Algorithms for Handwritten Digit Recognition

Michael J. M. Mazack
Adviser: Dr. Tjalling Ypma

Western Washington University

Master’s Project Colloquium
February 5th, 2009
The Problem

Automatically classify a single unknown handwritten digit using a database of known digits.

An Unknown Digit (Test Image)

- $16 \times 16$-pixel grayscale images (matrices) of digits 0, ..., 9.
- Application: Automatic mail sorting at the post office.
Images from the Database

- Scanned and rescaled to $16 \times 16$-pixel grayscale images.
- Pixels take floating point values between -1 (white) and 1 (black).

Michael Mazack

Algorithms for Handwritten Digit Recognition
The Database

- 7291 handwritten digits collected by the U.S. Postal Service. ¹

### Breakdown of Digits

<table>
<thead>
<tr>
<th>Digit</th>
<th>Sample Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1194</td>
</tr>
<tr>
<td>1</td>
<td>1005</td>
</tr>
<tr>
<td>2</td>
<td>731</td>
</tr>
<tr>
<td>3</td>
<td>658</td>
</tr>
<tr>
<td>4</td>
<td>652</td>
</tr>
<tr>
<td>5</td>
<td>556</td>
</tr>
<tr>
<td>6</td>
<td>664</td>
</tr>
<tr>
<td>7</td>
<td>645</td>
</tr>
<tr>
<td>8</td>
<td>542</td>
</tr>
<tr>
<td>9</td>
<td>644</td>
</tr>
</tbody>
</table>

We try two classification algorithms.

- Singular Value Decomposition Based Algorithm
- Tangent Distance Algorithm
First Algorithm:
The SVD Based Algorithm
How We Handle the Database

- Unroll the $16 \times 16$-pixel images into vectors in $\mathbb{R}^{256}$.
- Collect all the different types (0 through 9) of unrolled images.
- Place all unrolled images of type $i \in \{0, 1, ..., 9\}$ into the matrix $D_i$ as the columns.

\[
D_5 = \begin{bmatrix}
| & | & | & \cdots & | \\
5 & 5 & 5 & \cdots & 5 \\
| & | & | & \cdots & |
\end{bmatrix}
\]

$D_5 \in \mathbb{R}^{256 \times 556}$

Notice there are many more columns than rows.
Take a test image \( d \in \mathbb{R}^{256} \).

\[
D_5 = \begin{bmatrix}
5 & 5 & 5 & \ldots & 5 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\end{bmatrix}, \quad d = ?
\]

- Is \( d \) a linear combination of the columns of some \( D_i \)?
- How close is \( d \) to being a linear combination of the columns of \( D_i \)?

Solve a least squares problem!

\[
\rho_i = \min_x \| D_i x - d \|^2_2
\]

Observe: We are interested in the residual \( \rho_i \) and not the \( x \).
A Classification Algorithm

Let \( d \in \mathbb{R}^{256} \) be a test digit to classify and let \( i \in \{0, 1, \ldots, 9\} \).

- Form the \( D_i \) matrices (as described before) for every \( i \).
- For every \( i \), find \( \rho_i = \min_x \| D_i x - d \|_2^2 \).
- Compute \( \min_i \{ \rho_i \} \) and classify \( d \) as a digit of type “\( i \)”. 

The residual can be found by using brute force to solve the least squares problem, but this is expensive. We will show how to implement this algorithm efficiently by using properties of the SVD.
**Theorem (Singular Value Decomposition)**

Let \( A \in \mathbb{R}^{m \times n} \) be a nonzero matrix with rank \( r \). Then \( A \) can be expressed as a product \( A = U \Sigma V^T \), where \( U \in \mathbb{R}^{m \times m} \) and \( V \in \mathbb{R}^{n \times n} \) are orthogonal, and \( \Sigma \in \mathbb{R}^{m \times n} \) is a “diagonal” matrix with diagonal entries (called singular values)

\[
\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0 = \sigma_{r+1} = \ldots = \sigma_n.
\]

Furthermore, the columns of \( U \) (called singular vectors) corresponding to nonzero singular values form an orthogonal basis for the column space of \( A \).

\[
U = \begin{bmatrix} u_1 & \cdots & u_m \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots \\ \vdots & \ddots & \sigma_r & \ddots & \ddots \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}, \quad V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}
\]
Consider the following least squares problem for $A \in \mathbb{R}^{m \times n}$ of rank $r$ where $n \gg m$. The residual $\rho$ is given by

$$\rho = \min_x \|Ax - d\|_2^2 \iff A^T Ax = A^T d.$$ 

(Notice that $Ax$ is in the column space of $A$)

Using the SVD $A = U\Sigma V^T$ and the fact that the first $r$ columns of $U$ span the column space of $A$

$$\rho = \min_y \|U_r y - d\|_2^2 \iff U_r^T U_r y = U_r^T d \iff y = U_r^T d.$$ 

Substituting gives an easy formula for computing the residual

$$\rho = \|U_r U_r^T d - d\|_2^2.$$
Reducing Computation Requirements

\[ \rho = \| U_r U_r^T d - d \|_2^2 \]

1 \leq r \leq 256

The formula for the residual is nice, but for large values of \( r \) computation time and storage requirements are high.

One way to make it more efficient is by approximation.
The following theorem allows us to make the best possible rank $k$ approximation of a matrix $A$. For our purposes $k << r$ (low rank approximation).

**Theorem (SVD Approximation)**

Let $A \in \mathbb{R}^{m \times n}$ be a nonzero matrix with rank $r$. Let $\sigma_1, \ldots, \sigma_r$ be the singular values of $A$, with associated left and right singular vectors $u_1, \ldots, u_r$ and $v_1, \ldots, v_r$, respectively, and let $k \leq r$. Then $A = U\Sigma V^T = \sum_{j=1}^{r} \sigma_j u_j v_j^T$, and $A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^T$ is the best rank $k$ approximation for $A$ under the 2-norm.

What other uses does the theorem have?
SVD Image Compression

- 128 × 128 image.
- Left to right: (top) rank 1, 3, 10, (bottom) 20, 30, 98 (full).
Proof of the SVD Approximation Theorem

Proof.

It’s clear that $A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^T$ has rank $k$. Computing $\|A - A_k\|_2$ we have

$$\|A - A_k\|_2 = \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^T \right\|_2 = \|U\Sigma_{k+1} V^T\|_2 = \|\Sigma_{k+1}\|_2 = \sigma_{k+1}.$$ 

Let $B$ be a rank $k$ $m \times n$ matrix, so it’s null space has dimension $n - k$. The space spanned by $\{v_1, \ldots, v_{k+1}\}$ has dimension $k + 1$. Since $(n - k) + (k + 1) > n$, the intersection of $\mathcal{N}(B)$ and $\{v_1, \ldots, v_{k+1}\}$ must be non-trivial. Let $h$ be a unit vector in their intersection. Then $h = c_1 v_1 + \cdots + c_{k+1} v_{k+1} = V_{k+1} c$ with $\|h\|_2 = 1$.

$$\|A - B\|_2 \geq \|(A - B)h\|_2 = \|Ah\|_2 = \|U\Sigma V^T h\|_2 = \|\Sigma(V^T h)\|_2$$

$$= \|\Sigma(V^T V_{k+1} c)\|_2 = \left\| \Sigma \begin{bmatrix} I_{k+1} \\ 0 \end{bmatrix} c \right\|_2 \geq \sigma_{k+1} \|c\|_2 = \sigma_{k+1}.$$
Developing the SVD Based Algorithm

Corollary

Let \( A \in \mathbb{R}^{m \times n} \) be a nonzero matrix of rank \( r \) with singular value decomposition \( A = U \Sigma V^T \). Then the first \( k < r \) columns \( u_1, ..., u_k \) of \( U \) form an orthogonal basis for the column space of \( A_k \). Furthermore, \( U_k = [u_1 \ u_2 \ ... \ u_k] \) implies \( U_k^T U_k = I \).

Proof.

Let \( A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^T \) and \( y \in \mathbb{R}^n \). Observe that
\[
A_k y = \sum_{j=1}^{k} \sigma_j u_j (v_j^T y).
\]
This means \( u_1, ..., u_k \) form an orthogonal basis for the column space of \( A_k \). For the second part, notice that the rows of matrix \( U_k^T \) are orthogonal to the columns of \( U_k \) which means \( U_k^T U_k = I \).
The Singular Vectors of the Database

\[ u_1, \ldots, u_{10} \text{ for } D_2, D_3, D_5, D_7. \]
Developing the SVD Based Algorithm (cont.)

Corollary

Let $A \in \mathbb{R}^{m \times n}$ be a nonzero matrix of rank $r$ with a rank $k$ approximation $A_k$. The least squares problem $\min_x \| U_k x - d \|_2^2$ has the solution $x = U_k^T d$ with residual $\| U_k U_k^T d - d \|_2^2$. 

Before we used $A$ to find the residual.

$$\rho = \min_x \| Ax - d \|_2^2 \quad \Rightarrow \quad \rho = \| U_r U_r^T d - d \|_2^2$$

(Notice that $Ax$ is in the column space of $A$)

Now we use $A_k$ to approximate the residual.

$$q = \min_x \| A_k x - d \|_2^2 \quad \Rightarrow \quad q = \| U_k U_k^T d - d \|_2^2$$

(Notice that $A_k x$ is in the column space of $A_k$)
Why Do We Use Fixed Low Rank Approximation?

- Reduces the computation time (pre-compute $U_k$).
- Gives a cheap formula for the residual.
- Avoids disasters (some $D_i$ matrices span $\mathbb{R}^{256}$).
- Provides fairness (not all $D_i$ matrices have the same rank).
The SVD Based Algorithm

Let $i \in \{0, 1, \ldots, 9\}$.

Do once at startup:

- Form the $D_i$ matrices for every $i$.
- Compute the SVD of each $D_i$.
- Do a rank $k$ approximation of each $D_i$ and store each $U_{ik}$.

Let $d \in \mathbb{R}^{256}$ be a test digit to classify.

- For every $i$, compute $q_i = \|U_{ik}U_{ik}^T d - d\|_2^2$.
- Compute $\min_i \{q_i\}$ and classify $d$ as an “$i$”.
SVD Based Algorithm Results

The following data are the test results for the SVD based algorithm with a rank approximation of 10 on a sample of 2007 test digits.

<table>
<thead>
<tr>
<th>Digit</th>
<th>Sample Size</th>
<th>Correct</th>
<th>Incorrect</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>359</td>
<td>353</td>
<td>6</td>
<td>98.329%</td>
</tr>
<tr>
<td>1</td>
<td>264</td>
<td>260</td>
<td>4</td>
<td>98.485%</td>
</tr>
<tr>
<td>2</td>
<td>198</td>
<td>179</td>
<td>19</td>
<td>90.404%</td>
</tr>
<tr>
<td>3</td>
<td>166</td>
<td>143</td>
<td>23</td>
<td>86.145%</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>183</td>
<td>17</td>
<td>91.500%</td>
</tr>
<tr>
<td>5</td>
<td>160</td>
<td>145</td>
<td>15</td>
<td>90.625%</td>
</tr>
<tr>
<td>6</td>
<td>170</td>
<td>160</td>
<td>10</td>
<td>94.118%</td>
</tr>
<tr>
<td>7</td>
<td>147</td>
<td>138</td>
<td>9</td>
<td>93.878%</td>
</tr>
<tr>
<td>8</td>
<td>166</td>
<td>149</td>
<td>17</td>
<td>89.759%</td>
</tr>
<tr>
<td>9</td>
<td>177</td>
<td>168</td>
<td>9</td>
<td>94.915%</td>
</tr>
</tbody>
</table>

Average Success Rate: 93.572%. Run time: 76 seconds.
Second Algorithm: Tangent Distance Algorithm
Consider rotating a digit $p$ by an angle $\alpha_p$.

Using the vector form of $p$ (i.e. $p \in \mathbb{R}^{256}$), we can represent all rotations of the digit $p$ by a parameterized curve $s(p, \alpha_p) \subset \mathbb{R}^{256}$ where $\alpha_p$ is the angle of rotation. Notice $s(p, 0) = p$. 
The original distance between the curves and the minimum distance between
the parameterized curves (impossible to compute).

\[
\min_{\alpha_p, \alpha_d} \| s(p, \alpha_p) - s(d, \alpha_d) \|_2^2 \downarrow \mathbb{R}^{256}
\]
Taylor Series Approximation

The equation for the parameterized curve $s(p, \alpha_p)$ is unknown and nonlinear, but can be approximated by Taylor expansion

$$s(p, \alpha_p) = s(p, 0) + \frac{ds}{d\alpha_p}(p, 0)\alpha_p + O(\alpha_p^2) \approx p + t_p\alpha_p$$

where $t_p = \frac{ds}{d\alpha}(p, 0) \in \mathbb{R}^{256}$.

Now consider a test digit $d \in \mathbb{R}^{256}$ to classify and a parameterized curve for it.

$$s(d, \alpha_d) = s(d, 0) + \frac{ds}{d\alpha_d}(d, 0)\alpha_d + O(\alpha_d^2) \approx d + t_d\alpha_d$$

Good Thing: We have linear approximations for $s(p, \alpha_p)$ and $s(d, \alpha_d)$. 
What is Tangent Distance?

The tangent distance is an approximation to the minimum distance between the parameterized curves.

\[
\min_{\alpha_p, \alpha_d} \| s(p, \alpha_p) - s(d, \alpha_d) \|_2^2 \approx \min_{\alpha_p, \alpha_d} \| (p + t_p \alpha_p) - (d + t_d \alpha_d) \|_2^2.
\]

\[s(p, \alpha_p)\]

\[s(d, \alpha_d)\]
We can approximate the distance between the two curves by the tangent distance.

\[
\min_{\alpha_p, \alpha_d} \| s(p, \alpha_p) - s(d, \alpha_d) \|_2^2 \approx \min_{\alpha_p, \alpha_d} \| (p + t_p \alpha_p) - (d + t_d \alpha_d) \|_2^2
\]

\[
= \min_{\alpha_p, \alpha_d} \| (p - d) - (-t_p \quad t_d)(\alpha_p \quad \alpha_d)^T \|_2^2 = t_{pd}.
\]

This is to say that finding the tangent distance \( t_{pd} \) between \( p \) and \( d \) is the same as solving this least squares problem.
Consider doing $k$ transformations (rotation, scaling, translation, ...) on a digit $p$. How will the tangent distance change?

Now $s(p, a_p) \subset \mathbb{R}^{256}$ with $a_p = (\alpha_1 \ \alpha_2 \ \ldots \ \alpha_k)^T$. We can find a multivariate Taylor expansion for $s(p, a_p)$

$$s(p, a_p) = s(p, 0) + \sum_{i=1}^{k} \frac{\partial s}{\partial \alpha_i}(p, 0)\alpha_i + \mathcal{O}(\|a_p\|^2) \approx p + T_p a_p$$

$$T_p = \begin{pmatrix} \frac{\partial s}{\partial \alpha_1} & \frac{\partial s}{\partial \alpha_2} & \ldots & \frac{\partial s}{\partial \alpha_k} \end{pmatrix}$$
Computing Tangent Distance in the Multivariate Case

How does computing the tangent distance change in the multivariate case?

$$\min_{a_p, a_d} \| (p + T_P a_p) - (d + T_d a_d) \|_2^2$$

$$= \min_{a_p, a_d} \| (p - d) - (-T_P \ T_d)(a_p \ a_d)^T \|_2^2 = t_{pd}.$$

It’s still a least squares problem! Now the question is what exactly are $T_P$ and $T_d$?

$$T_P = \left( \begin{array}{cccc} \frac{\partial s}{\partial \alpha_1} & \frac{\partial s}{\partial \alpha_2} & \ldots & \frac{\partial s}{\partial \alpha_k} \end{array} \right)$$

Answer: $T_P$ is a Jacobian matrix consisting of derivatives of transformations evaluated at $(p, 0)$. $T_d$ is the same thing for $d$ at $(d, 0)$. 
Derivatives of Transformations

Let $f(x, y) \in \mathbb{R}$ be a differentiable function such that for a digit matrix $P \in \mathbb{R}^{16 \times 16}$, $f(i, j) = P_{ij}$ for all $i, j \in \{1, 2, \ldots, 16\}$ (e.g. $f(3, 4) = P_{3,4}$).

Let $p \in \mathbb{R}^{16 \times 16}$.

The derivatives of the transformations at $\alpha = 0$ are

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation in the x direction</td>
<td>$f_x$</td>
</tr>
<tr>
<td>Translation in the y direction</td>
<td>$f_y$</td>
</tr>
<tr>
<td>Rotation about the “origin”</td>
<td>$yf_x - xf_y$</td>
</tr>
<tr>
<td>Scaling</td>
<td>$xf_x + yf_y$</td>
</tr>
<tr>
<td>Stretch/compress along the horizontal and vertical axes</td>
<td>$xf_x - yf_y$</td>
</tr>
<tr>
<td>Stretch/compress along the diagonals</td>
<td>$yf_x + xf_y$</td>
</tr>
<tr>
<td>Thickening</td>
<td>$(f_x)^2 + (f_y)^2$</td>
</tr>
</tbody>
</table>
Derivation of the Scaling Derivative

Let \( f(x, y) \) be as before. Scaling is achieved by

\[
  s(p, \alpha_s)(x, y) = f((1 + \alpha_s)x, (1 + \alpha_s)y).
\]

Using the chain rule to differentiate and evaluating at \( \alpha_s = 0 \) we get

\[
  \frac{d}{d\alpha_s} (s(p, \alpha_s)(x, y))|_{\alpha_s=0} = xf_x + yf_y.
\]

The derivation of the other derivatives is mostly the same with the exception of the thickening derivative.
Let \( f(x, y) \) be as before. The thickened image is obtained by defining a new function

\[
g_\alpha(x, y) = \max_{\|r\|<\alpha} f(x + r_1, y + r_2)
\]

where \( r = (r_1, r_2) \) is a vector in \( \mathbb{R}^2 \).

For a complete derivation and discussion see Simard et. al.
How to Compute $f_x$ and $f_y$

- All derivatives can be formed from $f_x$ and $f_y$.
- Compute $f_x$ and $f_y$ using finite differences.
- Use two-sided finite differences in the interior.
- Use one-sided finite differences at the ends.

$$f_x(i,j) \approx \frac{f(i+1,j) - f(i-1,j)}{2}$$

$$f_x(1,j) \approx \frac{f(2,j) - f(1,j)}{1}$$

$$f_x(16,j) \approx \frac{f(16,j) - f(15,j)}{1}$$
Derivatives for Selected Digits

Selected Derivatives of Transformations

Listed Left to Right: Original image, x-translation, y-translation, rotation, scaling, stretch/compress along horizontal/vertical, stretch/compress along diagonals, thickening.
The Tangent Distance Algorithm

Do once at startup:
- Construct the tangent matrix $T_p$ for every $p$ in the database.

Let $d \in \mathbb{R}^{256}$ be a test digit to classify.
- Construct the tangent matrix $T_d$.
- Compute the tangent distance $t_{pd}$ between $d$ and every $p$.
- Find $r = \min_p \{ t_{pd} \}$.
- Classify $d$ as the digit corresponding to the $p$ that gives $r$. 
Tangent Distance Algorithm Results

The following data are the test results for the tangent distance algorithm on a sample of 2007 test digits.

<table>
<thead>
<tr>
<th>Digit</th>
<th>Sample Size</th>
<th>Correct</th>
<th>Incorrect</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>359</td>
<td>289</td>
<td>70</td>
<td>80.501%</td>
</tr>
<tr>
<td>1</td>
<td>264</td>
<td>255</td>
<td>9</td>
<td>96.591%</td>
</tr>
<tr>
<td>2</td>
<td>198</td>
<td>172</td>
<td>26</td>
<td>86.869%</td>
</tr>
<tr>
<td>3</td>
<td>166</td>
<td>145</td>
<td>21</td>
<td>87.349%</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>145</td>
<td>55</td>
<td>72.500%</td>
</tr>
<tr>
<td>5</td>
<td>160</td>
<td>143</td>
<td>17</td>
<td>89.375%</td>
</tr>
<tr>
<td>6</td>
<td>170</td>
<td>161</td>
<td>9</td>
<td>94.706%</td>
</tr>
<tr>
<td>7</td>
<td>147</td>
<td>137</td>
<td>10</td>
<td>93.197%</td>
</tr>
<tr>
<td>8</td>
<td>166</td>
<td>130</td>
<td>36</td>
<td>78.313%</td>
</tr>
<tr>
<td>9</td>
<td>177</td>
<td>166</td>
<td>11</td>
<td>93.785%</td>
</tr>
</tbody>
</table>

Average Success Rate: 86.846%. Run time: 25.5 hours. (7291 × 2007 = 14,633,037).
Closing Remarks
Summary of Results

Below are the test results for both algorithms. \(^2\)

**SVD Based Algorithm with Rank Approximation of 10:**
- Accuracy: 93.5%
- Run time: 76 seconds
- Suited for real time.

**Tangent Distance Algorithm:**
- Accuracy: 86.8% (91%)
- Run time: 25.5 hours (18 hours)
- Suited for “tie-breaking” (smallest \(\rho_i\) is close to another).

Omitting the thickening transformation yielded the numbers in parentheses.

\(^2\) The testing platform was a 2.4 GHz AMD Athlon 64 X2 processor machine with 2 GB of memory running Debian GNU/Linux. The software used to test the algorithms was Octave 3.0 running on a single core.
Further reading about the two algorithms.


The End!