Machine Learning: An Introduction to Neural Network using MXNet

Stat133 Post 2: Data Technologies

Junfang Jiang

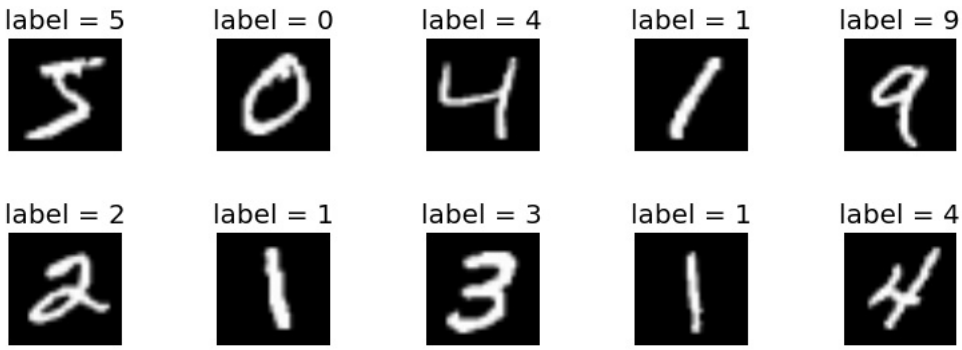
12/2/2017

Machine learning is a method of data analysis that automates the process of analytical model building. Instead of manually finding the underlying pattern of the data and feed it to the computer, the key concept of machine learning is based on the idea that machines should be able to learn and adapt through experience by itself.

In the previous post, (read here if interested), an extremely simplified version of machine learning is introduced with the example of iris species classification problem and KNN (k-Nearest Neighbors) algorithm, which computes relative "distances" (similarity) between samples and makes predictions by finding the closest sample in the training data based on the fact that iris of the same kind should look similar to each other. However, in most of real life problems, the pattern will be really hard to find and sometimes it is even impossible for a normal person to find or describe the exact rules to solve the problem. In that case, a more advanced machine learning algorithm (e.g. neural network) is required for the machine to learn the model.

Handwritten Digits Classification Problem

MNIST is a widely known handwritten digits image data set created by Yann LeCun. It contains a training set of 60,000 samples, and a test set of 10,000 samples, where each sample is a 28*28 image and its corresponding label (number of 0-9). Training & testing images and labels can be downloaded from its main page in ubyte format and transform into csv format by using the python script attached below.



(Note: color is inversed in the diagram above)

Source of data

Here is the training and testing set:

- CSV: mnist_train.csv, mnist_test.csv

If you want to generate the data by yourself, use the source files below:

- Original: train-images-idx3-ubyte, train-labels-idx1-ubyte, t10k-images-idx3-ubyte, t10k-labels-idx1-ubyte
- Python Script: convert.py

All samples in both training and testing data sets are in the following format, where pixXX is the RGB grey scale color for the XX-th pixel.

```
label, pix0, pix1, pix2, ... , pix783
```

Data preparation

Load the data from csv files.

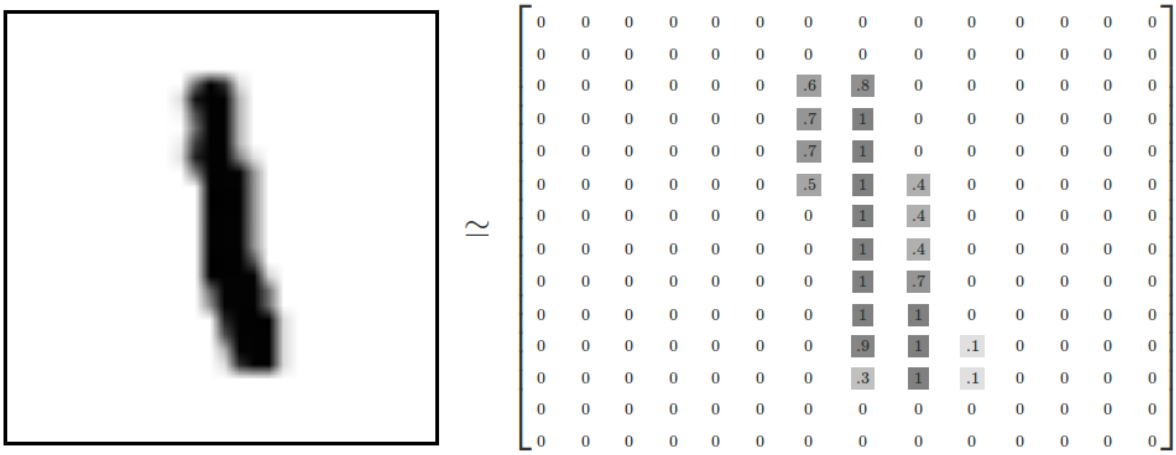
```
train <- read.csv('data/mnist_train.csv', header=TRUE)
test <- read.csv('data/mnist_test.csv', header=TRUE)


# Summary of Columns in Training & Test Data
head(train[,1:10])
```

```
##   label pix0 pix1 pix2 pix3 pix4 pix5 pix6 pix7 pix8
## 1     5    0    0    0    0    0    0    0    0    0
## 2     0    0    0    0    0    0    0    0    0    0
## 3     4    0    0    0    0    0    0    0    0    0
## 4     1    0    0    0    0    0    0    0    0    0
## 5     9    0    0    0    0    0    0    0    0    0
## 6     2    0    0    0    0    0    0    0    0    0
```

Store image pixels as inputs into `train_x` & `test_x` and corresponding labels as outputs into `train_y` & `test_y`. In addition, image pixels are stored as grey scale RGB color (255 as white, 0 as black). Divide it by 255 so that it now is under a range of 0 to 1.

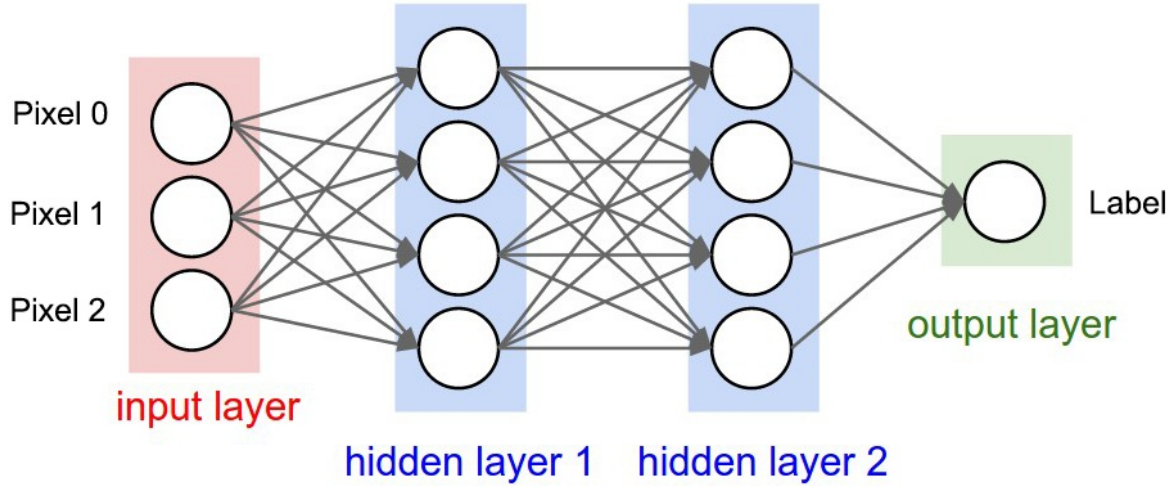
```
train_data = data.matrix(train)
train_x <- train_data[,-1]
train_y <- train_data[,1]


test_data = data.matrix(test)
test_x <- test_data[,-1]
test_y <- test_data[,1]


train_x <- train_x / 255
test_x  <- test_x / 255


table(train_y)
```

```
## train_y
##    0    1    2    3    4    5    6    7    8    9
## 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```



What is Neural Network

Neural network is a computing system inspired by biological neural networks that constitute animal brains (Wikipedia). It consists of a collection of neurons, each receiving signals from the neurons in the previous layer, sum them up by using a set of weights, and passing the result to the next layer. The weight between each pair of neurons is the variable that machine trying to learn from training data. The weights are randomly initialized at the beginning and updated based on the degree of error of the current predictions. In addition, a Relu Activation function [f(x) = max(0, x)] is applied after each layer to introduce non-linearity as the model is not necessarily linear (data points for handwritten digits are not linearly seperatable in high-dimentional space). The details of neural network are not going to be covered here, but instead, this post will show how to use MXNet library to complete the challenge.



The diagram above shows a simplified illustration of a neural network with three pixel inputs and two hidden layers of four neurons each. In the code below, a full-sized neural network is going to be constructed with three hidden layers, with 128, 64, 10 neurons respectively.

Build the network using MXNet

Install MXNet library using the following script.

```
# Install mxnet package
cran <- getOption("repos")
cran["dmlc"] <- "https://s3-us-west-2.amazonaws.com/apache-mxnet/R/CRAN/"
options(repos = cran)
install.packages("mxnet", repos = "http://cran.us.r-project.org", dependencies = T)
```

Now construct the structure of the neural network.

```
# Import library
library(mxnet)


# Create placeholder for input data
data <- mx.symbol.Variable("data")


# Construct three hidden layers with 128, 64, 10 neurons
layer1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
relu1  <- mx.symbol.Activation(layer1, name="relu1", act_type="relu")


layer2 <- mx.symbol.FullyConnected(relu1, name="fc2", num_hidden=64)
relu2  <- mx.symbol.Activation(layer2, name="relu2", act_type="relu")


layer3 <- mx.symbol.FullyConnected(relu2, name="fc3", num_hidden=10)


# Output layer
output <- mx.symbol.SoftmaxOutput(layer3, name="sm")
```

Train the model

The following code will train the model using the CPU only for adaptability (all computers must have a CPU). The training will take a while, but however, if your computer has a GPU (Graphics processing unit) installed, you can speed up the process of training by replacing the first line to `devices <- mx.gpu(i)` where `i` is the number of GPU you want to use. The training is based on `train_X`, `train_Y`, using `rowmajor` layout since our data samples are stored per row. 15 rounds of training are taken and the improvements in accuracy are displayed below.

```
# Train the model by using CPU
devices <- mx.cpu()


# Use random seed to ensure reproducibility
mx.set.seed(0)


# Train the model on training data
model <- mx.model.FeedForward.create(output, X=train_x, y=train_y, ctx=devices,
                                     array.layout="rowmajor",
                                     num.round=15,
                                     learning.rate=0.05,
                                     momentum=0.9,
                                     eval.metric=mx.metric.accuracy,
                                     initializer=mx.init.uniform(0.07),
                                     epoch.end.callback=mx.callback.log.train.metric(100))
```

```
## Start training with 1 devices
```

```
## [1] Train-accuracy=0.861695379273504
```

```
## [2] Train-accuracy=0.961204024520256
```

```
## [3] Train-accuracy=0.973780650319829
```

```
## [4] Train-accuracy=0.979977345415778
```

```
## [5] Train-accuracy=0.983825293176972
```

```
## [6] Train-accuracy=0.986290644989339
```

```
## [7] Train-accuracy=0.98890591684435
```

```
## [8] Train-accuracy=0.991321295309168
```

```
## [9] Train-accuracy=0.992004264392324
```

```
## [10] Train-accuracy=0.993453491471215
```

```
## [11] Train-accuracy=0.994552905117271
```

```
## [12] Train-accuracy=0.995452425373134
```

```
## [13] Train-accuracy=0.995818896588486
```

```
## [14] Train-accuracy=0.996685101279318
```

```
## [15] Train-accuracy=0.996668443496802
```

As we can see, the model is doing quite well as it achieves a relatively high accuracy (>99%) on the training set, which provides the expected output together with the input. Now, let's examine the performance of the model by testing it on the testing data set.

Evaluate the Model

Using the same model, make predictions about the testing images `test_x` and compare it with the correct solution `test_y`.

```
# Make predictions
predictions <- predict(model, test_x, array.layout = "rowmajor")
predictions <- data.frame(predictions, row.names = c(0:9))


# Summary of the predictions
print("Dimention of the output:")
```

```
## [1] "Dimention of the output:"
```

```
dim(predictions)
```

```
## [1]    10 10000
```

```
format(predictions[, 1:6], digits = 4)
```

```
##          X1        X2        X3        X4        X5        X6
## 0 1.172e-13 6.146e-12 8.991e-13 1.000e+00 3.176e-10 9.366e-16
## 1 3.438e-12 4.606e-11 9.999e-01 6.451e-17 3.340e-13 1.000e+00
## 2 2.669e-11 1.000e+00 6.020e-06 3.115e-11 3.198e-09 1.894e-09
## 3 1.517e-09 3.253e-07 2.315e-08 1.920e-15 2.037e-13 2.387e-11
## 4 1.481e-13 3.168e-18 6.973e-07 6.430e-12 9.996e-01 3.313e-08
## 5 1.057e-12 2.192e-17 1.304e-09 2.151e-13 2.764e-12 1.896e-14
## 6 1.113e-19 7.622e-13 9.137e-07 1.971e-09 5.728e-11 1.151e-11
## 7 1.000e+00 2.222e-17 1.774e-05 3.472e-11 1.710e-06 1.893e-06
## 8 9.729e-13 1.518e-08 8.414e-05 6.413e-14 1.931e-11 4.844e-09
## 9 2.594e-09 3.882e-21 2.602e-08 1.032e-07 4.039e-04 1.120e-10
```

As showed above, the output has 10 rows, corresponding to the probability of the ten numbers, and 10000 columns for all test samples. To extract the most probable label (row number with maximum value) for each sample, use `max.col`.

```
prediction_labels <- max.col(t(predictions)) - 1
table(prediction_labels)
```

```
## prediction_labels
##    0    1    2    3    4    5    6    7    8    9
##  992 1133 1039  973  995  870  955 1024  994 1025
```

And compute the accuracy of the model by comparing the predictions to the actual test labels.

```
# Put predictions and actual label in a same dataframe
compare <- data.frame(prediction_labels, test_y, prediction_labels == test_y)
names(compare) <- c("Predict", "Actual", "Matches")
head(compare, n = 10)
```

```
##    Predict Actual Matches
## 1        7      7    TRUE
## 2        2      2    TRUE
## 3        1      1    TRUE
## 4        0      0    TRUE
## 5        4      4    TRUE
## 6        1      1    TRUE
## 7        4      4    TRUE
## 8        9      9    TRUE
## 9        6      5   FALSE
## 10       9      9    TRUE
```

```
# Calculate the percentage of correct classification
print(sprintf("Accuracy: %f%%", sum(compare$Matches) / length(compare$Matches) * 100))
```

```
## [1] "Accuracy: 97.640000%"
```

The model achieves an accuracy of 97.640000%, which means the model is making reasonably good predictions.

Conclusion

As demonstrated in this post, neural network is proven to be a useful tool when facing data with unknown or complex underlying model. The weights and other parameters in the neural network are discovered automatically, which means human no longer need to think of what might be the relation between data, but instead the computer now has the ability to discover by itself and that also means now we don't immediately have an explanation of how the network does what it does.

How can we do better? It may have been noticed that neural network takes some time to train because most machine learning algorithms are designed to be slow in training, but fast when it is actually used to predict something. The neural network can also be constructed in other ways, such as including convolutional layers or other type of activation functions. Besides that, more iterations of training almost always improves performance. Hence better hardware such as more CPUs and GPUs is also a good way to increase the performance of the model.

Reference

- Wikipedia - Artificial Neural Network: https://en.wikipedia.org/wiki/Artificial_neural_network
- What is machine learning: https://www.sas.com/en_us/insights/analytics/machine-learning.html
- MNIST Data Set: http://yann.lecun.com/exdb/mnist/
- MNIST in CSV: https://pjreddie.com/projects/mnist-in-csv/
- MXNet Library: https://mxnet.incubator.apache.org/api/r/index.html
- MXNet Github Demo: https://github.com/apache/incubator-mxnet
- Handwritten Digits Classification Competition: https://mxnet.incubator.apache.org/tutorials/r/mnistCompetition.html
- A comparison of deep learning packages for R: http://blog.revolutionanalytics.com/2017/02/deep-learning-in-r.html