Convolutional Neural Neural Network with Backpropagation

Axel Ind (*4563539*)

November 19, 2017

Abstract

This document details work to create and evaluate a convolutional neural network that successfully classifies the MNSIST database of images into the numbers which each image represents. The network architecture consited of 2 convolutional layers, each followed by a pooling layer and a single fully connected softmax output layer. Accuracy of 99.2% is achieved.

1 Dataset

The data set used is the MNIST dataset of hand-written numbers represented by 28px $\times$ 28px grayscale images with intensity values in the range $(0, 256)$.

2 Network Architecture

Inputs

Stochastic Gradient Descent: $784 \times n$ (where n is the batch size).

Outputs

$Y = (y_0, y_1, y_2, ..., y_{10})$
Each class corresponds to a classification of: $(0, 9) \in \mathbb{Z}$.

3 Training

3.1 Propagation

All training results are achieved using backpropagation to update weights.

Optimization

The Dropout algorithm was applied to the final fully-connected layer. This addition significantly reduced the extent of over-fitting encountered.

Testing Hyperparameters

- **Network architecture**: Input$-\{$Conv-Relu-Pool$\}^2-$FullyConnectedLayer-Softmax
- **NumFilters**: 16
- **KernelSize**: 3×3
- **BatchSize**: 512

Testing Architecture

CPU	i7-6700HQ
GPU	GTX970 (6GB)
RAM	16GB
Hard Disk	SSD

4 Results

4.1 Variation of Learning Rate

Because overfitting was generally observed after about 10 epochs, learning rate data is shown for $numEpochs \leq 10$.

Figure 4.1 clearly illustrates that a learning rate of 0.1 is too high to converge on an acceptable accuracy.

By contrast, a learning rate of 0.0001 converges very slowly and does not offer an appreciable improvement in performance.

For this reason, a learning rate of between 0.01 and 0.001 is to be preferred.

Number of Parameters

Using the parameter calculation technique provided by Standford (`http://cs231n.github.io/convolutional-networks/`), the information in Graph 4.1 was generated. It shows an exponential increase in the number of parameters as the number of filters is increased.

Addition of the Dropout learning technique produced improved results as illustrated in Graph 4.1.

These results closely mirror the observed computation time for increased numbers of filters.

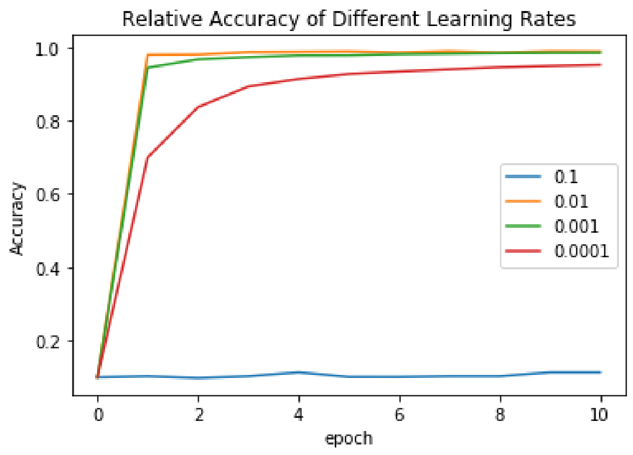


Figure 1: Graph showing the learning accuracy of different learning rates over time for the CNN architecture used.

4.2 Runtime Performance

Because the computation time for $epoch_0 \approx epoch_k$, runtime performance calculations were performed with an $numEpochs = 1$.

GPU Computation

GPU computation time appears to follow an almost linear pattern for $numFilters < 256$ ($r > 0.9959$). A product, no doubt of the exceptional ability of the GPU to simultaneously perform operations on multiple filters.

Futhermore, for $\exists$ numFilters > 1, GPU computation strictly outperforms CPU computation.

CPU Computation

CPU computation is shown to closely mirror an exponential equation of the form AB^x (Figure 4.2) with ($r = 0.9344$). An observation which strongly suggests that convolutional neural networks cannot have their number of filters efficiently scaled by CPU computation.

This conclusion is further corroborated by the extension of the filter count to 256 (Figure 4.2), wherein the exponential curve becomes even more obvious.

Furthermore, even at the base of 8 filters, the CPU computation is still almost 9 times slower than the equivalent GPU computation.

5 Conclusion

This paper has shown that:

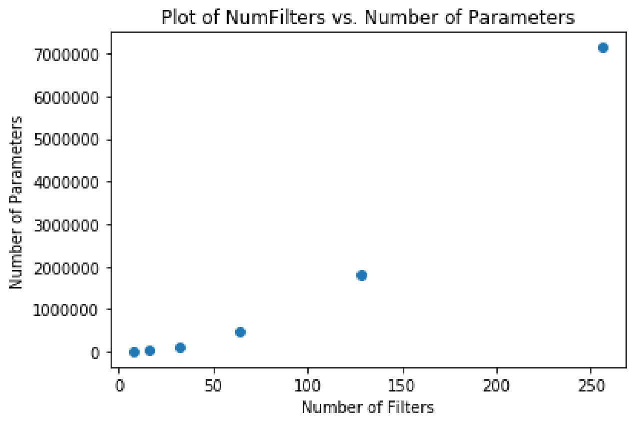


Figure 2: Figure illustrating the exponential increase in the number of parameters as the filter size increases.

1. Convolutional neural networks are an appropriate learning technique with which to classify MNIST data. In particular the architecture used in this experiment performs best with a learning rate $0.001 < \mu < 0.01$.

2. GPU computation of CNN filters progresses linearly for $numFilters \leq 256$, and strictly dominates CPU computation for all values $numFilters > 1$.

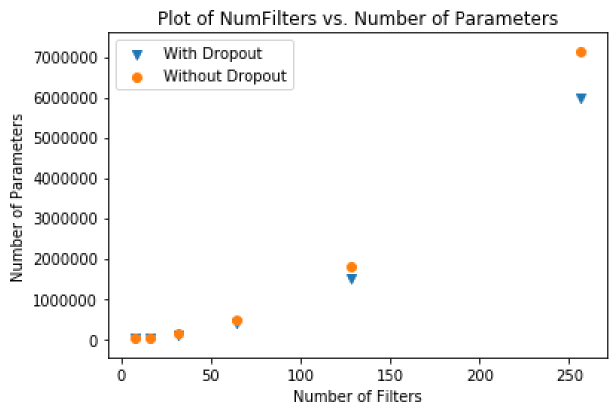


Figure 3: Figure illustrating the reduced parameters calculated per training example when Dropout trianing is used.

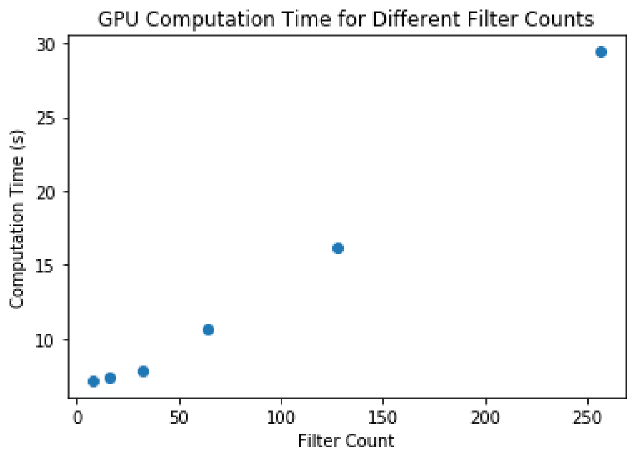
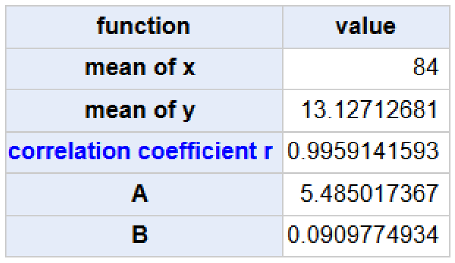


Figure 4: Figure showing the computation times for further increased filter sizes on the network when GPU computation is used.



function	value
mean of x	84
mean of y	13.12712681
correlation coefficient r	0.9959141593
A	5.485017367
B	0.0909774934

Figure 5: Table showing the key statistics of fitting a linear model to the observed value pairs for GPU computations.

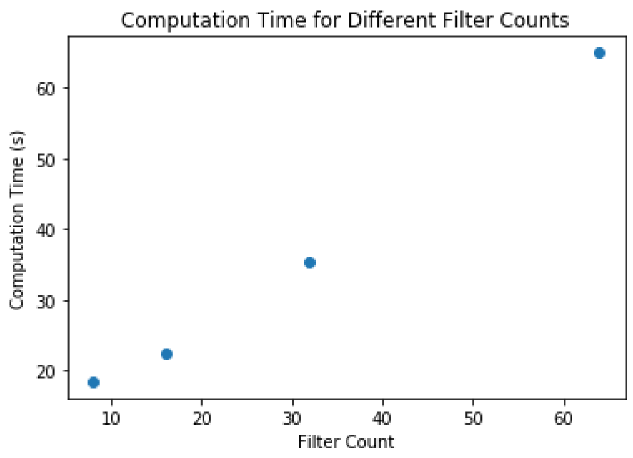


Figure 6: Figure showing the computation times for further increased filter sizes on the network when CPU computation is used.

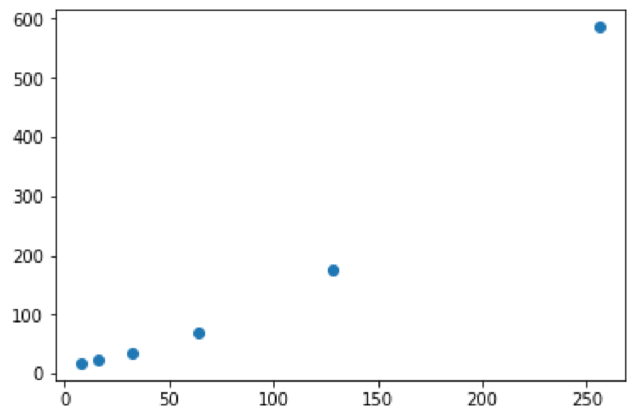
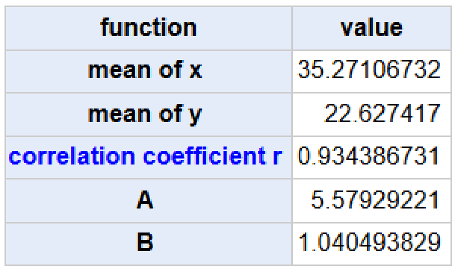


Figure 7: Figure showing the computation times for further increased filter sizes on the network when CPU computation is used.



function	value
mean of x	35.27106732
mean of y	22.627417
correlation coefficient r	0.934386731
A	5.57929221
B	1.040493829

Figure 8: Table showing the key statistics of fitting an exponential model to the observed value pairs for CPU computations.