Lab 1: Topics in Deep Learning

Lanston Chen

1. Linear regression model. Let's take another look at the Boston housing data from HW1. We will compare the performance of a linear model to a deep neural network on predicting **medv**.

```
set.seed(1)
Boston <- read.csv('https://zhang-datasets.s3.us-east-2.amazonaws.com/Boston.csv')
```

(a) Split the data into training (70%) and testing sets (30%)

```
train_id <- sample(1:nrow(Boston),nrow(Boston)*0.7)
test_id <- -train_id
```

(b) Fit a linear regression model using all covariates and report the testing error

```
model_lm <- lm(medv~.,data=Boston[train_id,])
pred_lm <- predict(model_lm,newdata=Boston[test_id,])
mean((pred_lm-Boston$medv[test_id])^2)  #var(Boston$medv)
```

```
## [1] 27.25409
```

(c) Fit a DNN and report the testing error

```
model_dnn <- keras_model_sequential()%>%
  layer_dense(input_shape = 12,units = 128,activation = 'relu') %>%
  layer_dropout(rate = 0.4) %>% #usually in small size data r1 and r2 are better than dropout
  layer_dense(units = 64,activation = 'relu')%>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1)

summary(model_dnn)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  dense_2 (Dense)                    (None, 128)                     1664
##  dropout_1 (Dropout)                (None, 128)                     0
##  dense_1 (Dense)                    (None, 64)                      8256
##  dropout (Dropout)                  (None, 64)                      0
##  dense (Dense)                      (None, 1)                       65
## ================================================================================
## Total params: 9,985
## Trainable params: 9,985
```

```
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
x <- model.matrix(medv~.,Boston)[,-1]
y <- Boston$medv

model_dnn %>% compile(loss = 'mse', optimizer = optimizer_rmsprop(), metrics = 'mean_absolute_e

history <- model_dnn %>% fit(x[train_id,], y[train_id], batch_size = 32, epochs = 50, validatio
```

```
## Epoch 1/50
## 9/9 - 1s - loss: 1083.1344 - mean_absolute_error: 24.8231 - val_loss: 240.6402 - val_mean_al
## Epoch 2/50
## 9/9 - 0s - loss: 812.1465 - mean_absolute_error: 22.2661 - val_loss: 272.4199 - val_mean_abs
## Epoch 3/50
## 9/9 - 0s - loss: 580.0920 - mean_absolute_error: 18.4052 - val_loss: 255.2390 - val_mean_abs
## Epoch 4/50
## 9/9 - 0s - loss: 412.9235 - mean_absolute_error: 15.8363 - val_loss: 330.0681 - val_mean_abs
## Epoch 5/50
## 9/9 - 0s - loss: 423.6916 - mean_absolute_error: 16.0580 - val_loss: 383.7601 - val_mean_abs
## Epoch 6/50
## 9/9 - 0s - loss: 310.7329 - mean_absolute_error: 14.1426 - val_loss: 238.7386 - val_mean_abs
## Epoch 7/50
## 9/9 - 0s - loss: 333.9290 - mean_absolute_error: 14.4182 - val_loss: 339.4520 - val_mean_abs
## Epoch 8/50
## 9/9 - 0s - loss: 314.2440 - mean_absolute_error: 14.2493 - val_loss: 366.4426 - val_mean_abs
## Epoch 9/50
## 9/9 - 0s - loss: 257.0594 - mean_absolute_error: 12.7168 - val_loss: 342.0568 - val_mean_abs
## Epoch 10/50
## 9/9 - 0s - loss: 250.0210 - mean_absolute_error: 12.5738 - val_loss: 349.3872 - val_mean_abs
## Epoch 11/50
## 9/9 - 0s - loss: 254.5619 - mean_absolute_error: 12.6325 - val_loss: 309.6800 - val_mean_abs
## Epoch 12/50
## 9/9 - 0s - loss: 233.6625 - mean_absolute_error: 12.0085 - val_loss: 360.9101 - val_mean_abs
## Epoch 13/50
## 9/9 - 0s - loss: 189.0828 - mean_absolute_error: 10.6177 - val_loss: 364.2737 - val_mean_abs
## Epoch 14/50
## 9/9 - 0s - loss: 203.0387 - mean_absolute_error: 11.0310 - val_loss: 313.5056 - val_mean_abs
## Epoch 15/50
## 9/9 - 0s - loss: 183.1197 - mean_absolute_error: 10.6606 - val_loss: 272.7821 - val_mean_abs
## Epoch 16/50
## 9/9 - 0s - loss: 208.7043 - mean_absolute_error: 11.4826 - val_loss: 282.9757 - val_mean_abs
## Epoch 17/50
## 9/9 - 0s - loss: 175.1482 - mean_absolute_error: 10.2954 - val_loss: 306.2222 - val_mean_abs
## Epoch 18/50
## 9/9 - 0s - loss: 179.7897 - mean_absolute_error: 10.0474 - val_loss: 280.6985 - val_mean_abs
## Epoch 19/50
## 9/9 - 0s - loss: 168.8600 - mean_absolute_error: 10.3628 - val_loss: 211.3609 - val_mean_abs
```

```
## Epoch 20/50
## 9/9 - 0s - loss: 180.9448 - mean_absolute_error: 10.2493 - val_loss: 307.1349 - val_mean_abs
## Epoch 21/50
## 9/9 - 0s - loss: 170.7650 - mean_absolute_error: 9.9133 - val_loss: 227.7525 - val_mean_abso
## Epoch 22/50
## 9/9 - 0s - loss: 146.0775 - mean_absolute_error: 9.4613 - val_loss: 218.1820 - val_mean_abso
## Epoch 23/50
## 9/9 - 0s - loss: 140.9657 - mean_absolute_error: 8.9436 - val_loss: 258.9810 - val_mean_abso
## Epoch 24/50
## 9/9 - 0s - loss: 146.7811 - mean_absolute_error: 9.5284 - val_loss: 212.8857 - val_mean_abso
## Epoch 25/50
## 9/9 - 0s - loss: 141.2902 - mean_absolute_error: 9.1922 - val_loss: 176.0727 - val_mean_abso
## Epoch 26/50
## 9/9 - 0s - loss: 157.4685 - mean_absolute_error: 9.6501 - val_loss: 185.9038 - val_mean_abso
## Epoch 27/50
## 9/9 - 0s - loss: 147.7150 - mean_absolute_error: 9.3558 - val_loss: 214.1011 - val_mean_abso
## Epoch 28/50
## 9/9 - 0s - loss: 139.9552 - mean_absolute_error: 9.1934 - val_loss: 192.2865 - val_mean_abso
## Epoch 29/50
## 9/9 - 0s - loss: 124.5697 - mean_absolute_error: 8.2209 - val_loss: 159.8399 - val_mean_abso
## Epoch 30/50
## 9/9 - 0s - loss: 116.0322 - mean_absolute_error: 8.1860 - val_loss: 218.4985 - val_mean_abso
## Epoch 31/50
## 9/9 - 0s - loss: 119.1671 - mean_absolute_error: 8.5145 - val_loss: 193.5421 - val_mean_abso
## Epoch 32/50
## 9/9 - 0s - loss: 125.1759 - mean_absolute_error: 8.5710 - val_loss: 188.6007 - val_mean_abso
## Epoch 33/50
## 9/9 - 0s - loss: 121.3494 - mean_absolute_error: 8.5148 - val_loss: 222.4068 - val_mean_abso
## Epoch 34/50
## 9/9 - 0s - loss: 107.9425 - mean_absolute_error: 7.8854 - val_loss: 166.0117 - val_mean_abso
## Epoch 35/50
## 9/9 - 0s - loss: 117.9403 - mean_absolute_error: 8.1304 - val_loss: 180.9890 - val_mean_abso
## Epoch 36/50
## 9/9 - 0s - loss: 110.5505 - mean_absolute_error: 7.9838 - val_loss: 162.4332 - val_mean_abso
## Epoch 37/50
## 9/9 - 0s - loss: 113.0313 - mean_absolute_error: 8.1526 - val_loss: 176.9709 - val_mean_abso
## Epoch 38/50
## 9/9 - 0s - loss: 111.6499 - mean_absolute_error: 7.8821 - val_loss: 172.0228 - val_mean_abso
## Epoch 39/50
## 9/9 - 0s - loss: 115.4506 - mean_absolute_error: 8.2724 - val_loss: 164.3115 - val_mean_abso
## Epoch 40/50
## 9/9 - 0s - loss: 115.1462 - mean_absolute_error: 8.0427 - val_loss: 187.3018 - val_mean_abso
## Epoch 41/50
## 9/9 - 0s - loss: 105.5983 - mean_absolute_error: 7.7639 - val_loss: 164.7424 - val_mean_abso
## Epoch 42/50
## 9/9 - 0s - loss: 99.7772 - mean_absolute_error: 7.3454 - val_loss: 202.0200 - val_mean_absol
## Epoch 43/50
## 9/9 - 0s - loss: 108.8730 - mean_absolute_error: 8.0864 - val_loss: 149.8183 - val_mean_abso
```

```
## Epoch 44/50
## 9/9 - 0s - loss: 107.1982 - mean_absolute_error: 7.7710 - val_loss: 198.6686 - val_mean_abso
## Epoch 45/50
## 9/9 - 0s - loss: 97.5604 - mean_absolute_error: 7.3683 - val_loss: 191.4267 - val_mean_absol
## Epoch 46/50
## 9/9 - 0s - loss: 84.3976 - mean_absolute_error: 7.2570 - val_loss: 162.8712 - val_mean_absol
## Epoch 47/50
## 9/9 - 0s - loss: 104.9332 - mean_absolute_error: 7.8211 - val_loss: 167.8739 - val_mean_abso
## Epoch 48/50
## 9/9 - 0s - loss: 95.2794 - mean_absolute_error: 7.1927 - val_loss: 154.1691 - val_mean_absol
## Epoch 49/50
## 9/9 - 0s - loss: 100.4315 - mean_absolute_error: 7.6725 - val_loss: 164.8013 - val_mean_abso
## Epoch 50/50
## 9/9 - 0s - loss: 87.0540 - mean_absolute_error: 6.8439 - val_loss: 117.4333 - val_mean_absol
```

2. Generalized linear regression model. Let's look at the Employee Attrition data from AT&T.

```r
d <- read.csv('https://zhang-datasets.s3.us-east-2.amazonaws.com/EmployeeAttrition.csv')



ncount <-c()
for ( j in 1:ncol(d)){
  ncount[j] <- length(unique(d[,j]))
}

d<-d[-c(9,10,20,25)]

m0 <- glm(Attrition~.,data = d,family = 'binomial')
summary(m0)
```

```
##
## Call:
## glm(formula = Attrition ~ ., family = "binomial", data = d)
##
## Coefficients:
##                                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -1.047e+01  3.846e+02  -0.027  0.97828
## Age                              -3.074e-02  1.352e-02  -2.273  0.02302 *
## BusinessTravelTravel_Frequently   1.915e+00  4.116e-01   4.652 3.29e-06 ***
## BusinessTravelTravel_Rarely       1.034e+00  3.796e-01   2.725  0.00643 **
## DailyRate                        -2.802e-04  2.198e-04  -1.274  0.20250
## DepartmentResearch & Development  1.276e+01  3.846e+02   0.033  0.97353
## DepartmentSales                   1.260e+01  3.846e+02   0.033  0.97386
## DistanceFromHome                  4.555e-02  1.075e-02   4.236 2.28e-05 ***
## Education                         2.363e-03  8.751e-02   0.027  0.97846
## EducationFieldLife Sciences      -7.405e-01  8.048e-01  -0.920  0.35755
## EducationFieldMarketing          -3.479e-01  8.531e-01  -0.408  0.68337
## EducationFieldMedical            -8.360e-01  8.045e-01  -1.039  0.29875
## EducationFieldOther              -8.216e-01  8.632e-01  -0.952  0.34122
```

```
## EducationFieldTechnical Degree    1.800e-01  8.226e-01   0.219  0.82679
## EnvironmentSatisfaction          -4.339e-01  8.281e-02  -5.240 1.61e-07 ***
## GenderMale                        3.923e-01  1.842e-01   2.130  0.03313 *
## JobInvolvement                   -5.237e-01  1.224e-01  -4.280 1.87e-05 ***
## JobLevel                         -7.756e-02  3.150e-01  -0.246  0.80550
## JobRoleHuman Resources            1.400e+01  3.846e+02   0.036  0.97096
## JobRoleLaboratory Technician      1.476e+00  4.833e-01   3.055  0.00225 **
## JobRoleManager                    3.644e-01  8.885e-01   0.410  0.68171
## JobRoleManufacturing Director     2.185e-01  5.322e-01   0.411  0.68138
## JobRoleResearch Director         -1.081e+00  1.004e+00  -1.077  0.28136
## JobRoleResearch Scientist         5.320e-01  4.948e-01   1.075  0.28230
## JobRoleSales Executive            1.185e+00  1.124e+00   1.054  0.29170
## JobRoleSales Representative       2.137e+00  1.178e+00   1.814  0.06960 .
## JobSatisfaction                  -4.129e-01  8.130e-02  -5.078 3.81e-07 ***
## MaritalStatusMarried              3.253e-01  2.664e-01   1.221  0.22201
## MaritalStatusSingle               1.147e+00  3.453e-01   3.323  0.00089 ***
## MonthlyIncome                     9.038e-06  8.134e-05   0.111  0.91153
## NumCompaniesWorked                1.952e-01  3.877e-02   5.035 4.78e-07 ***
## OverTimeYes                       1.983e+00  1.937e-01  10.239  < 2e-16 ***
## PercentSalaryHike                -2.135e-02  3.923e-02  -0.544  0.58622
## PerformanceRating                 1.147e-01  3.977e-01   0.288  0.77306
## RelationshipSatisfaction         -2.571e-01  8.248e-02  -3.117  0.00183 **
## StockOptionLevel                 -2.113e-01  1.577e-01  -1.340  0.18029
## TotalWorkingYears                -6.046e-02  2.932e-02  -2.062  0.03920 *
## TrainingTimesLastYear            -1.898e-01  7.308e-02  -2.598  0.00939 **
## WorkLifeBalance                  -3.688e-01  1.237e-01  -2.981  0.00287 **
## YearsAtCompany                    9.350e-02  3.887e-02   2.405  0.01616 *
## YearsInCurrentRole               -1.482e-01  4.544e-02  -3.260  0.00111 **
## YearsSinceLastPromotion           1.736e-01  4.223e-02   4.111 3.95e-05 ***
## YearsWithCurrManager             -1.353e-01  4.694e-02  -2.882  0.00396 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1298.58  on 1469  degrees of freedom
## Residual deviance:  859.22  on 1427  degrees of freedom
## AIC: 945.22
##
## Number of Fisher Scoring iterations: 14
```

(a) Split the data into training (70%) and testing sets (30%)

```r
x <- model.matrix(Attrition~.,data = d)[,-1]
y <- d$Attrition

set.seed(1)
train_id <- sample(1:nrow(x),nrow(x)*0.7)
```

```
test_id <-- train_id
```

(b) Fit a GLM using all covariates and report the classification accuracy

```
model_glm  <- glm(Attrition~., data = d[train_id,],family = 'binomial')
pred_glm <- predict(model_glm,newdata= d[test_id,],type='response')
```

```
accuracy <- function(pred,true){
  mean(as.numeric(pred)==true)
}

accuracy((pred_glm>0.5)*1,y[test_id])  #time 1 means change the logic value to number
```

```
## [1] 0.8866213
```

(c) Fit a GLM with lasso using all covariates and report the classification accuracy

```
cv.out <- cv.glmnet(x[train_id,],y[train_id],alpha=1,family='binomial')
lambda.best <- cv.out$lambda.min
predict_lasso <- predict(cv.out, s=lambda.best,newx=x[test_id,],type='response')
accuracy((predict_lasso>0.5)*1,y[test_id])
```

```
## [1] 0.8843537
```

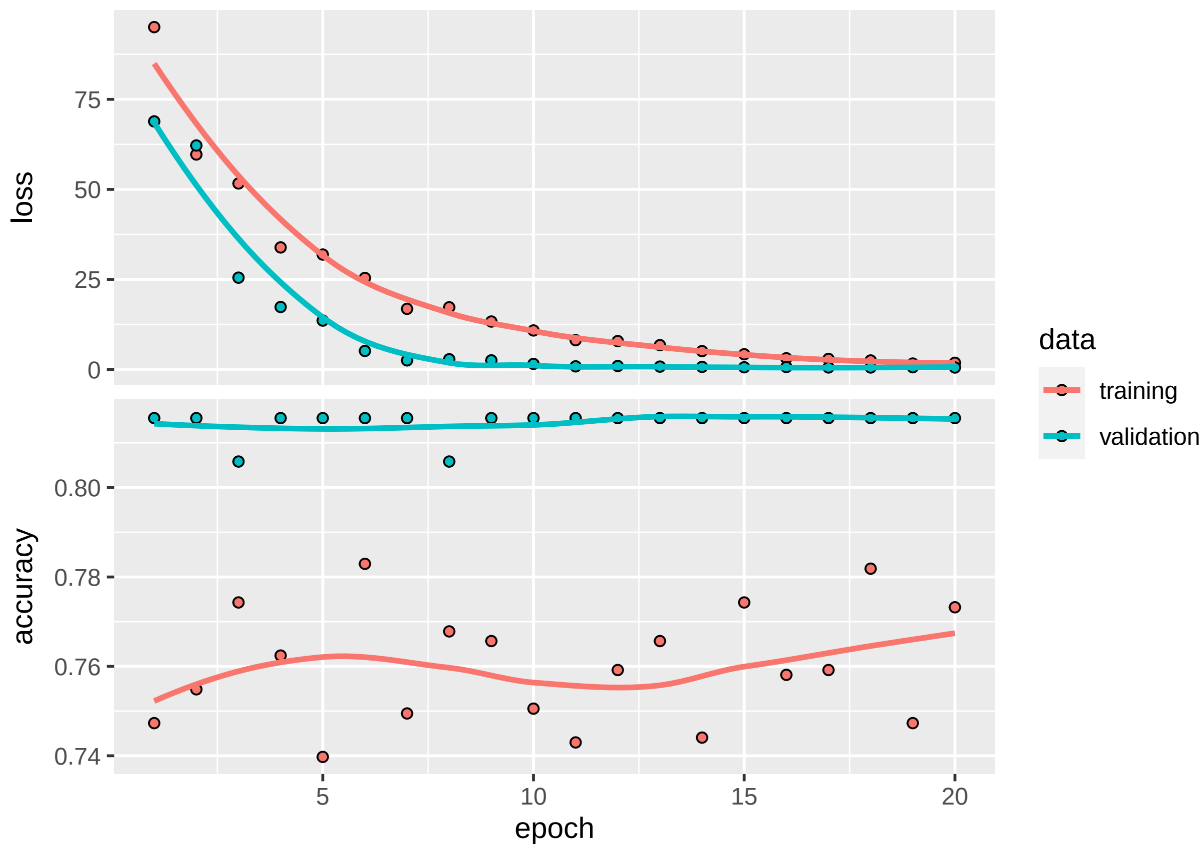
(d) Fit a DNN using all covariates and report the classification accuracy

```
modelnn <- keras_model_sequential() %>%
  layer_dense(input_shape = ncol(x),units = 128,activation = 'relu') %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128,activation = 'relu') %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 1,activation = 'sigmoid')

modelnn %>% compile(
  loss='binary_crossentropy',optimizer= optimizer_adam(),
  metrics = 'accuracy'  # Corrected typo
)

history <- modelnn %>% fit(
  x[train_id, ], y[train_id],
  batch_size = 32,
  epochs = 20,
  validation_split = 0.1
)
```

```
## Epoch 1/20
## 29/29 - 1s - loss: 95.0450 - accuracy: 0.7473 - val_loss: 68.8634 - val_accuracy: 0.8155 - 
## Epoch 2/20
## 29/29 - 0s - loss: 59.6742 - accuracy: 0.7549 - val_loss: 62.1736 - val_accuracy: 0.8155 - 
## Epoch 3/20
## 29/29 - 0s - loss: 51.6293 - accuracy: 0.7743 - val_loss: 25.4754 - val_accuracy: 0.8058 - 
```

```
## Epoch 4/20
## 29/29 - 0s - loss: 33.8682 - accuracy: 0.7624 - val_loss: 17.3309 - val_accuracy: 0.8155 - 
## Epoch 5/20
## 29/29 - 0s - loss: 31.8975 - accuracy: 0.7397 - val_loss: 13.5636 - val_accuracy: 0.8155 - 
## Epoch 6/20
## 29/29 - 0s - loss: 25.3980 - accuracy: 0.7829 - val_loss: 5.1307 - val_accuracy: 0.8155 - 1
## Epoch 7/20
## 29/29 - 0s - loss: 16.8203 - accuracy: 0.7495 - val_loss: 2.5158 - val_accuracy: 0.8155 - 1
## Epoch 8/20
## 29/29 - 0s - loss: 17.2522 - accuracy: 0.7678 - val_loss: 2.8273 - val_accuracy: 0.8058 - 1
## Epoch 9/20
## 29/29 - 0s - loss: 13.3012 - accuracy: 0.7657 - val_loss: 2.5140 - val_accuracy: 0.8155 - 1
## Epoch 10/20
## 29/29 - 0s - loss: 10.8173 - accuracy: 0.7505 - val_loss: 1.5358 - val_accuracy: 0.8155 - 1
## Epoch 11/20
## 29/29 - 0s - loss: 8.1198 - accuracy: 0.7430 - val_loss: 0.8541 - val_accuracy: 0.8155 - 12
## Epoch 12/20
## 29/29 - 0s - loss: 7.8723 - accuracy: 0.7592 - val_loss: 0.9452 - val_accuracy: 0.8155 - 12
## Epoch 13/20
## 29/29 - 0s - loss: 6.7446 - accuracy: 0.7657 - val_loss: 0.7863 - val_accuracy: 0.8155 - 13
## Epoch 14/20
## 29/29 - 0s - loss: 5.0829 - accuracy: 0.7441 - val_loss: 0.7056 - val_accuracy: 0.8155 - 13
## Epoch 15/20
## 29/29 - 0s - loss: 4.2133 - accuracy: 0.7743 - val_loss: 0.5931 - val_accuracy: 0.8155 - 15
## Epoch 16/20
## 29/29 - 0s - loss: 3.0909 - accuracy: 0.7581 - val_loss: 0.6687 - val_accuracy: 0.8155 - 13
## Epoch 17/20
## 29/29 - 0s - loss: 2.9126 - accuracy: 0.7592 - val_loss: 0.5477 - val_accuracy: 0.8155 - 13
## Epoch 18/20
## 29/29 - 0s - loss: 2.4869 - accuracy: 0.7819 - val_loss: 0.5418 - val_accuracy: 0.8155 - 12
## Epoch 19/20
## 29/29 - 0s - loss: 1.6305 - accuracy: 0.7473 - val_loss: 0.6096 - val_accuracy: 0.8155 - 13
## Epoch 20/20
## 29/29 - 0s - loss: 1.8446 - accuracy: 0.7732 - val_loss: 0.5331 - val_accuracy: 0.8155 - 13
```

```r
plot(history)
```




```r
accuracy(modelnn %>% predict(x[test_id,])>0.5*1,y[test_id])
```

```
## 14/14 - 0s - 118ms/epoch - 8ms/step
```

```
## [1] 0.8412698
```

3. Try the MNIST data set analyzed in class with a four layer DNN (you can experiment and decide the sizes and dropout rates), and report the classification accuracy.

```r
mnist <- dataset_mnist()
x_train <- mnist$train$x
g_train <- mnist$train$y
x_test <- mnist$test$x
g_test <- mnist$test$y
```

```r
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)
```

```r
x_train <- x_train / 255
x_test <- x_test / 255
```

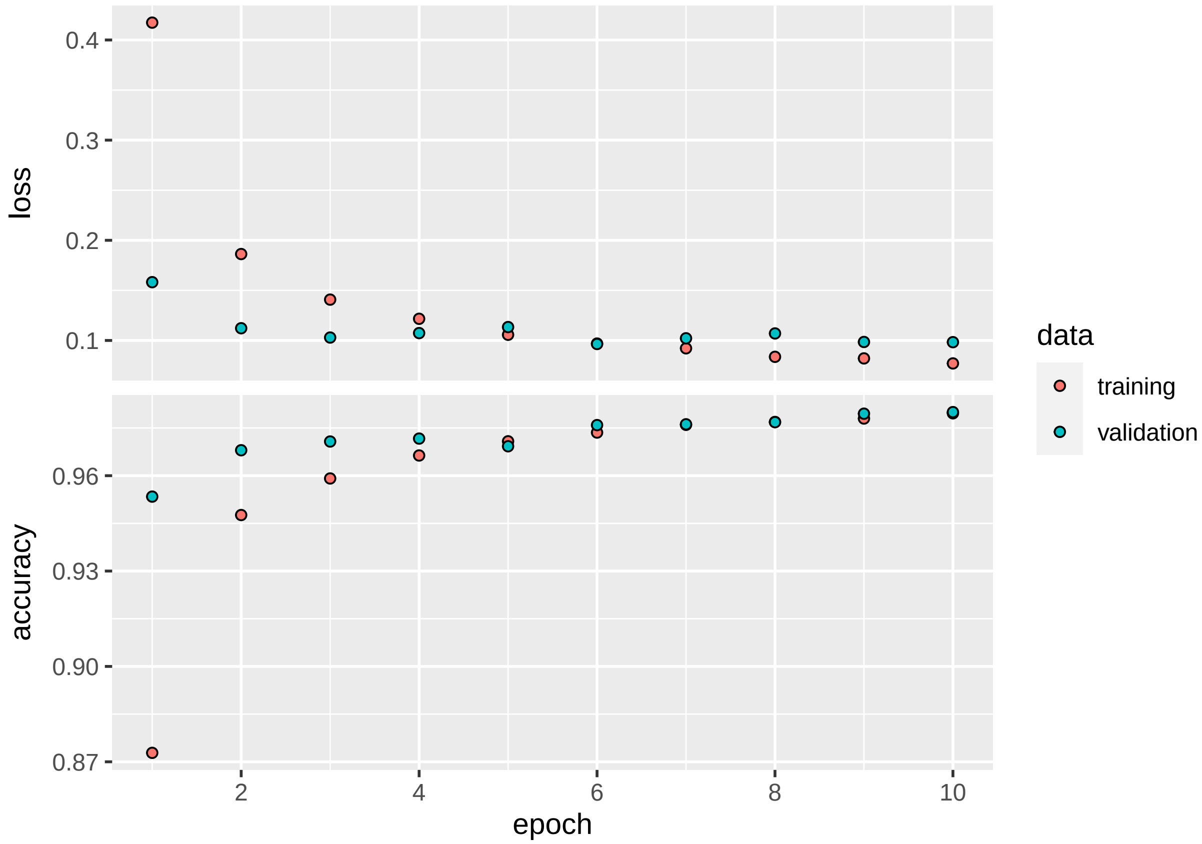
```r
modelnn <- keras_model_sequential()
modelnn %>%
```

```r
  layer_dense(units = 512, activation = "relu",input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = "softmax")

modelnn %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy")
  )

system.time(
  history <- modelnn %>%
     fit(x_train, y_train, epochs = 10, batch_size = 128,
       validation_split = 0.2)
 )
```

```
## Epoch 1/10
## 375/375 - 4s - loss: 0.4173 - accuracy: 0.8728 - val_loss: 0.1582 - val_accuracy: 0.9534 - 
## Epoch 2/10
## 375/375 - 3s - loss: 0.1863 - accuracy: 0.9476 - val_loss: 0.1122 - val_accuracy: 0.9680 - 
## Epoch 3/10
## 375/375 - 3s - loss: 0.1408 - accuracy: 0.9591 - val_loss: 0.1029 - val_accuracy: 0.9707 - 
## Epoch 4/10
## 375/375 - 4s - loss: 0.1216 - accuracy: 0.9664 - val_loss: 0.1074 - val_accuracy: 0.9717 - 
## Epoch 5/10
## 375/375 - 4s - loss: 0.1057 - accuracy: 0.9708 - val_loss: 0.1134 - val_accuracy: 0.9693 - 
## Epoch 6/10
## 375/375 - 4s - loss: 0.0969 - accuracy: 0.9736 - val_loss: 0.0965 - val_accuracy: 0.9759 - 
## Epoch 7/10
## 375/375 - 4s - loss: 0.0922 - accuracy: 0.9760 - val_loss: 0.1022 - val_accuracy: 0.9762 - 
## Epoch 8/10
## 375/375 - 4s - loss: 0.0837 - accuracy: 0.9769 - val_loss: 0.1070 - val_accuracy: 0.9768 - 
## Epoch 9/10
## 375/375 - 4s - loss: 0.0821 - accuracy: 0.9780 - val_loss: 0.0985 - val_accuracy: 0.9795 - 
## Epoch 10/10
## 375/375 - 4s - loss: 0.0771 - accuracy: 0.9796 - val_loss: 0.0983 - val_accuracy: 0.9800 - 
```

```
##    user  system elapsed
##  185.14  289.99   37.21
```

```r
plot(history, smooth = FALSE)
```




```r
accuracy <- function(pred, truth) {
   mean(drop(as.numeric(pred)) == drop(truth)) }
modelnn %>% predict(x_test) %>% k_argmax() %>% accuracy(g_test)
```

```
## 313/313 - 1s - 1s/epoch - 4ms/step
```

```
## [1] 0.9808
```

```r
head(g_test)
```

```
## [1] 7 2 1 0 4 1
```