SDSC3006 Lab
10-Deep Learning

Langming LIU langmiliu2-c@my.cityu.edu.hk

School of Data Science
City University of Hong Kong

Contents

- Single Layer Network
- Multilayer Network
- Convolutional Neural Network

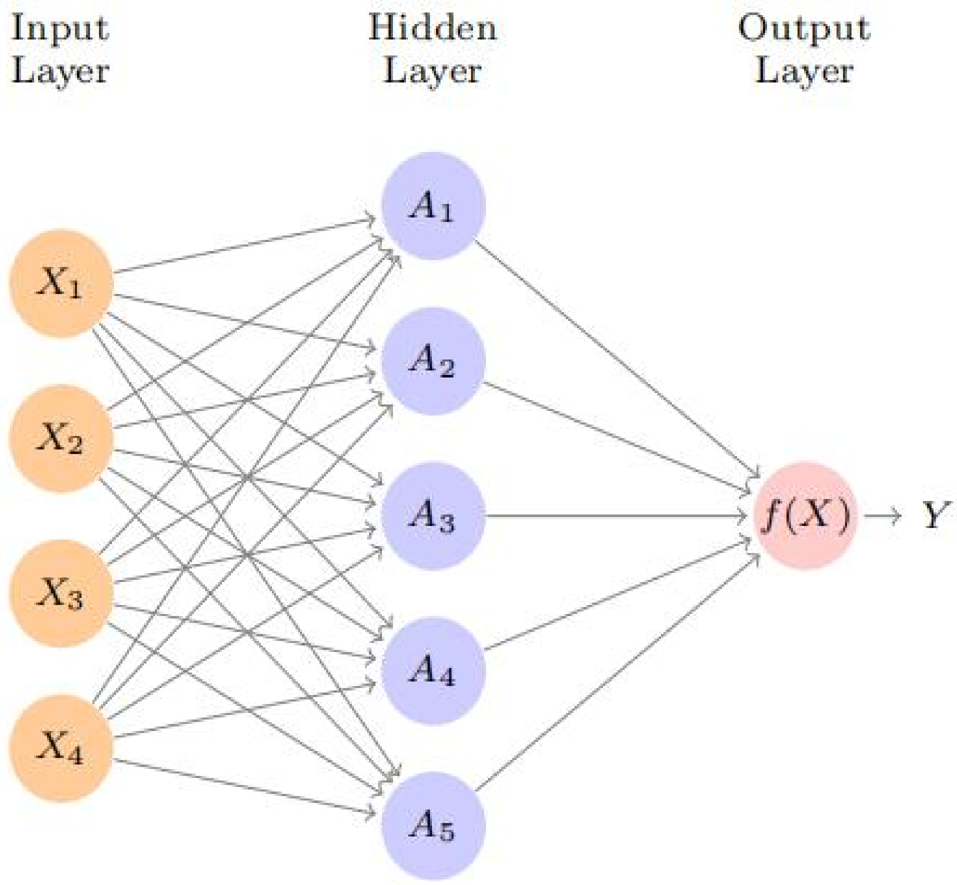
Single Layer Network

Introduction

- Layers: Input(1), Hidden(?), Output(1).
- Activation fuction: a non-linear function, such as sigmoid, tanh and ReLU.
- Loss: a measurement of NN, less is better. For instance, squared-error and cross-entropy.
- One-hot: a vector with one in the position related to its class and zeros elsewhere.

Introduction

- Training method: backpropagation. (gradient descent)
- Structure (for single one):



Implemention

```
#set up data and separate it
library (ISLR2)
Gitters = na.omit(Hitters)
n = nrow(Gitters)
set.seed (13)
ntest = trunc(n/3)
testid = sample(1:n, ntest)
x = scale(model.matrix(Salary~.-1, data = Gitters))
y = Gitters$Salary
#linear model for comparison
lfit = lm(Salary~., data = Gitters[-testid , ])
lpred = predict (lfit , Gitters[testid , ])
with (Gitters[testid , ], mean (abs (lpred - Salary)))
```

Implemention

##installation steps of tensorflow and keras: see lab02

##create model structure of NN using keras

```
library (keras)
modnn = keras_model_sequential() %>%
  layer_dense(units=50, activation="relu", input_shape=ncol(x)) %>%
  layer_dropout(rate=0.4) %>%
  layer_dense(units=1)
```

##pipe operator %>% passes the previous term as the first argument to the next function

##dropout: randomly drop the activations from previous layer(set to 0)

```
modnn %>% compile (loss = "mse",
optimizer = optimizer_rmsprop(),
metrics = list ("mean_absolute_error") )
```

##compile: not change R object, but relates to python

Implemention

```
##plot the history to display the mae for the training and test data
history = modnn %>% fit(x[-testid,], y[-testid], epochs=200, batch_size=32,
                    validation_data = list(x[testid,], y[testid]))
##batch_size: randomly choose training obs for SGD
install.packages('ggplots')
plot (history)

##calculate mae
npred =predict (modnn , x[testid , ])
mean ( abs (y[testid] - npred))
```

Multilayer Network

Implemention

```
mnist = dataset_mnist()
librag_train = mnist$train$y
x_train = mnist$train$x
g_train = mnist$train$y
x_test = mnist$test$x
g_test = mnist$test$y
dim(x_train)
dim(x_test)
# reshape the array into matrix and "one-hot"
x_train = array_reshape(x_train, c(nrow(x_train), 784))
x_test = array_reshape(x_test, c(nrow(x_test), 784))
y_train = to_categorical(g_train, 10)
y_test = to_categorical(g_test, 10)
# rescale the original data(0~255) into unit interval
x_train =x_train / 255
x_test = x_test / 255
```

Implemention

```
##structure of NN
modelnn = keras_model_sequential()
modelnn %>%
  layer_dense(units=256, activation="relu", input_shape=c(784))
%>%
  layer_dropout(rate=0.4) %>%
  layer_dense(units=128, activation="relu") %>%
  layer_dropout(rate=0.3) %>%
  layer_dense(units=10, activation="softmax")
##summary(modelnn)
```

Implemention

```
##structure of NN
modelnn %>% compile(loss="categorical_crossentropy",
                    optimizer=optimizer_rmsprop(),
                    metrics=c("accuracy"))
##supply training data, and fit the model
system.time(history <- modelnn %>%
              fit(x_train, y_train, epochs=30, batch_size=128,
                  validation_split=0.2))
plot(history, smooth=FALSE)

##evaluate the model by caculate the accuracy
accuracy = function (pred , truth)
           mean ( drop (pred) == drop (truth))
modelnn %>% predict_classes(x_test) %>% accuracy(g_test)
```

Convolutional Neural Network

Keys of CNN

- Convolution: made up of a large number of convolution filters(a little matrix).
- Max Pooling: condense a large image into a smaller pooling summary image.

$$\text{Max pool}\begin{bmatrix}1 & 2 & 5 & 3\\ 3 & 0 & 1 & 2\\ 2 & 1 & 3 & 4\\ 1 & 1 & 2 & 0\end{bmatrix}\rightarrow\begin{bmatrix}3 & 5\\ 2 & 4\end{bmatrix}.$$

- Architecture: convolve-then-pool squence and...
- Padding: guarantee same dimension
- Flattening: matrix -> vector
- Code