrix C with $C_{ij} = C(a_i, b_j)$. A valid path through the cost matrix is one that travels from index $(1, 1)$ to (n, n) using only moves of the form (i, j) to $(i + 1, j)$, $(i, j + 1)$, or $(i + 1, j + 1)$. The cost of the whole path is the sum of all cost matrix entries it traverses. We find the least cost path in $O(n^2)$ time using dynamic programming.

Note that since we did not specify what space $\{a_i\}$ and $\{b_i\}$ lie in, we can compute least cost path matches between multidimensional time series. For example, we can input the screen space points from two glyphs (assuming we normalize so that we have the same number of point samples in each glyph), we can use the cost function $C(P_i, Q_j) = ||P_i - Q_j||^2$.

IV. Supervised learning using curvatures

So far we have focused on the problem of single character recognition. We have created a few training sets with about 500 examples each (approximately 20 samples for each lowercase English character).

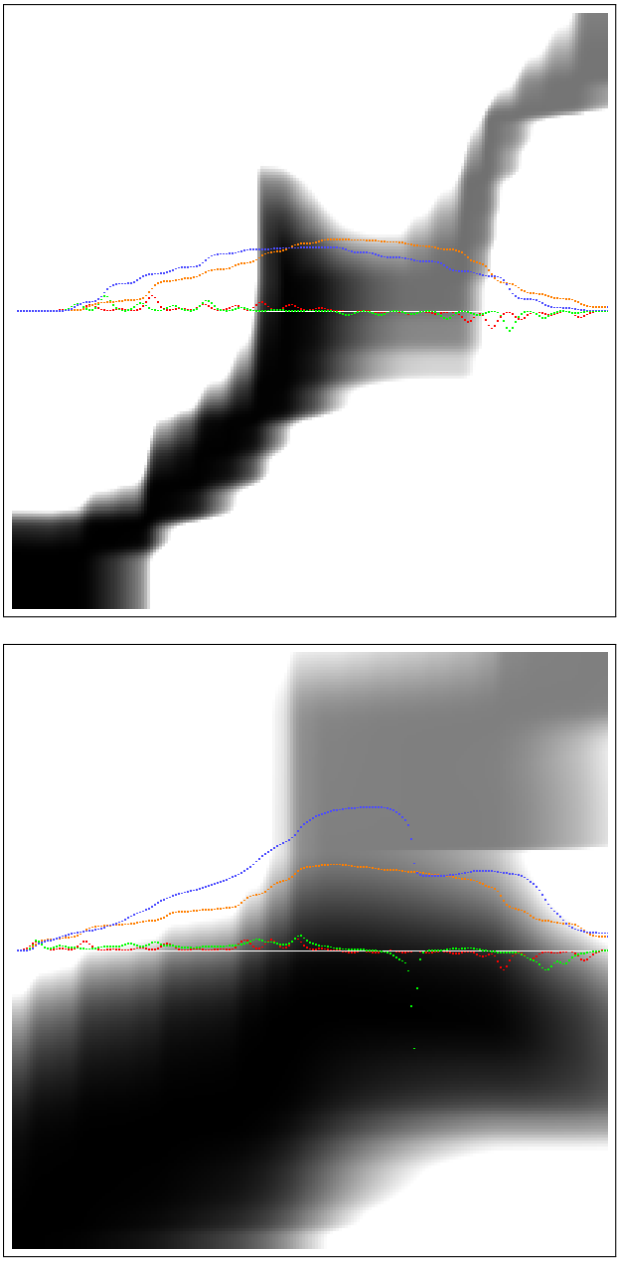

Fig. 1. Least cost path through integrated curvature time series for two 's' samples (top) and for an 's' and a 'g' sample (bottom). Also plotted are curvature (in red and green) and integrated curvature (in orange and blue).

To compare two glyphs, we start by smoothing them using the method described in part II to obtain with uniformly sampled time series of screen space points (usually of length 100). We use those points to compute curvature and integrated curvature, then we use the least cost path method to evaluate the similarity of the two glyphs.

Figure 1 graphically depicts the least cost path computation between integrated curvature time series, first for two 's' glyphs and then for an 's' glyph versus a 'g' glyph. The bottom left corner corresponds to index $(1, 1)$ in the cost matrix and the top right corner corresponds to (n, n). The color of a square in the grid is proportional to the sum of costs in the cheapest path to that matrix index. The colors are normalized so that the top right square (corresponding to the least cost path through the whole matrix) is half grey (RGB values $128, 128, 128$). Any square which costs more than twice as much as the top right square to reach is colored white. The least cost path for the top figure costs approximately 64 times less than the cheapest path for the bottom figure, which indicates that the two 's' glyphs are a much closer match than the 's' and 'g' glyphs. In addition, note that the least cost path of the top matrix is much more "diagonal" than the bottom one, which means the best correspondence between the two time series is nearly one-to-one, whereas the bottom figure is much more "murky," with a less clear connection between the data.

So far, we have only used this general comparison technique to naively create clusters of glyphs for supervised learning. Given a cluster, we calculate the cluster average by creating a composite glyph with all the $\{t_i, x_i, y_i\}$ data points from all glyphs in that cluster, then downsampling the composite glyph by taking the Gaussian weighted average centered at some number of uniform time samples. When analyzing a new glyph, we calculate its distance to the average glyph of each previously found cluster that has the same label as the new glyph. If the distance to the closest cluster is above some threshold, we create a new cluster for that label containing only the new glyph. We have not used any learning techniques or cross-validation to set the threshold; we have just been manually setting that parameter, as well as the weights for the various least cost path features (curvature, integrated curvature, etc.). Another potential idea to reduce variance is eliminating all single glyph clusters after training is complete.

When classifying a new glyph, we calculate its distance to every cluster and return the label of the closest cluster. This technique has proven fairly successful, even when setting parameters by hand. For example, when trained on a data set of 531 glyphs (about 20 examples per character) then tested on a set of 126 glyphs (about 4-6 examples per character), we achieved 92.8% accuracy (9 incorrect classifications).

V. Further work

We would like to make the single character supervised learning much more robust by automatically setting parameters like weights, new cluster threshold, and smoothing values. More generally, we would like to expand the recognition to entire English words in which the glyphs for each character may not be completely separated. One possible idea is expanding the least cost path method to try to "fit" a smaller time series into a larger one and then use that technique to figure out which letters are most likely to be at certain positions within a word, given the curvature data for the whole word. We could also use samples of English text or large vocabulary data sets to help determine which combinations of letters are actual words. Ideally, we could eventually use a paragraph of English text to train a model rather than single characters.

References

[1] C.L. Zitnick, 2013. Handwriting Beautification Using Token Means.