Using Neural Networks for Character Recognition

Kevin Luikey
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
kluikey@gatech.edu

Emily Chang
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
echang60@gatech.edu

Thao Dinh
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
thaolbdinh@gatech.edu

Yinghui Dong
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
ydong99@gatech.edu

Abstract

Neural Networks are a powerful tool for learning complex, nonlinear relationships between inputs. Vaguely inspired by the biological neural networks present in the human brain, neural networks are particularly useful in image recognition. Given sufficient numbers of labeled training examples, it can learn to recognize patterns and generalize to new data. Convolutional neural networks are the most popular neural network approach when it comes to computer vision, and are capable of classifying images with little or no preprocessing of the data.

1 Motivation

Optical character recognition has a number of practical applications such as, bank-check processing, postal address interpretation, license plate scanning, book scanning, digitizing paper records, etc. So many records in the world transmitted via paper, like passport documents, invoices, bank statements, receipts, business cards, mail, etc. Essentially any task which requires interpreting printed or handwritten records can be streamlined and improved by employing optical character recognition techniques.

2 Related Works

Feature extraction: Feature extraction relies on humans identifying features that they believe to be relevant to the image?s classification, such as aspect ratio, percent of pixels above horizontal half point, percent of pixels to the right of vertical half point, average distance from image centre, etc. then training on those features. This approach is more labor intensive to set up because humans have to manually identify features that may be relevant to the classification. This differs from our approach that does not involve any preprocessing of the data.

Clustering: The purpose of clustering is to divide a dataset into different clusters (subsets) given prior number of clusters. Clustering method should have two properties- homogeneity and heterogeneity. Homogeneity means that data belongs to one cluster should be as similar as possible while heterogeneity means that data belongs to different clusters should be as different as possible. K-means clustering has been used to group sets of characters. Clustering is an unsupervised learning approach which does not use class labels and also does not return prediction of a class label. Ours is

Preprint. Work in progress.

a supervised learning approach which trains on labeled data, and can output a predicted class for new data points.

Principal component analysis: Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components, thereby reducing the dimensionality of the training data. [3]

T-SNE: Visualization the data point in 2D or 3D can help people to get more intuitive understanding of the data, if the data can be correctly represented in low dimension space. T-SNE is such a technique that can do nonlinear dimensionality reduction and very suitable for visualization data in 2D or 3D.

3 Approach

3.1 Data Description

The expanded mnist dataset is a set of over 800,000 labeled examples of handwritten characters a to z, A to Z, and 0-9.

3.2 Techniques

3.2.1 Fully connected neural network

Our first approach was too build and train a fully connected neural net to classify the characters. There was no preprocessing of the data other than to reshape it to fit as an input to a model. We structured our neural net so that every pixel was an input neuron to the model, taking in the intensity of that pixel, and floating point value from 0 to 1. We initialized the starting weights randomly with a bias toward smaller value in hopes of reducing the complexity of the trained model, making it less likely to overfit. We primarily used two activation functions, sigmoid and reLU. Sigmoid was originally very popular in neural nets, but recently reLU, defined as f(x) = max(0, x), has been more commonly used. reLU has some useful properties that make it more appealing than sigmoid, such as being a more computationally efficient calculation, and dropping out useless neurons, as it outputs 0 when its input is negative, and will not change due to its derivative also being 0 when negative. We also experimented with different numbers of neurons per layer and number of layers. Finally, our output layer contained one neuron for each class, where the label vector for a given class would be a 1 for the corresponding element and 0 for all others.

3.2.2 Convolutional neural network

We also experimented with the effectiveness of convolutional neural network models, comparing their effectiveness with fully connected artificial neural networks in optical character recognition. Our convolutional neural network consists of 7 layers - input layer, convolutional layer, pooling layer, regularization layer, flatten layer, fully connected layer, and output layer. The convolutional layer creates feature maps, which are created by taking a filter matrix, smaller than the image, and doing element wise multiplication between the filter and the image and adds the outputs. After doing this over the whole image, we get a feature map. After that, the pooling layer reduces the dimensionality of the feature maps from the previous layer in a number of ways (e.g. maxing, averaging, summing, etc.). For our implementation, our regularization layer used dropout. Dropout randomly ignores neurons in the training phase of a certain set of neurons and is used to prevent overfitting. The flatten layer is used to convert the 2D matrix into a 1D matrix in order to be classified and processed in the fully connected layer. The fully connected layer connects every neuron from the previous layer into every neuron in this layer. Conceptually, this layer holds the the information that is most important from all of the convolutional layers. This way, our model is able to more easily classify the images by simplifying the features from the convolutional layers and at the same time, learns non-linear combinations of those features. In our case, we wanted 128 neurons in our fully connected layer, each neuron representing a single pixel from the original image. Our model uses a softmax activation function which makes all the outputs sum to 1, making each class have a probability-like value, and the output layer consists of one neuron for each class as with our first approach.

3.2.3 Cross validation

We trained our models using 5 fold cross validation. We divided the data set into 5 random groups and trained our model 5 separate time, once for each combination of 4 groups as the training set, and the 1 group not used to train as the validation set. This ensures that the model generalizes to new data it didn?t train on and that it doesn?t overfit, as if it does start to overfit, we would see the accuracy on the validation set decrease while the accuracy on the training set increases.

3.2.4 Principal component analysis

When train the fully connected neural network, we try to introduce PCA to reduce the input data dimension, since for neural network we originally use each pixel as input neuron which has 784 neurons. We do the experiment with different principal components on different layers neural network models.

3.2.5 t-SNE

We use t-SNE to visualize the MNIST data. Also we will try use PCA to reduce the MNIST data to 2D dimension to see the difference between two method in visualization. And finally we will try use PCA first to decrease to dimension to a relative low dimensions (such as 50 components PCA) and then use t-SNE down to 2D dimension.

4 Implementation details

4.1 Code and package

For our experiments we used cPickle in python to extract the training data from the data files into numpy arrays. Then for our fully connected neural network we coded the training and cross validation algorithms in python without any libraries. Although CNNs are generally difficult to implement due to their complex nature, we were able to use a convenient python package called Keras. Keras is a high level neural network API backed by, in our case, TensorFlow that allows us to create convolutional neural networks in a relatively simple way. We use Sklearn t-SNE and PCA package to do the t-SNE and PCA on origin MNIST input data.

4.2 Cross validation

In each of our approaches, we used cross validation to ensure our model can generalize, to reduce overfitting, and to get a more certain measure of accuracy, as just one run could produce better or worse results due to luck. To do this, we shuffled our training data to be in a random order, then divided it into k folds, where each fold was 1/kth of the data. Then we would run our training algorithm n=k times with the nth fold serving as the validation set for that run, and the rest of the data as the training set.

5 Experiments

5.1 Fully connected neural network

5.1.1 Fully connected neural network results

Our training algorithm used stochastic gradient descent to find a local minimum, calculating the gradient for one training example at a time, then stepping the weights and biases in that direction. After trying multiple different network structures, we found that 128x96x64 neurons in the hidden layers using the reLU activation function led to the best validation accuracy.

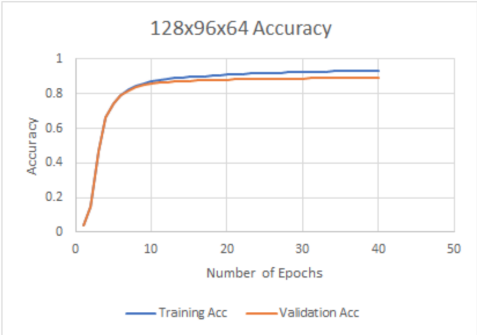

Figure 1: accuracy vs. epochs for 128*96*64 model

model	128*96*64	128*64*32	64*48*32
sigmoid	89.15%	88.58%	85.99%
reLU	88.24%	87.46%	86.31%

Table 1: average validation accuracy over 5 folds for fully connected neural network

5.1.2 Fully connected neural network analysis

Our model did not seem to overfit at all as evidenced by our validation accuracy continuing to increase continually, and not dropping off even after many epochs. This is likely due to the large number of inputs relative to the number of hidden layer neurons. We had 784 inputs, and usually around 128 to 32 neuros in the hidden layers. As the number of parameters decreases, the complexity of the functions it is able to model decreases, and since we had many more input nodes, 784, the actual function is much more complex than the model is able to perfectly capture, and so the limited number of neurons in the hidden layers prevented our model from overfitting the data, while still being able to capture patterns present in the data

5.2 Convolutional Neural Network

Our model is applied to NIST SD 19 handwritten dataset. The model consisted of one convolutional layer that creates 32 5x5 feature maps using a ReLU activation function. We also use max pooling with a pool size of 2x2. A dropout layer with a 0.20 probability of "dropping out". We used a ReLU activation function inside of our fully connected layer, but used the softmax activation function in the output layer to give us a nice probability output. Keras automatically performs back-propagation on our model.

5.2.1 Convolutional Neural Network analysis

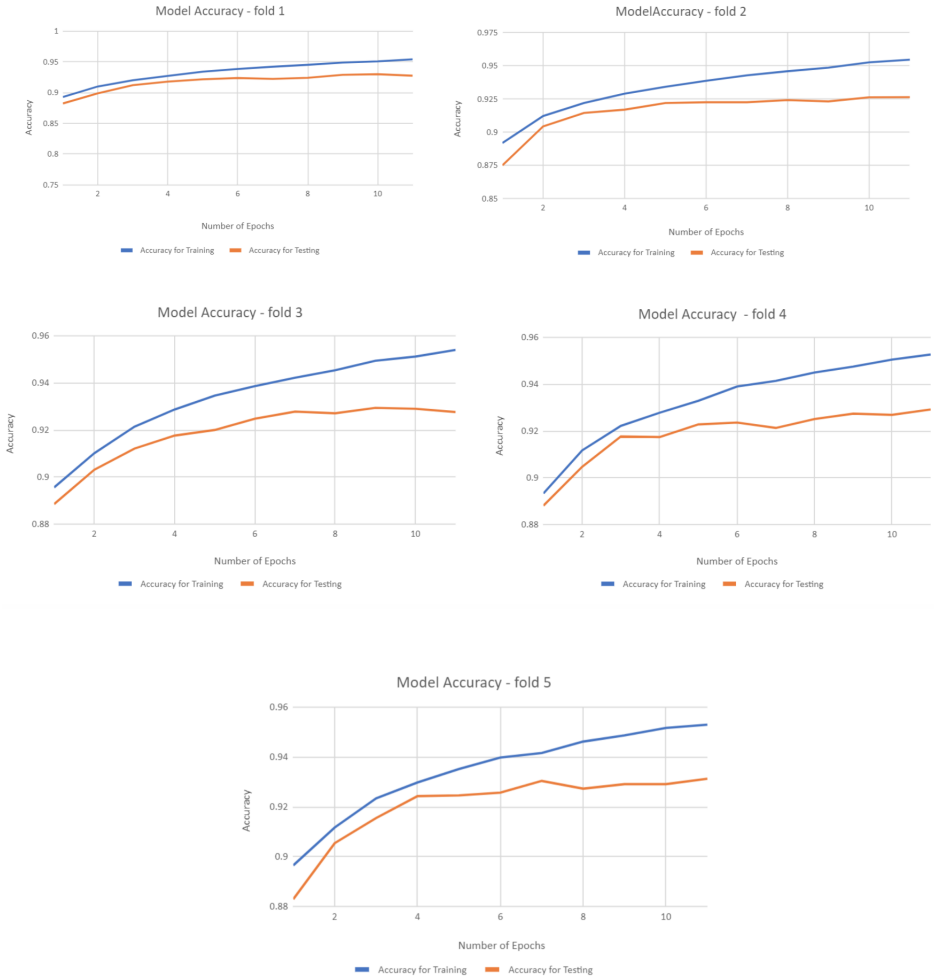

Figure 2: Accuracy vs number of epochs for Convolutional Neural Network for 5 folds

Average training accuracy over 5 folds	Average validation accuracy over 5 folds
95.37%	92.84%

Table 2: training and validation accuracy

5.3 Neural network with principal component analysis

The MNIST picture has 28*28=784 pixels, in NN it can be treated as 784 features or 784 dimensions, but not all features are as important as others. Paper [2] and article [1] shows some result that PCA can help improve the NN training process. So we want to test this, how PCA can influence the NN training process. We curious about some following issues:

- Will the data initially processed by PCA can reduce the NN model training time?
- Will the PCA always reduce the model accuracy and if true how the different number components of PCA will different influence
- If the NN model different with different number of layers, will the PCA has different influence?

For NN model, we first use the own coded NN model with 2 hidden layers and 16 neurons in each layer, the training data is 50000 MNIST data points and test on 10000 data points. Cross Validation

is not used here since we do PCA experiments mostly focus on the influence of PCA on NN training speed and accuracy loss.

With 2 hidden layers 16 neurons in each layer model, for input data, each time we will choose different number of principal components. We choose 4,8,16,32,48,64,96,128,256,784 as principal numbers, Input data will be reduced to those numbers of dimensions and then we check the accuracy and speed behaviour. Each model will be trained for 10 mins.

Below are some figures from experiments we do by using PCA with different number of components.

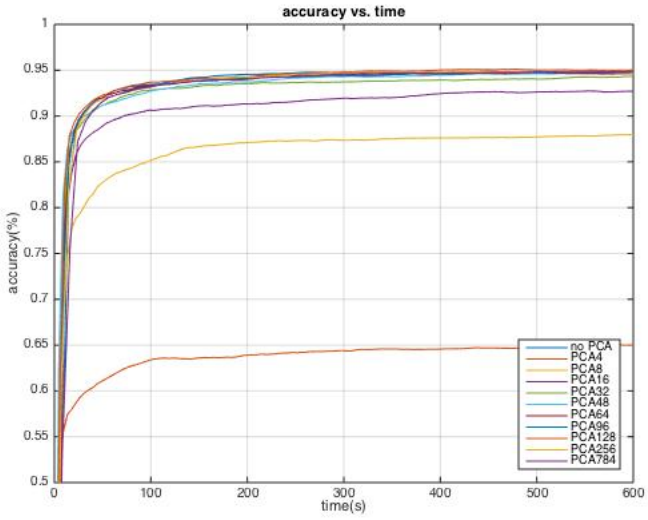

Figure 3: accuracy vs. time (2 hidden layers)

From this figure we see that once PCA components is lower (small or equal to 16), the accuracy is much lower than model without PCA. When the PCA components is larger than 64, the model loss curve with PCA and without PCA becomes similar.

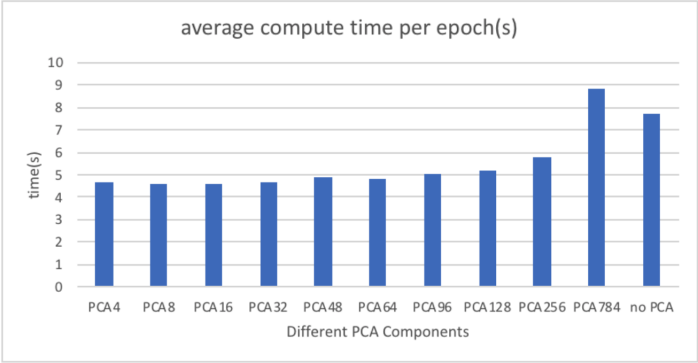

Figure 4: computing time per epoch for different PCA components (2 hidden layers)

For all PCA components smaller than 100, the average computing time for each epoch is less than 5s and not vary too much between each others.

We find that for the model we use, PCA doesn?t improve the training process in any aspect. In same time, model trained without PCA has higher accuracy. And PCA even doesn?t improve the training speed of one epoch.

One possible reason that PCA has no improvement on our model is that our model use two hidden layers, so maybe the first hidden layer with 16 neurons has the same effect with PCA, since they both reduce the input 784 dimensions into lower dimensions. From the mentioned paper and article, they both use one layer neural network and they get speed improvement and accuracy improvement from PCA. So we need to do some experiments on NN with only one hidden layer and see whether PCA has any influence.

Now we use NN model with only one hidden layer, and we still use 16 neurons in this layer, the training data set and test data set remaining.

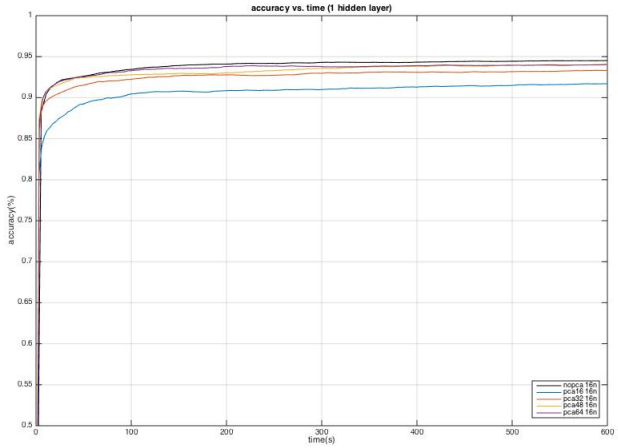

Figure 5: accuracy vs. time (1 hidden layers)

Below is the computing time for each epoch on average for different PCA components

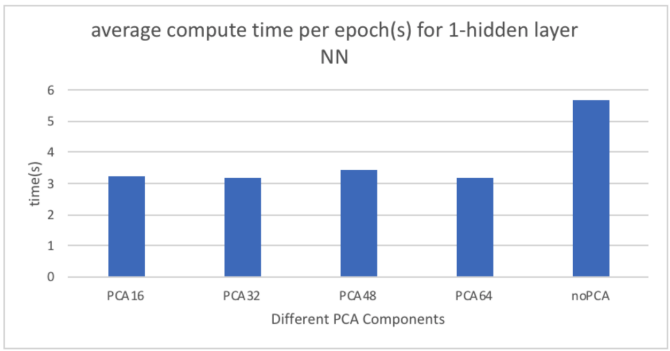

Figure 6: computing time per epoch for different PCA components (1 hidden layers)

This time we only use one hidden layer NN, but still the model without PCA trained faster on accuracy. Based on the computing time per epoch, PCA make the computing 40% faster per epoch, but since PCA loses information, in each epoch the accuracy improvement is decreased. Compare with the paper and article mentioned before, even we use the same one-hidden layer neural network, we still cannot recur the same experiment outputs.

5.4 t-SNE and PCA visualization experiements

First we use PCA reduce the MNIST data 784 dimensions into 2D space and visualized. We use 10000 data points. The variation is [0.09931776, 0.07755832]

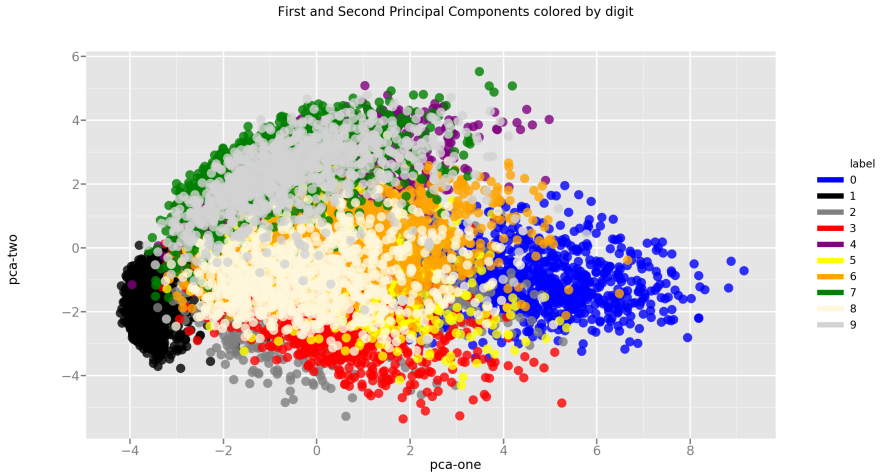

Figure 7: MNIST data in 2D space by PCA

From the figure we can see that only digit 0 and 1may be separateable. And most digit points are overlapped with each other.

Then we experiment on using t-SNE to reduce the dimensions to two and then plot

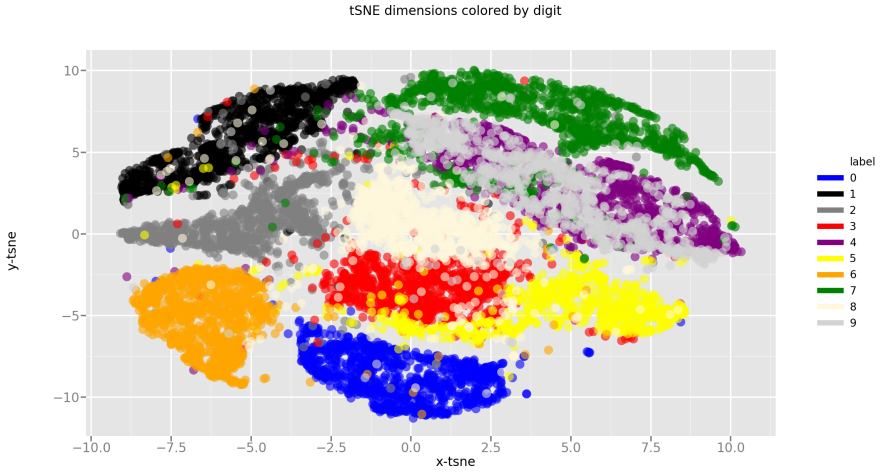

Figure 8: MNIST data in 2D space by t-SNE

From this figure we can easily find 9 separate blocks. Only the digit 4 and digit 9 seems still overlapped with each other.

References

[1] Meigarom Diego Fernandes. Does pca really improve classification outcome. `towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32`. [Online; accessed 25-July-2018].

[2] Vineet Singh and Sunil Pranit Lal. Digit recognition using single layer neural network with principal component analysis. In *Computer Science and Engineering (APWC on CSE), 2014 Asia-Pacific World Congress on*, pages 1–7. IEEE, 2014.

[3] Wikipedia contributors. Principal component analysis. `https://en.wikipedia.org/wiki/Principal_component_analysis`, 2018. [Online; accessed 25-July-2018].