Handwriting Recognition Technology in the Newton's Second Generation “Print Recognizer” (The One That Worked)

Larry Yaeger
Professor of Informatics, Indiana University
Distinguished Scientist, Apple Computer

World Wide Newton Conference
September 4-5, 2004
Handwriting Recognition Team

Core Team
Larry Yaeger (ATG)  Brandyn Webb (Contractor)
Dick Lyon (ATG)  Les Vogel (Contractor)
Bill Stafford (ATG)

Other Contributors
Rus Maxham  Kara Hayes  Gene Ciccarelli  Stuart Crawford
Chris Hamlin  George Mills  Dan Azuma  Boris Aleksandrovsky
Josh Gold  Michael Kaplan  Ernie Beernink  Giulia Pagallo

Testers
Polina Fukshansky  Glen Raphael  Julie Wilson  Emmanuel Euren
Ron Dotson  Denny Mahdik

WWNC 2004  2
Recognizer History

- ’92 ATG “Rosetta” project demos well at Stewart Alsop’s “Demo ’92” (blows socks off Nathan Myhrvold’s MS demo) and WWDC
- ’93 Head of ATG suggests abandoning handwriting recognition for interactive TV project
- ’93-’94 Rosetta nearly ships in “PenLite” pen-based Mac product
- Jan ’94 Port to Newton started
- ’94 Brief interest in Rosetta for abortive “Nautilus” Mac product
- ... testing with tethered Newtons, much accuracy improvement...
- 18 Nov ’94 Provided handful of untethered Newtons for testing
- 1 Feb ’95 Beta 1 build (Merry Xmas!)
- ’95 Rosetta ships as “Print Recognizer” in Newton (120?)
- ’95 Rosetta widely acknowledged as world’s first usable handwriting recognizer
Recognizer History

- 13 Nov ’95 John Markoff writes about Rosetta in NY Times
- Nov or Dec ’95 receive cease-and-desist demand for use of "Rosetta" name (Mac-based SmallTalk platform)
- Jan ’96 team picks “Mondello” codename, “Neuropen” product name
- ’96 Short-lived “Hollywood” pen-based Mac project
- Mar ’97 cursive almost working
- 18 Mar ’97 ATG laid off
- May ’00 “Inkwell” for Mac OS 9 declares beta
- May ’00 Marketing declares “no new features on 9”, OS X work begins
- Jul ’02 Inkwell for Mac OS X declares GM (10.2 / Jaguar)
- Sep ’03 Inkwell APIs and additional languages declare GM (10.3 / Panther)
- Apr ’04 Motion announced with gestural interface, including tablet and in-air ink-on-demand
Recognizer Overview

- Powerful state-of-the-art technology
  - Neural network character classifier
  - Maximum-likelihood search over letter segmentation, letter class, word, and word segmentation hypotheses
  - Flexible, loosely applied language model with very broad coverage
- Now part of “Inkwell” in Mac OS X
- Also provides gesture recognition
  - System
  - Application (Motion)
Recognition Block Diagram

(x,y) points & pen-lifts

Tentative Segmentation

Neural Network Classifier

Beam Search With Context

word probabilities

character segmentation hypotheses

character class hypotheses

WWNC 2004
Character Segmentation

- Which strokes comprise which characters?
- Constraints
  - All strokes must be used
  - No strokes may be used twice
- Efficient pre-segmentation
  - Avoid trying all possible permutations
  - Based on order, overlap, crossings, aspect ratio...
- Integrated with recognition
  - Forward & reverse “delays” implement implicit graph of hypotheses
Neural Network Character Classifier

- Inherently data-driven
- Learn from examples
- Non-linear decision boundaries
- Effective generalization
Context Is Essential

- Humans achieve 90% accuracy on characters in isolation (our database)
  - Word accuracy would then be ~ 60% ($0.9^5$)
- Variety of context models are possible
  - N-grams
  - Variable (Memory) Length Markov Model
  - Word lists
  - Regular expression graphs
- "Out of dictionary" writing also required
  - "xyzzy", unix pathnames, technical/medical terms, etc.
Recognition Technology

Tentative Segmentation → Neural Network Classifier → Beam Search With Context

(x,y) points & pen-lifts

(word probabilities)

(character segmentation hypotheses)

(character class hypotheses)

(clog)
### Character Segmentation

<table>
<thead>
<tr>
<th>Ink</th>
<th>Segment Number</th>
<th>Segment</th>
<th>Stroke Count</th>
<th>Forward Delay</th>
<th>Reverse Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c</td>
<td></td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>d</td>
<td></td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>d o</td>
<td></td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>d o</td>
<td></td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>o</td>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>o</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>g</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ i \rightarrow j \text{ is legal } \iff \text{ FD}_i + \text{ RD}_j = j - i \]

WWNC 2004
Network Design

- Variety of architectures tried
  - Single hidden layer, fully-connected
  - Multi-hidden layer, with receptive fields
  - Shared weights (LeCun)
  - Parallel classifiers combined at output layer
- Representation as important as architecture
  - Anti-aliased images
  - Baseline-driven with ascenders and descenders
  - Stroke features

WWNC 2004  12
Network Architectures
Neural Network Classifier

Stroke Count
Aspect Ratio
Image

Stroke Feature

14 x 14
1 x 1
5 x 1
95 x 1
104 x 1
112 x 1
72 x 1
20 x 9
7 x 7
7 x 2
7 x 7
5 x 5
(8x8)
(8x7;1,7) (7x8;7,1)
(8x6;1,8) (6x8;8,1)
(10x10)
(14x6)
(6x14)
(8x6)
(8x7;1,7) (7x8;7,1)
Normalizing Output Error

- Normalize “pressure towards zero”
- Based on recognition of the fact that most training signals are zero
- Training vector for letter "x"
  a ... w x y z A ... Z 0 ... 9 ! ... ~
  0 ... 0 1 0 0 0 ... 0 0 ... 0 0 ... 0
- Forces net to attempt to make unambiguous classifications
- Makes it difficult to obtain meaningful 2nd and 3rd choice probabilities
Normalized Output Error

- We reduce the BP error for non-target classes relative to the target class
- By a factor that "normalizes" the non-target error relative to the target error
- Based on the number of non-target vs. target classes
- For non-target output nodes
  \[ e' = e A \]
  where \[ A = \frac{1}{d (N_{outputs} - 1)} \]
- Allocates network resources to modeling of low-probability regime
Normalized Output Error

- Converges to MMSE estimate of $f( P(\text{class|input}), A )$
- We derived that function:
  $$\langle \hat{e}^2 \rangle = p (1-y)^2 + A (1-p) y^2$$
  where $p = P(\text{class|input})$, $y = \text{output unit activation}$
- Output $y$ for particular class is then:
  $$y = p / (A - A p + p)$$
- Inverting for $p$:
  $$p = y A / (y A - y + 1)$$
Normalized Output Error

Empirical \( p \) vs. \( y \) histogram for a net trained with \( A=0.11 \) (\( d=0.1 \)), with corresponding theoretical curve
Normalized Output Error

NormOutErr =

- 0.0
- 0.8

Error (%)

Character Error

Word Error

NormOutErr =

Error (%)

9.5 12.4

31.6 22.7

WWNC 2004 19
Stroke Warping

- Produce random variations in stroke data during training
- Small changes in skew, rotation, x and y linear and quadratic scaling
- Consistent with stylistic variations
- Improves generalization by effectively adding extra data samples
Stroke Warping

- Original
- Rotation
- X Linear
- X Quadratic
- X Skew
- Y Linear
Class Frequency Balancing

- Skip and repeat patterns
- Instead of dividing by the class priors
  - Eliminates noisy estimate of low freq. classes
  - Eliminates need for renormalization
  - Forces net to better model low freq. classes
- Compute normalized frequency, relative to average frequency
  \[ F_i = \frac{S_i}{\bar{S}} \]
  \[ \bar{S} = \frac{1}{c} \sum_{i=1}^{c} S_i \]
Class Frequency Balancing

- Compute repetition factor
  \[ R_i = \left( \frac{a}{F_i} \right)^b \]

- Where \( a \) (0.2 to 0.8) controls amount of skipping vs. repeating

- And \( b \) (0.5 to 0.9) controls amount of balancing
Stroke-Count Frequency Balancing

- Compute frequencies for stroke-counts in each class
- Modulate repetition factors by stroke-count sub-class frequencies

\[ R_{ij} = R_i \left[ \frac{(S_i/J)}{S_{ij}} \right]^b \]
Adding Noise to Stroke-Count

- Small percentage of samples use randomly selected stroke-count (as input to the net)
- Improves generalization by reducing bias towards observed stroke-counts
- Even improves accuracy on data drawn from training set
Negative Training

- Inherent ambiguities force segmentation code to generate false segmentations
- Ink can be interpreted in various ways...

```
clog
```

- "dog", "clog", "cbg", "%g"
- Train network to compute low probabilities for false segmentations
Negative Training

- Modulate negative training two ways...
  - Negative error factor (0.2 to 0.5)
    - Like A in normalized output error
  - Negative training probability (0.05 to 0.3)
    - Also speeds training
- Too much negative training
  - Suppresses net outputs for characters that look like elements of multi-stroke characters (I, 1, l, |, o, O, 0)
  - Slight reduction in character accuracy, large gain in word accuracy
Error Emphasis

- Probabilistically skip training for correctly classified patterns
- Never skip incorrectly classified patterns
- Just one form of error emphasis
  - Can reduce learning rate/error for correctly classified patterns
  - And/or increase learning rate/error for incorrectly classified patterns
  - Maintain pool of samples from which correctly classified patterns are flushed each epoch
## Training Probabilities and Error Factors

<table>
<thead>
<tr>
<th>Segment</th>
<th>Type</th>
<th>Prob. of Usage</th>
<th>Error Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POS</td>
<td>Correct 0.5</td>
<td>Target Class 1.0</td>
</tr>
<tr>
<td></td>
<td>POS</td>
<td>Incorrect 1.0</td>
<td>Other Classes 0.1</td>
</tr>
<tr>
<td></td>
<td>NEG</td>
<td>Correct 0.18</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>NEG</td>
<td>Incorrect NA</td>
<td>Other Classes 0.3</td>
</tr>
</tbody>
</table>
Annealing & Scheduling

- Start with large learning rate, then decay
  - When training set's total squared error increases
- Start with high error emphasis, then decay
- Start with minimal negative training, then increase
  - Mostly for pragmatic reasons
## Training Schedule

<table>
<thead>
<tr>
<th>Phase</th>
<th>Epochs</th>
<th>Learning Rate</th>
<th>Correct Train Prob</th>
<th>Negative Train Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>1.0 - 0.5</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>0.5 - 0.1</td>
<td>0.25</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>0.1 - 0.01</td>
<td>0.5</td>
<td>0.18</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.01 - 0.001</td>
<td>1.0</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Quantized Weights

- Forward/classification pass requires less precision than backward/learning pass
- Use one-byte weights for classification
  - Saves both space and time
  - $\pm 3.4$ (-8 to +8 with 1/16 Steps)
- Use three-byte weights for learning
  - $\pm 3.20$
- First Newton version
  - ~200KB ROM (~85KB for weights)
  - ~5KB-100KB RAM
  - ~3.8 char/second
Quantized Weights

count per bin of width 1/16

weight value \( w \)
User Adaptation

- Neural net classifier based on an inherently learning technology
- Learning not used in Newton due to memory constraints
- Learning not (yet) used in Mac OS X due to limited human resources
- Can reduce error rates by factor of 2 to 5, yet user-independent "walk-up" performance is maintained!
User Adaptation

- User training scenario
  - 15-20 min. of data entry
    - Less for problem characters alone
  - Possibly < 1 min. network learning
    - One-shot learning may suffice (single epoch)
    - May learn during data entry
    - Maximum of a few minutes (~10-12 Epochs)

- Learn on the fly
  - Can continuously adapt
  - Need system hooks
  - Choosing what to train on is key system issue
User Adaptation

<table>
<thead>
<tr>
<th>User-Independent</th>
<th>User-Specific</th>
<th>User-Adapted</th>
<th>Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.8</td>
<td>6.1</td>
<td>21.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.3</td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.7</td>
<td>11.7</td>
<td></td>
</tr>
<tr>
<td>13.9</td>
<td>5.1</td>
<td>17.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.7</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.7</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>24.8</td>
<td>5.1</td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.7</td>
<td>11.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.7</td>
<td>11.7</td>
<td></td>
</tr>
<tr>
<td>13.9</td>
<td>5.1</td>
<td>17.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.7</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.7</td>
<td>6.3</td>
<td></td>
</tr>
</tbody>
</table>
Integration with Character Segmentation

- Search takes place over segmentation hypotheses (as well as character hypotheses)
- Stroke recombinations are presented in regular, predictable order
- Forward and reverse "delay" parameters suffice to indicate legal time-step transitions
Integration with Word Segmentation

- Search also takes place over word segmentation hypotheses
- Word-space becomes an optional segment/character
- Weighted by probability ("SpaceProb") derived from statistical model of gap sizes and stroke centroid spacing
- Non-space hypotheses are weighted by 1-SpaceProb
Word Segmentation
Statistical Model

\[ P_{\text{WordBreak}} = \frac{\Gamma_{\text{WordGap}}}{(\Gamma_{\text{StrokeGap}} + \Gamma_{\text{WordGap}})} \]
Recognition Ambiguity

scuba
Geometric Context

“if” from User vs Table

Error vector of eight differences

(User data scaled and offset to minimize error magnitude)
Language Model

- Dictionaries
  - Word lists
  - Regular expression grammars
- BiGrammars - combinations of dictionaries
  - Probabilistically weighted
  - Flexible starts, stops, and transitions
Regular Expression Grammars

Telephone numbers example:

\[
\begin{align*}
\text{dig} & = [0123456789] \\
\text{digm01} & = [23456789] \\
\text{acodenums} & = (\text{digm01 [01] dig}) \\
\text{acode} & = \{ ("1-"? \text{acodenums }"-"):40, \\
& \quad ("1"? "(" \text{acodenums }")):60 \} \\
\text{phone} & = (\text{acode? digm01 dig dig }"-" \text{ dig dig dig dig dig})
\end{align*}
\]
Bigrammars

- Limited context telephone example:

  BiGrammar2 Phone

  [phone.lang 1. 1. 1.]
BiGrammars

(Fairly) general context example:

BiGrammar2 FairlyGeneral
[EndPunct.lang 0. 0.9 0.5 EndPunct.lang 1.]
(.8
 (.6
 [WordList.dict 0.5 0.8 1. EndPunct.lang 0.2]
 [User.dict 0.5 0.8 1. EndPunct.lang 0.2]
 )
 (.4
 [Phone.lang 0.5 0.8 1. EndPunct.lang 0.2]
 [Date.lang 0.5 0.8 1. EndPunct.lang 0.2]
 )
)
(.2
 [OpenPunct.lang 0. 0. 1.]
 (.6
 WordList.dict 0.5
 User.dict 0.5
 )
 (.4
 Phone.lang 0.5
 Date.lang 0.5
 )
)
]
when Year-old Arabian retire tipped off the Christmas wrapping, No (square with delights) Santa brought the Attacking hit too dathe would. Problem was, Joe talked Bobbie. His doll stones at the really in its army Antiques I machine gun and hand decades At its side. But it says things like 3 "Want to go shopping" The Pro has claimed responsibility that's Bobbie Liberation Organization. Make up of more than 50 Concerned parents & Machinist 5 and other activists 5 the Pro claims to have crop if Housed switched the voice boxes on 300 hit, Joe and Bobbie foils across the United States this holiday Season "we have operations All over the country" said one pro member 5 who wished to remain autonomous. "Our goal is to cereal and correct Thu problem of exposed stereo in editorials toys."
Mondello Writing Example

When 7-year-old Zachariah Zelin ripped off the Christmas wrapping, he squealed with delight. Santa brought the talking G.I. Joe doll he wanted. Problem was, Joe talked like Barbie. His doll stands at the ready in its Army fatigues, machine gun and hand grenades at its side. But it says things like, "Want to go shopping?" The BLO has claimed responsibility. That's Barbie Liberation Organization. Made up of more than 50 concerned parents, feminists and other activists, the BLO claims to have surreptitiously switched the voice boxes on 300 G.I. Joe and Barbie dolls across the United States this holiday season. "We have operatives all over the country," said one BLO member, who wished to remain anonymous. "Our goal is to reveal and Correct the problem of gender-based stereotyping in children's toys."
Apple-Newton
Handwriting Recognition

The Power of the Best