deep learning model

Dr.metales

12/5/2019

```
knitr::opts_chunk$set(echo = TRUE, warning=FALSE,error=FALSE,message=FALSE)
```

Prepare the data

Deep learning models are becoming the most importatnt predictive models uesd by the well known companies in the world. In this paper we will use the deep model to predict the competition titanic data set presentd by kaggle. Let's call this data.

```
library(tidyverse)
data <- read_csv("train.csv")
```

first I will call some packages, **tidyverse** to manipulate the data, **keras** package for deep learning models,**caret** for randomly spliting the data and creating the confusion matrix.

```
library(keras)
library(caret)
```

The first step in modeling is to clean and prepare the data. the following code shows the structure of this data.

```
glimpse(data)
```

```
## Observations: 891
## Variables: 12
## $ PassengerId <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
## $ Survived    <dbl> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,...
## $ Pclass      <dbl> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3,...
## $ Name        <chr> "Braund, Mr. Owen Harris", "Cumings, Mrs. John Bra...
## $ Sex         <chr> "male", "female", "female", "female", "male", "mal...
## $ Age         <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, ...
## $ SibSp       <dbl> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4,...
## $ Parch       <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1,...
## $ Ticket      <chr> "A/5 21171", "PC 17599", "STON/O2. 3101282", "1138...
## $ Fare        <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, ...
## $ Cabin       <chr> NA, "C85", NA, "C123", NA, NA, "E46", NA, NA, NA, ...
## $ Embarked    <chr> "S", "C", "S", "S", "S", "Q", "S", "S", "S", "C", ...
```

From this data we want to predict the variable **Survived** using the remaining variables.We see that some variables have unique values such as **PassengerId**,**Name**, and **ticket**. Thus,they cannot be used as predictors. the same note applies to the vraiable **Cabin** with the additional probleme of missing values. these variables will be removed as follows:

```r
mydata<-data[,-c(1,4,9,11)]
head(mydata)
```

```
## # A tibble: 6 x 8
##   Survived Pclass Sex      Age SibSp Parch  Fare Embarked
##      <dbl>  <dbl> <chr>  <dbl> <dbl> <dbl> <dbl> <chr>
## 1        0      3 male      22     1     0  7.25 S
## 2        1      1 female    38     1     0 71.3  C
## 3        1      3 female    26     0     0  7.92 S
## 4        1      1 female    35     1     0 53.1  S
## 5        0      3 male      35     0     0  8.05 S
## 6        0      3 male      NA     0     0  8.46 Q
```

As we see some variables should be of factor type such as **Pclass** (which is now doouble), **Sex** (character), and "Embarked** (character). thus, we convert them to factor type:

```r
mydata$Pclass<-as.factor(mydata$Pclass)
mydata$Embarked<-as.factor(mydata$Embarked)
mydata$Sex<-as.factor(mydata$Sex)
glimpse(mydata)
```

```
## Observations: 891
## Variables: 8
## $ Survived <dbl> 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,...
## $ Pclass   <fct> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 2,...
## $ Sex      <fct> male, female, female, female, male, male, male, male,...
## $ Age      <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, 39,...
## $ SibSp    <dbl> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4, 0,...
## $ Parch    <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1, 0,...
## $ Fare     <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, 51....
## $ Embarked <fct> S, C, S, S, S, Q, S, S, S, C, S, S, S, S, S, S, Q, S,...
```

Now let's get some summary about this data

```r
summary(mydata)
```

```
##     Survived      Pclass      Sex           Age            SibSp
##  Min.   :0.0000   1:216   female:314   Min.   : 0.42   Min.   :0.000
##  1st Qu.:0.0000   2:184   male  :577   1st Qu.:20.12   1st Qu.:0.000
##  Median :0.0000   3:491                Median :28.00   Median :0.000
##  Mean   :0.3838                        Mean   :29.70   Mean   :0.523
##  3rd Qu.:1.0000                        3rd Qu.:38.00   3rd Qu.:1.000
##  Max.   :1.0000                        Max.   :80.00   Max.   :8.000
##                                        NA's   :177
##      Parch             Fare        Embarked
##  Min.   :0.0000   Min.   :  0.00   C   :168
##  1st Qu.:0.0000   1st Qu.:  7.91   Q   : 77
##  Median :0.0000   Median : 14.45   S   :644
##  Mean   :0.3816   Mean   : 32.20   NA's:  2
##  3rd Qu.:0.0000   3rd Qu.: 31.00
##  Max.   :6.0000   Max.   :512.33
##
```

We have only two variables that have missing values, **Age** with large number 177 , followed by **Embarked** with 2 missing values. To deal with this issue we have two options:

- the first and easy one is to remove the entire rows that have any missing value but with the cost of may losing valuable informations specially when we have large number of missing values which is in our case.
- the second option is to impute this missing values using the other complete cases, for instance we can replace a missing value of a peticular column by the mean of this column (for numeric variable) or we use multinomial method to predict the categorical variables.

fortunately , there is a usefull package called **mice** which will do this imputation for us. However, applying this imputation on the entire data would lead us to fall on a probleme called **train-test comtamination** ,which means that when we split the data , the missing values of the training set are imputed using cases in the test set, and this violates a crucial concept in machine learning for model evaluation, the test set should never be seen by the model during the training process.

To avoid this problem we apply the imputation seperatly on the training set and on the test set. So let's partition the data using **caret** package function.

Partition the data & impute the missing values.

we randomly split the data into two sets , 80% of samples will go to the training set and the remaining 20% will be kept as test set.

```
set.seed(1234)
index<-createDataPartition(mydata$Survived,p=0.8,list=FALSE)
train<-mydata[index,]
test<-mydata[-index,]
```

Now we are ready to impute the missing values for both train and test set.

```
library(mice)
impute_train<-mice(train,m=1,seed = 1111)
```

```
##
##  iter imp variable
##   1   1  Age  Embarked
##   2   1  Age  Embarked
##   3   1  Age  Embarked
##   4   1  Age  Embarked
##   5   1  Age  Embarked
```

```
train<-complete(impute_train,1)

impute_test<-mice(test,m=1,seed = 1111)
```

```
##
##  iter imp variable
##   1   1  Age  Embarked
##   2   1  Age  Embarked
##   3   1  Age  Embarked
##   4   1  Age  Embarked
##   5   1  Age  Embarked
```

```r
test<-complete(impute_test,1)
```

Convert data into a normalized matrix.

in deep learning all the variables should of numeric type, so first we convert the factors to integer type and recode the levels in order to start from 0, then we convert the data into matrix. After that we pull out the target variable into a separate vector, and finally we normalize our matrix.

We do this transformation for both sets (train and test).

```r
train$Embarked<-as.integer(train$Embarked)-1
train$Sex<-as.integer(train$Sex)-1
train$Pclass<-as.integer(train$Pclass)-1

test$Embarked<-as.integer(test$Embarked)-1
test$Sex<-as.integer(test$Sex)-1
test$Pclass<-as.integer(test$Pclass)-1
glimpse(test)
```

```
## Observations: 178
## Variables: 8
## $ Survived <dbl> 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,...
## $ Pclass   <dbl> 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 0, 0, 2, 2, 2, 1, 2,...
## $ Sex      <dbl> 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,...
## $ Age      <dbl> 35.0, 2.0, 27.0, 55.0, 38.0, 23.0, 38.0, 3.0, 28.0, 3...
## $ SibSp    <dbl> 0, 3, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 4, 5, 0, 0,...
## $ Parch    <dbl> 0, 1, 2, 0, 0, 0, 5, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0,...
## $ Fare     <dbl> 8.0500, 21.0750, 11.1333, 16.0000, 13.0000, 7.2250, 3...
## $ Embarked <dbl> 2, 2, 2, 2, 2, 0, 2, 0, 2, 1, 2, 2, 0, 2, 2, 2, 2, 2,...
```

we convert the tow sets into matrix form. (we also remove the column names)

```r
trained<-as.matrix(train)
dimnames(trained)<-NULL

tested<-as.matrix(test)
dimnames(tested)<-NULL
str(trained)
```

```
##  num [1:713, 1:8] 0 1 1 1 0 0 1 1 1 0 ...
```

Now we pull out the target variabeles

```r
trainy<-trained[,1]
testy<-tested[,1]
trainx<-trained[,-1]
testx<-tested[,-1]
```

Apply one hot encoding on the target variable

```r
trainlabel<-to_categorical(trainy)
testlabel<-to_categorical(testy)
```

The final step now is normalizing the matrices (trainx and testx)

```r
trainx<-normalize(trainx)
testx<-normalize(testx)
summary(testx)
```

```
##        V1                 V2                 V3                 V4
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.02114   Min.   :0.00000
##  1st Qu.:0.01781   1st Qu.:0.00000   1st Qu.:0.62924   1st Qu.:0.00000
##  Median :0.05052   Median :0.01681   Median :0.89791   Median :0.00000
##  Mean   :0.04924   Mean   :0.01898   Mean   :0.74744   Mean   :0.01284
##  3rd Qu.:0.07560   3rd Qu.:0.03292   3rd Qu.:0.95154   3rd Qu.:0.01134
##  Max.   :0.24380   Max.   :0.05572   Max.   :0.99827   Max.   :0.20525
##        V5                  V6                V7
##  Min.   :0.000000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.2945   1st Qu.:0.02804
##  Median :0.000000   Median :0.4324   Median :0.04562
##  Mean   :0.008293   Mean   :0.5168   Mean   :0.04756
##  3rd Qu.:0.000000   3rd Qu.:0.7753   3rd Qu.:0.06829
##  Max.   :0.102624   Max.   :0.9977   Max.   :0.17216
```

Train the model.

Now it is time time to buil our model. Th first step is to define the model type and the number of layers that will be used with the prespecified parameters. We will choose a simple model with one hidden layer with 10 unites (nodes). Since we have 7 predictors the input_shape will be 7, and the activation function is **relu** which is the most used one, but for the output layer we choose sigmoid function since we have binary classification.

Creat the model

```r
model <- keras_model_sequential()

model %>%
    layer_dense(units=10,activation = "relu",
              kernel_initializer = "he_normal",input_shape =c(7))%>%
    layer_dense(units=2,activation = "sigmoid")

summary(model)
```

```
## Model: "sequential"
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense (Dense)                    (None, 10)                    80
```

```
## ___________________________________________________________________________
## dense_1 (Dense)                  (None, 2)                     22
## ===========================================================================
## Total params: 102
## Trainable params: 102
## Non-trainable params: 0
## ___________________________________________________________________________
```

We have in total 102 parameters to estimate, since we have 7 inputs and 10 nodes and 10 biases, so the parameters number of the hidden layer is 80 (7*10+10).By the same way get the parameters number of the output layer.

Compile the model

In the compile function (from keras) we spacify the loss function, the optimizer and the metric type that will be used. In our case we use the **binary crossentropy**, the optimizer is the popular one **adam** and for the metric we use **accuracy**.

```
model %>%
  compile(loss="binary_crossentropy",
          optimizer="adam",
          metric="accuracy")
```

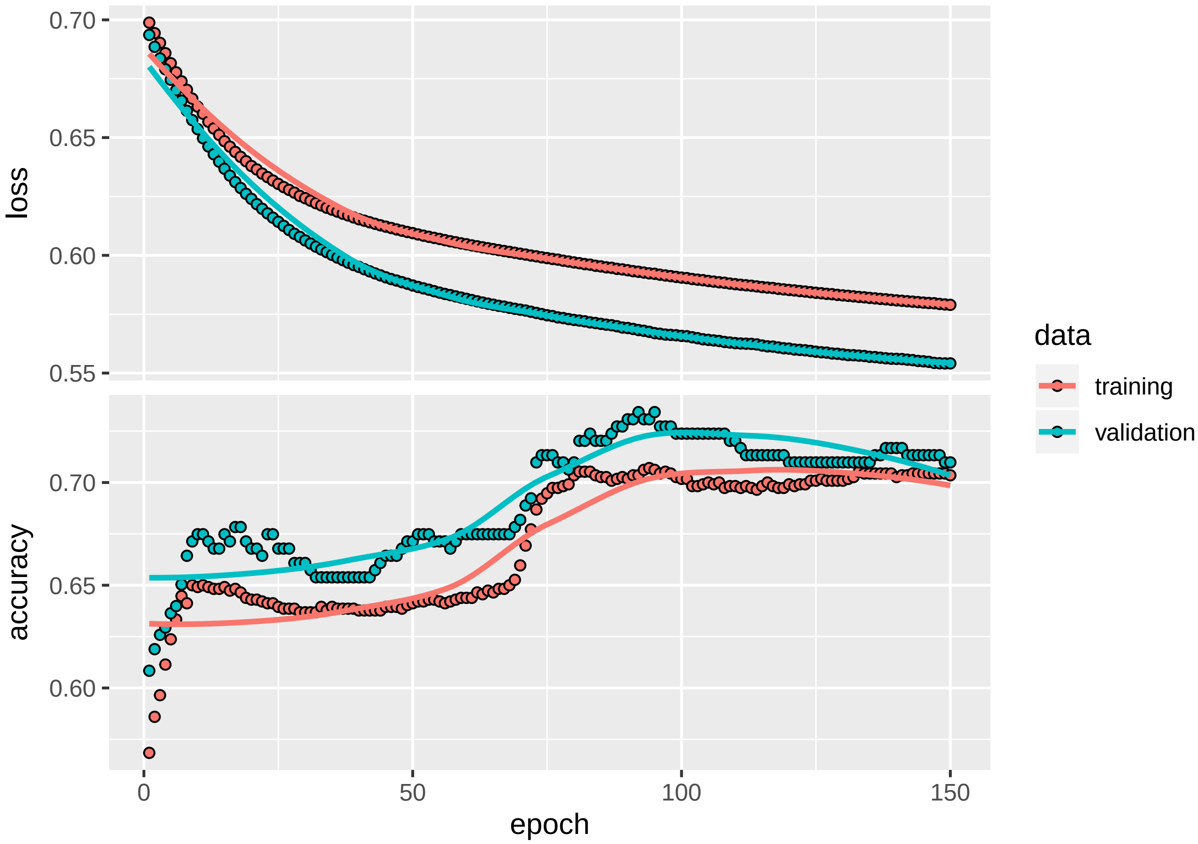
Execute the model

Now it is time to run our model and we can follow the dynamic evolution of the process in the plot windwo on the right lower corner of the screen. and you can also plot the model in a static way. for our model we choose 100 epochs (iterations), for the stochastic gradient we use 50 samples at each iteration, and we hold out 20% of the training data to asses the model.

```
history<- model %>%
  fit (trainx,trainlabel,epoch=150,batch_size=100,validation_split=0.2)
```

for the last iteration we see that the loss is about 0.5516 and the accuracy is 72.28%.

```
plot(history)
```



It should be noted here that since the accuracy lines are more or less closer to each other we do not have to be wory about overfiting.

The model evaluation

```
model %>%
  evaluate(testx, testlabel)
```

```
## $loss
## [1] 0.6062116
##
## $accuracy
## [1] 0.6573034
```

The accuracy rate of the model using the test set is 64.61 which much lower than that of the training set.

prediction and confusion matrix

we get the prediction on the test set as follows.

```
pred<-predict_classes(model,testx)
head(pred)
```

```
## [1] 0 1 0 0 0 0
```

Using the **caret** package we get the confusion matrix

```r
confusionMatrix(as.factor(pred),as.factor(testy))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 91 37
##          1 23 27
##
##                Accuracy : 0.6629
##                  95% CI : (0.5884, 0.7319)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : 0.29414
##
##                   Kappa : 0.2312
##
##  Mcnemar's Test P-Value : 0.09329
##
##             Sensitivity : 0.7982
##             Specificity : 0.4219
##          Pos Pred Value : 0.7109
##          Neg Pred Value : 0.5400
##              Prevalence : 0.6404
##          Detection Rate : 0.5112
##    Detection Prevalence : 0.7191
##       Balanced Accuracy : 0.6101
##
##        'Positive' Class : 0
##
```

As we see the moderate accuracy rate ** 64.61% ** leads us to think about refiting our model by tuning again some parameters. To do so we try first to increase the nodes number and see what will happen. And then we will add another hidden layer and check if any improvment happens.

tune the model by increasing the number of nodes to 40 nodes

```r
model1 <- keras_model_sequential()

model1 %>%
    layer_dense(units=40,activation = "relu",
              kernel_initializer = "he_normal",input_shape =c(7))%>%
    layer_dense(units=2,activation = "sigmoid")

model1 %>%
  compile(loss="binary_crossentropy",
          optimizer="adam",
          metric="accuracy")
```

```r
history1<- model1 %>%
  fit (trainx,trainlabel,epoch=200,batch_size=40,validation_split=0.2)
```

with this new model the accuracy in the validation set at the last iteration is about 79.72% which is a large improvment compared to our simple model.

Let's check the accuracy for the test set.

```r
pred<-predict_classes(model1,testx)
confusionMatrix(as.factor(pred),as.factor(testy))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 96 36
##          1 18 28
##
##                Accuracy : 0.6966
##                  95% CI : (0.6234, 0.7632)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : 0.0676
##
##                   Kappa : 0.298
##
##  Mcnemar's Test P-Value : 0.0207
##
##             Sensitivity : 0.8421
##             Specificity : 0.4375
##          Pos Pred Value : 0.7273
##          Neg Pred Value : 0.6087
##              Prevalence : 0.6404
##          Detection Rate : 0.5393
##    Detection Prevalence : 0.7416
##       Balanced Accuracy : 0.6398
##
##        'Positive' Class : 0
##
```

wr get also a large improvment for the test set which is now more than 82%.

tune the model by increasing the number of layers This time we will add two hidden layer with 20 aand 10 nodes respectively.

```r
model2 <- keras_model_sequential()

model2 %>%
    layer_dense(units=40,activation = "relu",
              kernel_initializer = "he_normal",input_shape =c(7))%>%
```

```
    layer_dense(units=20,activation = "relu")%>%
    layer_dense(units=10,activation = "relu")%>%
    layer_dense(units=2,activation = "sigmoid")

model2 %>%
  compile(loss="binary_crossentropy",
          optimizer="adam",
          metric="accuracy")


history2 <- model2 %>%
  fit (trainx,trainlabel,epoch=200,batch_size=50,validation_split=0.2)
```

we get a slight improvment for the accuracy of the validation set in the training data from 79.72% to 82,17%. For the test set

```
pred<-predict_classes(model2,testx)
confusionMatrix(as.factor(pred),as.factor(testy))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 94 15
##          1 20 49
##
##                Accuracy : 0.8034
##                  95% CI : (0.7373, 0.8591)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : 1.629e-06
##
##                   Kappa : 0.5802
##
##  Mcnemar's Test P-Value : 0.499
##
##             Sensitivity : 0.8246
##             Specificity : 0.7656
##          Pos Pred Value : 0.8624
##          Neg Pred Value : 0.7101
##              Prevalence : 0.6404
##          Detection Rate : 0.5281
##    Detection Prevalence : 0.6124
##       Balanced Accuracy : 0.7951
##
##        'Positive' Class : 0
##
```

using this model our test accuracy has been improved from 82.02% to 84.27%.