AI Magazine Volume 19 Number 1 (1998) (© AAAI)

Combining Neural Networks and Context-Driven Search for Online, Printed Handwriting Recognition in the NEWTON

Larry S. Yaeger, Brandyn J. Webb, and Richard F. Lyon

■ While online 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, and so on, 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.

Pen-based hand-held computers are heavily dependent on fast and accurate handwriting recognition because 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, but they proved unacceptable in real-world applications, generating disturbing and seemingly random word substitutions—known colloquially within Apple and NEWTON as "The Doonesbury Effect" because of Gary Trudeau's satirical look at first-generation NEWTON recognition performance. However, 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 were clearly delineated by pen lifts. By simultaneously providing accurate character-level recognition, dictionaries exhibiting wide coverage of the language, and the ability to write entire-

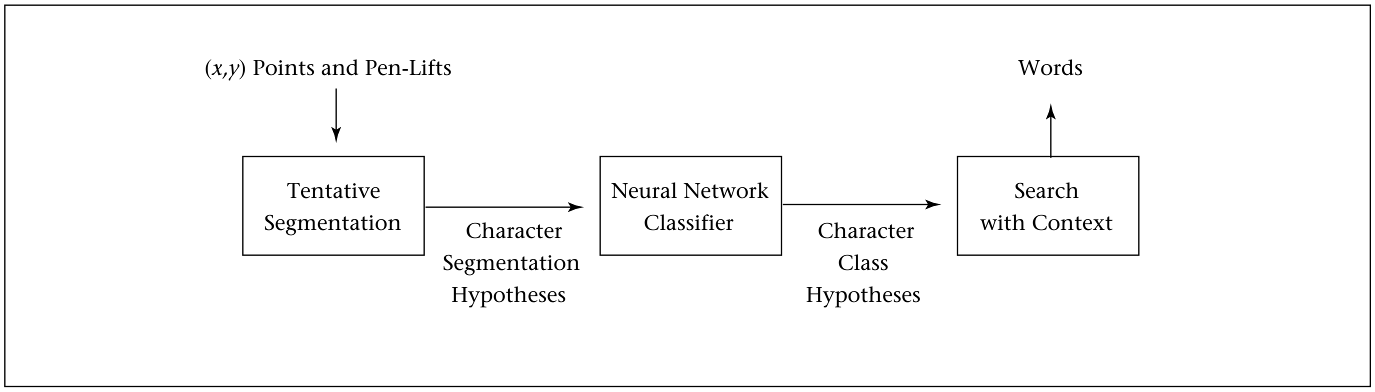

(x,y) Points and Pen-Lifts
Tentative Segmentation
Character Segmentation Hypotheses
Neural Network Classifier
Character Class Hypotheses
Search with Context
Words

Figure 1. A Simplified Block Diagram of Our Hand-Print Recognizer.

ly outside these 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, and telephone numbers), and a means of combining all these dictionaries into a comprehensive language model. In addition, 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," which 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 units, since 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, Suen, and Wakahara 1990) and speech (Renals et al. 1992). In addition, 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). However, these approaches leave a large number of open-ended questions about how to achieve acceptable (to a real user) levels of performance. In this article, we survey some of our experiences in exploring refinements and improvements to these techniques.

System Overview

Apple's print recognizer (APR) consists of three conceptual stages—(1) tentative segmentation, (2) classification, and (3) context-driven search—as indicated in figure 1. The primary data on which we operate are simple sequences of (x,y) coordinate pairs plus pen-up–pen-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 that 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.

Tentative Segmentation

Character segmentation—the process of deciding which strokes make up 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 these 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 can be used twice") and some less obvious filters (to cull impossible segments 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. Because of 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 (for each 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	c	1	3	0
	2	cl	2	4	1
	3	clo	3	4	2
clog	4	l	1	2	0
	5	lo	2	2	1
	6	o	1	1	0
	7	g	1	0	0

Figure 2. Segmentation of Strokes into Tentative Characters, or Segments.

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 later, a fairly standard multilayer perceptron trained with error backpropagation 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 (Renals and Morgan 1992; Richard and Lippman 1991; Gish 1990; 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).

Representation

A recurring theme in ANN research is the extreme importance of the representation of the data that are given as input to the network. We experimented with a variety of input representations, including stroke features both antialiased (gray scale) and not (binary) and images both antialiased and not, and with various schemes for positioning and scaling the ink within the image input window. In every case, antialiasing was a significant win. This result is consistent with others' findings: ANNs perform better when presented with smoothly varying, distributed input than they do when presented with binary, localized input. 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, it helped to define the architecture of our nets. For our final, hand-optimized system, we use 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.

Input Feature	Resolution	Description
Image	14 × 14	Antialiased, scale to window, scale limited
Stroke	20 × 9	Antialiased, limited-resolution tangent slope, resampled to fixed number of points
Aspect Ratio	1 × 1	Normalized and capped to [0,1]
Stroke Count	5 × 1	Dithered thermometer code

Table 1. Input Representations Used in the Apple Print Recognizer.

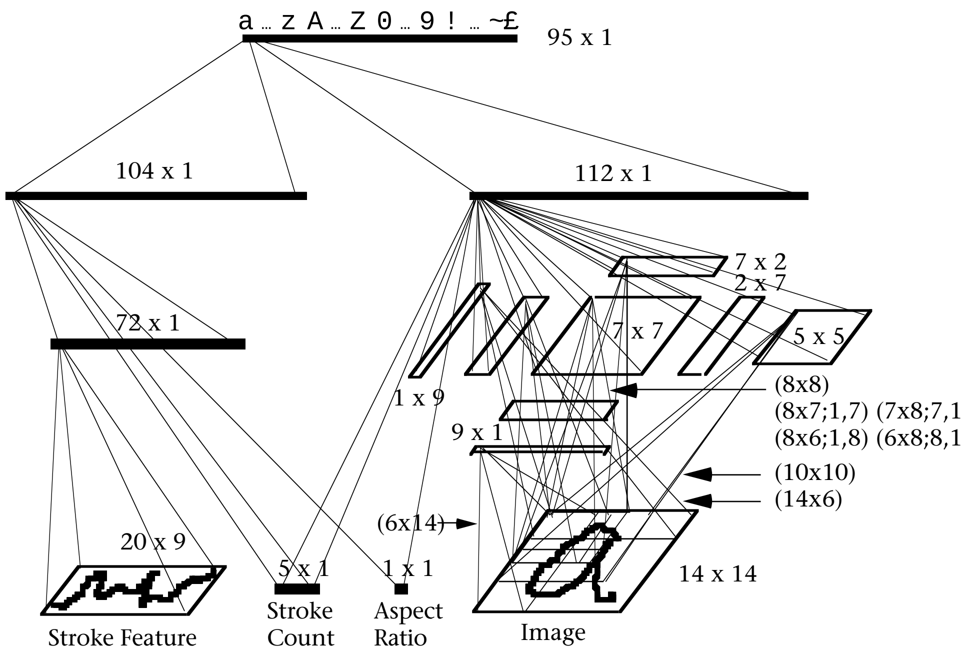


Figure 3. Final English-Language Net Architecture.

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.

Layers are fully connected except for the input to the first hidden layer on the image side. This first hidden layer on the image side consists of eight separate grids, each of which accepts input from the image input grid with its own receptive field sizes and strides, shown parenthetically in figure 3 as (*x*-size × *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 7 × 2 and 2 × 7 side panels (surrounding the central 7 × 7 grid) pay special attention to the edges of the image. The 9 × 1 and 1 × 9 side panels specifically examine full-size vertical and horizontal features, respectively. The 5 × 5 grid observes features at a different spatial scale than the 7 × 7 grid.

Combining the two classifiers at the output layer, rather than, say, averaging the output of completely independent classifiers, allows generic backpropagation to learn the best way to combine them, which is both convenient and powerful. However, our integrated multiple-representation 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).

Normalizing Output Error

Analyzing a class of errors involving words that were misrecognized as a result of, 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 backpropagation, coupled with the typical training vector, which consists of all 0s 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 toward 0" versus the "pressure toward 1" introduced by the output error during training. We refer to this technique as *NormOutErr* because of its normalizing effect on target versus nontarget output error.

We reduce the backpropagation error for nontarget classes relative to the target class by a factor that normalizes the total nontarget error seen at a given output unit relative to the total target error seen at the unit. If the training set consisted of an equal representation of classes, then this normalization should be based on the number of nontarget versus target classes in a typical training vector or, simply, the number of output units (minus one). Hence for nontarget 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 as

$$A = 1 \ / \ [d(N_{output} - 1)] \ ,$$

where N_{output} is the number of output units, and is our tuning parameter, typically ranging from 0.1 to 0.2. Error at the target output unit is unchanged. Overall, this error modulation raises the activation values at the output units because of the reduced pressure toward zero, particularly for low-probability samples. Thus, the learning algorithm no longer converges to a least mean-squared error (LMSE) estimate of $P(class \mid input)$ but to an LMSE estimate of a nonlinear function $f(P(class \mid input), A)$ 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 the resulting nonlinear function. The net will attempt to converge to minimize the modified quadratic error function

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

by setting its output y for a particular class to

$$y = p \ / \ (A - Ap + p) \ ,$$

where $p = P(class \mid input)$, and A is as previously defined. 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

$$p = yA \ / \ (yA + 1 - y) \ .$$

We verified the fit of this function by looking at histograms of character-level empirical percentage correct versus y, as in figure 4. 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 significant increase in word accuracy with 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-percent 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.)

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. *Negative training* consists of presenting invalid segments to the net during training with all-zero target vectors.

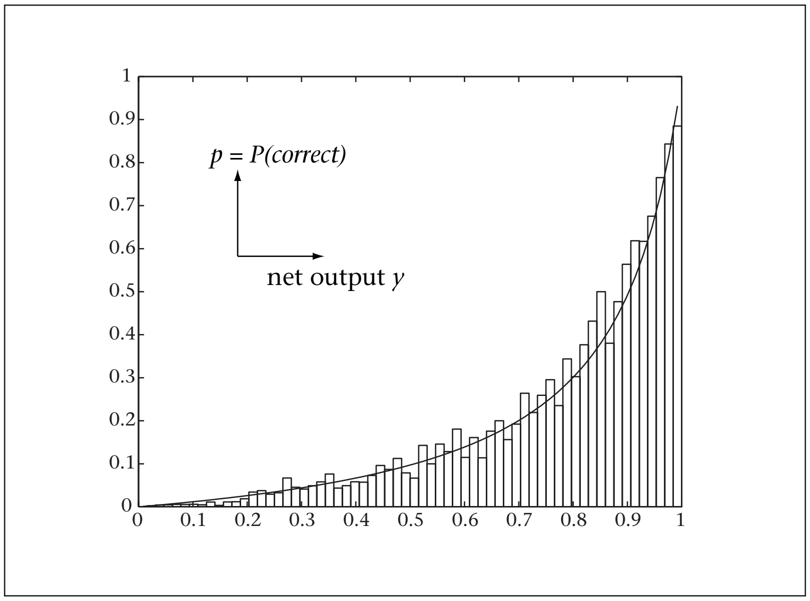


Figure 4. Empirical p *Versus* y *Histogram for a Net Trained with* A = *0.11 (*d = *.1), with the Corresponding Theoretical Curve.*

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. Lowering the learning rate for negative patterns reduces the impact of negative training on positive training, thus modulating the impact on whole characters that specifically look like pieces of multistroke characters (for example, I, 1, l, o, O, 0). Second, 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). Probabilistically skipping negative patterns both reduces the overall impact of negative training and significantly reduces training time, because invalid segments are more numerous than valid segments. As with Norm OutErr, this modification hurts character-level accuracy a little bit but helps word-level accuracy a lot.

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 stroke warping 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 backpropagation. The amounts of each distortion to apply were chosen through cross-validation experiments as just the amounts needed to yield optimum generalization. (*Cross-validation* is a standard technique for early stopping of ANN training to prevent overlearning of the training set. Such overlearning of extraneous detail in the training samples reduces the ANN's ability to generalize and, thus, reduces its accuracy on new data. The technique consists of partitioning the available data into a training set and a cross-validation set. Then, during the learning process, the training set is used in the usual fashion to train the network while accuracy on the cross-validation set is simply measured periodically. Training is terminated when accuracy on the cross-validation set is maximized, despite the fact that further training might continue to improve the accuracy measured on the training set. Cross-validation is a very effective and widely accepted means of determining the optimal amount of learning for a given ANN and body of data.) 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.

Our stroke warping scheme is somewhat related to the ideas of TANGENT DIST and TANGENT PROP (Simard, LeCun, and Denker 1993; Simard et al. 1992) 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.

Frequency Balancing

Training data from natural English words and phrases exhibit nonuniform 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 signifi-

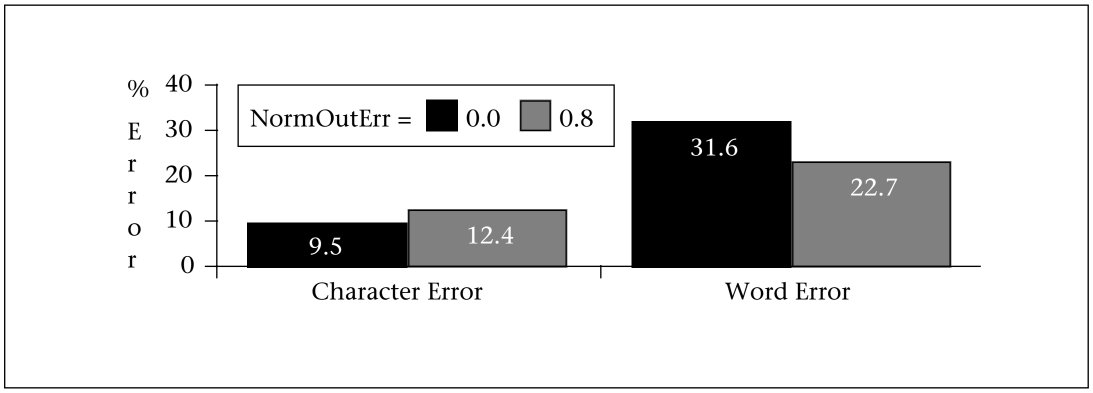


Figure 5. Character and Word Error Rates for Two Different Values of NormOutErr (d)*.*

A value of 0.0 disables NormOutErr, yielding normal backpropagation. The unusually high value of 0.8 (*A* = 0.014) produces nearly equal pressures toward 0 and 1.

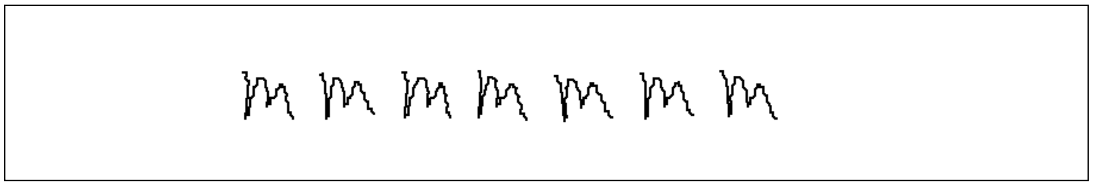


Figure 6. A Few Random Stroke Warpings of the Same Original m *Data.*

cant benefit to the overall word-recognition process. To this end, we explicitly (partially) balance the frequencies of the classes during training. We obtain this frequency balancing 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 this 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:

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

with *C* the number of classes. Our repetition factor is then defined as

$$R_i = (a / Fi)^b ,$$

with *a* and *b* as adjustable controls over the amount of skipping versus repeating and the degree of prior normalization, respectively. Typical values of *a* range from 0.2 to 0.8, and *b* ranges from 0.5 to 0.9. The factor $a < 1$ lets us do more skipping than repeating. For example, 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 no balancing, giving $R_i = 1.0$ for all classes, and 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. This division, however, may produce noisier estimates for low-frequency classes, resulting in a set of estimates that are not really optimized in an LMSE sense (as the net output are). In addition, output activations that are naturally bounded between 0 and 1 because of the sigmoid convert to potentially large probability estimates, requiring a renormalization step. Our method of frequency balancing during training eliminates both these concerns. Perhaps more significantly, frequency balancing also allows the standard backpropagation training process to dedicate more network resources to the classification of the lower-frequency classes, although we have no current method for characterizing or quantifying this benefit.

Phase	Epochs (Approximate)	Learning Rate	Correct Training Probability	Negative Training Probability
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

Table 2. Typical Multiphase Schedule of Learning Rates and Other Parameters for Training a Character-Classifier Net.

Error Emphasis

Although frequency balancing corrects for underrepresented classes, it cannot account for underrepresented writing styles. We use a conceptually related probabilistic skipping of patterns, but of 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), which 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. This only applies to correctly segmented, or *positive* patterns, and misclassified patterns are never skipped.

Especially during the 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 error emphasis is the only way we were able to get the net to learn to correctly classify unusual character variants, such as a three-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).

Annealing

Although some discussions of backpropagation espouse explicit formulas for modulating the learning rate over time, many seem to assume the use of a single, fixed learning rate. We view the stochastic backpropagation process as a kind of simulated annealing, with a learning rate starting very high and decreasing only slowly to a very low value. However, 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 is 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 these counts. Thus, even though we are using online 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 near 1.0 in order to avoid introducing 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 table 2.

Quantized Weights

The work of Asanović and Morgan (1991) shows that 2-byte (16-bit) weights are about the smallest that can be tolerated in training large ANNs using backpropagation. However, memory is expensive in small devices, and reduced instruction set computer 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 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, weights limited to the 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 7 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. Because 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, 1-byte weights (a ±3.4 fixed-point representation), we find no noticeable degradation relative to floating-point, 4-byte, or 2-byte weights using this scheme.

We have also developed a scheme for training with 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.

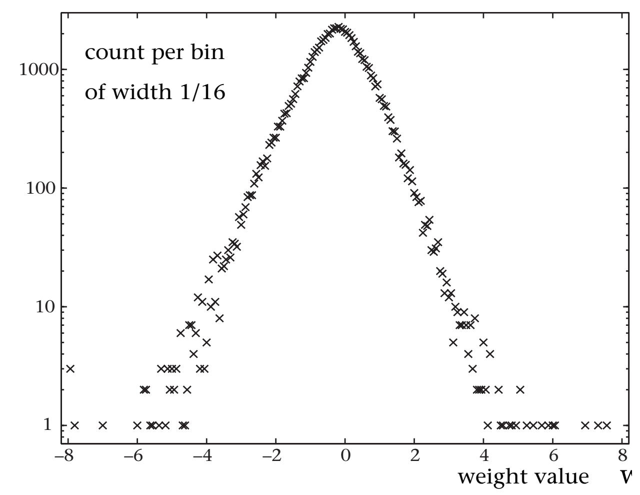


Figure 7. Distribution of Weight Values in a Net with One-Byte Weights on a Log Count Scale.

Weights with magnitudes greater than four are sparse but important.

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 10 (or fewer) scored character-class hypotheses for each 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 that 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 full language model, a simple letter-case model, and a geometric-context model discussed later. The fully integrated search process takes place over a space of character- and word-segmentation hypotheses as well as character-class hypotheses.

Lexical Context

Context is essential to accurate recognition, even if this context takes the form of a broad language model. Humans achieve just 90-percent accuracy on isolated characters from our database. Without any context, this character accuracy would translate to a word accuracy of

```
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 8. Sample of the Regular-Expression Grammar Used to Define a Simple Model of Telephone Numbers.

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 this alternative; an appended question mark means “zero or one occurrence”; and the final symbol definition represents the graph or grammar expressed by this dictionary.

```
BiGrammar Phone

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

Figure 9. 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 numeric values).

not much more than 60 percent (0.9^5), assuming an average word length of 5 characters. We need to obtain word accuracies much higher than this 60 percent, despite basing our recognition on character accuracies that may never reach human levels. We achieve this accuracy by careful 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 accuracy to around 77 percent.

The next large gain in accuracy requires a genuine language model. We provide this model by means of dictionary graphs and assemblages of these graphs combined into what we refer to as *BiGrammars*. BiGrammars are essentially scored lists of dictionaries together with specified legal (scored) transitions between these dictionaries. This scheme allows us to use word lists, prefix and suffix lists, and punctuation models and 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, and so on, as shown in figure 8.

All 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 simple BiGrammar, which might be useful to specify context for a field that only accepts telephone num-

```
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 10. 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 numeric values following each dictionary name), as well as any dictionaries to which this dictionary can legally transition and the probability of taking this 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.

bers, is shown in figure 9. A more complex BiGrammar (although still far short of the complexity of our final general-input context) is shown in figure 10.

We refer to our language model as "weakly applied" because in parallel with all the word list–based dictionaries and regular-expression grammars, we simultaneously search both an alphabetic-character grammar (wordlike) and a completely general, any-character-anywhere grammar (symbols). These more flexible models, although 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 such as phone numbers) but can also recognize arbitrary, nondictionary character strings, especially if they are written neatly enough that the character classifier can be confident of its classifications.

We have also experimented with bigrams, trigrams, N grams, and we are continuing experiments with other, more data-driven language models; to date, however, our generative approach has yielded the best results.

Geometric Context

We have never found a way to reliably estimate a baseline or topline for characters, independent of classifying these characters in a word. Non-recognition-integrated estimates of these

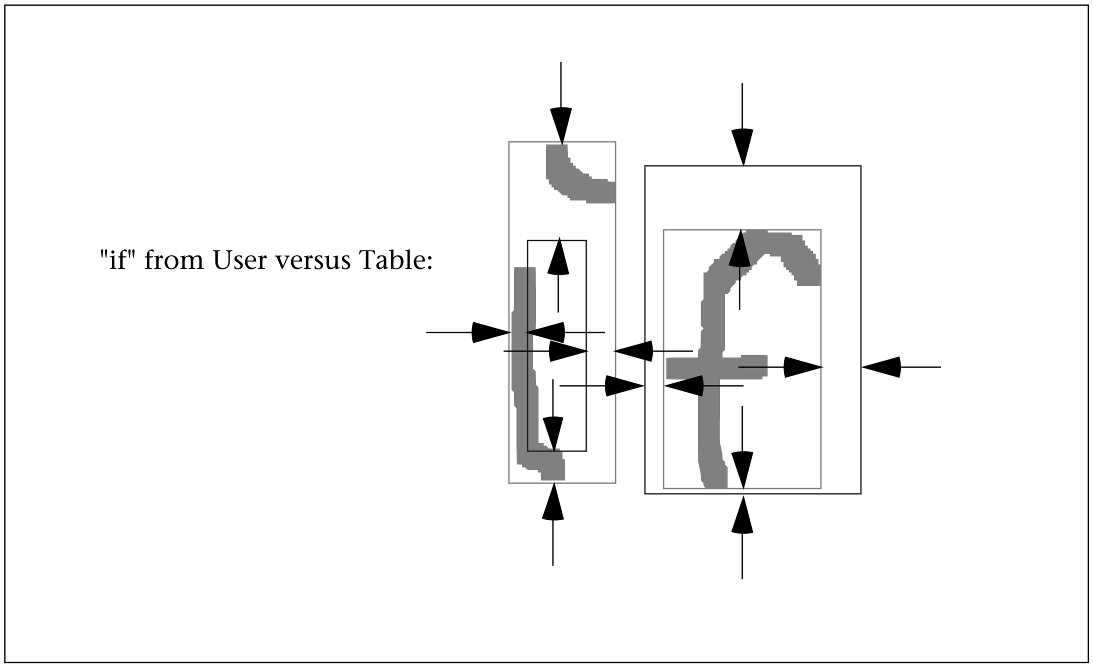


Figure 11. The Eight Measurements That Contribute to the GeoContext Error Vector and Corresponding Score for Each Letter Pair.

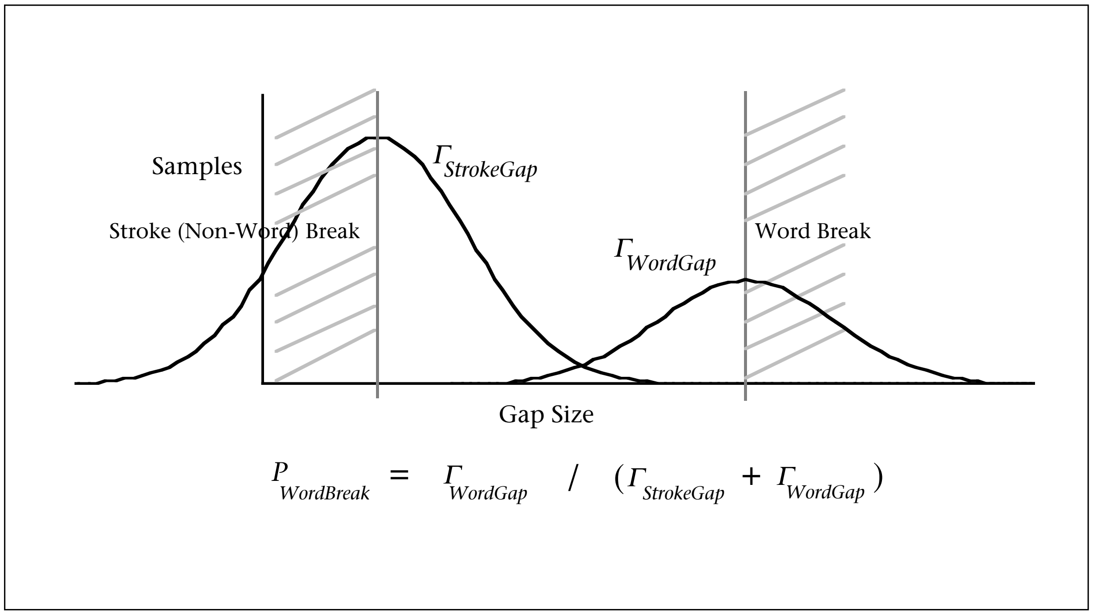
line positions, based on strictly geometric features, have too many pathological failure modes, which produce erratic recognition failures. However, the geometric positioning of characters most certainly bears information important to the recognition process. Our system factors the problem by letting 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 during both learning and recognition. This *CharHeight* score directly multiplies the scores emitted by the ANN classifier and helps significantly in case disambiguation.

We also use a *GeoContext* module that scores adjacent characters based on the classification hypotheses for these characters and 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, top line = 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 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 because it is applied in the negative log probability space of the search engine). Figure 11 illustrates the bounding boxes derived from the user's ink versus the table-driven model with the associated error measures for our GeoContext module.

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, and so on, in our standardized space from data that had no labeled baselines or other absolute referent points. Because we had a technique for generating an error vector from the table of parameters, we decided to use a backpropagation 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 and disambiguating case and a major role in recognition in general. A more extended discussion of GeoContext was provided by Lyon and Yaeger (1996).



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

Figure 12. 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.

Integration with Word Segmentation

Just as it is necessary to integrate character segmentation with recognition using the search process, so is it essential to integrate word segmentation with recognition and search to obtain accurate estimates of word boundaries and reduce the large class of errors associated with missegmented 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 (nonword-gap) models, as illustrated in figure 12. Because any given gap is, by definition, either a word gap or a nonword gap, the simple ratio defined in figure 12 provides a convenient, self-normalizing estimate of the word-gap probability. In practice, this 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, that is, when a particular gap is more likely to be a word gap than a nonword 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.

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, although 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 net-

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 underrepresented classes and writing styles.

work 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 he/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.

Although we provided legitimate accuracy statistics for certain comparative tests of some of our algorithms, we deliberately shied away from claiming specific levels of accuracy in general. Neat printers, who are familiar with the system, can achieve 100-percent 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 as writing with otherwise cursive characters but separated by large spaces in a wholly unnatural style), can yield word-level accuracies as low as 80 percent. 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 news groups and some personal testing that suggest accuracies of 97 percent to 98 percent in regular use. For scientific purposes, none of these numbers have any real meaning because our testing data sets 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 analyzing large numbers of experienced users of the systems in the field—a difficult project that 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 underrepresented classes and writing styles. Many of these techniques—stroke-count dithering, normalization of output error, frequency balancing, and 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 overrepresented non-target classes from biasing the net against underrepresented target classes. Frequency balancing prevents overrepresented classes from biasing the net against underrepresented classes. Stroke-count dithering and error emphasis prevent overrepresented writing styles from biasing the net against underrepresented writing styles. One could even argue that negative training eliminates an absolute bias toward properly segmented characters and that stroke warping reduces the bias toward those writing styles found in the training data, although these techniques also provide wholly new information to the system.

Although 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 might be related to the technique of dividing out priors, as is sometimes done to convert from *p(class | input)* to *p(input | class)*. However, we also believe that forcing the net during learning to allocate resources to represent less frequent sample types might 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 article 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; that is, 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 because even a person who normally prints (as opposed to writing in cursive script)

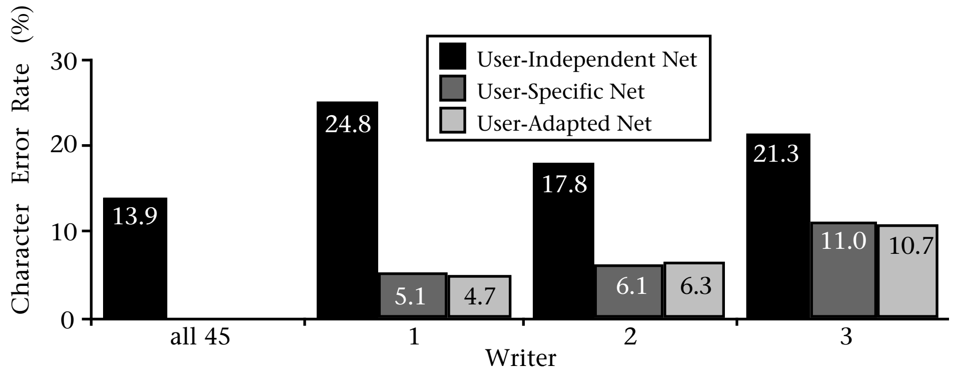


Figure 13. User-Adaptation Test Results for Three Individual Writers with Three Different Nets Each Plus the Overall Results for 45 Writers Tested on a User-Independent Net Trained on All 45 Writers.

might 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 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, in the fashion routinely used in print recognition. This technique has proven to work well, and the version of the print recognizer in the most recent NEWTONS, since 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 article. Because of certain assumptions and constraints in the current release of the software, APR is not yet a full cursive recognizer, although this is an obvious next direction to explore.

The net architecture discussed earlier 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 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. In 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 the same. In addition, training data for all compound characters sharing a common base or a common accent contribute to the network's ability to learn this base or accent as opposed to contributing only to an explicit base + accent combination. Here again, however, the fundamental recognizer technology has not changed significantly from that presented in this article.

Future Extensions

We are optimistic that our algorithms, having proven themselves to work essentially well for connected characters as for disconnected characters, might extend gracefully to full cursive script. On a more speculative note, we believe that the technique might 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, because of random-access memory 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 13 shows a comparison of the average performance on an old user-independent net trained on data from 45 writers and the performance for 3 individuals using (1) the user-independent net, (2) a net trained on data exclusively from this individual, and (3) 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.)

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 this 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 two to a factor of five 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 small amount of training data from an individual will allow us to adapt the user-independent net to the user and improve the overall accuracy for the user significantly, especially for individuals whose writing is more stylized or whose writing style is underrepresented in our user-independent training corpus. Even for writers with common 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 might 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 article are the subject of pending U.S. and foreign patent applications.

References

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

Bengio, Y.; LeCun, Y.; Nohl, C.; and Burges, C. 1995. LEREC: A NN-HMM Hybrid for Online Handwriting Recognition. *Neural Computation* 7:1289–1303.

Bourlard, H., and Wellekens, C. J. 1990. Links between Markov Models and Multilayer Perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12:1167–1178.

Chang, E. I., and Lippmann, R. P. 1995. Using Voice Transformations to Create Additional Training Talkers for Word Spotting. In *Advances in Neural Information Processing Systems 7*, eds. G. Tesauro, D. S. Touretzky, and T. K. Leen, 875–882. Cambridge, Mass.: MIT Press.

Gish, H. 1990. A Probabilistic Approach to Understanding and Training of Neural Network Classifiers. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 1361–1364. Washington, D.C.: IEEE Computer Society.

Guyon, I.; Henderson, D.; Albrecht, P.; LeCun, Y.; and Denker, P. 1992. Writer-Independent and Writer-Adaptive Neural Network for Online Character Recognition. In *From Pixels to Features III*, ed. S. Impedovo, 493–506. New York: Elsevier.

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

Lippmann, R. P. 1994. Neural Networks, Bayesian a Posteriori Probabilities, and Pattern Classification. In *From Statistics to Neural Networks—Theory and Pattern*

Recognition Applications, eds. V. Cherkassky, J. H. Friedman, and H. Wechsler, 83–104. Berlin: Springer-Verlag.

Lyon, R. F., and Yaeger, L. S. 1996. Online Hand-Printing Recognition with Neural Networks. Paper presented at the *Fifth* International Conference on Microelectronics for Neural Networks and Fuzzy Systems, 12–14 February, Lausanne, Switzerland.

Morgan, N., and Bourlard, H. 1995. Continuous Speech Recognition—An Introduction to the Hybrid HMM-Connectionist Approach. *IEEE Signal Processing* 13(3): 24–42.

Parizeau, M., and Plamondon, R. 1993. Allograph Adjacency Constraints for Cursive Script Recognition. Paper presented at the Third International Workshop on Frontiers in Handwriting Recognition, 25–27 May, Buffalo, New York.

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

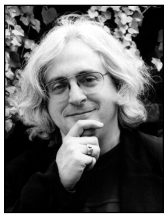
Renals, S.; Morgan, N.; Cohen, M.; and Franco, H. 1992. Connectionist Probability Estimation in the DECIPHER Speech-Recognition System. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, I-601–I-604. Washington, D.C.: IEEE Computer Society.

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

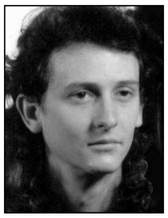
Simard, P.; Victorri, B.; LeCun, Y.; and Denker, J. 1992. Tangent Prop—A Formalism for Specifying Selected Invariances in an Adaptive Network. In *Advances in Neural Information Processing Systems 4*, eds. J. E. Moody, S. J. Hanson, and R. P. Lippman, 895–903. San Francisco, Calif.: Morgan Kaufmann.

Simard, P.; LeCun, Y.; and Denker, J. 1993. Efficient Pattern Recognition Using a New Transformation Distance. In *Advances in Neural Information Processing Systems 5*, eds. S. J. Hanson, J. D. Cowan, and C. L. Giles, 50–58. San Francisco, Calif.: Morgan Kaufmann.

Tappert, C. C.; Suen, C. Y.; and Wakahara, T. 1990. The State of the Art in Online Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12:787–808.



Larry Yaeger (//pobox.com/~larryy) studied aerospace engineering with a focus on computers. He was director of software development at Digital Productions and generated the computer graphic special effects for *The Last Starfighter*, *2010*, and *Labyrinth* as well as a number of television commercials. At Apple Computer, as part of Alan Kay's Vivarium Program, he designed and programmed a computer voice for Koko the gorilla, helped introduce MACINTOSHes into routine production on *Star Trek: The Next Generation*, and created a widely respected artificial-life computational ecology (POLYWORLD). Most recently, in Apple's Advanced Technology Group, he was technical lead in the development of the neural network handprint-recognition system in second-generation NEWTONS.



Brandyn Webb (www.brainstorm.com/~brandyn) is an independent consultant specializing in neural networks, computer graphics, and simulation. He received a degree in computer science from the University of California at San Diego at the age of 18. He was responsible for the San Diego Supercomputer Center's SYNU renderer as well as numerous scientific visualization tools and SIGGRAPH demonstrations for Ardent Computer. He created Pertek's ADME package to create realistic animated models of facilities operations involving human decision and interaction. He designed hardware and software algorithms for Neural Semiconductor and cofounded Apple's handprint-recognition project.

Richard F. Lyon (www.pcmp.caltech.edu/~dick) is known for his research in very large system integration and machine perception at Xerox PARC, Schlumberger Palo Alto Research, and Apple Computer. His achievements include the development of a computational model of the cochlea and auditory brain stem and the application of the neuromorphic system approach to problems in handwriting recognition, optical mouse tracking, and machine hearing. He recently directed the team at Apple that developed the improved NEWTON handwriting-recognition technology. Currently, Lyon continues his research affiliation with the Computation and Neural Systems Program at the California Institute of Technology and serves as a technology-development consultant to several companies in the information industry.

IOS Press Ad
No electronic copy. Shoot from CRC