Classifying Handwritten Digits with a Convolutional Deep Neural Network

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural networks excel at solving classification problems. A simple implementation of a deep neural network with convolutional and pooling layers saw success at classifying members of the MNIST dataset: labeled images of handwritten digits.

1 Introduction

In order to classify the digits in the MNIST dataset, a deep convolutional neural network was used, but what exactly does that mean? The network contained convolutional, pooling, and fully connected layers, each serving either to increase efficiency of training or accuracy of classification. Training was performed on randomly selected batches of 100 digits. 5000 of these batches were used for training. This process was repeated twice more in order to achieve a clearer idea of the networks performance.

2 The MNIST Dataset

The MNIST dataset consists of 70,000 images of handwritten digits, and labels defining which digit they represent. There are 60,000 in the training set, and 10,000 in the testing set. Each image has a height and width of 28 pixels on a single black/white color channel.

3 Network Structure

The network consisted of of two convolutional layers, two pooling layers, and one fully connected layer.

3.1 Convolutional Layers

Each convolutional layer applied a number of convolution filters to 5x5 subregions of the image. This forced the neural network to consider regions of the image in the same way a human might from its start, instead of allowing it to learn that groups of nearby pixels have significance over many training steps.

The first convolutional layer applied 16 filters, which multiplied the size of the data by 16. The second applied 32 filters, which similarly increased the size by 32.

The second convolutional layer was acting not on pixels, but on the data generated by the first convolutional layer. Essentially, it drew inferences about the input not from relationships between groups of pixels, but from relationships between groups of those groups.

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

3.2 Pooling Layers

Between the two convolutional layers, the amount of data being processed grows massively, which greatly increases the amount of processing time required for training. This was remedied in part by the inclusion of pooling layers after each convolutional layer, which reduced the resolution of the data by a factor of four. This comes at a cost to quality of classification, but with such a great decrease in processing time, its inclusion is well worth it.

The specific method of pooling used was max pooling over 2x2 regions with a step size of 2. This means that the data entering the layer was broken into discrete 2x2 regions, and each was reduced to its single highest value. Another method, such as taking the average of the values within each region, or taking the most extreme value coud have been used. This process can be thought of in much the same way one can think of reducing the resolution on any image. Quality is indisputably lost, but without massive reduction in resolution, the image is still much the same.

3.3 Dense Layer

Finally, after all convolution and pooling is complete, the data passes through one dense, or fully connnected layer, which simply draws patterns from the data passed in, which is finally reduced to ten values, one for each possible label. The greatest output value acts as the network's prediction. The specific dense layer used here contained 512 nodes.

This layer had a dropout rate of 40%, which means that 40% of values, randomly selected, are simply not passed forwards. This encourages a robustness in the network that should allow it to succeed at classification even missing 40% of data, and should help to avoid total reliance on any one feature.

4 Training

In order to train the network, a batch of randomly selected training samples is selected and run through the network. Outputs are generated by the network for each member of the batch, and loss is calculated. Loss is simply the difference between the desired output and the actual output of the network. In this case, the desired output is a zero for each label except for the correct one, which should have a value of one.

This loss is used to perform gradient descent on the network, by which weight values are changed from the output layer working back towards the beginning, such that, should the same input arrive again, the actual output will be closer to the desired output. The learning rate for this process was selected as 0.001. This value was selected arbitrarily, and could very well be changed for better results. Empirical testing is required to find the best learning rate.

This process was repeated 5000 times, and then the network was reset, and learning was done again from scratch twice more.

5 Results

Training steps took on average 27.3 seconds each. The average accuracy after training was 93.55%. Compared to some attempts at solving the same problem with neural networks, this was a bit of a bust. As of 2013, some attempts were seeing accuracy as high as 99.79%. Very impressive. With much more training time and number fiddling based on empirical testing, the implimentation described here could possibly have reached similar quality.