Tooling around with MNIST: what's the best model architecture?

Alec J. Hoyland
Center for Systems Neuroscience
Boston University
Boston, MA 02215
`ahoyland@bu.edu`

Abstract

This is the abstract.

1 Introduction

Optical character recognition (OCR) is the ability to take as input an image including handwritten or printed text, and output the sequence of characters contained within the image. OCR engines have many applications, including data entry (e.g. for checks, passports, invoices), book-scanning (e.g. for Project Gutenberg), pen computing, and in assistive devices (e.g. for blind and visually-impaired individuals). In an OCR engine, text is read from a scanned document or image, and translate the images into a form manipulable by the computer, such as plain text, enabling a multitude of other analyses and operations. As the saying goes, "You can't grep dead trees!"

A successful model is one which accurately predicts the correct label for the image (i.e. recognizes the digit), and does so quickly. A model that fails to accurately label is not useful, as is one that labels accurately, but too slowly to be useful in real-world applications.

We describe two measures of efficiency, training time and testing time. The former is the total runtime of the training phase, and the latter is the total runtime of the testing phase. Since the model need only be trained once (or perhaps fine-tuned later), minimizing runtime on the testing set is more important, since in a real-world scenario, the "testing" set will be monumentally larger, consisting of any documents or images the model is used to annotate.

Intuition indicates that simpler models, which involve less computationally-involved operations, will perform better on measures of efficiency, though it is possible that the decrease in accuracy would be unacceptable.

In this paper, we will explore a convolutional neural network and a feedforward neural network, with various training schedules, and benchmark accuracy and runtime speed.

2 Related Works

The MNIST dataset [1, 2] has been used as a standard machine learning benchmark for more than two decades. It is comprised of 60,000 28x28 grayscale images of handwritten digits 0-9. The handwriting samples themselves were written by American high school students, and American Census Bureau employees. While concerns about the construct validity of the dataset have been raised in recent years [3] the dataset and its variants remains a staple of machine learning benchmarking, and serves well for evaluating the quality of various model architectures.

Preprint. Work in progress.

The MNIST dataset is suited to a classification task, where the image is read as input, and the output is the numerical digit label (i.e. 0-9). Many different model architectures have been implemented to achieve highest accuracy on the classification task.

Many different model architectures have been tested against the MNIST dataset. Linear classifiers comprise some of the simplest of models, and traditionally fare poorly on digit identification [1]. Some of the first attempts based on multilayer neural networks with backpropagation [1, 4], were performed by the original curators of the MNIST dataset. Many variations have been tested, including limited receptive field models [5], and convolutional neural networks [6]. The latter exploits elastic training image deformations, treating the input as a full matrix, rather than a vector of features. Image deformation has also been explored in hidden Markov models [7]. Methods have improved in GPU-based computations in deep, big simple neural networks [8, 9], resulting in a broad family of neural network models. Support vector machines [10] have also been used for image-recognition tasks.

3 Methods

In this paper, eight models are implemented and compared to each other. The goal is to determine what sort of basic model architecture performs the best, in efficiency of training/evaluating, as well as in accuracy of the resultant classification. We consider a convolutional neural network and a fully-connected feedforward neural network, with modifications to the loss function and training schedule.

In all cases, Adam is used as the optimizer [11]. It computes a bias-corrected gradient using the first two moments, and can be thought of as a trust-region stochastic gradient descent algorithm.

The cross-entropy function is used as the loss function.

$$H(p, q) = -\mathbb{E}(\log q) = -\sum_{s\in\mathcal{S}} p_s \log q_s \tag{1}$$

This function measures the number of bits required to encode an input if an encoding scheme is optimized for probability distribution q rather than the true distribution p. In the case where $p = q$, the cross-entropy is the entropy, and the loss is minimized.

3.1 Convolutional neural network

The convolutional neural network (CNN) consists of three convolutional layers, a fully-connected layer, and a softargmax output.

The first convolutional layer operates on a 28x28 image, with a 3x3 window, padding of (1, 1), and a ReLU activation. This leads to 16 units in the second convolution layer, which operate with a 3x3 window and padding of (1, 1) on 14x14 images. This convolutional layer passes through ReLU activation to a third convolution layer, which acts on 7x7 images, with the same padding and ReLU activation as before.

Each convolutional layer is separated by a 2x2 maximum pooling layer, for selecting the most representative features.

Finally, the 3-dimensional tensor is flattened to 2 dimensions, and passed to a 288-unit fully-connected layer with linear activation, and 10 outputs (corresponding to the digits). The softargmax function is used to squash the outputs to normalized probabilities.

Loss is computed by the cross-entropy function, and the Adam optimizer with $\beta = 0.001$ is used for training [11].

3.2 Fully-connected neural network

A multi-layer neural network was also tested. This network consists of 2 fully-connected, feedforward layers each with 32 hidden units. ReLU activation is used between the hidden layers, and the softargmax function is used to squash outputs.

Cross-entropy loss is used, and Adam is the optimizer algorithm.

3.3 Modifications to existing models

Several alterations were made to the base models to attempt to improve accuracy and efficiency.

First, the input data were fuzzed with Gaussian noise ($\xi = 0.1$) to try to make the model more robust. Second, a modified training schedule was implemented, with early-stopping at 99.9% accuracy to prevent overfitting, and an adaptive learning rate.

In the adaptive learning rate schedule, while the loss function was used to train the model, stopping was determined by the accuracy on the testing dataset. If no improvement was seen in 5 epochs, the learning rate was decreased by 90% to a minimum of 10^{-6}. If no improvement was seen after 10 epochs, training stops early, as the model is assumed to have converged to a local minimum.

These modifications result in a total of eight models:

- CONV, the base convolutional neural network
- CONVFUZZ, CNN with input fuzzing
- CONVADPT, CNN with adaptive learning rate
- CONVFULL, CNN with both input fuzzing and adaptive learning rate
- DENSE, the base fully-connected feedforward neural network
- DENSEFUZZ, feedforward network with input fuzzing
- DENSEADPT, feedforward network with adaptive learning rate
- DENSEFULL, feedforward network with both input fuzzing and adaptive learning rate

3.4 Evaluating quality of models

Models were evaluated for accuracy and efficiency.

Accuracy is defined as the number of correct labels divided by the size of the test set. Efficiency was measured by training time, and testing time. Training time is defined as the amount of time for which the model is trained. Testing time is the amount of time it takes for the model to label the testing set.

Unless training time is prohibitively long, testing time is more important. This is because possessing a model that can identify characters and glyphs quickly, is important for real-world applications of optical character recognition. A model does not need only to be accurate, it should also be fast.

We use two testing datasets. One is the original, MNIST testing dataset. The other is the MNIST testing dataset with Gaussian noise ($\xi = 0.1, \mu = 0, \sigma = 1$) added to each image. This "fuzzed" dataset is designed to test how well the models can handle noisy data, such as in an optical character recognition task on an old manuscript.

4 Results

Models were written in JULIA [12] using the FLUX package for autodifferentiation and machine learning [13]. Computations were performed on a 16-core computer at 3.7 GHz, with 32 GB of RAM. No GPU-acceleration was used.

Models were compiled, and timed using built-in macros.

Training time is defined as the amount of time it took to train the model to a stopping point. We tested $n = \{5, 10, 20\}$ epochs, each time, training through minibatches of the entire training dataset. In non-adaptive models, this is the time of $n.$ epochs through the training set (60,000 images). In adaptive models, the models ran until assumed convergence.

Testing time is defined as the amount of time it took to run the model on the 10,000 image testing set. We define testing speed as the number of digits identified per second (in digits/second).

Testing accuracy is the ratio of mean number of correct classifications divided by the total for all images in the testing set. We define testing efficiency as the mean number of digits identified correctly per second (in digits/second).

$$\text{testing speed} = \frac{10,000}{\text{testing time}} \tag{2}$$

$$\text{testing accuracy} = \frac{\#\text{ labeled correctly}}{10,000} \tag{3}$$

$$\text{testing efficiency} = \text{testing accuracy} \times \text{testing speed} \tag{4}$$

We see that the CONV-family of models perform the best in accuracy, but take the longest to run, both in training time and in testing time. Convolutional neural networks perform well on image-classification tasks because they treat the image as a matrix, which can be filtered over several convolutional layers. In contrast, traditional feedforward DENSE networks accept rasterized image input. The convolutional layers allow sub-features to be extracted out, leading to better classification. However, convolution is an expensive operation, requiring more time.

We also see that DENSE adaptive models perform terribly in accuracy. The testing accuracy does not decrease fast enough from an untrained (randomly-initialized) model for the adaptive learning rate heuristic to be of any use. Instead, the model quits training too early, resulting in a poorly-trained model.

Of the models tested, the adaptive learning rate is useful when training computationally-expensive models for many epochs. In these cases, the model will quit when the accuracy is maximized, avoiding overfitting, or performing excessive training without an increase in accuracy. Since the number of epochs trained for was low, the adaptive learning rate did not make a significant difference.

Table 1: Results of simulations (5 epochs)

Model Name	Training Time (s)	Testing Time (s)	Testing Accuracy
CONV	320	3.72	0.985
CONVFUZZ	314	3.42	0.984
CONVADPT	336	3.78	0.987
CONVFULL	376	3.38	0.986
DENSE	67.9	0.053	0.906
DENSEFUZZ	144	0.055	0.901
DENSEADPT	9.51	0.058	0.116
DENSEFULL	0.114	0.103	0.117

Table 2: Results of simulations (10 epochs)

Model Name	Training Time (s)	Testing Time (s)	Testing Accuracy
CONV	762	3.83	0.986
CONVFUZZ	713	3.48	0.988
CONVADPT	627	3.30	0.987
CONVFULL	642	3.34	0.987
DENSE	68.3	0.062	0.900
DENSEFUZZ	145	0.0625	0.902
DENSEADPT	22.8	0.059	0.114
DENSEFULL	29.1	0.103	0.117

5 Discussion

5.1 Convolutional neural networks: accurate and slow

The results of the model evaluations indicate that CNNs outperform feedforward neural networks on classifying images in the MNIST dataset, in terms of accuracy. The convolutional neural network

Table 3: Results of simulations (20 epochs)

Model Name	Training Time (s)	Testing Time (s)	Testing Accuracy	Testing Efficiency
CONV	1215	3.314	0.988	2982
CONVFUZZ	1223	3.43	0.988	2879
CONVADPT	1338	3.44	0.99	2876
CONVFULL	1395	3.73	0.988	2652
DENSE	68.1	0.053	0.899	168 498
DENSEFUZZ	143	0.054	0.903	167 210
DENSEADPT	35.4	0.057	0.111	20 115
DENSEFULL	33.7	0.085	0.116	13 811

Table 4: Results of simulations (20 epochs) with fuzzing

Model Name	Testing Accuracy	Testing Accuracy (fuzzed)	Testing Efficiency
CONV	0.989	0.982	2963
CONVFUZZ	0.988	0.987	2876
CONVADPT	0.990	0.986	2865
CONVFULL	0.989	0.989	2651
DENSE	0.899	0.895	167 730
DENSEFUZZ	0.903	0.900	166 544
DENSEADPT	0.115	0.111	19 483
DENSEFULL	0.117	0.116	13 752

achieved $< 98\%$ accuracy after 5 epochs for all four models tested. This is an order of magnitude better than the dense networks. However, the models took significantly longer to train, and perform about 60 times slower in the classification task. Therefore, it is better to use a convolutional neural network when the main objective is accuracy, or when the training/running time are unimportant.

A better method might be to train two networks, both a CNN and a feedforward NN, and pass the input through the feedforward neural network. One can establish a confidence measure, such as a probability cutoff, as a measure of "unsureness". Images whose labels about which the feedforward network is uncertain, would be passed to the CNN for classification. This has the advantage of speeding up classification, while only increasing the error rate by a small amount.

5.2 Input fuzzing makes models more robust

Input fuzzing makes models more robust to noise. In all cases, models perform worse on classifying the fuzzed testing data than on the vanilla testing images. Models which were trained with the fuzzed loss function (CONVFUZZ and DENSEFUZZ), suffered smaller hits to their accuracy, than the models trained with the normal loss function. This indicates that training using a fuzzy loss function improves a model's robustness to noise.

While the fuzzing reduces the deleterious effect of noise on classification accuracy, training models with the fuzzed loss function took much longer.

This is partly due to the vectorized operations to generate the fuzzy input data each iteration adding to the total training time, and because the fuzzy loss function was implemented using global variables in a script. Using global variables in functions makes it hard for the JIT Julia compiler to optimize vectorized operations. In comparison, the model training itself was extremely fast.

Overall, fuzzing does not add significantly to training time, and produces a more robust model.

5.3 Adaptive learning rates are not for everything

The adaptive learning scheme comes with many theoretical advantages, but in practice, does not work well for all problems. Since the scheme was conditioned on the model's accuracy with the testing data, adaptive learning of this form is only possible when testing data is readily available during the training process. Furthermore, adaptive learning prevents overfitting by exiting when the accuracy is maximized, but doesn't prevent underfitting. The dense models failed to converge fast enough, and so the adaptive learning scheme assumed a local minimum had been reached, and training exited, resulting in underfit models.

None of the models were trained long enough for the adaptive learning scheme to pay off. We can see after about 20 epochs, the adaptive model matches the fixed learning rate model. If models were trained for more than 20 epochs, the adaptive learning rate would be advantageous, since early exiting would save time and prevent overfitting.

6 Conclusion

In this paper, we discuss using feedforward, dense neural networks and convolutional neural networks on optical character recognition tasks, using the well-studied MNIST dataset of handwritten digits. We find that dense networks are much faster to train and during classification, but result in less accurate models than CNNs. In addition, while input fuzzing in the loss function adds to the computational load during training, resultant models are more robust to noisy inputs. Adaptive learning and early exiting are found to be a effective methods for preventing overfitting, but may be more parsimonious than necessary, resultig in underfit models. Furthermore, we propose a hybrid model, in which all images are classified by a dense network, and images for which the dense network is uncertain (low maximal probability), are passed to an auxiliary CNN for better classification. This framework has the advantage of improving classification speed without strongly decreasing the accuracy.

References

[1] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: A case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 77–82 vol.2.

[2] Handwritten digit database.

[3] Chhavi Yadav and Léon Bottou. Cold Case: The Lost MNIST Digits.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 86(11):2278–2324.

[5] Ernst Kussul and Tatiana Baidyk. Improved method of handwritten digit recognition tested on MNIST database. 22(12):971–981.

[6] Patrice Y. Simard, Dave Steinkraus, and John Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis.

[7] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Ney. Deformation Models for Image Recognition. 29(8):1422–1435.

[8] Dan Claudiu Cirean, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. 22(12):3207–3220.

[9] Dan Cirean, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.

[10] Dennis Decoste and Bernhard Schölkopf. Training Invariant Support Vector Machines. 46(1):161–190.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization.

[12] Jeff. Bezanson, Alan. Edelman, Stefan. Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. 59(1):65–98.

[13] Michael Innes. Flux: Elegant machine learning with Julia. 3:602.