Combining Neural Networks and Context-Driven Search for On-Line, Printed Handwriting Recognition in the Newton

Larry S. Yaeger
Apple Computer
5540 Bittersweet Rd.
Beanblossom, IN 46160
larryy@pobox.com

Brandyn J. Webb
The Future
4578 Fieldgate Rd.
Oceanside, CA 92056
brandyn@sifter.org

Richard F. Lyon
Foveonics, Inc.
10131-B Bubb Rd.
Cupertino, CA 95014
dicklyon@acm.org

Abstract

While on-line handwriting recognition is an area of long-standing and ongoing research, the recent emergence of portable, pen-based computers has focused urgent attention on usable, practical solutions. We discuss a combination and improvement of classical methods to produce robust recognition of hand-printed English text, for a recognizer shipping in new models of Apple Computer's Newton MessagePad® and eMate®. Combining an artificial neural network (ANN), as a character classifier, with a context-driven search over segmentation and word recognition hypotheses provides an effective recognition system. Long-standing issues relative to training, generalization, segmentation, models of context, probabilistic formalisms, etc., need to be resolved, however, to achieve excellent performance. We present a number of recent innovations in the application of ANNs as character classifiers for word recognition, including integrated multiple representations, normalized output error, negative training, stroke warping, frequency balancing, error emphasis, and quantized weights. User-adaptation and extension to cursive recognition pose continuing challenges.

1 INTRODUCTION

Pen-based hand-held computers are heavily dependent upon fast and accurate handwriting recognition, since the pen serves as the primary means for inputting data to such devices. Some earlier attempts at handwriting recognition have utilized strong, limited language models to maximize accuracy. However, this approach has proven to be unacceptable in real-world applications, generating disturbing and seemingly random word substitutions—known colloquially within Apple and Newton as "The Doonesbury Effect", due to Gary Trudeau's satirical look at first-generation Newton recognition performance. But the original handwriting recognition technology in the Newton, and the current, much-improved "Cursive Recognizer" technology, both of which were licensed from ParaGraph International, Inc., are not the subject of this article.

In Apple's Advanced Technology Group (aka Apple Research Labs), we pursued a different approach, using bottom-up classification techniques based on trainable artificial neural networks (ANNs), in combination with comprehensive but weakly-applied language models. To focus our work on a subproblem that was tractable enough to lead to usable products in a reasonable time, we initially restricted the domain to hand-printing, so that strokes are clearly delineated by pen lifts. By simultaneously providing accurate character-level recognition, dictionaries exhibiting very wide coverage of the language, and the ability to write entirely outside those dictionaries, we have produced a hand-print recognizer that some have called the "first usable" handwriting recognition system. The ANN character classifier required some innovative training techniques to perform its task well. The dictionaries required large word lists, a regular expression grammar (to describe special constructs such as date, time, phone numbers, etc.), and a means of combining all these dictionaries into a comprehensive language model. And well balanced prior probabilities had to be determined for in-dictionary and out-of-dictionary writing. Together with a maximum-likelihood search engine, these elements form the basis of the so-called "Print Recognizer", that was first shipped in Newton OS 2.0 based MessagePad 120 units in December, 1995, and has shipped in all subsequent Newton devices.

In the most recent unit, the MessagePad 2000, despite retaining its label as a "Print Recognizer", it has been extended to handle connected characters (as well as a full Western European character set).

There is ample prior work in combining low-level classifiers with dynamic time warping, hidden Markov models, Viterbi algorithms, and other search strategies to provide integrated segmentation and recognition for writing (Tappert *et al* 1990) and speech (Renals *et al* 1992). And there is a rich background in the use of ANNs as classifiers, including their use as low-level character classifiers in a higher-level word recognition system (Bengio *et al* 1995). But these approaches leave a large number of open-ended questions about how to achieve acceptable (to a real user) levels of performance. In this paper, we survey some of our experiences in exploring refinements and improvements to these techniques.

2 SYSTEM OVERVIEW

Apple's print recognizer (APR) consists of three conceptual stages—Tentative Segmentation, Classification, and Context-Driven Search—as indicated in Figure 1. The primary data upon which we operate are simple sequences of (*x*,*y*) coordinate pairs, plus pen-up/down information, thus defining stroke primitives. The Segmentation stage decides which strokes will be combined to produce *segments*—the tentative groupings of strokes that will be treated as possible characters—and produces a sequence of these segments together with legal transitions between them. This process builds an implicit graph which is then labeled in the Classification stage and examined for a maximum likelihood interpretation in the Search stage. The Classification stage evaluates each segment using the ANN classifier, and produces a vector of output activations that are used as letter-class probabilities. The Search stage then uses these class probabilities together with models of lexical and geometric context to find the *N* most likely word or sentence hypotheses.

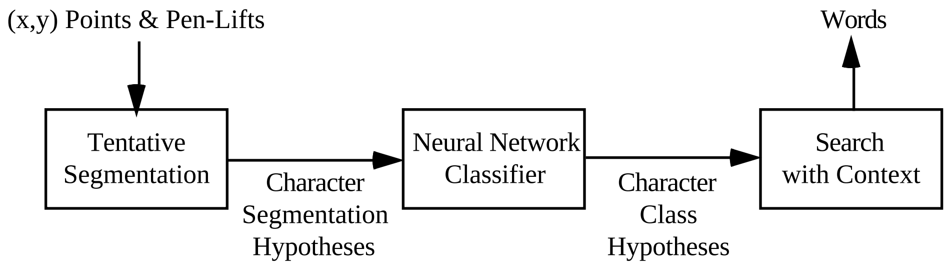

(x,y) Points & Pen-Lifts
Words
Tentative Segmentation
Character Segmentation Hypotheses
Neural Network Classifier
Character Class Hypotheses
Search with Context

Figure 1: A simplified block diagram of our hand-print recognizer.

3 TENTATIVE SEGMENTATION

Character segmentation—the process of deciding which strokes comprise which characters—is inherently ambiguous. Ultimately this decision must be made, but, short of writing in boxes, it is impossible to do so (with any accuracy) in advance, external to the recognition process. Hence the initial segmentation stage in APR produces multiple, tentative groupings of strokes, and defers the final segmentation decisions until the search stage, thus integrating those segmentation decisions with the overall recognition process.

APR uses a potentially exhaustive, sequential enumeration of stroke combinations to generate a sequence of viable character-segmentation hypotheses. These *segments* are subjected to some obvious constraints (such as “all strokes must be used” and “no strokes may be used twice”), and some less obvious filters (to cull “impossible” segmentations for the sake of efficiency). The resulting algorithm produces the actual segments that will be processed as possible characters, along with the legal transitions between these segments.

The legal transitions are defined by *forward* and *reverse delays*. The forward delay indicates the next possible segment in the sequence. The reverse delay indicates the start of the current batch of segments, all of which share the same leading stroke. Due to the enumeration scheme, a segment's reverse delay is the same as its stroke count minus one, unless preceding segments (sharing the same leading stroke) were eliminated by the filters mentioned previously. These two simple delay parameters (per segment) suffice to define an implicit graph of all legal segment transitions. For a transition from segment number *i* to segment number *j* to be legal, the sum of segment *i*'s forward delay plus segment *j*'s reverse delay must be equal to *j* - *i*. Figure 2 provides an example of some ambiguous ink and the segments that might be generated from its strokes, supporting interpretations of “dog”, “clog”, “cbg”, or even “%g”.

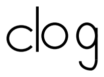
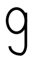
Ink	Segment Number	Segment	Stroke Count	Forward Delay	Reverse Delay
	1		1	3	0
	2		2	4	1
	3		3	4	2
	4		1	2	0
	5		2	2	1
	6		1	1	0
	7		1	0	0

Figure 2: Segmentation of strokes into tentative characters, or segments.

4 CHARACTER CLASSIFICATION

The output of the segmentation stage is a stream of segments that are then passed to an ANN for classification as characters. Except for the architecture and training specifics detailed below, a fairly standard multi-layer perceptron trained with error back-propagation (BP) provides the ANN character classifier at the heart of APR. A large body of prior work exists to indicate the general applicability of ANN technology as a classifier providing good estimates of *a posteriori* probabilities of each class given the input (Gish 1990, Richard and Lippman 1991, Renals and Morgan 1992, and others cited herein). Compelling arguments have been made for why ANNs providing posterior probabilities in a probabilistic recognition formulation should be expected to outperform other recognition approaches (Lippman 1994), and ANNs have performed well as the core of speech recognition systems (Morgan and Bourlard 1995).

4.1 REPRESENTATION

A recurring theme in ANN research is the extreme importance of the representation of the data that is given as input to the network. We experimented with a variety of input representations, including stroke features both anti-aliased (gray-scale) and not (binary), and images both anti-aliased and not, and with various schemes for positioning and scaling the ink within the image input window. In every case, anti-aliasing was a significant win. This is consistent with others' findings, that ANNs perform better when presented with smoothly varying, distributed inputs than they do when presented with binary, localized inputs. Almost the simplest image representation possible, a non-aspect-ratio-preserving, expand-to-fill-the-window image (limited only by a maximum scale factor to keep from blowing dots up to the full window size), together with either a single unit or a thermometer code (some number of units turned on in sequence to represent larger values) for the aspect ratio, proved to be the most effective single-classifier solution. However, the best overall classifier accuracy was ultimately obtained by combining multiple distinct representations into nearly independent, parallel classifiers, joined at a final output layer. Hence representation proved not only to be as important as architecture, but, ultimately, to help define the architecture of our nets. For our final, hand-optimized system, we utilize four distinct inputs, as indicated in Table 1. The stroke count representation was dithered (changed randomly at a small probability), to expand the effective training set, prevent the network from fixating on this simple input, and thereby improve the network's ability to generalize. A schematic of the various input representations can be seen as part of the architecture drawing in Figure 3 in the next section.

Table 1: Input representations used in APR.

Input Feature	**Resolution**	**Description**
Image	14x14	anti-aliased, scale-to-window, scale-limited
Stroke	20x9	anti-aliased, limited resolution tangent slope, resampled to fixed number of points
Aspect Ratio	1x1	normalized and capped to [0,1]
Stroke Count	5x1	dithered thermometer code

4.2 ARCHITECTURE

As with representations, we experimented with a variety of architectures, including simple fully-connected layers, receptive fields, shared weights, multiple hidden layers, and, ultimately, multiple nearly independent classifiers tied to a common output layer. The final choice of architecture includes multiple input representations, a first hidden layer (separate for each input representation) using receptive fields, fully connected second hidden layers (again distinct for each representation), and a final, shared, fully-connected output layer. Simple scalar features—aspect ratio and stroke count—connect to both second hidden layers. The final network architecture, for our original English-language system, is shown in Figure 3.

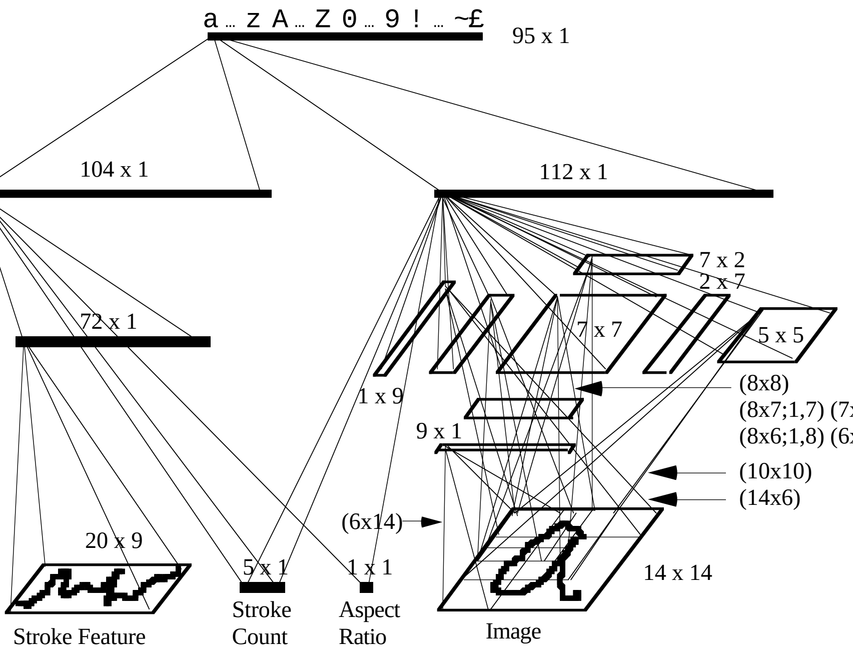


Figure 3: Final English-language net architecture. (See the text for an explanation of the notation.)

Layers are fully connected, except for the inputs to the first hidden layer on the image side. This first hidden layer on the image side consists of 8 separate grids, each of which accepts inputs from the image input grid with its own receptive field sizes and strides, shown parenthetically in Figure 3 as (*x*-size x *y*-size; *x*-stride, *y*-stride). A *stride* is the number of units (pixels) in the input image space between sequential positionings of the receptive fields, in a given direction. The 7x2 and 2x7 side panels (surrounding the central 7x7 grid) pay special attention to the edges of the image. The 9x1 and 1x9 side panels specifically examine full-size vertical and horizontal features, respectively. The 5x5 grid observes features at a different spatial scale than the 7x7 grid.

Combining the two classifiers at the output layer, rather than, say, averaging the outputs of completely independent classifiers, allows generic BP to learn the best way to combine them, which is both convenient and powerful. But our *integrated multiple-representations* architecture is conceptually related to and motivated by prior experiments at combining nets such as Steve Nowlan's “mixture of experts” (Jacobs *et al* 1991).

4.3 NORMALIZING OUTPUT ERROR

Analyzing a class of errors involving words that were misrecognized due to perhaps a single misclassified character, we realized that the net was doing a poor job of representing second and third choice probabilities. Essentially, the net was being forced to attempt unambiguous classification of intrinsically ambiguous patterns due to the nature of the mean squared error minimization in BP, coupled with the typical training vector which consists of all 0's except for the single 1 of the target. Lacking any viable means of encoding legitimate probabilistic ambiguity into the training vectors, we decided to try “normalizing” the “pressure towards 0” vs. the “pressure towards 1” introduced by the output error during training. We refer to this technique as *NormOutErr*, due to its normalizing effect on target versus non-target output error.

We reduce the BP error for non-target classes relative to the target class by a factor that normalizes the total non-target error seen at a given output unit relative to the total target error seen at that unit. Assuming a training set with equal representation of classes, this normalization should then be based on the number of non-target versus target classes in a typical training vector, or, simply, the number of output units (minus one). Hence for *non-target* output units, we scale the error at each unit by a constant:

$$e' = Ae$$

where *e* is the error at an output unit, and *A* is defined to be:

$$A = 1 \big/ \left[d(N_{outputs} - 1) \right]$$

where $N_{outputs}$ is the number of output units, and d is our tuning parameter, typically ranging from 0.1 to 0.2. Error at the *target* output unit is unchanged. Overall, this raises the activation values at the output units, due to the reduced pressure towards zero, particularly for low-probability samples. Thus the learning algorithm no longer converges to a least mean-squared error (LMSE) estimate of $P(class|input)$, but to an LMSE estimate of a nonlinear function $f(P(class|input), A)$ depending on the factor *A* by which we reduced the error pressure toward zero.

Using a simple version of the technique of Bourlard and Wellekens (1990), we worked out what that resulting nonlinear function is. The net will attempt to converge to minimize the modified quadratic error function

$$\left\langle \tilde{E}^2 \right\rangle = p(1-y)^2 + A(1-p)y^2$$

by setting its output *y* for a particular class to

$$y = p \big/ \left(A - Ap + p \right)$$

where $p = P(class|input)$, and *A* is as defined above. For small values of *p*, the activation *y* is increased by a factor of nearly 1/*A* relative to the conventional case of $y = p$, and for high values of p the activation is closer to 1 by nearly a factor of *A*. The inverse function, useful for converting back to a probability, is

$$p = yA \big/ \left(yA + 1 - y \right)$$

We verified the fit of this function by looking at histograms of character-level empirical percentage-correct versus *y*, as in Figure 4.

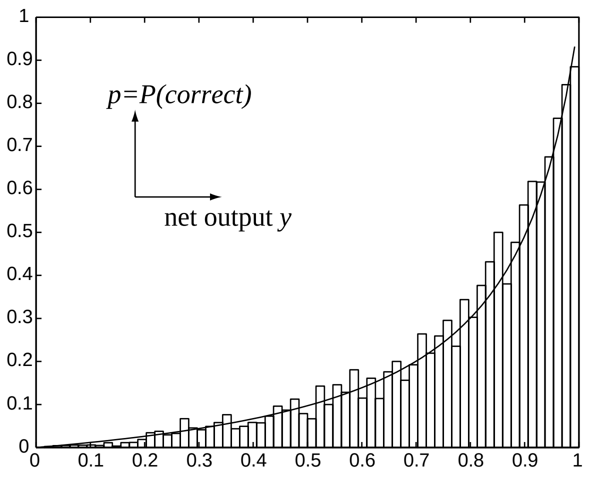


Figure 4: Empirical *p vs. y* histogram for a net trained with *A*=0.11 (*d*=.1), with the corresponding theoretical curve.

Even for this moderate amount of output error normalization, it is clear that the lower-probability samples have their output activations raised significantly, relative to the 45˚ line that *A* = 1 yields.

The primary benefit derived from this technique is that the net does a much better job of representing second and third choice probabilities, and low probabilities in general. Despite a small drop in top choice character accuracy when using NormOutErr, we obtain a very significant increase in word accuracy by this technique. Figure 5 shows an exaggerated example of this effect, for an atypically large value of *d* (0.8), which overly penalizes character accuracy; however, the 30% decrease in word error rate is normal for this technique. (Note: These data are from a multi-year-old experiment, and are not necessarily representative of current levels of performance on any absolute scale.)

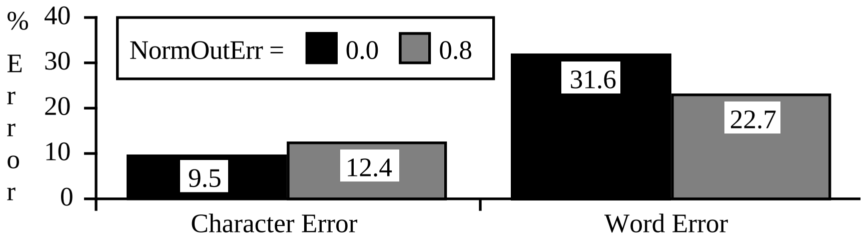


Figure 5: Character and word error rates for two different values of NormOutErr (*d*). A value of 0.0 disables NormOutErr, yielding normal BP. The unusually high value of 0.8 (*A*=0.013) produces nearly equal pressures towards 0 and 1.

4.4 NEGATIVE TRAINING

The previously discussed inherent ambiguities in character segmentation necessarily result in the generation and testing of a large number of invalid segments. During recognition, the network must classify these invalid segments just as it would any valid segment, with no knowledge of which are valid or invalid. A significant increase in word-level recognition accuracy was obtained by performing *negative training* with these invalid segments. This consists of presenting invalid segments to the net during training, with all-zero target vectors. We retain control over the degree of negative training in two ways. First is a *negative-training factor* (ranging from 0.2 to 0.5) that modulates the learning rate (equivalently by modulating the error at the output layer) for these negative patterns. This reduces the impact of negative training on positive training, thus modulating the impact on characters that specifically look like elements of multi-stroke characters (e.g., I, 1, l, o, O, 0). Secondly, we control a *negative-training probability* (ranging between 0.05 and 0.3), which determines the probability that a particular negative sample will actually be trained on (for a given presentation). This both reduces the overall impact of negative training, and significantly reduces training time, since invalid segments are more numerous than valid segments. As with NormOutErr, this modification hurts character-level accuracy a little bit, but helps word-level accuracy a lot.

4.5 STROKE WARPING

During training (but not during recognition), we produce random variations in stroke data, consisting of small changes in skew, rotation, and *x* and *y* linear and quadratic scalings. This produces alternate character forms that are consistent with stylistic variations within and between writers, and induces an explicit aspect ratio and rotation invariance within the framework of standard back-propagation. The amounts of each distortion to apply were chosen through cross-validation experiments, as just the amount needed to yield optimum generalization. (*Cross-validation* is a standard technique for *early stopping* of ANN training, to prevent over-learning of the training set, and thus reduced accuracy on new data outside that training set. The technique consists of keeping aside some subset of the available data, the cross-validation set, and testing on it at some interval, but never training on it, and then stopping the training when accuracy ceases to improve on this cross-validation set, despite the fact that accuracy might continue to improve on the training set.) We chose relative amounts of the various transformations by testing for optimal final, converged accuracy on a cross-validation set. We then increased the amount of all stroke warping being applied to the training set, just to the point at which accuracy on the training set ceased to diverge from accuracy on the cross-validation set.

We also examined a number of such samples by eye to verify that they represent a natural range of variation. A small set of such variations is shown in Figure 6.

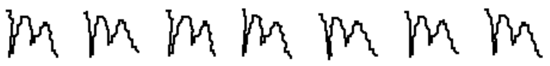


Figure 6: A few random stroke warpings of the same original “m” data.

Our stroke warping scheme is somewhat related to the ideas of Tangent Dist and Tangent Prop (Simard *et al* 1992, 1993), in terms of the use of predetermined families of transformations, but we believe it is much easier to implement. It is also somewhat distinct in applying transformations on the original coordinate data, as opposed to using distortions of images. The voice transformation scheme of Chang and Lippmann (1995) is also related, but they use a static replication of the training set through a small number of transformations, rather than dynamic random transformations of an essentially infinite variety.

4.6 FREQUENCY BALANCING

Training data from natural English words and phrases exhibit very non-uniform priors for the various character classes, and ANNs readily model these priors. However, as with NormOutErr, we find that reducing the effect of these priors on the net, in a controlled way, and thus forcing the net to allocate more of its resources to low-frequency, low-probability classes is of significant benefit to the overall word recognition process. To this end, we explicitly (partially) balance the frequencies of the classes during training. We do this by probabilistically skipping and repeating patterns, based on a precomputed *repetition factor*. Each presentation of a repeated pattern is "warped" uniquely, as discussed previously.

To compute the repetition factor for a class *i*, we first compute a normalized frequency of that class:

$$F_i = S_i \,/\, \bar{S}$$

where S_i is the number of samples in class *i*, and $\bar{S}$ is the average number of samples over all classes, computed in the obvious way:

$$\bar{S} = \left(\frac{1}{C}\sum_{i=1}^{C} S_i\right)$$

with *C* being the number of classes. Our repetition factor is then defined to be:

$$R_i = \left(a/F_i\right)^b$$

with a and b being adjustable controls over the amount of skipping vs. repeating and the degree of prior normalization, respectively. Typical values of *a* range from 0.2 to 0.8, while *b* ranges from 0.5 to 0.9. The factor $a < 1$ lets us do more skipping than repeating; e.g. for $a = 0.5$, classes with relative frequency equal to half the average will neither skip nor repeat; more frequent classes will skip, and less frequent classes will repeat. A value of 0.0 for *b* would do nothing, giving R_i = 1.0 for all classes, while a value of 1.0 would provide "full" normalization. A value of *b* somewhat less than one seems to be the best choice, letting the net keep some bias in favor of classes with higher prior probabilities.

This explicit prior-bias reduction is conceptually related to Lippmann's (1994) and Morgan and Bourlard's (1995) recommended method for converting from the net's estimate of posterior probability, *p(class|input)*, to the value needed in an HMM or Viterbi search, *p(input|class)*, which is to divide by *p(class)* priors. Using that technique, however, should produce noisier estimates for low frequency classes, due to the divisions by low frequencies, resulting in a set of estimates that are not really optimized in a LMSE sense (as the net outputs are). In addition, output activations that are naturally bounded between 0 and 1, due to the sigmoid, convert to potentially very large probability estimates, requiring a re-normalization step. Our method of frequency balancing during training eliminates both of these concerns. Perhaps more significantly, frequency balancing also allows the standard BP training process to dedicate more network resources to the classification of the lower-frequency classes, though we have no current method for characterizing or quantifying this benefit.

4.7 ERROR EMPHASIS

While frequency balancing corrects for under-represented classes, it cannot account for under-represented writing styles. We utilize a conceptually related probabilistic skipping of patterns, but this time for just those patterns that the net correctly classifies in its forward/recognition pass, as a form of "error emphasis", to address this problem. We define a *correct-train probability* (ranging from 0.1 to 1.0) that is used as a biased coin to determine whether a particular pattern, having been correctly classified, will also be used for the backward/training pass or not. This only applies to correctly segmented, or "positive" patterns, and misclassified patterns are never skipped.

Especially during early stages of training, we set this parameter fairly low (around 0.1), thus concentrating most of the training time and the net's learning capability on patterns that are more difficult to correctly classify. This is the only way we were able to get the net to learn to correctly classify unusual character variants, such as a 3-stroke "5" as written by only one training writer.

Variants of this scheme are possible in which misclassified patterns would be repeated, or different learning rates would apply to correctly and incorrectly classified patterns. It is also related to techniques that use a training subset, from which easily-classified patterns are replaced by randomly selected patterns from the full training set (Guyon *et al* 1992).

4.8 ANNEALING

Though some discussions of back-propagation espouse explicit formulae for modulating the learning rate over time, many seem to assume the use of a single, fixed learning rate. We view the stochastic back-propagation process as a kind of simulated annealing, with a learning rate starting very high and decreasing only slowly to a very low value. But rather than using any prespecified formula to decelerate learning, the rate at which the learning rate decreases is determined by the dynamics of the learning process itself. We typically start with a rate near 1.0 and reduce the rate by a multiplicative *decay factor* of 0.9 until it gets down to about 0.001. The rate decay factor is applied following any epoch in which the total squared error increased on the training set, relative to the previous epoch. This "total squared error" is summed over all output units and over all patterns in one full epoch, and normalized by those counts. So even though we are using "on-line" or stochastic gradient descent, we have a measure of performance over whole epochs that can be used to guide the "annealing" of the learning rate. Repeated tests indicate that this approach yields better results than low (or even moderate) initial learning rates, which we speculate to be related to a better ability to escape local minima.

In addition, we find that we obtain best overall results when we also allow some of our many training parameters to change over the course of a training run. In particular, the correct train probability needs to start out very low to give the net a chance to learn unusual character styles, but it should finish up near 1.0 in order to not introduce a general posterior probability bias in favor of classes with lots of ambiguous examples. We typically train a net in four “phases” according to parameters such as in Figure 7.

Phase	Epochs	Learning Rate	Correct Train Prob	Negative Train Prob
1	25	1.0 - 0.5	0.1	0.05
2	25	0.5 - 0.1	0.25	0.1
3	50	0.1 - 0.01	0.5	0.18
4	30	0.01 - 0.001	1.0	0.3

Figure 7: Typical multi-phase schedule of learning rates and other parameters for training a character-classifier net.

4.9 QUANTIZED WEIGHTS

The work of Asanovic and Morgan (1991) shows that two-byte (16-bit) weights are about the smallest that can be tolerated in training large ANNs via back-propagation. But memory is expensive in small devices, and RISC processors, such as the ARM-610 in the first devices in which this technology was deployed, are much more efficient doing one-byte loads and multiplies than two-byte loads and multiplies, so we were motivated to make one-byte weights work.

Running the net for recognition demands significantly less precision than does training the net. It turns out that one-byte weights provide adequate precision for recognition, if the weights are trained appropriately. In particular, a dynamic range should be fixed, and weights limited to that legal range during training, and then rounded to the requisite precision after training. For example, we find that a range of weight values from (almost) –8 to +8 in steps of 1/16 does a good job. Figure 8 shows a typical resulting distribution of weight values. If the weight limit is enforced during high-precision training, the resources of the net will be adapted to make up for the limit. Since bias weights are few in number, however, and very important, we allow them to use two bytes with essentially unlimited range. Performing our forward/recognition pass with low-precision, one-byte weights (a ±3.4 fixed-point representation), we find no noticeable degradation relative to floating-point, four-byte, or two-byte weights using this scheme.

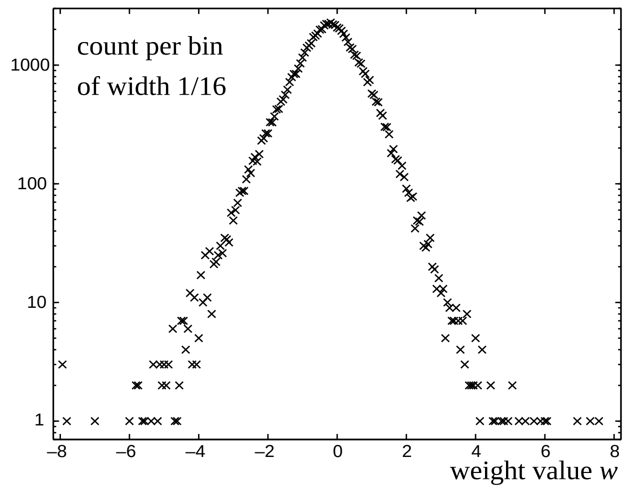


Figure 8: Distribution of weight values in a net with one-byte weights, on a log count scale. Weights with magnitudes greater than 4 are sparse, but important.

We have also developed a scheme for training with augmented one-byte weights. It uses a temporary augmentation of the weight values with two additional low-order bytes to achieve precision in training, but runs the forward pass of the net using only the one-byte high-order part. Thus any cumulative effect of the one-byte rounded weights in the forward pass can be compensated through further training. Small weight changes accumulate in the low-order bytes, and only occasionally carry into a change in the one-byte weights used by the net. In a personal product, this scheme could be used for adaptation to the user, after which the low-order residuals could be discarded and the temporary memory reclaimed.

5 CONTEXT-DRIVEN SEARCH

The output of the ANN classifier is a stream of probability vectors, one vector for each segmentation hypothesis, with as many potentially nonzero probability elements in each vector as there are characters (that the system is capable of recognizing). In practice, we typically only pass the top ten (or fewer) scored character-class hypotheses, per segment, to the search engine, for the sake of efficiency. The search engine then looks for a minimum-cost path through this vector stream, abiding by the legal transitions between segments, as defined in the tentative-segmentation step discussed previously. This minimum-cost path is the APR system's best interpretation of the ink input by the user, and is returned to the system in which APR is embedded as the recognition result for whole words or sentences of the user's input.

The search is driven by a somewhat *ad hoc*, generative language model, which consists of a set of graphs that are searched in parallel. We use a simple beam search in a negative-log-probability (or *penalty*) space for the best N hypotheses. The beam is based on a fixed maximum number of hypotheses, rather than a particular value. Each possible transition token (character) emitted by one of the graphs is scored not only by the ANN, but by the language model itself, by a simple letter-case model, and by geometric-context models discussed below. The fully integrated search process takes place over a space of character- and word-segmentation hypotheses, as well as character-class hypotheses.

5.1 LEXICAL CONTEXT

Context is essential to accurate recognition, even if that context takes the form of a very broad language model. Humans achieve just 90% accuracy on isolated characters from our database. Lacking any context this would translate to a word accuracy of not much more than 60% (0.9^5), assuming an average word length of 5 characters. We obviously need to do much better, with even lower isolated-character accuracy, and we accomplish this by the application of our context models.

A simple model of letter case and adjacency—penalizing case transitions except between the first and second characters, penalizing alphabetic-to-numeric transitions, and so on—together with the geometric-context models discussed later, is sufficient to raise word-level accuracy up to around 77%.

The next large gain in accuracy requires a genuine language model. We provide this model by means of dictionary graphs, and assemblages of those graphs combined into what we refer to as *BiGrammars*. BiGrammars are essentially scored lists of dictionaries, together with specified legal (scored) transitions between those dictionaries. This scheme allows us to use word lists, prefix and suffix lists, and punctuation models, and to enable appropriate transitions between them. Some dictionary graphs are derived from a regular-expression grammar that permits us to easily model phone numbers, dates, times, etc., as shown in Figure 9.

```
dig    = [0123456789]
digm01 =   [23456789]

acodenums = (digm01 [01] dig)

acode  = { ("1-"?    acodenums "-"):40 ,
           ("1"? "(" acodenums ")"):60 }

phone = (acode? digm01 dig dig "-" dig dig dig dig)
```

Figure 9: Sample of the regular-expression language used to define a simple telephone-number grammar. Symbols are defined by the equal operator; square brackets enclose multiple, alternative characters; parentheses enclose sequences of symbols; curly braces enclose multiple, alternative symbols; an appended colon followed by numbers designates a prior probability of that alternative; an appended question mark means "zero or one occurrence"; and the final symbol definition represents the graph or grammar expressed by this dictionary.

All of these dictionaries can be searched in parallel by combining them into a general-purpose BiGrammar that is suitable for most applications. It is also possible to combine subsets of these dictionaries, or special-purpose dictionaries, into special BiGrammars targeted at more limited contexts. A very simple BiGrammar, which might be useful to specify context for a field that only accepts telephone numbers, is shown in Figure 10. A more complex BiGrammar (though still far short of the complexity of our final general-input context) is shown in Figure 11.

```
BiGrammar Phone

[Phone.lang 1. 1. 1.]
```

Figure 10: Sample of a simple BiGrammar describing a telephone-only context. The BiGrammar is first named (Phone), and then specified as a list of dictionaries (Phone.lang), together with the probability of starting with this dictionary, ending with this dictionary, and cycling within this dictionary (the three numerical values).

```
BiGrammar FairlyGeneral
(.8
   (.6
      [WordList.dict .5  .8  1. EndPunct.lang .2]
      [User.dict     .5  .8  1. EndPunct.lang .2]
   )
   (.4
      [Phone.lang    .5  .8  1. EndPunct.lang .2]
      [Date.lang     .5  .8  1. EndPunct.lang .2]
   )
)

(.2
   [OpenPunct.lang  1.  0.  .5
      (.6
         WordList.dict .5
         User.dict     .5
      )
      (.4
         Phone.lang    .5
         Date.lang     .5
      )
   ]
)

[EndPunct.lang  0.  .9  .5  EndPunct.lang .1]
```

Figure 11: Sample of a slightly more complex BiGrammar describing a fairly general context. The BiGrammar is first named (FairlyGeneral), and then specified as a list of dictionaries (the *.dict and *.lang entries), together with the probability of starting with this dictionary, ending with this dictionary, and cycling within this dictionary (the first three numerical values following each dictionary name), plus any dictionaries to which this dictionary may legally transition, along with the probability of taking that transition. The parentheses permit easy specification of multiplicative prior probabilities for all dictionaries contained within them. Note that in this simple example, it is not possible (starting probability = 0) to start a string with the EndPunct (end punctuation) dictionary, just as it is not possible to end a string with the OpenPunct dictionary.

We refer to our language model as being "weakly applied" because in parallel with all of the wordlist-based dictionaries and regular-expression grammars, we simultaneously search both an alphabetic-characters grammar ("wordlike") and a completely general, any-character-anywhere grammar ("symbols"). These more flexible models, though given fairly low *a priori* probabilities, permit users to write any unusual character string they might desire. When the prior probabilities for the various dictionaries are properly balanced, the recognizer is able to benefit from the language model, and deliver the desired level of accuracy for common in-dictionary words (and special constructs like phone numbers, etc.), yet can also recognize arbitrary, non-dictionary character strings, especially if they are written neatly enough that the character classifier can be confident of its classifications.

We have also experimented with bi-grams, tri-grams, N-grams, and we are continuing experiments with other, more data-driven language models; so far, however, our generative approach has yielded the best results.

5.2 GEOMETRIC CONTEXT

We have never found a way to reliably estimate a baseline or topline for characters, independent of classifying those characters in a word. Non-recognition-integrated estimates of these line positions, based on strictly geometric features, have too many pathological failure modes, which produce erratic recognition failures. Yet the geometric positioning of characters most certainly bears information important to the recognition process. Our system factors the problem by letting the ANN classify representations that are independent of baseline and size, and then using separate modules to score both the absolute size of individual characters, and the relative size and position of adjacent characters.

The scoring based on absolute size is derived from a set of simple Gaussian models of individual character heights, relative to some running scale parameters computed both during learning and during recognition. This *CharHeight* score directly multiplies the scores emitted by the ANN classifier, and helps significantly in case disambiguation.

We also employ a *GeoContext* module that scores adjacent characters, based on the classification hypotheses for those characters and on their relative size and placement. GeoContext scores each tentative character based on its class and the class of the immediately preceding letter (for the current search hypothesis). The character classes are used to look up expected character sizes and positions in a standardized space (baseline=0.0, topline=1.0). The ink being evaluated provides actual sizes and positions that can be compared directly to the expected values, subject only to a scale factor and offset, which are chosen so as to minimize the estimated error of fit between data and model. This same quadratic error term, computed from the inverse covariance matrix of a full multivariate Gaussian model of these sizes and positions, is used directly as GeoContext's score (or penalty, since it is applied in the –log probability space of the search engine). Figure 12 illustrates the bounding boxes derived from the user's ink *vs.* the table-driven model, with the associated error measures for our GeoContext module.

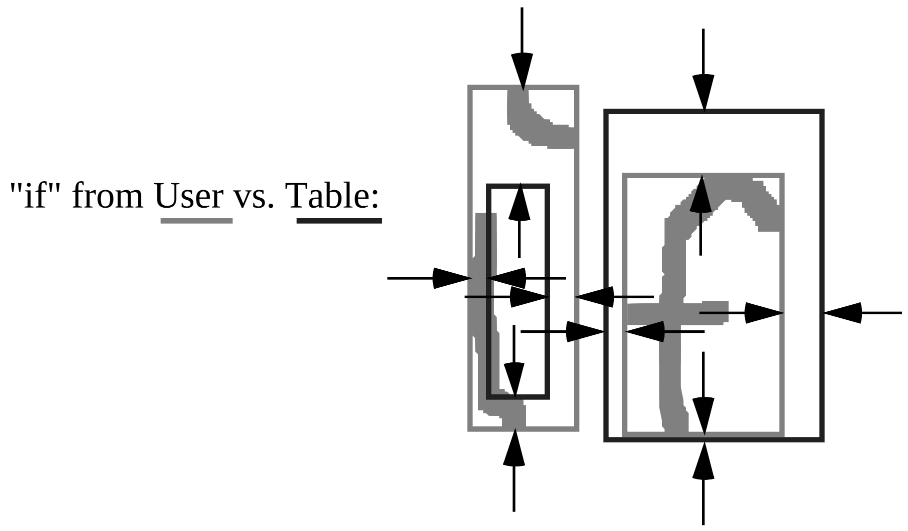


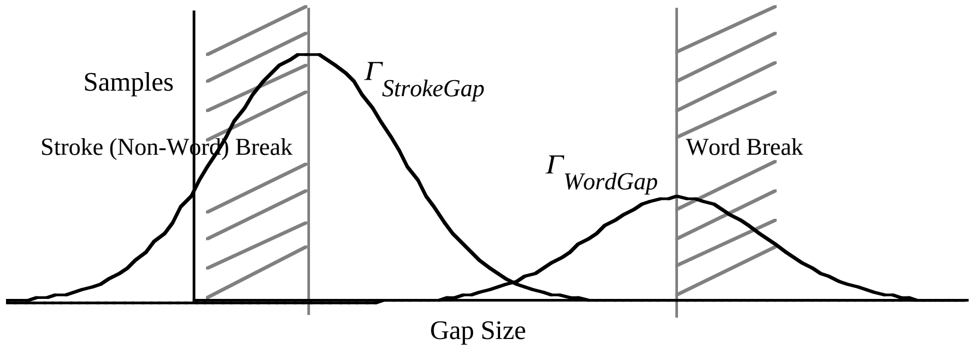
Figure 12: The eight measurements that contribute to the GeoContext error vector and corresponding score for each letter pair.

GeoContext's multivariate Gaussian model is learned directly from data. The problem in doing so was to find a good way to train per-character parameters of top, bottom, width, space, etc., in our standardized space, from data that had no labeled baselines, or other absolute referent points. Since we had a technique for generating an error vector from the table of parameters, we decided to use a back-propagation variant to train the table of parameters to minimize the squared error terms in the error vectors, given all the pairs of adjacent characters and correct class labels from the training set.

GeoContext plays a major role in properly recognizing punctuation, in disambiguating case, and in recognition in general. A more extended discussion of GeoContext has been provided by Lyon and Yaeger (1996).

5.3 INTEGRATION WITH WORD SEGMENTATION

Just as it is necessary to integrate character segmentation with recognition via the search process, so is it essential to integrate word segmentation with recognition and search, in order to obtain accurate estimates of word boundaries, and to reduce the large class of errors associated with mis-segmented words. To perform this integration, we first need a means of estimating the probability of a word break between each pair of tentative characters. We use a simple statistical model of gap sizes and stroke-centroid spacing to compute this probability (*spaceProb*). Gaussian density distributions, based on means and standard deviations computed from a large training corpus, together with a prior probability scale factor, provide the basis for the word-gap and stroke-gap (non-word-gap) models, as illustrated in Figure 13. Since any given gap is, by definition, either a word gap or a non-word gap, the simple ratio defined in Figure 13 provides a convenient, self-normalizing estimate of the word-gap probability. In practice, that equation further reduces to a simple sigmoid form, thus allowing us to take advantage of a lookup-table-based sigmoid derived for use in the ANN. In a thresholding, non-integrated word-segmentation model, word breaks would be introduced when spaceProb exceeds 0.5; i.e., when a particular gap is more likely to be a word-gap than a non-word-gap. For our integrated system, both word-break and non-word-break hypotheses are generated at each segment transition, and weighted by spaceProb and (1–spaceProb), respectively. The search process then proceeds over this larger hypothesis space to produce best estimates of whole phrases or sentences, thus integrating word segmentation as well as character segmentation.



$$P_{WordBreak} = \Gamma_{WordGap} \;/\; (\Gamma_{StrokeGap} + \Gamma_{WordGap})$$

Figure 13: Gaussian density distributions yield a simple statistical model of word-break probability, which is applied in the region between the peaks of the StrokeGap and WordGap distributions. Hashed areas indicate regions of clear cut decisions, where $P_{WordBreak}$ is set to either 0.0 or 1.0, to avoid problems dealing with tails of these simple distributions.

6 DISCUSSION

The combination of elements described in the preceding sections produces a powerful, integrated approach to character segmentation, word segmentation, and recognition. Users' experiences with APR are almost uniformly positive, unlike experiences with previous handwriting recognition systems. Writing within the dictionary is remarkably accurate, yet the ease with which people can write outside the dictionary has fooled many people into thinking that the Newton's "Print Recognizer" does not use dictionaries. As discussed previously, our recognizer certainly does use dictionaries. Indeed, the broad-coverage language model, though weakly applied, is essential for high accuracy recognition. Curiously, there seems to be little problem with dictionary *perplexity*—little difficulty as a result of using very large, very complex language models. We attribute this fortunate behavior to the excellent performance of the neural network character classifier at the heart of the system. One of the side benefits of the weak application of the language model is that even when recognition fails and produces the wrong result, the answer that is returned to the user is typically understandable by the user—perhaps involving substitution of a single character. Two useful phenomena ensue as a result. First, the user learns what works and what doesn't, especially when she refers back to the ink that produced the misrecognition, so the system trains the user gracefully over time. Second, the meaning is not lost the way it can be, all too easily, with whole word substitutions—with that "Doonesbury Effect" found in first-generation, strong-language-model recognizers.

Though we have provided legitimate accuracy statistics for certain comparative tests of some of our algorithms, we have deliberately shied away from claiming specific levels of accuracy in general. Neat printers, who are familiar with the system, can achieve 100% accuracy if they are careful. Testing on data from complete novices, writing for the first time using a metal pen on a glass surface, without any feedback from the recognition system, and with ambiguous instructions about writing with "disconnected characters" (intended to mean printing, but often interpreted to mean writing with otherwise cursive characters but separated by large spaces in a wholly unnatural style), can yield word-level accuracies as low as 80%. Of course, the entire interesting range of recognition accuracies lies between these two extremes. Perhaps a slightly more meaningful statistic comes from common reports on usenet newsgroups, and some personal testing, that suggest accuracies of 97% to 98% in regular use. But for scientific purposes, none of these numbers have any real meaning, since our testing datasets are proprietary, and the only valid tests between different recognizers would have to be based on results obtained by processing the exact same bits, or by analyzing large numbers of experienced users of the systems in the field—a difficult project which has not been undertaken.

One of the key reasons for the success of APR is the suite of innovative neural network training techniques that help the network encode better class probabilities, especially for under-represented classes and writing styles. Many of these techniques—stroke count dithering, normalization of output error, frequency balancing, error emphasis—share a unifying theme: Reducing the effect of *a priori* biases in the training data on network learning significantly improves the network's performance in an integrated recognition system, despite a modest reduction in the network's accuracy for individual characters. Normalization of output error prevents over-represented non-target classes from biasing the net against under-represented target classes. Frequency balancing prevents over-represented classes from biasing the net against under-represented classes. And stroke-count dithering and error emphasis prevent over-represented writing styles from biasing the net against under-represented writing styles. One could even argue that negative training eliminates an absolute bias towards properly segmented characters, and that stroke warping reduces the bias towards those writing styles found in the training data, although these techniques also provide wholly new information to the system.

Though we've offered arguments for why each of these techniques, individually, helps the overall recognition process, it is unclear why prior-bias reduction, in general, should be so consistently valuable. The general effect may be related to the technique of dividing out priors, as is sometimes done to convert from *p(class|input)* to *p(input|class)*. But we also believe that forcing the net, during learning, to allocate resources to represent less frequent sample types may be directly beneficial. In any event, it is clear that paying attention to such biases and taking steps to modulate them is a vital component of effective training of a neural network serving as a classifier in a maximum-likelihood recognition system.

The majority of this paper describes a sort of snapshot of the system and its architecture as it was deployed in its first commercial release, when it was, indeed, purely a "Print Recognizer". Letters had to be fully "disconnected"; i.e., the pen had to be lifted between each pair of characters. The characters could overlap to some extent, but the ink could not be continuous. Connected characters proved to be the largest remaining class of errors for most of our users, since even a person who normally prints (as opposed to writing in cursive script) may occasionally connect a pair of characters—the cross-bar of a "t" with the "h" in "the", the "o" and "n" in any word ending in "ion", and so on. To address this issue, we experimented with some fairly straightforward modifications to our recognizer, involving the *fragmenting* of user-strokes into multiple system-strokes, or *fragments*. Once the ink representing the connected characters is broken up into fragments, we then allow our standard integrated segmentation and recognition process to stitch them back together into the most likely character and word hypotheses, as always. This technique has proven itself to work quite well, and the version of the "Print Recognizer" in the most recent Newton, the MessagePad 2000®, supports recognition of printing with connected characters. This capability was added without significant modification of the main recognition algorithms as presented in this paper. Due to certain assumptions and constraints in the current release of the software, APR is not yet a full cursive recognizer, though that is an obvious next direction to explore.

The net architecture discussed in section 4.2 and shown in Figure 3 also corresponds to the true printing-only recognizer. The final output layer has 95 elements corresponding to the full printable ASCII character set plus the British Pound sign. Initially for the German market, and now even in English units, we have extended APR to handle diacritical marks and the special symbols needed for most European languages (although there is only very limited coverage of foreign languages in the English units). The main innovation that permitted this extended character set was an explicit handling of any compound character as a *base* plus an *accent*. This way only a few nodes needed to be added to the neural network output layer, representing just the bases and accents, rather than all combinations and permutations of same. And training data for all compound characters sharing a common base or a common accent contributed to the network's ability to learn that base or accent, as opposed to contributing only to the explicit base+accent combination. Here again, however, the fundamental recognizer technology has not changed significantly from that presented in this paper.

7 FUTURE EXTENSIONS

We are optimistic that our algorithms, having proven themselves to work essentially as well for connected characters as for disconnected characters, may extend gracefully to full cursive.

On a more speculative note, we believe that the technique may extend well to ideographic languages, substituting radicals for characters, and ideographic characters for words.

Finally, a note about learning and user adaptation: For a learning technology such as ANNs, user adaptation is an obvious and natural fit, and was planned as part of the system from its inception. However, due to RAM constraints in the initial shipping product, and the subsequent prioritization of European character sets and connected characters, we have not yet deployed a learning system. We have, however, done some testing of user adaptation, and believe it to be of considerable value. Figure 14 shows a comparison of the average performance on an old user-independent net trained on data from 45 writers, and the performance for three individuals using A) the user-independent net, B) a net trained on data exclusively from that individual, and C) a copy of the user-independent net adapted to the specific user by some incremental training. (Note: These data are from a multi-year-old experiment, and are not necessarily representative of current levels of performance on any absolute scale.)

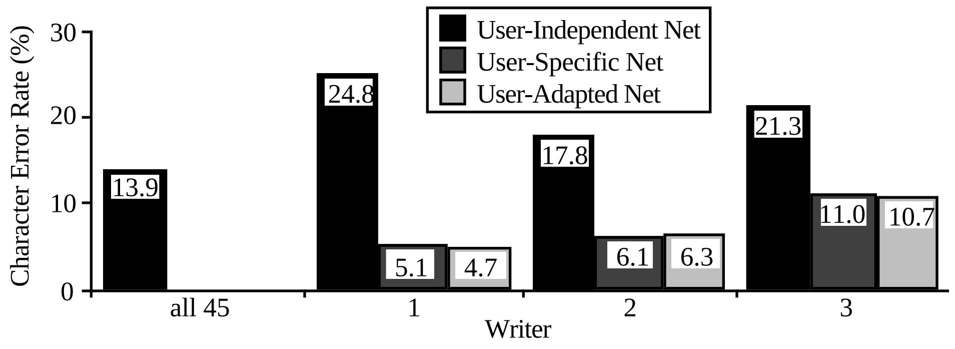


Figure 14: User-adaptation test results for three individual writers with three different nets each, plus the overall results for 45 writers tested on a user-independent net trained on all 45 writers.

An important distinction is being made here between "user-adapted" and "user-specific" nets. "User-specific" nets have been trained with a relatively large corpus of data exclusively from that specific user. "User-adapted" nets were based on the user-independent net, with some additional training using limited data from the user in question. All testing was performed with data held out from all training sets.

One obvious thing to note is the reduction in error rate ranging from a factor of 2 to a factor of 5 that both user-specific and user-adapted nets provide. An equally important thing to note is that the user-adapted net performs essentially as well as a user-specific net—in fact, slightly *better* for two of the three writers. Given ANNs' penchant for local minima, we were concerned that this might not be the case. But it appears that the features learned during the user-independent net training served the user-adapted net well. We believe that a very small amount of training data from an individual will allow us to adapt the user-independent net to that user, and improve the overall accuracy for that user significantly, especially for individuals with more stylized writing, or whose writing style is under-represented in our user-independent training corpus. And even for writers with common and/or neat writing styles, there is inherently less ambiguity in a single writer's style than in a corpus of data necessarily doing its best to represent essentially all possible writing styles.

These results may be exaggerated somewhat by the limited data in the user-independent training corpus at the time these tests were performed (just 45 writers), and at least two of the three writers in question had particularly problematic writing styles. We have also made significant advances in our user-independent recognition accuracies since these tests were performed. Nonetheless, we believe these results are suggestive of the significant value of user adaptation, even in preference to a user-specific solution.

Acknowledgments

This work was done in collaboration with Bill Stafford, Apple Computer, and Les Vogel, Angel Island Technologies. We would also like to acknowledge the contributions of Michael Kaplan, Rus Maxham, Kara Hayes, Gene Ciccarelli, and Stuart Crawford. We also received support and assistance from the Newton Recognition, Software Testing, and Hardware Support groups. We are also indebted to our many colleagues in the connectionist community for their advice, help, and encouragement over the years, as well as our many colleagues at Apple who pitched in to help throughout the life of this project.

Some of the techniques described in this paper are the subject of pending U.S. and foreign patent applications.

References

K. Asanovic and N. Morgan, "Experimental Determination of Precision Requirements for Back-Propagation Training of Artificial Neural Networks," TR-91-036, International Computer Science Institute, Berkeley, 1991.

Y. Bengio, Y. LeCun, C. Nohl, and C. Burges, "LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition," *Neural Computation*, Vol. 7, pp. 1289-1303, 1995.

H. Bourlard and C. J. Wellekens, “Links between Markov Models and Multilayer Perceptrons,” *IEEE Trans. PAMI*, Vol. 12, pp. 1167–1178, 1990.

E. I. Chang and R. P. Lippmann, “Using Voice Transformations to Create Additional Training Talkers for Word Spotting,” in *Advances in Neural Information Processing Systems 7*, Tesauro et al. (eds), pp. 875–882, MIT Press, 1995.

H. Gish, “A Probabilistic Approach to Understanding and Training of Neural Network Classifiers,” *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing* (Albuquerque, NM), pp. 1361–1364, 1990.

I. Guyon, D. Henderson, P. Albrecht, Y. LeCun, and P. Denker, “Writer independent and writer adaptive neural network for on-line character recognition,” in *From pixels to features III*, S. Impedovo (ed.), pp. 493–506, Elsevier, Amsterdam, 1992.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive Mixtures of Local Experts,” *Neural Computation*, Vol. 3, pp. 79–87, 1991.

R. P. Lippmann, “Neural Networks, Bayesian *a posteriori* Probabilities, and Pattern Classification,” pp. 83–104 in: *From Statistics to Neural Networks—Theory and Pattern Recognition Applications*, V. Cherkassky, J. H. Friedman, and H. Wechsler (eds.), Springer-Verlag, Berlin, 1994.

R. F. Lyon and L. S. Yaeger, “On-Line Hand-Printing Recognition with Neural Networks”, *Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems* (proceedings), pp. 201–212, 1996.

N. Morgan and H. Bourlard, “Continuous Speech Recognition—An introduction to the hybrid HMM/connectionist approach,” IEEE Signal Processing Mag., Vol. 13, no. 3, pp. 24–42, May 1995.

M. Parizeau and R. Plamondon, “Allograph Adjacency Constraints for Cursive Script Recognition,” *Third International Workshop on Frontiers in Handwriting Recognition* (Pre-Proceedings) , pp. 252–261, 1993.

S. Renals and N. Morgan, “Connectionist Probability Estimation in HMM Speech Recognition,” TR-92-081, International Computer Science Institute, 1992.

S. Renals, N. Morgan, M. Cohen, and H. Franco “Connectionist Probability Estimation in the Decipher Speech Recognition System,” *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing* (San Francisco), pp. I-601–I-604, 1992.

M. D. Richard and R. P. Lippmann, “Neural Network Classifiers Estimate Bayesian *a Posteriori* Probabilities,” *Neural Computation*, Vol. 3, pp. 461–483, 1991.

P. Simard, B. Victorri, Y. LeCun and J. Denker, “Tangent Prop—A Formalism for Specifying Selected Invariances in an Adaptive Network,” in *Advances in Neural Information Processing Systems 4*, Moody et al. (eds.), pp. 895–903, Morgan Kaufmann, 1992.

P. Simard, Y. LeCun and J. Denker, “Efficient Pattern Recognition Using a New Transformation Distance,” in *Advances in Neural Information Processing Systems 5*, Hanson et al. (eds.), pp. 50–58, Morgan Kaufmann, 1993.

C. C. Tappert, C. Y. Suen, and T. Wakahara, “The State of the Art in On-Line Handwriting Recognition,” *IEEE Trans. PAMI*, Vol. 12, pp. 787–808, 1990.