Handwriting-number Recognition

Linqiao Shang, Xin Li, Hanyi Zhang

1. Introduction

The human visual system is one of the wonders of the world. Most of the people can recognize handwriting digits or words easily and effortlessly. Humans have a primary visual cortex, in each hemisphere of our brain, also known as V1, contains 140 million neurons, with tens billions of connections between them. And yet human vision involves not just V1, but an entire series of visual cortices doing progressively more complex image processing. We carry in our heads a supercomputer, tuned by evolution over hundreds of millions of years, and superbly adapted to understand the visual world. Recognizing handwritten digits isn't easy. Humans are good at making sense of what our eyes show us. There is a goof based on the idea that any 3-year-old child can recognize a photo of a bird, but figuring out how to make a computer recognize objects has puzzled the very best computer scientists for over 50 years. Since 1998, human have finally found approaches to object recognition. That sounds like a bunch of made up words from a William Gibson Sci-Fi novel, but the ideas are totally understandable if you break them down one by one.

Neural networks approach the problem in a brand new way. The idea is to take a large number of handwritten digits, known as training examples, and then develop a system which can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy. So we could build a better handwriting recognizer by using thousands or even millions or billions of training examples.

In this article we'll introducing two computer program implementing neural network and convolution neural network that learns to recognize handwritten digits. Both two are short program can recognize digits with an accuracy over 97 percent, without human intervention.

We're focusing on handwriting recognition because it's an excellent prototype problem for learning about neural networks in general. As a prototype it hits a sweet spot: it's challenging - it's no small feat to recognize handwritten digits - but it's not so difficult as to require an extremely complicated solution, or tremendous computational power. Furthermore, it's a great way to develop more advanced techniques, such as deep learning. And so throughout the book we'll return repeatedly to the problem of handwriting recognition. Later in the book, we'll discuss how these ideas may be applied to other problems in computer vision, and also in speech, natural language processing, and other domains.

We'll develop many key ideas about neural networks, including two important types of artificial neuron (the perceptron and the sigmoid neuron), and the standard learning algorithm for neural networks, known as stochastic gradient descent. Throughout, we focus on explaining why things are done the way they are, and on building your neural networks intuition. Also deep learning will be discussed in this article.

2. Related work

Some researchers have achieved "near-human performance" on the MNIST database, using a committee of neural networks; in the same paper, the authors achieve performance double that of humans on other recognition tasks. The highest error rate been recorded is the research in 1998, using Pairwise Linear Classifier, has an error rate as 7.6 percent.

In 2004, a best-case error rate of 0.42 percent was achieved on the database by researchers using a new classifier called the LIRA, which is a neural classifier with three neuron layers based on Rosenblatt's perceptron principles.

Some researchers have tested artificial intelligence systems using the database put under random distortions. The systems in these cases are usually neural networks and the distortions used tend to be either affine distortions or elastic distortions. Sometimes, these systems can be very successful; one such system achieved an error rate on the database of 0.39 percent.

In 2011, an error rate of 0.27 percent, improving on the previous best result, was reported by researchers using a similar system of neural networks. In 2013, an approach based on regularization of neural networks using DropConnect has been claimed to achieve a 0.21 percent error rate. Recently, the single convolutional neural network best performance was 0.31 percent error rate. Currently, the best performance of a single convolutional neural network trained in 74 epochs on the expanded training data is 0.27 percent error rate. Also, the Parallel Computing Center (Khmelnitskiy, Ukraine) obtained an ensemble of only 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate in 2016.

3. Implementation

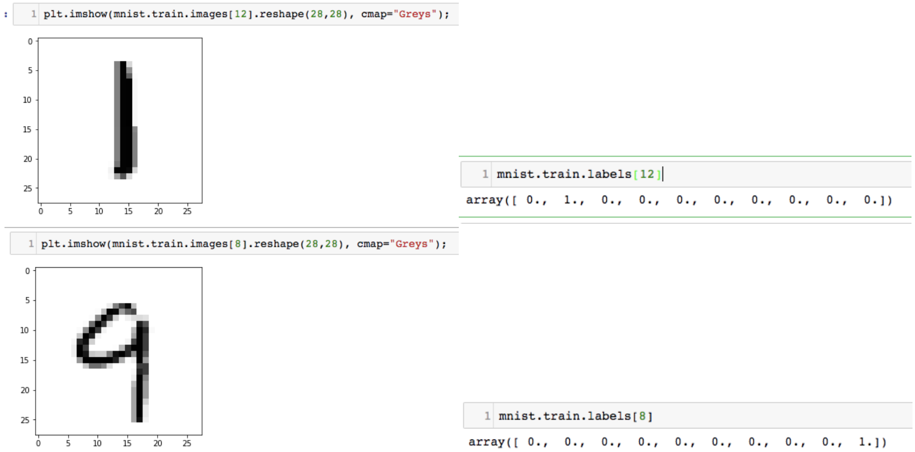
3.1. data set

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

samples from MNIST

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23 percent. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support vector machine to get an error rate of 0.8 percent.

The data format of MNIST database is handwritten digit. Each individual digit is an array with 784 (28x28) values. Each value represents the color for one pixel. If we reshape a training sample to 28x28 and plot it, we can see the original digit, and we could also see the label of this digit. The label format is the *one-hot encoding* style. This means that the label corresponds to the index of the array where the value is 1.



```
plt.imshow(mnist.train.images[12].reshape(28,28), cmap="Greys");
```

```
mnist.train.labels[12]
```

```
array([ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

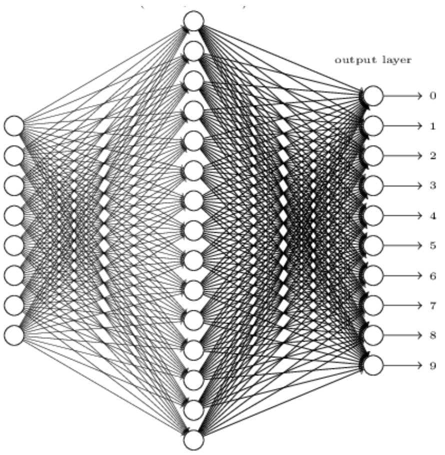
```
plt.imshow(mnist.train.images[8].reshape(28,28), cmap="Greys");
```

```
mnist.train.labels[8]
```

```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.])
```

3.2. Neural Network

The Neural Network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy.



Input layer. The input layer of the network contains neurons encoding the values of the input pixels. our hand-writing samples in the MNIST data set are 28*28 greyscale-pixel images, so we'd have 784 neurons in input layer. For every input pixel, the value of 0.0 represents white and the value of 1.0 represents black, and the between values means gradually darkening from 0.0 to 1.0.

The second layer of the network is a hidden layer. We denote the number of neurons in this hidden layer by n, which is our hyper parameter, and we'll experiment with different values for n.

Output layer. There are 10 Arabic numerals in total [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. Then we need 10 types of output for each number. So the output layer contains 10 neurons, when a sample is classified to a number, the corresponding neuron will be set to 1, the other 9 neurons remain 0, i.e. [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] represents 0.

So each training input x as a 784-dimensional vector, each entry in the vector represents the grey value for a single pixel in the image. Assuming the corresponding desired output y=f(x), where y is a 10-dimensional vector.

To quantify how well we're achieving this goal we define a cost function:

$$C(w,b) \equiv \frac{1}{2n}\sum_{x} ||y-a||^2$$

w donates the collection of weights in the network, b means the biases, n is the total number of training inputs, a is the vector of outputs from the network when x is input, and the sum is over all training inputs x.

When C (w, b) becomes small, i.e. when C (w, b) approaches to 0, y is approximately equal to the output a, for all training inputs, so our training method has been good. By contrast, it's not doing well when C (w, b) is large. The aim of our training algorithm is minimizing the cost function of weights and biases. In other words, we need to find the set of key parameters (weights and biases) which make the cost function as small as possible.

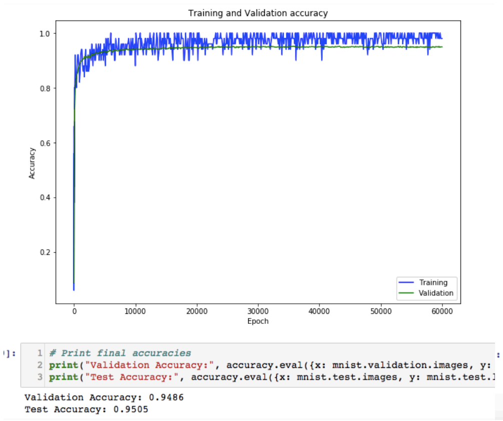
We're going to use stochastic gradient descent to find and adjust these two variables:

$$w_k' = w_k - \frac{\eta}{m}\sum_{j}\frac{\partial C_{Xj}}{\partial w_k}$$

$$b_l' = b_l - \frac{\eta}{m}\sum_{j}\frac{\partial C_{Xj}}{\partial b_l}$$

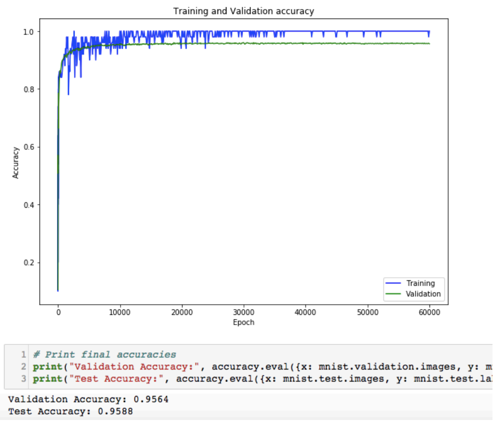
Where η is the learning rate, m is the mini batch size.

Then we're running the code and tuning the hyper parameters (hidden layer neurons n, learning rate η, mini batch size m).



```
# Print final accuracies
print("Validation Accuracy:", accuracy.eval({x: mnist.validation.images, y:
print("Test Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.l
```

```
Validation Accuracy: 0.9486
Test Accuracy: 0.9505
```

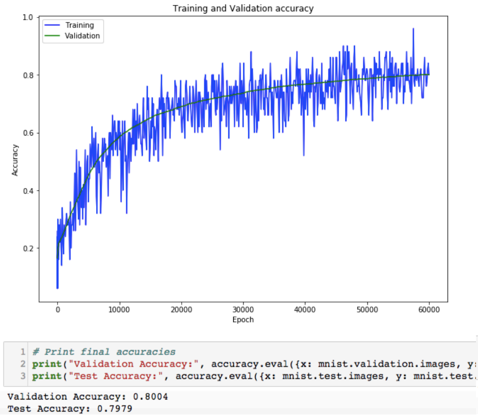


```
# Print final accuracies
print("Validation Accuracy:", accuracy.eval({x: mnist.validation.images, y: m
print("Test Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.la
```

```
Validation Accuracy: 0.9564
Test Accuracy: 0.9588
```

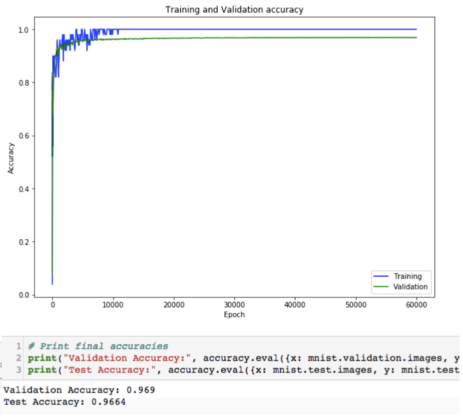
In the first running, n = 30, η = 1.0, m = 50 and we got the accuracy of this model is 94.86%, it seems good but when this model runs in a complex system where millions of digits are processed, this 5% difference would be significant. From what we have learned before, we assume that the ability of recognition of a neural network is related to the ability of its hidden layer to detect details of inputs. So, we tune the n to 100 to verify the assumption. As the result shows, the accuracy increases to 95.64%. The performance of our model does improve, but not too much. To avoid over-fitting and exceeded running time, we finally set the neurons of hidden layer to 300.

In the next step, we need to tune the learning rate. If it's too small, it will take a long time to run and potentially never reaching the optimum. But if it's too big, then the optimization may be unstable and bouncing around it.



```
# Print final accuracies
print("Validation Accuracy:", accuracy.eval({x: mnist.validation.images, y:
print("Test Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.
```

```
Validation Accuracy: 0.8004
Test Accuracy: 0.7979
```

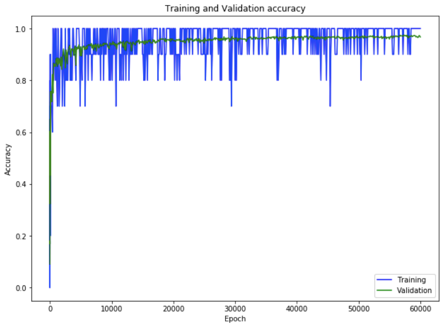


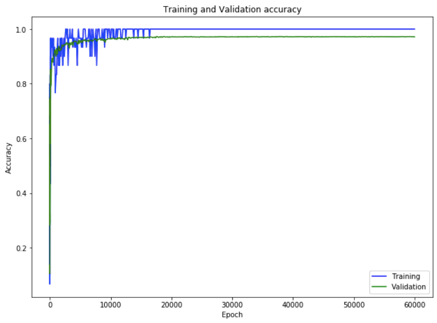
```
# Print final accuracies
print("Validation Accuracy:", accuracy.eval({x: mnist.validation.images, y
print("Test Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test
```

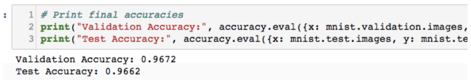
```
Validation Accuracy: 0.969
Test Accuracy: 0.9664
```

After tuning, we found that the performance fixed between a certain threshold when learning rate is [1.5, 1.8], and we finally choose to use 1.6 as learning rate.

SGD uses a subset of training data set called mini batch to evaluate the cost function. This subset changes each iteration. When the size of a mini batch is too small, it will cause a slow convergence (the parameters may jump around a lot rather than smoothly approaching the optimum outcome). And the advantage of speed may get lost as more of the training data gets used.

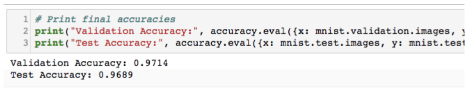






```
# Print final accuracies
print("Validation Accuracy:", accuracy.eval({x: mnist.validation.images,
print("Test Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.te
```

```
Validation Accuracy: 0.9672
Test Accuracy: 0.9662
```



```
# Print final accuracies
print("Validation Accuracy:", accuracy.eval({x: mnist.validation.images, y
print("Test Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test
```

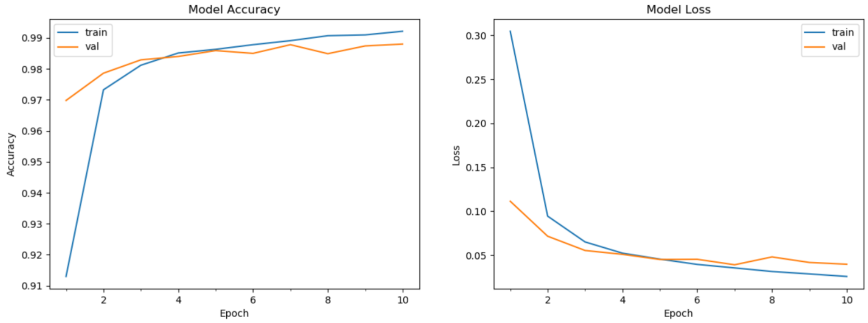
```
Validation Accuracy: 0.9714
Test Accuracy: 0.9689
```

The final accuracy of this model with parameters of m=30, m=300, η=1.6 is 97.14%. A 97.14% accuracy is not bad because it uses only one hidden layer and we could also replicate it in a proof-of-concept application. If we run the scikit-learn's SVM classifier using default setting for recognition of handwritten digits, we will get the accuracy of 94%, which means our one hidden neural network is a little better than the default SVM.

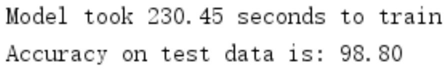
3.3. Convolution Neural Network

Convolutional neural networks (CNNs) emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s. In the last few years, thanks to the increase in computational power, CNNs have managed to achieve superhuman performance on some complex visual tasks.

This is our first model. We just add a single convolutional layer which has 32 filters and each filter's size is 5 * %. And the single pooling layer has a filter which size is 2 * 2. Due to there are 10 categories of numbers, so the activation function is softmax rather than relu. Next, the number of filters should be 2^n, because computer can calculate it quickly. So we choose 32 (n = 5).

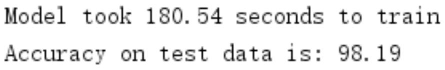


5 * 5 filter:



```
Model took 230.45 seconds to train
Accuracy on test data is: 98.80
```

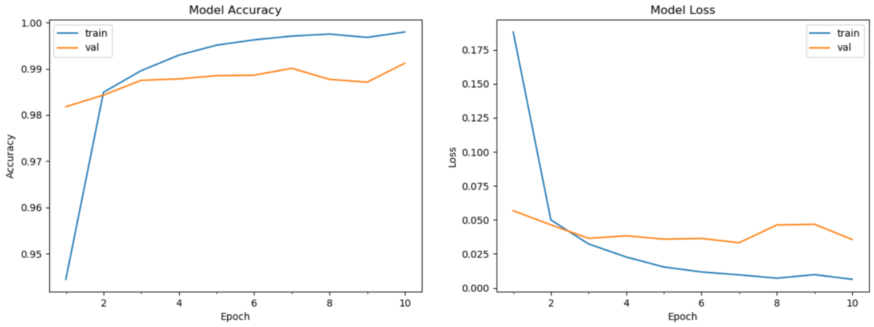
3 * 3 filter:

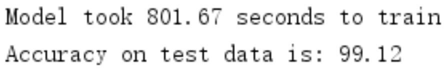


```
Model took 180.54 seconds to train
Accuracy on test data is: 98.19
```

The accuracy of our first model is 98.8%, it is not a bad result. And these two images show that the curve of model accuracy tends to be gentle when epoch is bigger than 6. Also, this model doesn't take much time to calculate, about 230 seconds to train. We don't choose 7 * 7 because the principle of CNN is small and deep, if filter's size is large, it need to calculate more and more parameters.

Then, we tried to add more convolutional layer in the model. As a result, we implement another layer which is same as first layer and a fully connected layer which has 128 neurons. The result is the following picture.

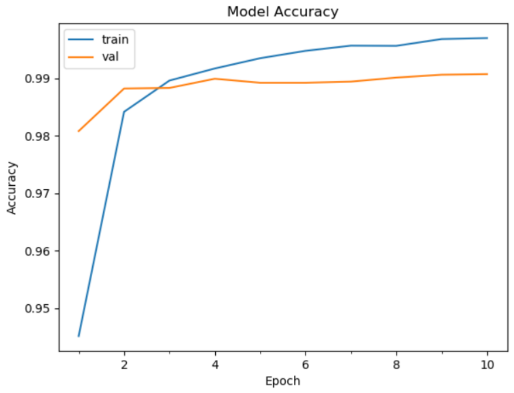


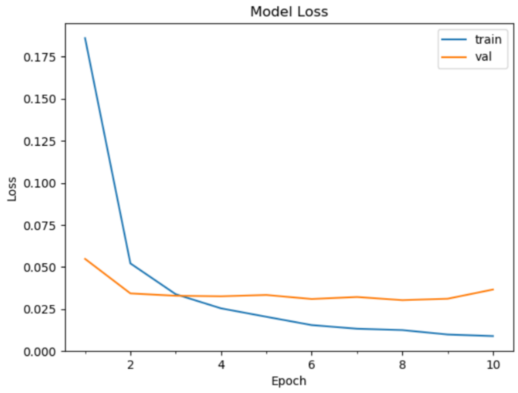


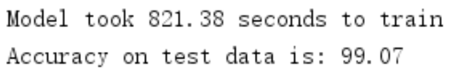
```
Model took 801.67 seconds to train
Accuracy on test data is: 99.12
```

99.12% is the second model's accuracy, so the deeper network can not only increase the test accuracy but also increase the training time.

Next, we decide to add the dropout layer which parameter is 0.1.

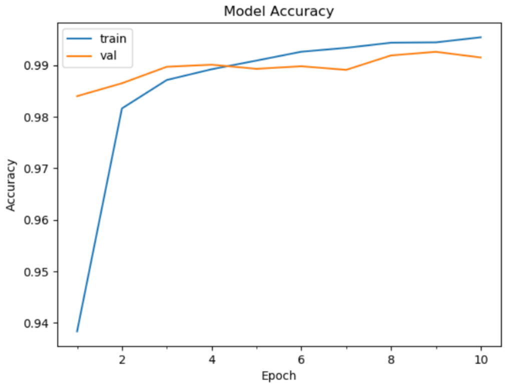


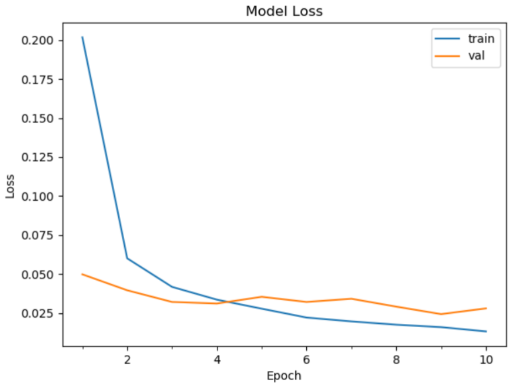


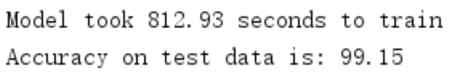


```
Model took 821.38 seconds to train
Accuracy on test data is: 99.07
```

The accuracy is 99.07%, so adding the dropout layer increases the test accuracy while increasing the training time. Dropout layer adds regularization to the network by preventing weights to converge at the same position. During forward propagation, nodes are turned off randomly while all nodes are turned on during forward propagation.







```
Model took 812.93 seconds to train
Accuracy on test data is: 99.15
```

Finally, we change this parameter from 0.1 to 0.2. And we find the accuracy changes from 99.07% to 99.15%. It is a small increase, so this model can resolve this problem perfectly.

4. Comparison

So clearly as the result, the CNN has reached the accuracy of 0.9915 and the Neural Network reached 0.9714. Although we can always adjust some of the parameters to optimize the result. But the result clearly shows that the CNN is better than Neural Network in handwriting digit recognition problem.

From the training time point of view, the Neural Network takes about 150s to train, but the CNN takes 230 seconds if using 5*5 filter. It may seem like neural network method trains faster than CNN. But theoretically the CNN should be faster than neural network. We consider this as a lack of input dimensions. Here we only take 28x28 as the input diameters. If we add one more dimension of the picture---color. The result should be CNN faster than neural network.

Compare to the parameters, they both use quite different parameters, due to their characteristics. But it clearly shows that the CNN has less parameters than Neural Network. And quite easy to understand what the parameter is and easy to adjust.

So according to those parameters. We can see that, CNN slide a kernel across dimensions and can have an input of any size, this kernel's parameters are shared in every slide for that layer. By contrast, neural networks tend to have far more parameters because they have associated weights for every single input, in addition, they require a fixed input size. CNN require less input layers than Neural Network. Which makes it easy to implement.

5. Future Research Directions

From the approaches we discussed above. We can see that neural break down all the parts into pieces and determine whether this piece satisfied this situation. If yes, then out put the answer. And of-cause, in these two method we only use 2 hidden layers. We can always improve further and further through multiple layers. Ultimately, we'll be working with sub-networks that answer questions so simple they can easily be answered at the level of single pixels. Those questions might, for example, be about the presence or absence of very simple shapes at particular points in the image. Such questions can be answered by single neurons connected to the raw pixels in the image. The end result is a network which breaks down a very complicated question into very simple questions answerable at the level of single pixels. It does this through a series of many layers, with early layers answering very simple and specific questions about the input image, and later layers building up a hierarchy of ever more complex and abstract concepts. To train those neural, we could use learning algorithms so that the network can automatically learn the weights and biases - and thus, the hierarchy of concepts - from training data.

Researchers in the 1980s and 1990s tried using stochastic gradient descent and back propagation to train deep networks. Unfortunately, except for a few special architectures, they didn't have much luck. The networks would learn, but very slowly, and in practice often too slowly to be useful. Since 2006, a set of techniques has been developed that enable learning in deep neural nets. These deep learning techniques are based on stochastic gradient descent and back propagation, but also introduce new ideas. These techniques have enabled much deeper (and larger) networks to be trained - people now routinely train networks with 5 to 10 hidden layers. And, it turns out that these perform far better on many problems than shallow neural networks, i.e., networks with just a single hidden layer. The reason, of course, is the ability of deep nets to build up a complex hierarchy of concepts. It's a bit like the way conventional programming languages use modular design and ideas about abstraction to enable the creation of complex computer programs. Comparing a deep network to a shallow network is a bit like comparing a programming language with the ability to make function calls to a stripped down language with no ability to make such calls.

And furthermore, due to the simplicity of the number output, from 0-9, we can also expand the recognition problem to handwriting words. Like recognizing a-z, A-Z, so that we can recognize the written word. Which means the output will increase.

Besides the increment of output size, we can also improve the input size or minimize the input size to generate more accuracy or reduce the training time. Due to the data characteristic in MNIST database. The input has to be 28*28=784. But if the size increased. The number of input will be dramatically increased. And we can use some methods like dichotomy to minimize the input data to a considerable size.

6. Conclusion

We have studied both Convolution Neural Network and Neural Network. And has reached the highest accuracy of 0.9915 (CNN) with a single convolutional layer which has 32 filters and each filter's size is 5 *5. And for the neural network, we reached accuracy of 0.9714 with parameters of mini batch size equals to 30, hidden layer equals to 300 and learning rate equals to 1.6. Also, through the comparison, the CNN is a better algorithm to perform a picture recognition problem. Further improvement can also be made to increase the result of the accuracy.

Convolutional Neural Net is a popular deep learning technique for current visual recognition tasks. Like all deep learning techniques, CNN is very dependent on the size and quality of the training data. Given a well prepared dataset, CNNs are capable of surpassing humans at visual recognition tasks.

7. Individual Contribution

Xin Li: read reference about Neural Network, learn about MNIST database, construct NN model and tune the parameters to optimize, write datasets and Neural Networks parts of report and slides.

Hanyi Zhang: read reference about CNN, construct model and optimize parameters, write CNN part of report and slides.

Linqiao Shang: make comparison of two algorithms, write other parts of report and slides.

8. Reference

LeCun, Yann; Corinna Cortes; Christopher J.C. Burges. "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges". Retrieved 17 August 2013.

Ciresan, Dan; Ueli Meier; Jürgen Schmidhuber (2012). "Multi-column deep neural networks for image classification" (PDF). 2012 IEEE Conference on Computer Vision and Pattern Recognition: 3642–3649. arXiv:1202.2745. doi:10.1109/CVPR.2012.6248110. ISBN 978-1-4673-1228-8.

LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-Based Learning Applied to Document Recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324. doi:10.1109/5.726791. Retrieved 18 August 2013.

Kussul, Ernst; Tatiana Baidyk (2004). "Improved method of handwritten digit recognition tested on MNIST database" (PDF). Image and Vision Computing. 22 (12): 971–981. doi:10.1016/j.imavis.2004.03.008. Retrieved 20 September 2013.

Ranzato, Marc'Aurelio; Christopher Poultney; Sumit Chopra; Yann LeCun (2006). "Efficient Learning of Sparse Representations with an Energy-Based Model" (PDF). Advances in Neural Information Processing Systems. 19: 1137–1144. Retrieved 20 September 2013.

Ciresan, Dan Claudiu; Ueli Meier; Luca Maria Gambardella; Jürgen Schmidhuber (2011). "Convolutional neural network committees for handwritten character classification" (PDF). 2011 International Conference on Document Analysis and Recognition (ICDAR): 1135–1139. doi:10.1109/ICDAR.2011.229. ISBN 978-1-4577-1350-7. Retrieved 20 September 2013.

Wan, Li; Matthew Zeiler; Sixin Zhang; Yann LeCun; Rob Fergus (2013). Regularization of Neural Network using DropConnect. International Conference on Machine Learning (ICML).

Romanuke, Vadim. "The single convolutional neural network best performance in 18 epochs on the expanded training data at Parallel Computing Center, Khmelnitskiy, Ukraine". Retrieved 16 November 2016.

Romanuke, Vadim. "Parallel Computing Center (Khmelnitskiy, Ukraine) gives a single convolutional neural network performing on MNIST at 0.27 percent error rate". Retrieved 24 November 2016.

Romanuke, Vadim. "Parallel Computing Center (Khmelnitskiy, Ukraine) represents an ensemble of 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate". Retrieved 24 November 2016.