Convolutional Neural Networks for Classifying Handwritten Digits

Favyen Bastani
fbastani@perennate.com

Zhengli Wang
wzl@mit.edu

December 5, 2014

1 Introduction

Image recognition algorithms have become pervasive: on Google Maps Street View, image recognition protects privacy by automatically blurring faces and license plate numbers [1]; robots use object recognition to carry out various tasks; meanwhile, individuals routinely employ optical character recognition and facial recognition. Over the last decade, image recognition approaches have changed dramatically. Initially, feature extraction techniques such as HOG and FAST were typically combined with support vector machine (SVM) classifiers, with research often focusing on selection of SVM kernels and defining features for specific applications. Recently, though, neural networks have received renewed attention and achieved high classification performance.

Cells in the mammalian visual cortex are able to identify local patterns in the visual field [2]. Convolutional neural networks (CNNs) aim to do the same through *convolutional and pooling layers*. While traditional neural networks are composed of densely connected layers, where each neuron in the previous layer connects to each neuron in the next layer, convolutional layers restrict filters to be applied on local sub-regions of an input image. By training these layers through back-propogation, these filters eventually detect specific features in the image that help with classification.

In this project, we implement a fast CPU-based CNN in Python using the `numpy` scientific computing library, and apply it to the MNIST handwritten digits database [3]. We evaluate the performance of various neural network parameters (including layer configuration, output layer activation function, and training algorithm) in terms of classification accuracy and speed, and compare CNN to feature extraction approaches with SVM. We find that ...

In Sections 2 and 3, we detail convolution and pooling functions, the convolutional neural network structure and training algorithm, and our implementation. In Section 4, we experiment with various neural network parameters, and in Section 5, we compare with SVM-based algorithms.

2 Neural networks

An artificial neural network (ANN) consist of neuron units that, when composed in a layered network, compute some output function from an input vector $y = h(x)$. Each neuron has a specific set of input val-

ues (which may be from the input vector of the network, or from other neurons) and a single output. The output is generally computed as $n_{w,b}(z) = f(w \cdot z + b)$ for some *activation function* f, weight vector w, and bias parameter b. The sigmoid function is often used as the activation function, i.e. $f(l) = \frac{1}{1+e^{-l}}$, as it mimics the hard threshold activation pattern of a biological neuron.

Neurons are connected to form a neural network. Typically, the network is composed of layers of neurons, where neurons in the first layer accept inputs from the input vector x, and neurons in each following layer take inputs from the outputs of the neurons in the preceding layer. Then, the network output vector y consists of the outputs from neurons in the last layer of the network. In fully connected neural networks, a neuron takes inputs from every neuron in the previous layer.

Forward propogation refers to the process of propogating some input vector x through each layer of the neural network to compute $y = h(x)$. In a fully connected neural network, this can be done using efficient matrix operations (which can take advantage of fast matrix libraries such as `numpy`). Let Z_l be the output vector of the lth neural network layer (where $Z_0 = x$) and n_{li} be the ith neuron in layer l; then, $Z_{l+1} = f(W^{\mathrm{T}} Z_l + B)$ where W_{ij} is the weight from n_{li} to $n_{l+1,j}$ and B_j is the bias parameter for neuron $n_{l+1,j}$.

Neural networks can be trained using back propogation. At a high level, we compute a cost function on the outputs from the neural network given some input and desired output, and propogate the error (in the form of partial derivatives of the cost) starting at the last layer to the first layer. These error terms are then used to compute the derivative with respect to specific weight and bias parameters, which can then be updated via gradient descent.

2.1 ANN for multi-label classification

Artificial neural networks are often used for multi-label classification. However, the sigmoid activation function is not well suited for this purpose, where we want to assign exactly one label c for each input vector x. Instead, we use *softmax activation function*, defined as

$$f(L, i) = \frac{e^{L_i}}{\sum_j e^{L_j}}$$

Here, L is a vector containing inputs to the activation function for each neuron in the softmax layer. For back propogation, the cost function given a desired output vector o is

$$E = -\sum_i o_i \log(y_i)$$

2.2 Stochastic gradient descent

In normal gradient descent neural network training, we update weights based on performing averaging derivatives from back propogation across all training samples. Stochastic gradient descent instead randomly selects mini-batches of a predetermined size from the training samples on each iteration and updates based on the selected samples (the batch size is generally selected to yield optimal performance with matrix libraries, although in our implementation we perform back propogation separately for each sample). This increases solution sparsity, as well as convergence speed.

We implement one variation of stochastic gradient descent that significantly reduces training time in many cases. Rather

than update weights directly from the computed gradient Δ, we apply a momentum that retains gradients from previous iterations. For a training rate α and momentum m, we update parameters θ as

$$v = mv + \alpha\Delta$$

$$\theta = \theta - v$$

3 Convolutional neural networks

Images, along with many other data types, exhibit strong local patterns that need to be identified in order to correctly classify images. Fully connected neural networks may ignore these patterns, or training them to identify local patterns will take a significant number of iterations. Convolutional neural networks take an alternate approach, where the localized structure is directly encoded into the structure of the neural network. In this section, we discuss the functionality of convolution and pooling layers, and strategies of incorporating them in neural networks.

3.1 Convolution

Images can often be hierarchically broken down into features and sub-features that support image recognition. For example, a flower may contain a stem and a petal, each of which consist of edges at various positions and angles. Additionally, these features may exist at various locations in the image: if a photograph is shifted in one direction, an image recognition algorithm should still be able to classify it in the same way.

3.2 Pooling

3.3 Network

4 CNN evaluation

5 Comparison with SVM-based recognition

6 Conclusion