Comparison of Learning Methods for Handwriting Recognition

Introduction

Even in this era of new technologies, handwriting has remained an important form of communication that sees use in bank checking, addressing postage and filling out physical forms, among other applications. Given the pervasive use of handwriting in daily life, building systems which can automatically read handwriting has been found to be very useful. Yet the task of recognizing characters, while trivial to humans, becomes very difficult with all varieties of character size, character shape, applied writing pressure, and other factors that vary by individual.

Our goal is to build an optical handwriting recognition (OHR) system with artificial neural networks (ANNs), which can learn from various samples of human handwriting. We plan to compare the results of this system with those of a support vector machine (SVM) OHR system.

Related work

One way of classifying handwriting recognition methods, like ANNs and SVMs, distinguishes between offline learning, where the algorithm is trained on examples ahead of time, and online learning, where the algorithm can learn one sample at a time, and adapt at runtime (we attempted the latter approach).

A more classical approach for OHR using SVMs has been described, and has been improved with different kernels, such as the Gaussian time-warping kernel [1][2][3]. A cited issue with SVMs, however, is their high training complexity. Even when kernel-based techniques are used to reduce over-fitting, working with SVMs is a highly memory intensive process.

Various other techniques of handwriting recognition have been implemented and compared for signature recognition, such as multi-layer perceptrons, minimal distance classifiers, nearest neighbor classifiers and hidden Markov models [4]. Online ANNs have been used for classifying numerals and other characters, and have been enhanced with adaboosting, a technique that makes the algorithm more sensitive to misclassified samples [3][5]. There will likely be challenges with applying this to handwritten character recognition, however, because of the larger number of target classes (26 uppercase characters versus 10 digits).

Approach

ANNs

The primary algorithm that we implemented was a feedforward classification ANN with perceptron nodes, using MATLAB. We chose this because it was the most straightforward ANN to implement, and we trained the parameters using backpropagation. We compared this to an SVM algorithm, using the same set of data.

We used 1867 training examples and 468 test samples of uppercase letters, all of which we wrote by hand and transferred to 20×20 pixel images (see Figure 1). The features for our algorithm represented the pixel values of each image, and were thus treated as vectors in $\mathbb{R}^{400}$. While mini-strokes of characters have also been used as features, using the pixel values was the more straightforward method. The algorithm output a vector $a^{(L)} \in \mathbb{R}^{26}$, with a dimension for each possible character, and we then took the dimension with the highest value as the output (for example, if the highest value in the output vector was in the 26^{th} entry, we would predict that the input image was of the letter Z).

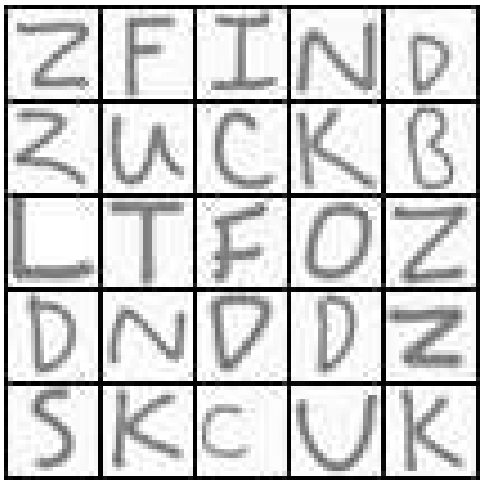

Figure 1: Random sample of our handwritten character data.

The perceptron ANN works by feeding an input vector of features, $a^{(1)}$, into the next layer, $a^{(2)} \in \mathbb{R}^n$. This can then be fed to any number of hidden layers, before finally leading to the output layer, in our case $a^{(L)} \in \mathbb{R}^{26}$. There can be an arbitrary number of hidden layers, and each layer can have an arbi-

trary number of units, though in each trial we set the hidden layers be of the same size n (see Figure 2).

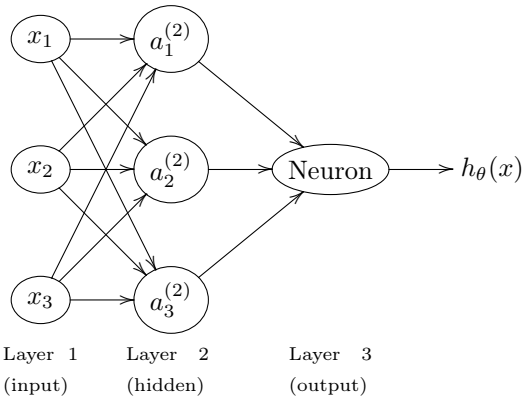

Figure 2: Representation of artificial neural network with an input layer of three nodes (x), a single hidden layer of three nodes ($a^{(2)}$) and a single node in the output layer. Inputs are fed to the hidden layer by $a^{(2)} = g(\Theta^{(1)}x)$, and then to the output layer and finally to the activation function, which is the sigmoid hypothesis function $h_\theta(x)$ in this case.

In our implementation, the values of each layer, $a^{(l)}$ are calculated by multiplying the previous layer $a^{(l-1)}$ by a parameter matrix $\Theta^{(l-1)}$, and taking the sigmoid function $g(z) = \frac{1}{1-\exp(-z)}$ of this. In general, then, layer l is calculated by the equation $a^{(l)} = g(\Theta^{(l-1)}a^{(l-1)})$. The key part of the algorithm was determining the neural network parameters $\Theta^{(l)}$, which we found by backpropagation with a gradient descent-like function.

SVMs

Classifying is often a problem solved by logistic regression, but while logistic regression requires a large number of training examples for each feature, SVMs solve this problem by using a kernel to map the data to higher dimensional kernel-induced feature space. We used the radial basis function (RBF) kernel in our project, which is given by

$$k(x_a, x_b) = e^{\frac{-.5\|x_a - x_b\|^2}{\sigma^2}},$$

and the optimization task is reduced to an iterative greedy algorithm. For the SVM implementation, we used the MATLAB optimization toolbox and a MDM (Mitchell-Demyanov-Malozemov) solver for optimizing the RBF kernel.

Results

Below are the tables with the results for the ANN and SVM. The key metrics we used were the training time and the test accuracy. Our results show that our ANN had the superior performance, reaching 84.5% accuracy on the test set, compared to the maximum 81.5% of the SVM. It also took less time to train on average, with a typical time of less than 100 minutes, and with even the best performing trial taking less time to train than any of the SVM trials.

We first tested the ANN algorithm for different numbers of hidden layers, then hidden units, and finally for the regularization parameter λ, at each step finding the optimum setting for each parameter. We found that the accuracy was greatest with 2 hidden layers of 400 hidden units, and with the regularization parameter of $\lambda = 1$.

The key metrics for the SVM's performance were also the training time and the test accuracy, which varied based on the threshold parameter ϵ and regularization parameter C (the optimization function iterated until reaching a threshold of $10^{-\epsilon}$). The chart shows that the best result of 81.9% test accuracy occurred while optimizing to the threshold $10^{-\epsilon} = 10^{-5}$ with regularization parameter $C = 10$.

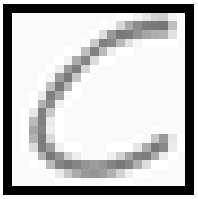

Figure 3: Example of error from the test set: letter C, misidentified as L by the neural net.

Discussion

The results from the neural network clearly show that increasing the number of hidden layers only increases the recognition rate up to a certain point before it starts decreasing. The number of hidden layers for ANNs is a convex function which had a maximal value of 2 for our OHR dataset. Increasing the number of hidden units also improved the level accuracy as expected, until after about 400 units.

As expected, the training time for the ANN increased as we raised the number of hidden units and number of hidden layers. For the SVM, both the ϵ and regularization parameter are convex functions with optimal values, as evident from our results, with low values of C allowing the SVM to overfit, and higher values preventing the SVM from obtaining a

ANN Results

Hidden Layers	Hidden Units	Training Time (min)	Training set Accuracy (%)	Test Set Accuracy (%)	λ
2	80	40	71.8	44.6	1
2	100	45	84.6	61.1	1
2	150	76	91.9	65.6	1
2	300	137	96	81	1
2	400	44	94	81.6	0.3
2	400	101	97	**84.5**	1
2	400	64	94.4	70.85	3
2	400	71	93.7	73.5	10
2	500	104	96	77.6	1
3	200	60	86	63	1
4	300	217	77.7	53.65	1

SVM Results

ϵ	C	Training Time (min)	Training Accuracy (%)	Test Accuracy (%)
5	10	123	84.3	**81.9**
4	10	127	81.5	77.9
3	10	110	75.3	70.2
7	10	112	72.1	62.4
6	10	119	80.2	76.8
5	1	135	85.8	70.2
5	0.1	140	90.3	65.4
5	100	110	75.3	70.3
5	1000	105	70.2	60.1

proper model. The ϵ parameter was used to control the senstivity used to fit the training data; a higher value caused fewer support vectors to be selected, while a lower value than the optimum only marginally increased the accuracy. The optimal parameters are only specific to our dataset, however, and the relative and absolute performance of the algorithms would likely be much improved by a significantly larger dataset.

Due to the lack of time, we were not able to use other variations in these techniques. A possible improvement could be found in choosing a different kernel for the SVM, such as a linear, polynomial or sigmoid kernel. Additionally, we could have also used kernel functions to improve the ANN, though it would have significantly increased the training time, which was already exceeding 2 hours per trial in the case with several hidden layers. Because the performance of the ANN was already surpassing that of the SVM, adding an RBF kernel to the ANN would have increased the gap further. Another variation could have been to use mini-strokes as features rather than raw pixel values, which would have increased the accuracy of the OHR in both the techniques.

A potential extension of the neural network algorithm for optical character recognition could be the detection of entire words, provided the words were written in discrete printed format (as opposed to cursive). This extension would require the training on an entire English dictionary, which would, however, be excessively time-consuming with our limited computing power.

References

[1] C. Bahlmann, B. Haasdonk, and H. Burkhardt, "Online handwriting recognition with support vector machines - a kernel approach," in *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, 2002, pp. 49-54.

[2] A. Abdul Rahim, M. Khalia, C. Viard-Gaudin, and E. Poisson, "Online handwriting recognition using support vector machine," in TENCON 2004. 2004 *IEEE Region 10 Conference*, 2004, pp. 311-314 Vol. 1.

[3] S. Cho, "Neural-Network Classifiers for Recognizing Totally Unconstrained Handwritten Numerals," *Neural Networks, IEEE Transactions on*, vol. 8, pp 43-53, 1997.

[4] R. Plamondon and S. N. Srihari, "Online and off-line handwriting recognition: a comprehensive survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 63-84, 2000.

[5] H. Schwenk, Y. Bengio, "AdaBoosting Neural Networks: Application to on-line Character Recognition," in International Conference on Artificial Neural Networks (ICANN '97), pp. 967-972, Springer, 1997.