Convolutional Neural Network for Image Classification

Chen Wang
Johns Hopkins University
Baltimore, MD 21218, USA
cwang107@jhu.edu

Yang Xi
Johns Hopkins University
Baltimore, MD 21218, USA
yxi5@jhu.edu

Abstract

Neural network, as a fundamental classification algorithm, is widely used in many image classification issues. With the rapid development of high performance computing device and parallel computing devices, convolutional neural network also draws increasingly more attention from many researchers in this area.

In this project, we deduced the theory behind back-propagation neural network and implemented a back-propagation neural network from scratch in Java. Then we applied our neural network classifier to solve a tough image classification problem CIFAR-10. Moreover, we proposed a new approach to do the convolution in convolutional neural network and made some experiments to test the functionality of dropout layer and rectified linear neuron.

1 Introduction

As a significant application of machine learning and pattern recognition theory, image classification has become an important topic in individual's life. Face recognition, vehicle detection and signature verification are all excellent examples. In this project, we choose a CIFAR-10 database which is a general image classification problem contains 10 different classes of images, such as airplane, truck, frog and etc. Due to the low resolution and complex image content, this problem is frequently treated as a hard issue in this area. Through the implementation and application of the neural network algorithm, we wish to deeply figure out how neural network will work for this image classification issue. Moreover, we made sufficient experiments about how convolutional neural works in this issue from different aspect. Through applying the knowledge we learned in this course, we also proposed a improved version of convolutional neural network comparing with the basic convolutional neural network.

2 Background

Image classification has been one of the most important topics in the field of computer vision and machine learning. As a popular benchmark in this field, the CIFAR-10 databases (Krizhevsky, 2009) are frequently used as a benchmark to judge the performance of an classification algorithm. Many researchers paid neuromus efforts in this problem . Even though the best known result has achieved a 94% of accuracy (Karpathy, 2011), it is still a quite challenging issue and many other well designed algorithms in the performance ranking list can only achieve around 70% of accuracy.

Neural network and a series of derivative algorithms (Mitchell, 1997) have been applied for image classification problems for a long time. The whole network consists of multiple layers of neurons and each of them can be treated as a single processing unit. The basic idea of neural network is inspired by the biological neural networks (Aleksander and Morton, 1990). The backpropagation algorithm, the most popular way to train a multi-layered neural network, was proposed by Bryson and Yu-Chi (1969) and further improved by Werbos (1974), Rumelhart et al (1986) .

Instead of simple multiplication between neuron

outputs and weights, convolutional neural network incorporates more properties such as convolution and down-pooling (Simard et al, 2003). Due to the rapid development of computing platform like GPU, increasingly more researchers start to apply this algorithm in complex image classification. According to many other researchers? work, we realize the difficulty of this dataset and classification issue. Normally it takes more than dozens of hours of time to train a good performance model, even with high performance GPU and some other parallel computing techniques.

3 Model

In this section, at first, since we have implemented an neural network without using any open souce, so we'll give a brief introduction to neural network by explaining the feedforword and back propagation steps in mathematics. Then we will describe how we use the basic neural network to do the classifying job on CIFAR-10 dataset.

3.1 Neural Network

Neural networks are made up many layers, and each layer was consisted of different number of neurons which have trainable weights and biases. All the neurons are fully connected to the neurons in previous and post layers. The first layer is input layer which was viewed as a signle vector. The last layer is output layer whose output view be as predict result. Other layers between input and output layer are call hidden layer which will process and pass the 'message' from previous layer to post layer. Every neuron will receive some inputs from neurons in previous layer. Then it performs a dot product of all the inputs, following with a non-linearity optionally function as output of this neuron.

3.1.1 How does the neural network work

1. Initalize all weight $w_{ij}^{(l)}$ in the neural network, and $w_{ij}^{(l)}$ stands for the weight on the path from the ith neuron in $(l-1)$th layer to the jth neuron in lth layer

2. Feedforword:

(a) Take one train data, set the input values of each neuron $y_i^{(0)}$ in the 0th (input) layer and the label in output layer.

(b) Compute total net input from pre-layer to each hidden layer neuron $x_j^{(l)}$ in the next layer, and squash the total net input using an activation function as the output in next layer $y_j^{(l)}$ (here we use the logistic function), then repeat the process with the output layer neurons.
$X^{(l)} = Y^{(l-1)}W^{(l-1)}, W^{(l-1)} \in \mathbf{R}(d(l-1) \times d(l))$
$Y^{(l)} = \sigma(X^{(l)}), X^{(l)}$ and $Y^{(l)} \in \mathbf{R}(d(l) \times 1)$, in which $d(l)$ means the number of the neuron in lth layer.

3. Back progagation
$W^{(l)} = W^{(l)} - \eta dW^{(l)}$

(a) Compute the gradients dW matrix on weights from the second last layer to last output layer $(l = n)$. Let error $E = \frac{1}{2}(label_j^{(n)} - y_j^{(n)})^2$, then calculate the gradient of each weight between last layer and its pre-layer.
$\delta^{(n)} = -(E - Y^{(n)})\sigma'(X^{(n)}), \delta^{(n)} \in \mathbf{R}(d(n) \times 1)$
$dW^{(n-1)} = (Y^{(n-1)})^T \delta^{(n)}$

(b) Compute gradient in previous layers $(l = n-1)$
$\delta^{(l)} = W^{(l)}(\delta^{(l+1)})$
$dW^{(l-1)} = (Y^{(l-1)})^T \delta^{(l)}$

Repeat the process until whole weights in the neural network were updated.

4. Return to the second step (feedforward), keep updating the weights until reach the iteration.

3.2 Convolutional Neural Network

Convolutional neural network works based on basic neural networks which was described above. So what does the CNNs change? There are several variations on CNNs layers architecture: Convolutional Layer, Pooling Layer and Fully-Connected Layer. Fully-Connected Layer is just acting as neural network which we have already covered in previous. CNN algorithm has two main processes: convolution and sampling, which will happen on convolutional layers and max pooling layers.

3.2.1 Convolution process

Every neuron takes inputs from a rectangular $n \times n$ section of the previous layer, the rectangular sec-

tion is called local receptive field.
$x_{i,j}^{(l)} = \sigma(b + \sum_{r=0}^{n} \sum_{c=0}^{n} w_{r,c} x_{i+r,j+c}^{(l-1)})$

Since the every local receptive field takes same weights $w_{r,c}$ and biases b from the equation above, the parameters could be viewed as a trainable filter or kernel F, the convolution process could be considered as acting an image convolution and the convolutional layer is the convolution output of the previous layer. We sometimes call the trainable filter from the input layer to hidden layer a feature map with shared weights and bias.

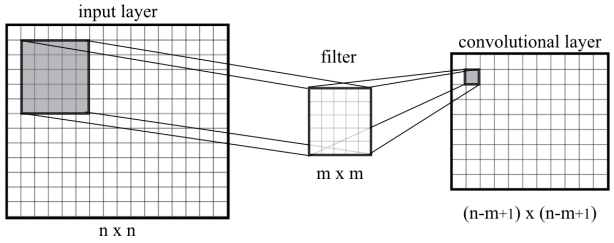


Figure 1: Convolution process

3.2.2 Sampling process

After each convolutional layer, there may be following a pooling layer. So the sampling process happens between convolutional layer and pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it. In our work, we will take the maximum of the block as the single output to pooling layer.

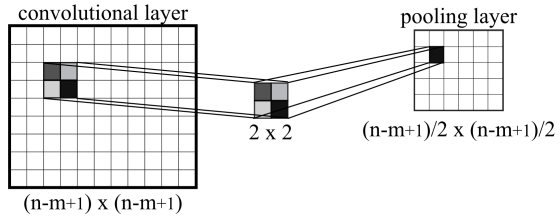


Figure 2: Sample process

CNNs have much fewer connections and parameters, and also they are easier to train. Discarding the fully connected strategy means to pay more attention on the regional structure, which is very meaningful when we take image processing into consideration, since there are less relations between different region of an image.

3.3 Activation Units

As usual, the standard activation of a neurons output are $y(x) = (1 + e^{-x})^{-1}$ or $y(x) = tanh(x)$ where x as the input.

3.3.1 Vanishing gradient problem

Vanishing gradients occurs when higher layer units are nearly saturated at -1 or 1, leading lower layers of a Deep Neural Network to have gradients of nearly 0. Such vanishing gradients cause slow optimization convergence, and in some cases the final trained network may fall off to a poor local minimum station.

3.3.2 Rectified Linear Neuron

A Units using sigmoid or tanh functions can suffer from the vanishing gradient problem.

$$y = max0, b + \sum_{i=1}^{k} x_i w_i$$

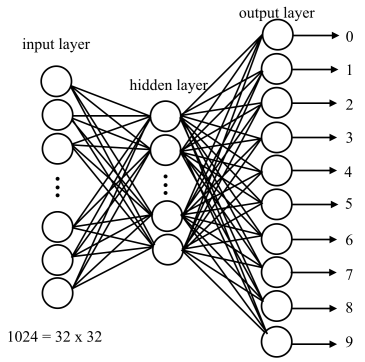
ReLU doesn't face gradient vanishing problem as with sigmoid and tanh function. Rectified linear units don't saturate for large inputs.

3.4 Regulation Dropout

Dropout is a ways of controlling the capacity of Neural Networks to prevent overfitting. It's extremely simple and effective. During training, the dropped out neurons do not participate in feedforward and also backpropagation. Dropout could be viewed as sampling a neural network within the full Neural Network, and only updating the parameters of the sampled network based on the input data. Dropout layer was always included in between fully-connected layers. In dropout layer, the choice of which units to drop is random. In our experiment, we set all the units with a fixed probability 0.5.

3.5 How we use the neural network

The input layer of the network contains neurons encoding the values of the input pixels. Our training data for the network will consist of 32 by 32 pixel images from CIFAR-10 dataset, and so the input layer contains 1024 (32*32) neurons.



The second layer of the network is a hidden layer. We denote the number of neurons in this hidden layer by n, and we'll experiment with different values for n. The example shown illustrates a small hidden layer containing just n=15 neurons.

The output layer of the network contains 10 neurons which stand for 10 types of image label. We number the output neurons from 0 through 9 and select the neuron with the highest activation value as predicting result.

4 Our work

4.1 Data acquisition

The CIFAR-10 database can be obtained from https://www.cs.toronto.edu/ kriz/cifar.html. This resource consists of 6 batches and each contains 10000 32*32 color images in 10 classes and the data is given in the format of 10000*3072 numpy array for each batch to represent 10000 RGB image. To apply this data in our works, we wrote some python scripts to reshape this data into two different formats for both our Java version of neural network and the python version of convolution-al neural network. Because we are not focusing on getting best performance as a course project, we choose to change the RGB image into grayscale to save computation time and make it easy to process.

4.2 Implementation of Neural Network

As the first part of our work, after a detailed deduction of back-propagation, we implemented a Java version of neural network from scratch with-out using any external machine learning libraries. Then we applied our neural network to this CIFAR-10 image classification problem and test the performance for different parameters.

In our implementation, we used the Java learning framework provided by the class as the skeleton of our code. Through writing python scripts, we transform the CIFAR-10 data to the same format with our homework data and write it to disk. Therefore we can use the existing library to load our data. After that, each single pixel in our grayscale image plays a rule of single neuron input at the first layer. From the perspective of algorithm itself, we use mini-batch stochastic gradient de-scent to train the model, and more specifically the back-propagation algorithm mentioned above is applied to calculate the gradients.

During the process of our implementation, the time efficiency for our Java program bothered us a lot. Relatively speaking, Java has less support for complex and large scale of matrix operations. After we firstly applied our implemented program to the CIFAR-10 dataset, through Eclipse debugger we went through the output of each layer and the result seems reasonable. However, even only using one batch of data set (10000 images), a single iteration of three-layer network takes several minutes to run. As a comparison, we configured the same parameters and data size to a python neural network library, the output result is quite similar, but a single iteration only takes a few seconds instead.

After many testing, we figured out this is caused by Java's garbage collection mechanism. We followed a Princeton?s code to do the matrix operation in Java. It creates a new array object in the memory for every matrix operation. When the computation become really complex, like in this neural network application, it would make JVM frequently collect garbage in the heap, which takes significant amount of time. To solve this problem, we also pass the result array to the matrix operation method as a parameter. Due to the reference passing for object in Java, we can avoid creating many new objects in this case. The running time turns out to be much more efficient, but still not as good as python. This is also an important reason for us to move to python for the rest of our work, convolutional neural network.

4.3 Convolutional filter

Through the exploration of how convolutional neural networks and how it is different from a normal neural network, we find an interesting problem

about the convolution layer. Generally speaking, convolution can be treated as a way of feature extraction in computer vision and image processing. The difference is that this kind "feature extraction" can also "learn". A traditional way to initialize convolutional filters, which also means the parameters of the first layer, is through some "random" method, such as uniform distribution. Then during the process enough amount of training, these initial weights would become increasingly more reasonable. Below is an example of a convolution layer which contains 10 different initial filters.

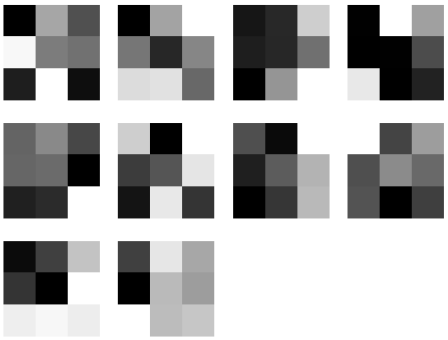


Figure 3: Convolutional filters generated by uniform distribution

Based on this, we tried to make some improvement and then apply it to our CIFAR-10 problem. In-stead of traditional convolution kernel, we wish to use some different image filter to implement the convolutional layer. For example, different laplacian filter, sobel filter, prewitt filter and etc.

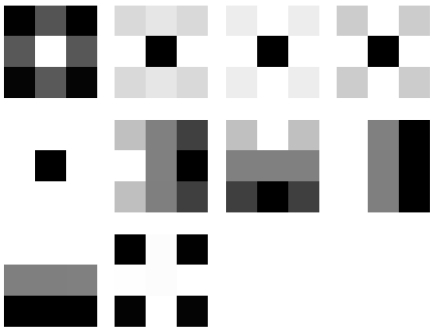


Figure 4: Our convolutional filters

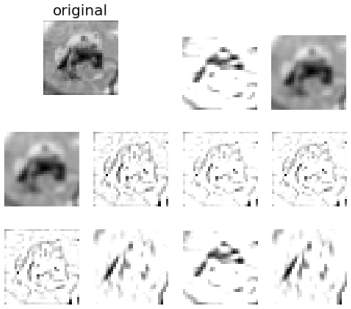


Figure 5: the convoluted result after a hundred iterations

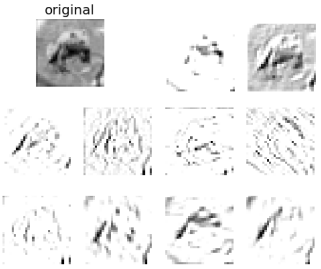


Figure 6: the convoluted result after a hundred iterations

4.4 Dropout

We have tried to add dropout layer in the basic convolutional neural network between last convolutional layer and fully connected layer (output layer). We wish to explore how well the dropout layer has regularied.

4.5 ReLU

To contrast the difference by using sigmoid and rectified function as activation, we tried to use sigmoid and rectified function respectively to see distinct result.

5 Result

5.1 Neural Network

From the table below, it shows the validation accuracy under five neural network with different structure: From the experiment, all the accuracy was picked by maximum value near the maximum point of (tran cost/validation cost) in timeline. The table indicated that the neural network with more hidden layer mill perform better than the neural network with less hidden layer in some limit. (Since training

a neural network with much more layer will cost numerous time which is impossible to run on our normal device, we only diccuss the this issue within our experiment.) We also found out that training with small mini batch size takes more time to reach the optimal, and it's more possible that traing with large mini batch size may fall to local optimal.

5.2 Convolutional Neural Network

Convolutional Neural Network performs much better than the original Neural Network on classifying the CIFAR-10 dataset. CNNs has increased almost 20 percent accuracy with less parameters, and also the training time was reduced, though it's not obvious.

According to Figure n, it is obviously that our proposed new convolution approach achieves a much better performance at first, and after a bunch of iterations the difference between the two ap-proaches? performances become less and less. However, even after two methods start to converge respectively, our new approach still has a ad-vantage over the traditional method.

This result is fully consistent with our expectation, because after all our selected filter would help much more in the information extraction comparing with parameters generated from uniform distribution. The reason they can do it this ways is that the model is ?learning?, during the training pro-cess. What we did can be treated as giving some pre-knowledge to the classifier before training.

5.2.1 ReLU

From the figure 9, it shows that using ReLU to train convolutional neural networks faster the almost 5 times than their equivalents with sigmoid units.

5.2.2 Dropout

From the figure 10, adding dropout layer is performing extremely effective in regularization. Under the same structure of CNNs, dropout maintein the ratio of train cost and validation cost

6 Compared to proposal

6.1 Those works are on the track

1. We have finished the implementation of basic neural network by Java without any open souce library.

Batch Size / NN Structure	10	20	50
64	22.32%	21.65%	24.45%
512	25.68%	26.08%	25.08%
32, 16	27.15%	28.12%	27.79%
64, 32	28.08%	29.21%	28.60%
128, 64, 32	29.29%	30.15%	29.87%

Figure 7: validation accuracy under different neural network structure. Because all the shared same number of neurons in input layer and output layer, table only give out the the structure of hidden layer, 64 stands for 1 hidden layer with 64 neurons, (32, 16) stands for two hidden layer, first hidden layer owns 32 neurons and second layer owns 16 neurons

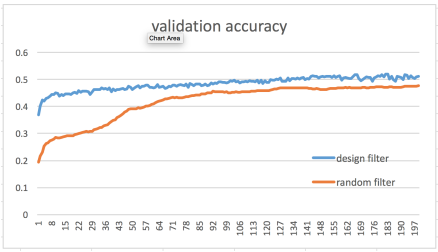


Figure 8: Accuracy comparison between our improved CNN and traditional CNN

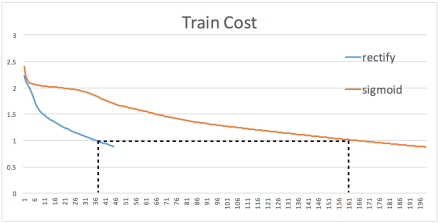


Figure 9: Accuracy comparison between our improved CNN and traditional CNN

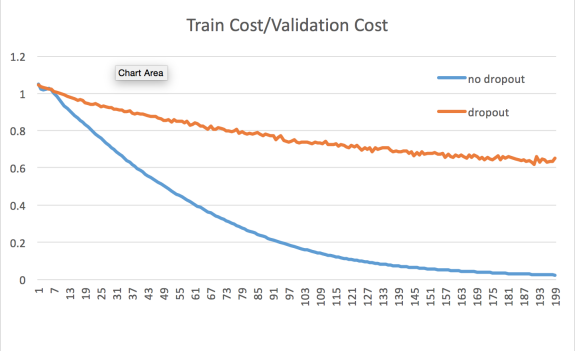


Figure 10: Accuracy comparison between our improved CNN and traditional CNN

2. We have toned the parameters under basic neural network.

6.2 Those works are on the track

We find out new some idea about the convolution network, and make some further progress on that.

7 Reference

References

Simard, P.Y., Steinkraus, D. and Platt, J.C. 2003 *Best practices for convolutional neural networks applied to visual document analysis*, volume 1. IEEE..

Rumelhart, D.E., Hinton, G.E. and Williams, R.J.,. 1988. *Learning representations by back-propagating errors.*. Cognitive modeling, 5, p.3.

Bryson, A.E.. 1975. *Applied optimal control: optimization, estimation and control. CRC Press.* , 24(11):503–512.

Kohonen.. 1988. *An introduction to neural computing. Neural networks* , 24(1), pp.3-16.

Krizhevsky, A. and Hinton, G.. 2009. *Learning multiple layers of features from tiny images.* , 24(1), pp.3-16.

Karpathy, A. . 2011. *Lessons learned from manually classifying CIFAR-10.* , 24(1), pp.3-16.

Thomas M. Mitchell. . 1997. . *Machine Learning.* , McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Paul J. Werbos. . 1974. . *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* , Harvard University Press.