Neural Network and Deep Learning based Handwriting recognition

Furong Bai, Danlu Pan,
Xiangxi Kong, Shuangqiu Liu

Project Introduction

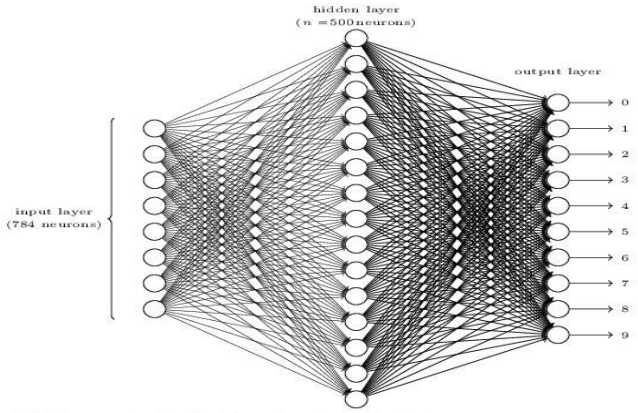
The Handwriting digit recognition is a field of research in artificial intelligence and computer vision. In recent years, there is a trend to apply machine learning, in specific, deep learning to the field. A computer performing handwriting recognition is said to be able to acquire and detect digits in paper documents, images and other sources and convert them into machine-encoded form. The project aims to build a 10-class classifier that can recognize the handwritten digits using the data from the MNIST dataset. In MNIST dataset, each image is a 28 by 28 pixel square and a standard spit of the dataset is used to evaluate and compare models, where 60,000 images are available to train a model and a separate set of 10,000 images are used to test it. The models were built using multiple machine learning techniques in neural network, including DNN and CNN. The goal is to use the least amount of training data to achieve the highest possible testing accuracy at the meantime.

Data Preparation and Cleaning

In the data preparation part, Brendan O'Connor's code with the specific functions was used to upload the dataset into R. Images of digits in this dataset were taken from a variety of scanned documents and other sources. We use MNIST data set as our data resource. It is an excellent largely formerly-used dataset for evaluating and building models, allowing us to focus on the machine learning with little data cleaning.

Neural Networks/ Project Overview

The structure of basic neural network we developed in this project is like following. Each layer is fully connected. There are 10 neurons in input layer, multiple hidden layers, and 10 neurons in output layer. The model parameters were trained through general optimization algorithm, in order to minimize the cost function. The method evolved from shallow to deep neural network learning. We also added convolutional layers to the structure in addition. However, the best result achieved was the model of basic deep neural network structure with fully connected layers. Detailed model design and final results will be discussed in following paragraphs.



* Original image was retrieved from https://timgsa.baidu.com/timg?image&quality=80&-size=b9999_10000&sec=1493590442375&di=60eb1619bb064d3d50d5c58ea2d88278&imgtype=0&src=http%3A%2F%2Fs6.si

In most models, linear activation function had been used. We also tested logistic activation function in replacement of linear. In such cases, the objective was changed to minimize the Cross-entropy function due to the sigmoid output neurons. Final model applied Rectifier Linear function as the activation function. Rectifier has been demonstrated high performance in image recognition tasks (Candel, 2015). Compared to other activation functions, it attained higher accuracy with same training set in this project.
To fulfill the NNs in R language, we use tools in R's community. There are several popular and mature packages, which are nnet, mxnet, nerualnet, H2O, and deepnet, etc. In this project, mxnet and H20 packages were applied to explore the handwriting recognition, specific process and advantages were explained in the following paper.

Hyperparameter initialization

To initialize the network, how to best define initialization strategy is an open problem. For this paper, we experimented three larged used strategies: normalize(straightforward strategy), uniform and uniform adaptive distribution strategies. Since uniform adaptive distribution strategy yielded the best result, it was chosen in initializing weights in this project.

Shallow neural network (By Mxnet Package)

The first attempt in this project was building a basic neural network with 1 hidden layer and all neurons are fully connected. "mxnet" package had been utilized in this step. It is one of the available architectures of applications generating feed-forward neural network and convolutional neural network (CNN). It allows the users to integrate the detailed requirements in the neural network. The predominant function employed by this project is called, mx.mlp, which is a convenience interface for multiple layer perceptron (Blogisr part 2).

```
trainIndex = createDataPartition(inTrain$y, p = 0.15,list=FALSE)
```

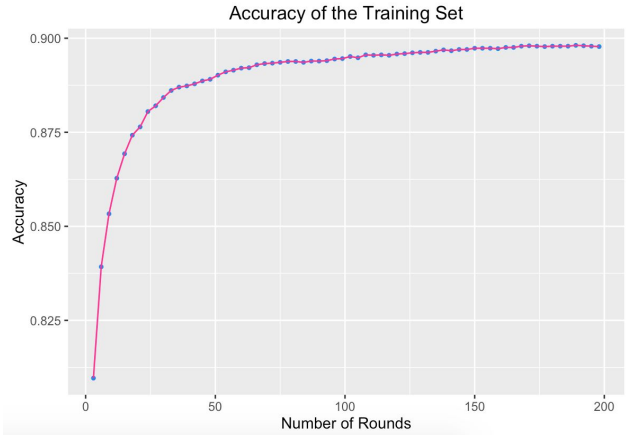
As shown in the above code, the size of training set was 15% of the 60000 observations, i.e. 9005 data. Apparently, there were many options of the size of training set that this project could used to activate the model. However, if a larger size of training set was chosen, it will be time-costly to run the whole application. On the contrary, if a smaller size of training set was selected, the running time would be saved substantially. But, the accuracy of handwriting recognition would go downstairs.

The following code shows the optimal arguments and hyperparameters of this classifier. To attain higher accuracy, we examined on dozens of different values The initial values were dramatically different from the current ones. For instance, the learning rate was set to be 0.01 at the very beginning with the number of hidden nodes being 10 and the number of rounds being 10. Its test accuracy was only 0.10. In this regard, we tuned the model parameters to achieve a relatively high accuracy.

This model constructed with 500 hidden neurons within one hidden layer. Softmax layer was used to transfer outputs into a probability distribution. In this 10-class neural network, the output class is determined by the maximum value for all output neurons. (Xiong, 2017)

```
model = mx.mlp(train.x, train.y, hidden_node = 500, out_node = 10,
               out_activation="softmax", num.round=200,
               learning.rate=0.0008, momentum=0.7,
               eval.metric=mx.metric.accuracy)
```

Nevertheless, the final result was not very satisfying. The highest training accuracy that this model can get was approximately 0.8978 using 9005 observations as training set. A slightly larger size of training set was also bee tested. But, the result did not improved substantially. It was very hard to reach a 90% or higher accuracy using this model. Thus we evaluated model structure by adding more hidden layers in later work.




```
[195] Train-accuracy=0.897667253521127
[196] Train-accuracy=0.897667253521127
[197] Train-accuracy=0.897777288732394
[198] Train-accuracy=0.897777288732394
[199] Train-accuracy=0.897777288732394
[200] Train-accuracy=0.897777288732394
```

```
> sum(diag(table(testing[, 1], pred.label)))/10000
[1] 0.8471
```

Another problem caused by raising the number of rounds was the processing time. For this case, the running time for 200 rounds was 226.826 seconds, or roughly 4 minutes. But, if the number of rounds was lifted to 300, the processing time would be doubled. This is a significant trade-off that people should deliberate when doing academic research.

```
> proc.time()
   user  system elapsed
226.826  81.903 270.641
```

In next step, we tested the model built on testing set containing a fixed number of observations (10000). The test accuracy was 0.8471 and the FOM was 0.2279. In later paper, we described the progress in achieving higher test accuracy with lower FOM through developing a DNN model based on the insights from this simple neural network classifier.

Deep neural network (DNN)

In second part of the analysis, this project focused on deep neural network with multiple hidden layers. In tuning the hyper parameters, we used different number of hidden layers and number of hidden neurons in each layer. The following table summarized part of the topologies we examined, and the corresponding results. It helped to explain how we get to the final model.

Package used in building and testing DNN models was H2O, which is an open-source platform that people can employ to generate feed-forward neural network, deep autoencoders as well as other machine learning subjects. However, H2O's core was embedded in Java, which means that users must download the package ahead of time from the following website https://www.java.com/en/download/ (Blogisr part 6).

First model in the table was the shallow neural network that discussed in previous section. The initial DNN model was designed based on output from the shallow network. The table showed the growing in test accuracy as we continued adjusting model topology. Larger training set was also used (not included in the table) before reaching a minimum size of training data. The minimum was 4025 training data while maintaining an acceptable test accuracy.

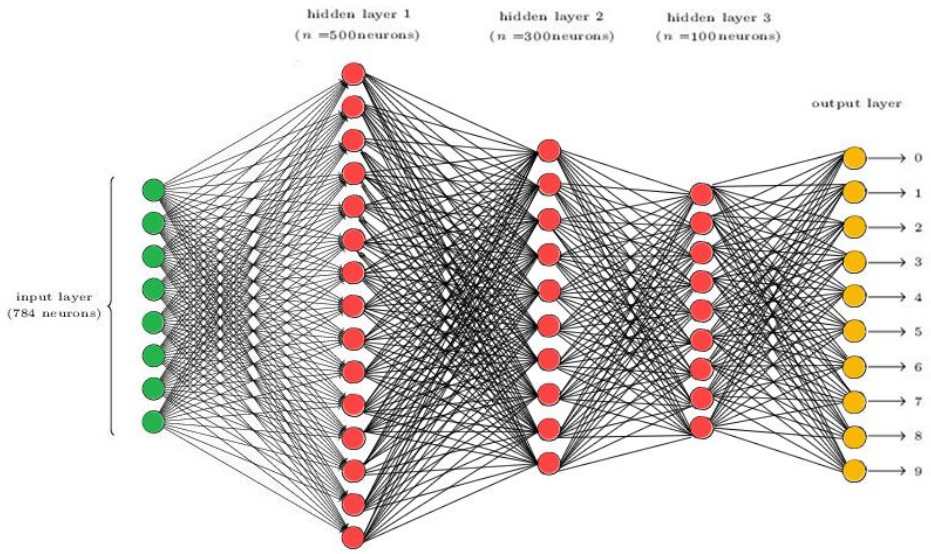
Two different activation functions were been tested, one was the Rectifier linear function and the other was the Tank logistic function. Also different cost functions were used corresponding to two activation functions. Rectifier performed better than Tanh because it produced higher accuracy and lower FOM if any other parameters were same. In general, Rectifier has been exhibited high performance in image recognition tasks. Followings are formulas of these two activation functions. (Candel, 2015).

Function	Formula
Tanh	$f(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}}$
Rectified Linear	$f(\alpha) = \max(0, \alpha)$

where $\alpha = \sum_i w_i x_i + b.$

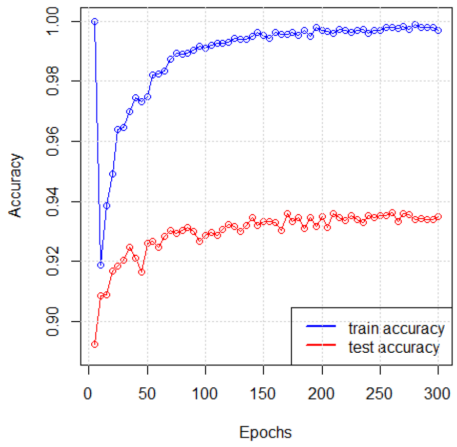
Topology					Results		
No. hidden layers	No. hidden neurons	Activation Func.	Learning rate	epoch	Training Size /P1 P1= train size/60k	Test Accuracy (P2)	FOM
1	500	Linear	0.0008	200	9005 /0.1501	0. 8471	0.2279
2	100,100	Rectifier	0.00005	200	4025/0.0671	0.9017	0.1319
2	100,100	Rectifier	0.00001	300	4025/0.0671	0.9350	0.0985
2	100,100	Rectifier	0.00001	100	4025/0.0671	0.9286	0.1049
2	100,100	Tanh	0.00001	100	4025/0.0671	0.9026	0.1309
2	500,100	Rectifier	0.00001	100	4025/0.0671	0.9453	0.0882
2	500,100	Tanh	0.00001	100	4025/0.0671	0.9065	0.1606
3	100,100,100	Rectifier	0.00001	100	4025/0.0671	0.9296	0.1039
3	500,100,100	Rectifier	0.00001	100	4025/0.0671	0.9424	0.0911
3	**500,300,100**	**Rectifier**	**0.00001**	**100**	**4025/0.0671**	**0.9463**	**0.0872**
3	500,500,300	Rectifier	0.00001	100	4025/0.0671	0.9456	0.0880
3	800,500,100	Rectifier	0.00001	100	4025/0.0671	0.9470	0.0866
3	800,800,800	Rectifier	0.00001	100	4025/0.0671	0.9477	0.0859
4	500,300,100, 100	Rectifier	0.00001	100	4025/0.0671	0.9455	0.0881

C-E: Cross-Entropy Function
Rectifier: Rectifier Linear Function
Tanh: Tank Logistic Function



Structure of final DNN model

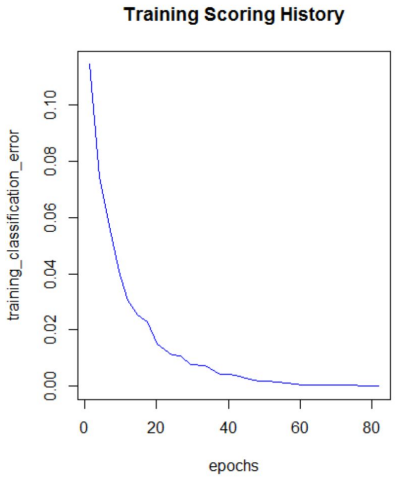
Following graph showed the learning curve of a DNN model. (shown in 2nd row of the table above) With 2 hidden layers and 100 neurons in each hidden layer, it converged after 100 iterations. It showed the growth of accuracies corresponding to increasing number of iterations. However, there is a little concern about overfitting. Regulations were applied in later models. Detailed techniques will be explained in final result.



Final result

Final results were marked red in the table above. The best FOM value we obtained in experiment was 0.0872, with a test accuracy of 0.9463 (P2) and 4025 (P1=0.0671) data used in training. This final model contained 3 hidden layers with 500,300,100 neurons in each. Learning rate was 0.00001. The activation function used was the Rectifier Linear function. Considering the issue of overfitting, L1 & L2 regulations were applied to help. In addition, we also used dropout as another regulation technique which was commonly applied neural network. Input dropout ratio is 0.2 and hidden layer dropout ratio was 0.5 which implied temporarily randomly ignored half of the hidden neurons in the network. The model gradually converged as epoch increased towards to 100. Followings are the code when used in training the final DNN model and test accuracy obtained. The graph showed the training scoring history as number of iterations increased.

```
######## dnn.m2 ################
dnn.m1 = h2o.deeplearning(
  x = 2:785,
  y = 1,   # column number for label
  training_frame = train_h2o, # data in H2O format
  hidden = c(500,300,100),
  activation = "RectifierWithDropout", # activation function
  input_dropout_ratio = 0.2, # % of inputs dropout
  hidden_dropout_ratios = c(0.5,0.5,0.5), # % for nodes dr
  nesterov_accelerated_gradient = T, # use it for speedn
  #loss =c("CrossEntropy"),
  l1=1e-5,  ## add some L1/L2 regularization
  l2=1e-5,
  epochs = 100)
'
> # Accuracy
> sum(diag(table(as.vector(test_h2o[, 1]), as.vector(pred[,1]) )))/10000
[1] 0.9463
```



One interesting finding during experiment was that the test accuracy wouldn't improved apparently even using more hidden neurons than 500 in each hidden layer. However, using different methods in initialization would lead to very different result. In the final model, the initial weights were uniform adaptive distributed.

Convolutional neural network (CNN)

A CNN is a special case of the neural network. This kind of network consists of one or more convolutional layers, often with a subsampling layer, which are followed by one or more fully connected layers as in a standard neural network.

Convolutional neural networks allow networks to have fewer model parameter (the weights and bias are shared between neurons). They are given a very effective tool - convolutions - for digit recognition. Besides, in the convolutional layer, the same coefficients are used across different locations in the space, so the memory requirement is reduced. In a CNN, since the number of hyperparameters is significantly reduced, training time will proportionately cut down.

In this section, The network configuration of CNN was built with two convolutions, three activation layers, two pooling layer, one flatten later and two fully connected layers. The size of training set was 12% of the 60,000 observations, i.e. 7,200. And at the end of the process, the output was transformed by using the Softmax Output symbol function in the mxnet package. The details of this CNN structure shows as below:

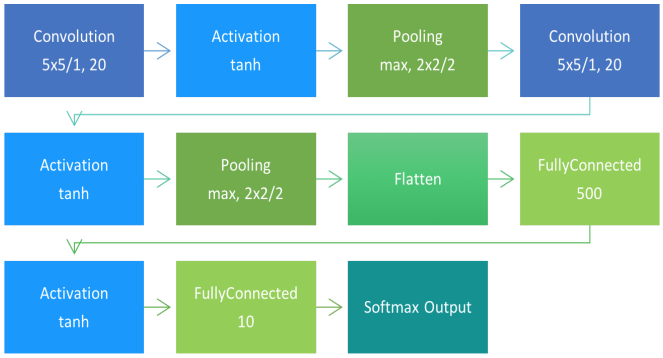


Figure.CNN configuration

This project employed the built-in 'mx.model.FeedForward.create' function from 'mxnet' package to train the CNN model.

```
cnn.model <- mx.model.FeedForward.create(cnn, X=train.array,
  y=training.y, ctx=devices, num.round=30,
  array.batch.size=100, learning.rate=0.06,
  momentum=0.7, eval.metric=mx.metric.accuracy,
  initializer=mx.init.uniform(0.05),
  epoch.end.callback=mx.callback.log.train.metric(100)
  )
```

```
[22] Train-accuracy=0.89972602739726
[23] Train-accuracy=0.90013698630137
[24] Train-accuracy=0.89986301369863
[25] Train-accuracy=0.9
[26] Train-accuracy=0.9
[27] Train-accuracy=0.90027397260274
[28] Train-accuracy=0.90013698630137
[29] Train-accuracy=0.90027397260274
[30] Train-accuracy=0.90027397260274
```

The model was trained as above, hyperparameters were adjusted by many times to attain relatively high testing accuracy. The learning rate was set to be 0.06, the number of the iterations was 30, and momentum was 0.7. In testing the CNN model on testing set, the test accuracy was 0.6025.

```
> sum(diag(table(testing[,1],pred.label)))/10000
[1] 0.6025
```

While the testing accuracy was 60.25% which far smaller compared to the training accuracy showed above, it meant that the model was overfitting. Final result was obtained through build deep neural network as described above.

Group Contribution

Our group created dynamic team work in this project. We determined our objective and work flow together and then divided the entire project to various distributions to each member. Each member showed their distributions in this work. Shuangqiu used the Mxnet in building a basic neural network. Furong utilized H2O package to develop DNN model. Danlu and Xiangxi used Mxnet package to build the CNN model. Each one reported their work in this paper.

Conclusion

Compared to CNN, our DNN that fulfilled with H2O package obtained the best result: Test accuracy: 0.9463 and FOM: 0.08723 ,with three hidden layers of 500,300,100 neurons in each, using 4025 training observations. The biggest advantage of DNN is to extract and learn features automatically by deep layers architecture, especially for these complex and high-dimensional data which are hardly to be captured (Zhao, 2016, February 13).

This project was significantly crucial to go over the essential theories and applications of neural networks as well as deep learning, especially provided practice in all aspects of AI. We processed rare dataset, trained different NN models in R language and obtained great accuracy of deep learning.
The most challenging section in this project was to reach the best accuracy using the smallest training data set. Also, turning model parameters and dealing with the overfitting issues were challenging in this project.

For future improvement, we can explore more effective methods to improve accuracy. For instance, we can artificially generate more training data by slightly converting some numbers to provide more reliable training data.

Reference

Blogisr. "Deep Learning in R." R-bloggers. WordPress, 06 Feb. 2017. Web. 29 Apr. 2017. <https://www.r-bloggers.com/deep-learning-in-r-2/>.

OZAKI, T. J. (2016, March 30). {mxnet} R package from MXnet, an intuitive Deep Learning framework including CNN & RNN. Retrieved April 29, 2017, from http://tjo-en.hatenablog.com/entry/2016/03/30/233848

O'Connor, Brendan. (2008) - gist.github.com/39760 - anyall.org

Xiong, Jinjun. (2017) Machine Learning: Concepts & Application: Lecture 10 - Neural Network & Deep Learning Part I, Columbia University Course, NYC NY.

Xiong, Jinjun. (2017) Machine Learning: Concepts & Application: Lecture1 - Neural Network & Deep Learning Part II, Columbia University Course, NYC NY.

Zhao, P. (2016, February 13). R for Deep Learning (I): Build Fully Connected Neural Network from Scratch – ParallelR. Retrieved April 29, 2017, from http://www.parallelr.com/r-deep-neural-network-from-scratch/