Homework 4

Brandon Amaral, Monte Davityan, Nicholas Lombardo, Hongkai Lu

2022-10-03

1 Murphy Chapter 13 & ISLR Chapter 10 Summaries

Section 10.6 When to Use Deep Learning

The text shows three way to determine when to use deep learning: 1: Occam's razor principle: when faced with several methods that give roughly equivalent performance, pick the simplest. 2: According to the text, "if we can produce models with the simpler tools that perform as well, they are likely to be easier to fit and understand, and potentially less fragile than the more complex approaches." 3: When the sample size of the data is extremely large and the interpretability of the model is not our main priority, we should use Deep Learning to be our first choice.

Section 10.7 Fitting a Neural Network

When fitting a neural netwrok model, the text provide two general strategies: 1. Slow Learning 2. Regularization

10.7.1 Backpropagation Backpropagation is a process that the act of differentiation assigns a fraction of the residual to each of the parameters via the chain rule.

10.7.2 Regularization and Stochastic Gradient Descent Stochastic Gradient Descent is a process that we sample a small fraction of n observations' summation of the derivatives of $R_i(\theta)$ with respect to β_k and w_kj each time we compute a gradient step if n is large. In this process, Regularization is essential to avoid overfitting.

10.7.3 Dropout Learning Dropout learning is a way to randomly remove a fraction ϕ of the units in a layer when fitting the model.

10.7.4 Network Tuning There are three ways provided for network tuning in the text, which are "The number of hidden layers, and the number of units per layer", "The number of hidden layers, and the number of units per layer", and "The number of hidden layers, and the number of units per layer".

Section 10.8 Interpolation and Double Descent

Double descent is a phenomenon that the test error has a U-shape before the interpolation threshold is reached, then it descends again as an increasingly flexible model is fit.That may applied to deep learning as well. However, even though double descent can occasionally occur in neural networks model, we do not rely on this behavior typically; meanwhile, it is important to know that the bias-variance trade-off always holds.

2 Reproduction of ISLR Lab 10.9

10.9.1 A Single Layer Network on the Hitters Data

First, We load the Hitters data and split it to training and test set.

```
Gitters <- na.omit(Hitters)
n <- nrow(Gitters)
set.seed(13)
ntest <- trunc(n/3)
testid <- sample(1:n, ntest)
```

Then, we fit a linear model and obtain the mean absolute prediction error, which is 254.6687.

```
lfit <- lm(Salary ~., data=Gitters[-testid,])
lpred <- predict(lfit, Gitters[testid,])
with(Gitters[testid,], mean(abs(lpred - Salary)))
```

```
## [1] 254.6687
```

Next, we fit a lasso model. Similarly, we calculate the absolute prediction error of the lasso model, which is slightly less than the result of linear model.

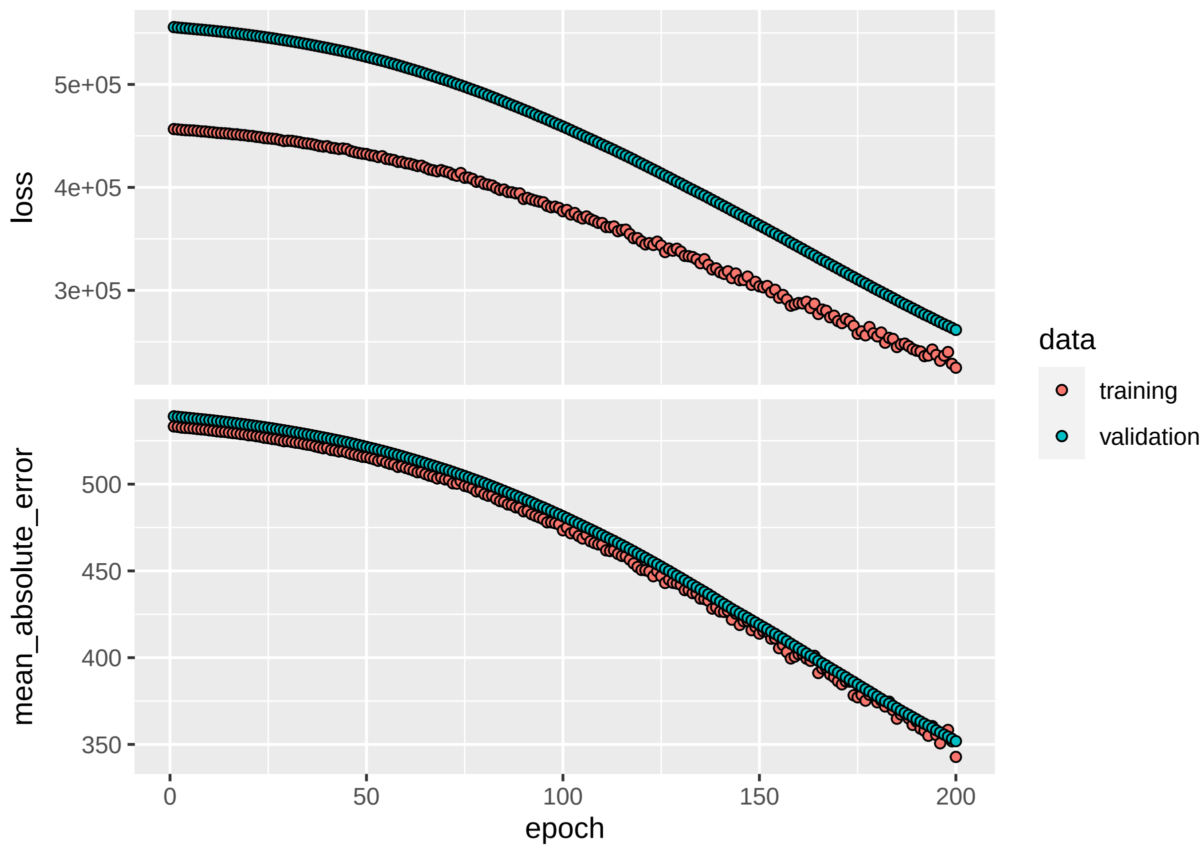
```
library(glmnet)
x <- scale(model.matrix(Salary ~. -1, data=Gitters))
y <- Gitters$Salary

cvfit <- cv.glmnet(x[-testid, ], y[-testid], type.measure='mae')
cpred <- predict(cvfit, x[testid,], s='lambda.min')
mean(abs(y[testid] - cpred))
```

```
## [1] 252.2994
```

Last, we fit the neural network with library keras. As the text written, it is a challenge to get keras running on computer. The error I got is "CondaSSLError: Encountered an SSL error. Most likely a certificate verification issue." Still figure out a way to get this installing.

```
modnn <- keras_model_sequential() %>%
  layer_dense(units=50, activation='relu', input_shape = ncol(x)) %>%
  layer_dropout(rate=0.4) %>%
  layer_dense(units=1)
```

```
modnn %>% compile(loss='mse', optimizer=optimizer_rmsprop(), metrics=list('mean_absolute_error'))
history <- modnn %>% fit(
  x[-testid, ], y[-testid], epochs=200, batch_size=32,
  validation_data=list(x[testid,], y[testid])
)
plot(history, smooth = FALSE)
```




```
npred <- predict(modnn, x[testid,])
mean(abs(y[testid] - npred))
```

```
## [1] 351.8675
```

10.9.2 A Multilayer Network on the MNIST Digit Data

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

```
## [1] 60000    28    28
```

```
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)
```

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

```
modelnn <- keras_model_sequential()
modelnn %>%
  layer_dense(units=256, activation="relu", input_shape=c(784)) %>%
```

```r
  layer_dropout(rate=0.4) %>%
  layer_dense(units=128, activation='relu') %>%
  layer_dropout(rate=0.3) %>%
  layer_dense(units=10, activation='softmax')

#summary(modelnn)
```

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

```r
system.time(
  history <- modelnn %>%
    fit(x_train, y_train, epochs=30, batch_size=128, validation_split=0.2)
)
```

```
##    user  system elapsed
## 109.581   7.724  48.001
```

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

```
## [1] 0.9814
```

```r
modellr <- keras_model_sequential() %>%
   layer_dense(input_shape = 784, units = 10,
       activation = "softmax")
#summary(modellr)
```

```r
modellr %>% compile(loss = "categorical_crossentropy",
     optimizer = optimizer_rmsprop(), metrics = c("accuracy"))
modellr %>% fit(x_train, y_train, epochs = 30,
      batch_size = 128, validation_split = 0.2)
modellr %>% predict(x_test) %>% k_argmax() %>% accuracy(g_test)
```

```
## [1] 0.9285
```

10.9.3 Convolutional Neural Networks

```r
cifar100 <- dataset_cifar100()
names(cifar100)
```

```
## [1] "train" "test"
```

```r
x_train <- cifar100$train$x
g_train <- cifar100$train$y
x_test <- cifar100$test$x
g_test <- cifar100$test$y
dim(x_train)
```

```
## [1] 50000    32    32     3
```

```r
range(x_train[1,,, 1])
```

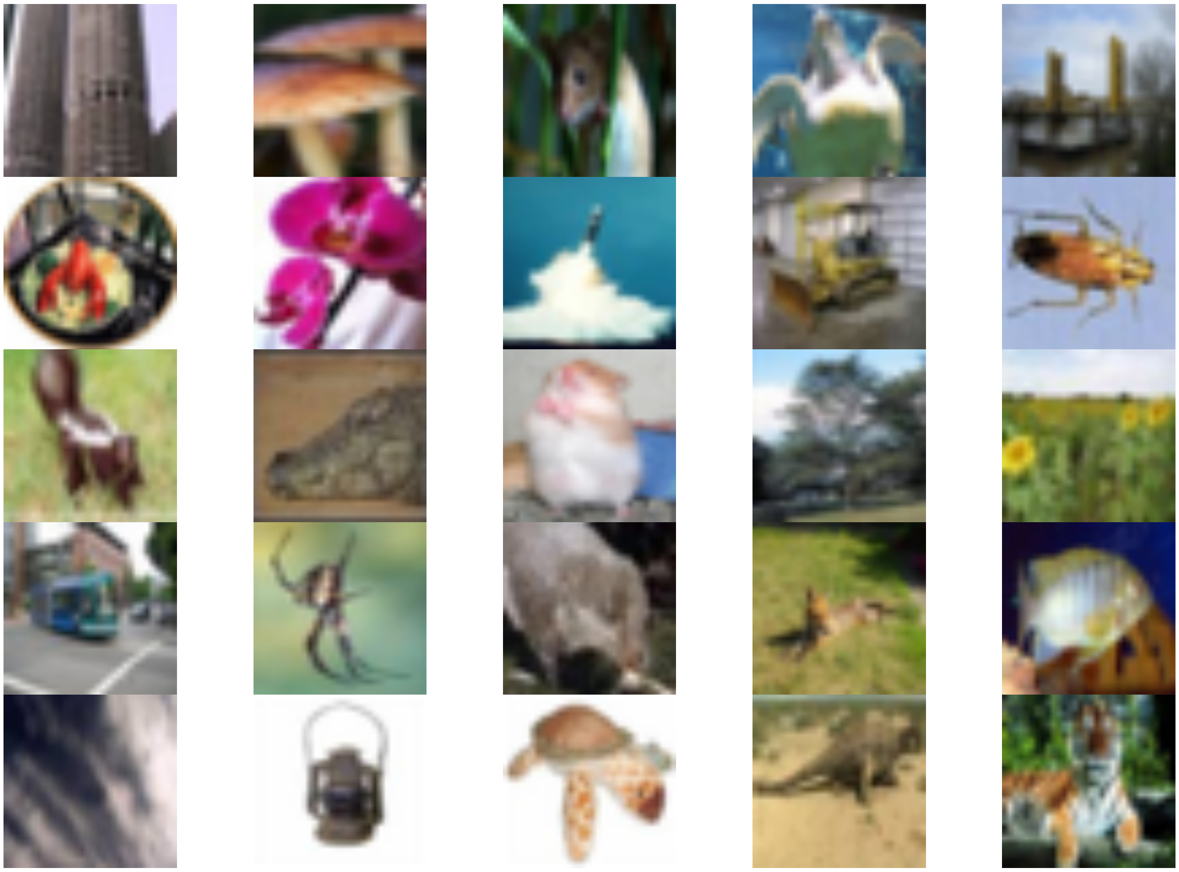
```
## [1]  13 255
```

```r
x_train <- x_train / 255
x_test <- x_test / 255
y_train <- to_categorical(g_train, 100)
```

```r
dim(y_train)
```

```
## [1] 50000   100
```

```r
library(jpeg)
par(mar = c(0, 0, 0, 0), mfrow = c(5, 5))
index <- sample(seq(50000), 25)
for (i in index) plot(as.raster(x_train[i,,, ]))
```



```r
model <- keras_model_sequential() %>%
   layer_conv_2d(filters = 32, kernel_size = c(3, 3),
      padding = "same", activation = "relu",
      input_shape = c(32, 32, 3)) %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_conv_2d(filters = 64, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_conv_2d(filters = 128, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_conv_2d(filters = 256, kernel_size = c(3, 3),
      padding = "same", activation = "relu") %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_flatten() %>%
   layer_dropout(rate = 0.5) %>%
   layer_dense(units = 512, activation = "relu") %>%
   layer_dense(units = 100, activation = "softmax")
```

```r
#summary(model)
```

```r
model %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy"))
#history <- model %>% fit(x_train, y_train, epochs = 30,
history <- model %>% fit(x_train, y_train, epochs = 10,
    batch_size = 128, validation_split = 0.2)
model %>% predict(x_test) %>% k_argmax() %>% accuracy(g_test)
```

```
## [1] 0.4058
```

10.9.4 Using Pretrained CNN Models

```r
img_dir <- "book_images"
image_names <- list.files(img_dir)
num_images <- length(image_names)
x <- array(dim = c(num_images, 224, 224, 3))
for (i in 1:num_images) {
   img_path <- paste(img_dir, image_names[i], sep = "/")
   img <- image_load(img_path, target_size = c(224, 224))
   x[i,,, ] <- image_to_array(img)
}
x <- imagenet_preprocess_input(x)
```

```r
model <- application_resnet50(weights = "imagenet")
#summary(model)
```

```r
pred6 <- model %>% predict(x) %>%
   imagenet_decode_predictions(top = 3)
names(pred6) <- image_names
print(pred6)
```

```
## $flamingo.jpg
##   class_name class_description       score
## 1  n02007558          flamingo 0.926349819
## 2  n02006656         spoonbill 0.071699291
## 3  n02002556       white_stork 0.001228209
##
## $hawk_cropped.jpeg
##   class_name class_description      score
## 1  n01608432              kite 0.72270876
## 2  n01622779    great_grey_owl 0.08182590
## 3  n01532829       house_finch 0.04218867
##
## $hawk.jpg
##   class_name class_description     score
## 1  n03388043          fountain 0.2788654
## 2  n03532672              hook 0.1785550
## 3  n03804744              nail 0.1080727
##
## $huey.jpg
##   class_name           class_description      score
## 1  n02097474             Tibetan_terrier 0.50929713
## 2  n02098413                       Lhasa 0.42209816
```

```
## 3  n02098105 soft-coated_wheaten_terrier 0.01695861
##
## $kitty.jpg
##   class_name    class_description      score
## 1  n02105641 Old_English_sheepdog 0.83265942
## 2  n02086240             Shih-Tzu 0.04513892
## 3  n03223299              doormat 0.03299790
##
## $weaver.jpg
##   class_name class_description      score
## 1  n01843065           jacamar 0.49795377
## 2  n01818515             macaw 0.22193290
## 3  n02494079   squirrel_monkey 0.04287881
```

10.9.5 IMDb Document Classification

```
max_features <- 10000
imdb <- dataset_imdb(num_words = max_features)
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb
```

```
x_train[[1]][1:12]
```

```
##  [1]    1   14   22   16   43  530  973 1622 1385   65  458 4468
```

```
word_index <- dataset_imdb_word_index()
decode_review <- function(text, word_index) {
   word <- names(word_index)
   idx <- unlist(word_index, use.names = FALSE)
   word <- c("<PAD>", "<START>", "<UNK>", "<UNUSED>", word)
   idx <- c(0:3, idx + 3)
   words <- word[match(text, idx, 2)]
   paste(words, collapse = " ")
}
decode_review(x_train[[1]][1:12], word_index)
```

```
## [1] "<START> this film was just brilliant casting location scenery story direction everyone's"
```

```
library(Matrix)
one_hot <- function(sequences, dimension) {
   seqlen <- sapply(sequences, length)
   n <- length(seqlen)
   rowind <- rep(1:n, seqlen)
   colind <- unlist(sequences)
   sparseMatrix(i = rowind, j = colind,
      dims = c(n, dimension))
}
```

```
x_train_1h <- one_hot(x_train, 10000)
x_test_1h <- one_hot(x_test, 10000)
dim(x_train_1h)
```

```
## [1] 25000 10000
```

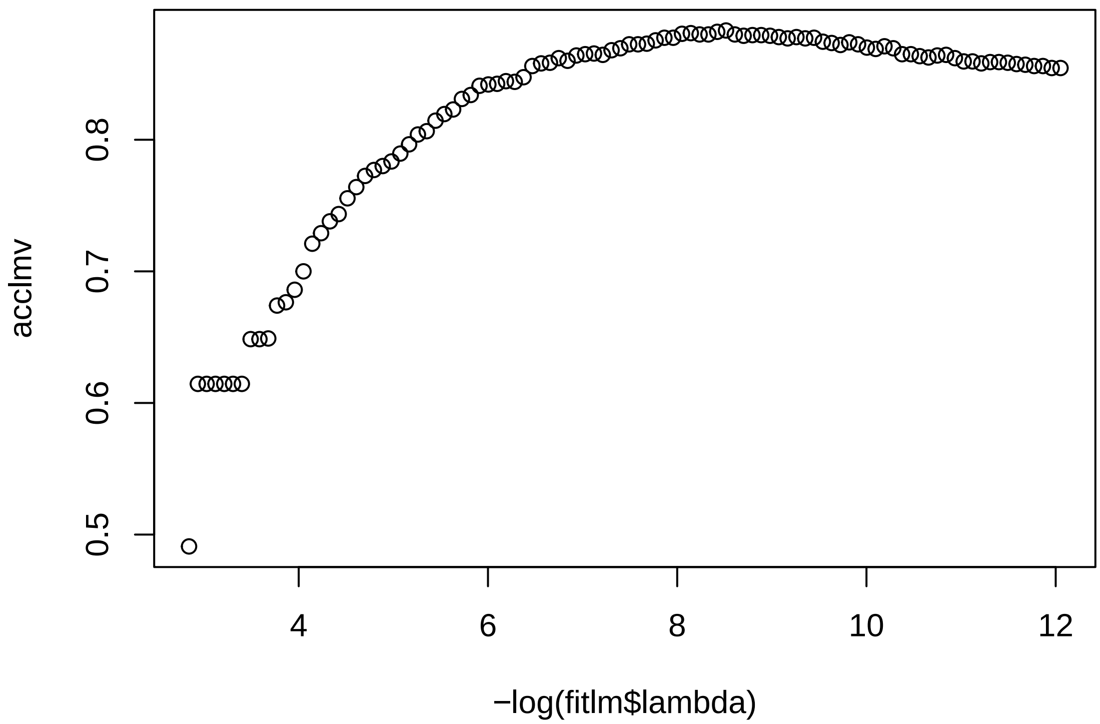
```
nnzero(x_train_1h) / (25000 * 10000)
```

```
## [1] 0.01316987
```

```r
set.seed(3)
ival <- sample(seq(along = y_train), 2000)
```

```r
library(glmnet)
fitlm <- glmnet(x_train_1h[-ival, ], y_train[-ival],
    family = "binomial", standardize = FALSE)
classlmv <- predict(fitlm, x_train_1h[ival, ]) > 0
acclmv <- apply(classlmv, 2, accuracy,  y_train[ival] > 0)
```

```r
par(mar = c(4, 4, 4, 4), mfrow = c(1, 1))
plot(-log(fitlm$lambda), acclmv)
```




```r
model <- keras_model_sequential() %>%
   layer_dense(units = 16, activation = "relu",
      input_shape = c(10000)) %>%
   layer_dense(units = 16, activation = "relu") %>%
   layer_dense(units = 1, activation = "sigmoid")
model %>% compile(optimizer = "rmsprop",
   loss = "binary_crossentropy", metrics = c("accuracy"))
history <- model %>% fit(x_train_1h[-ival, ], y_train[-ival],
   epochs = 20, batch_size = 512,
   validation_data = list(x_train_1h[ival, ], y_train[ival]))
```

```r
history <- model %>% fit(
    x_train_1h[-ival, ], y_train[-ival], epochs = 20,
    batch_size = 512, validation_data = list(x_test_1h, y_test)
    )
```

10.9.6 Recurrent Neural Networks

```
wc <- sapply(x_train, length)
median(wc)
```

Sequential Models for Document Classification

```
## [1] 178
```

```
sum(wc <= 500) / length(wc)
```

```
## [1] 0.91568
```

```
maxlen <- 500
x_train <- pad_sequences(x_train, maxlen = maxlen)
x_test <- pad_sequences(x_test, maxlen = maxlen)
dim(x_train)
```

```
## [1] 25000   500
```

```
dim(x_test)
```

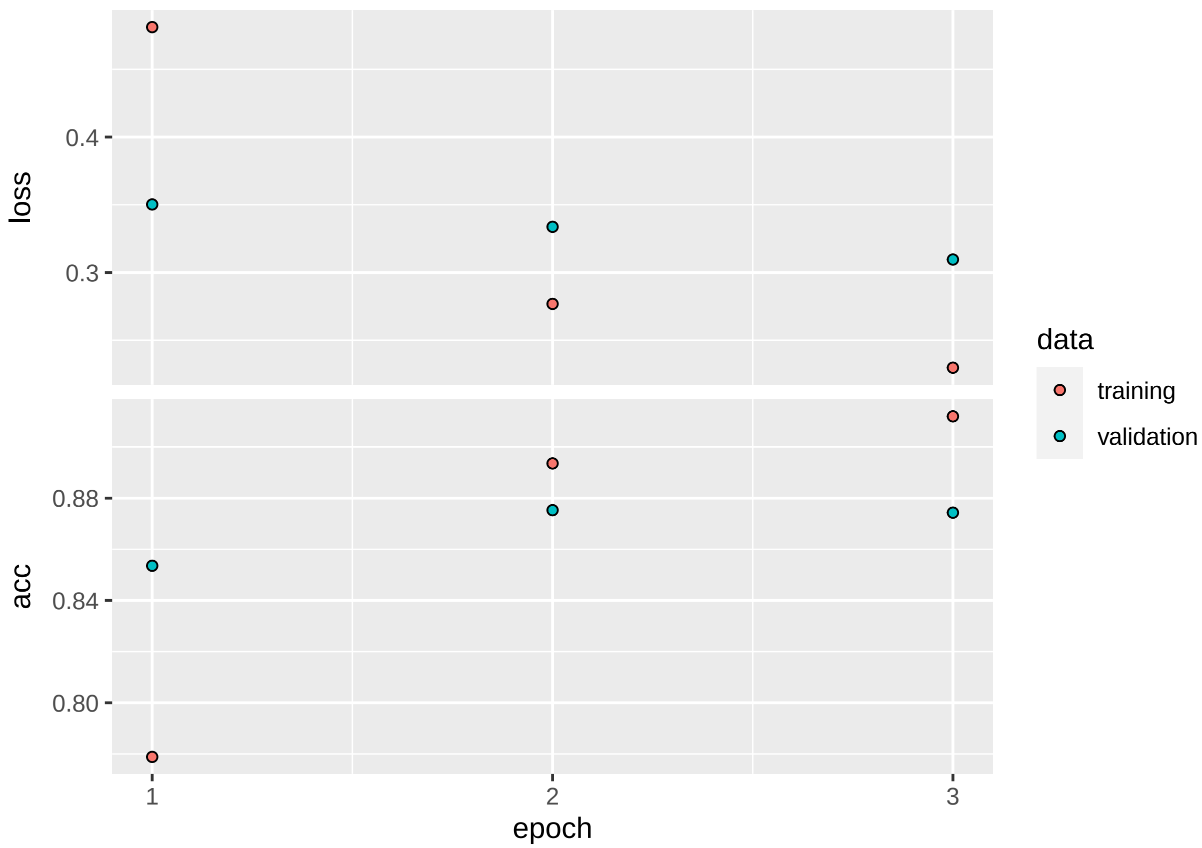
```
## [1] 25000   500
```

```
x_train[1, 490:500]
```

```
##  [1]   16 4472  113  103   32   15   16 5345   19  178   32
```

```
model <- keras_model_sequential() %>%
   layer_embedding(input_dim = 10000, output_dim = 32) %>%
   layer_lstm(units = 32) %>%
   layer_dense(units = 1, activation = "sigmoid")
```

```
model %>% compile(optimizer = "rmsprop",
    loss = "binary_crossentropy", metrics = c("acc"))
#history <- model %>% fit(x_train, y_train, epochs = 10,
history <- model %>% fit(x_train, y_train, epochs = 3,
    batch_size = 128, validation_data = list(x_test, y_test))
plot(history, smooth = FALSE)
```




```
xdata <- data.matrix(
 NYSE[, c("DJ_return", "log_volume","log_volatility")]
 )
istrain <- NYSE[, "train"]
xdata <- scale(xdata)
```

```
lagm <- function(x, k = 1) {
   n <- nrow(x)
   pad <- matrix(NA, k, ncol(x))
   rbind(pad, x[1:(n - k), ])
}
```

```
arframe <- data.frame(log_volume = xdata[, "log_volume"],
   L1 = lagm(xdata, 1), L2 = lagm(xdata, 2),
   L3 = lagm(xdata, 3), L4 = lagm(xdata, 4),
   L5 = lagm(xdata, 5)
 )
```

```
arframe <- arframe[-(1:5), ]
istrain <- istrain[-(1:5)]
```

```
arfit <- lm(log_volume ~ ., data = arframe[istrain, ])
```

```
arpred <- predict(arfit, arframe[!istrain, ])
V0 <- var(arframe[!istrain, "log_volume"])
1 - mean((arpred - arframe[!istrain, "log_volume"])^2) / V0
```

Time Series Prediction

```
## [1] 0.413223
```

```
arframed <-
    data.frame(day = NYSE[-(1:5), "day_of_week"], arframe)
arfitd <- lm(log_volume ~ ., data = arframed[istrain, ])
arpredd <- predict(arfitd, arframed[!istrain, ])
1 - mean((arpredd - arframe[!istrain, "log_volume"])^2) / V0
```

```
## [1] 0.4598616
```

```
n <- nrow(arframe)
xrnn <- data.matrix(arframe[, -1])
xrnn <- array(xrnn, c(n, 3, 5))
xrnn <- xrnn[,, 5:1]
xrnn <- aperm(xrnn, c(1, 3, 2))
dim(xrnn)
```

```
## [1] 6046    5    3
```

```
model <- keras_model_sequential() %>%
   layer_simple_rnn(units = 12,
      input_shape = list(5, 3),
      dropout = 0.1, recurrent_dropout = 0.1) %>%
   layer_dense(units = 1)
model %>% compile(optimizer = optimizer_rmsprop(),
    loss = "mse")
```

```
history <- model %>% fit(
    xrnn[istrain,, ], arframe[istrain, "log_volume"],
#    batch_size = 64, epochs = 200,
    batch_size = 64, epochs = 75,
    validation_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )
kpred <- predict(model, xrnn[!istrain,, ])
1 - mean((kpred - arframe[!istrain, "log_volume"])^2) / V0
```

```
## [1] 0.407035
```

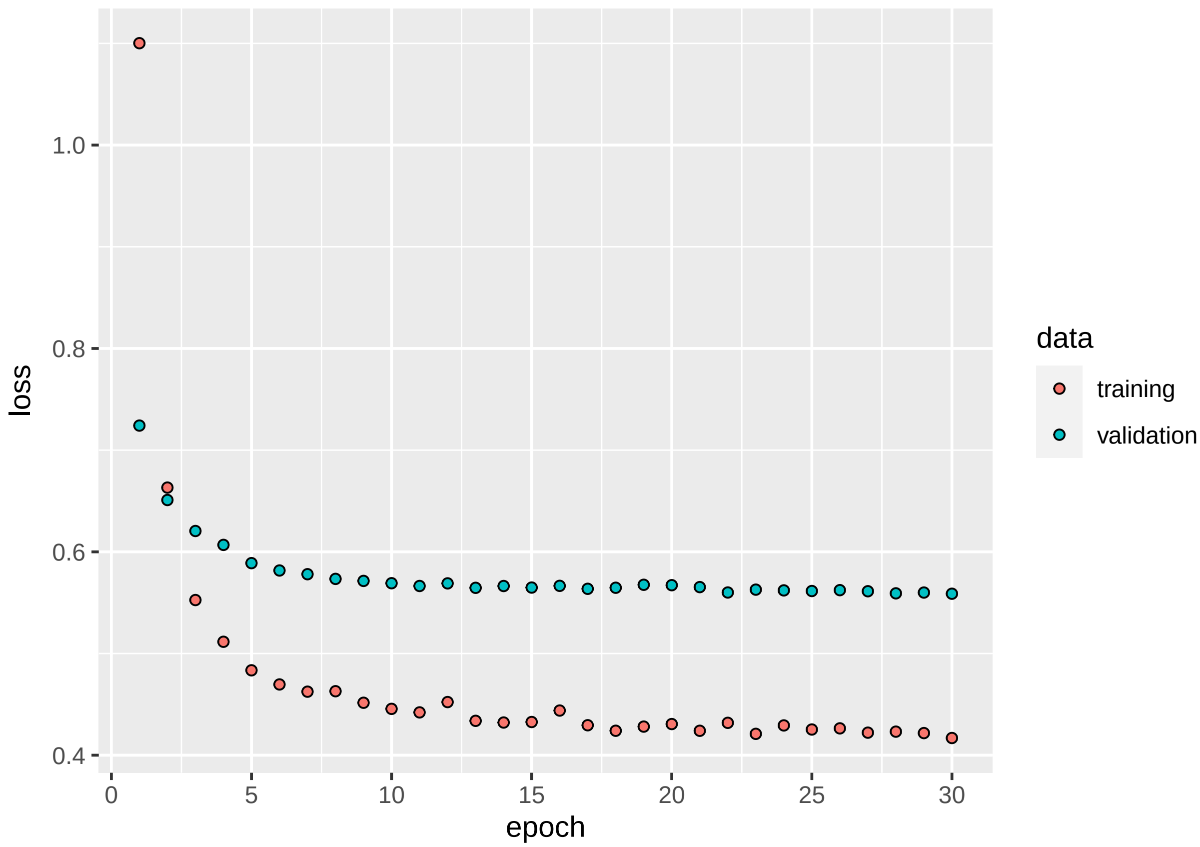
```
model <- keras_model_sequential() %>%
   layer_flatten(input_shape = c(5, 3)) %>%
   layer_dense(units = 1)
```

```
x <- model.matrix(log_volume ~ . - 1, data = arframed)
colnames(x)
```

```
##  [1] "dayfri"          "daymon"          "daythur"
##  [4] "daytues"         "daywed"          "L1.DJ_return"
##  [7] "L1.log_volume"   "L1.log_volatility" "L2.DJ_return"
## [10] "L2.log_volume"   "L2.log_volatility" "L3.DJ_return"
## [13] "L3.log_volume"   "L3.log_volatility" "L4.DJ_return"
## [16] "L4.log_volume"   "L4.log_volatility" "L5.DJ_return"
```

```
## [19] "L5.log_volume"     "L5.log_volatility"
```

```
arnnd <- keras_model_sequential() %>%
   layer_dense(units = 32, activation = 'relu',
      input_shape = ncol(x)) %>%
   layer_dropout(rate = 0.5) %>%
   layer_dense(units = 1)
arnnd %>% compile(loss = "mse",
    optimizer = optimizer_rmsprop())
history <- arnnd %>% fit(
#    x[istrain, ], arframe[istrain, "log_volume"], epochs = 100,
    x[istrain, ], arframe[istrain, "log_volume"], epochs = 30,
    batch_size = 32, validation_data =
      list(x[!istrain, ], arframe[!istrain, "log_volume"])
  )
plot(history, smooth = FALSE)
```




```
npred <- predict(arnnd, x[!istrain, ])
1 - mean((arframe[!istrain, "log_volume"] - npred)^2) / V0
```

```
## [1] 0.4698843
```

3 ISLR Problems

10.7

We fit a neural network with a single hidden layer and 10 units to the `Default` data, and compare its performance to a logistic regression model. First, we scale our data in a model matrix, excluding the usual first column of 1's for an intercept term, and create a separate response vector of 1's and 0's.

```
x <- model.matrix(default ~ . -1, data = Default) |> scale()
y <- ifelse(Default$default == "Yes",1,0)
```

Next, we partition our data into training and test datasets, training on a random set of 80% of the data, and testing on the remaining 20%.

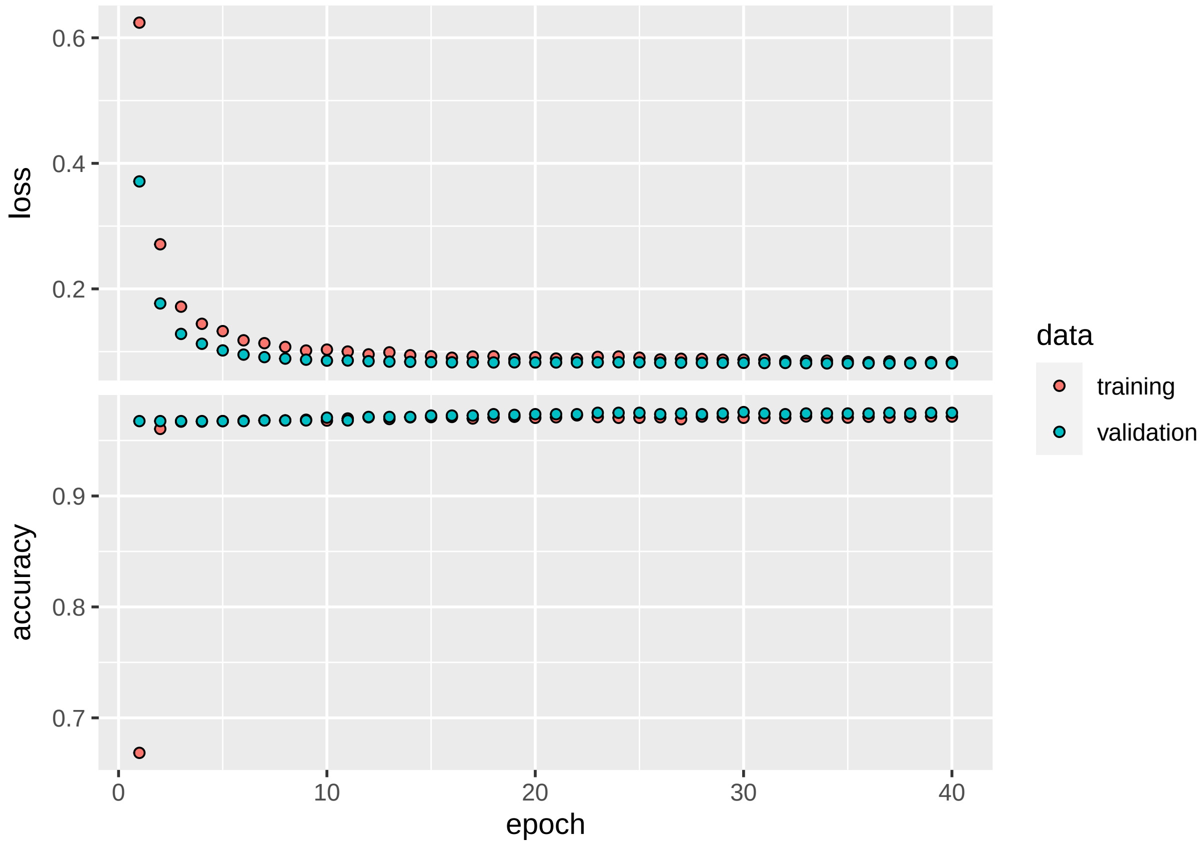
```
n <- nrow(x)
test_ind <- sample(1:n, n/5)

x_train <- x[-test_ind,]
x_test <- x[test_ind,]
y_train <- y[-test_ind]
y_test <- y[test_ind]
```

Now, we build the model architecture, defining a single layer model with 10 units in the hidden layer, with the usual ReLU activation function and a dropout rate of 20% for regularization. Since we are predicting a binary response, we use the sigmoid activation function in the output layer. Then, we compile the model, using the binary cross entropy loss function as a measure of fit.

```
nnmodel <- keras_model_sequential() |>
  layer_dense(units = 10, activation = "relu", input_shape = ncol(x)) |>
  layer_dropout(rate = 0.20) |>
  layer_dense(units = 1, activation = "sigmoid")

nnmodel |> compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = "accuracy")
```

We fit the model using a batch size of 30 with 40 epochs. We also further split the data into a third validation set, again with 20%, so we fit the model on 6400 training observations. Then, each epoch is $6400/30 \approx 214$ stochastic gradient descent steps. We plot the loss of the model alongside its

```
history <- nnmodel |>
  fit(x_train, y_train, epochs = 40, batch_size = 30, validation_split = 0.2)

plot(history, smooth = FALSE)
```



Now, we make our predictions on the test set and find the model's misclassification rate.

```
y_test_pred <- nnmodel |> predict(x_test)
NN_misclass_rate <- 1/length(y_test)*sum(y_test != round(y_test_pred))
```

Now, we compare this with a logistic regression model.

```
log_model <- glm(y_train ~ x_train, family = binomial(link = logit))
y_log_test_pred <- round(predict(log_model, newdata = data.frame(x_test), type = "response"))
logistic_misclass_rate <- 1/length(y_test)*sum(y_test != round(y_log_test_pred))
```

The table shows the misclassification rates for both models

```
misclass <- data.frame(Logistic = logistic_misclass_rate, NeuralNet = NN_misclass_rate)
knitr::kable(misclass)
```

Logistic	NeuralNet
0.1915	0.029

In the table, we see that the logistic regression model performed considerably worse, with a misclassification rate of about 19.15% versus the neural net's misclassification rate of about 2.9%.

10.8

```
img_dir <- "MyAnimalPics"
image_names <- list.files (img_dir)
num_images <- length (image_names)
```

```r
x <- array ( dim = c(num_images , 224, 224, 3))
for (i in 1:num_images) {
  img_path <- paste (img_dir , image_names[i], sep = "/")
  img <- image_load (img_path, target_size = c(224, 224))
  x[i,,, ] <- image_to_array(img)
}
x <- imagenet_preprocess_input (x)
```

```r
model <- application_resnet50(weights = "imagenet")
#summary (model)
```

```r
pred6 <- model %>% predict (x) %>%
  imagenet_decode_predictions (top = 5)
names (pred6) <- image_names
print (pred6)
```

```
## $Bear.jpg
##   class_name class_description      score
## 1  n02363005            beaver 0.63836390
## 2  n02077923          sea_lion 0.15228654
## 3  n02504458  African_elephant 0.04073466
## 4  n02444819             otter 0.02228249
## 5  n02396427         wild_boar 0.01866793
##
## $Cat.jpg
##   class_name class_description      score
## 1  n03786901            mortar 0.36851746
## 2  n03794056         mousetrap 0.20677969
## 3  n04399382             teddy 0.05336932
## 4  n02328150            Angora 0.04163421
## 5  n02950826            cannon 0.03490378
##
## $Coyote.jpg
##   class_name class_description       score
## 1  n04275548        spider_web 0.738270223
## 2  n02325366       wood_rabbit 0.097948611
## 3  n02494079   squirrel_monkey 0.020395182
## 4  n02457408  three-toed_sloth 0.018195676
## 5  n02486410            baboon 0.008232873
##
## $Dog.jpg
##   class_name  class_description       score
## 1  n02106662    German_shepherd 0.846454084
## 2  n02105412             kelpie 0.093938135
## 3  n02105162           malinois 0.021421773
## 4  n02106550         Rottweiler 0.007880141
## 5  n02099712 Labrador_retriever 0.005973050
##
## $Fish.jpg
##   class_name class_description      score
## 1  n03908714  pencil_sharpener 0.30948049
## 2  n03127747       crash_helmet 0.06694938
## 3  n04311174        steel_drum 0.05349804
## 4  n02843684         birdhouse 0.04055086
```

```
## 5  n03443371            goblet 0.04000980
##
## $Hamster.jpg
##   class_name class_description      score
## 1  n03793489             mouse 0.34858209
## 2  n02233338         cockroach 0.06264669
## 3  n02909870            bucket 0.04642543
## 4  n03794056         mousetrap 0.04376448
## 5  n04192698            shield 0.03958284
##
## $Lizard.jpg
##   class_name class_description      score
## 1  n13044778         earthstar 0.16892505
## 2  n07749582             lemon 0.09082820
## 3  n13054560            bolete 0.07577771
## 4  n07754684         jackfruit 0.04505501
## 5  n02219486               ant 0.04071513
##
## $Monkey.jpg
##   class_name class_description      score
## 1  n02489166  proboscis_monkey 0.18063334
## 2  n02493509              titi 0.14519861
## 3  n02492035          capuchin 0.14259827
## 4  n02492660      howler_monkey 0.10484368
## 5  n03017168             chime 0.09478551
##
## $Parrot.jpg
##   class_name class_description      score
## 1  n03991062               pot 0.32824290
## 2  n01580077               jay 0.22052953
## 3  n01882714             koala 0.11813702
## 4  n01818515             macaw 0.03252555
## 5  n02843684         birdhouse 0.02874277
##
## $Penguin.jpg
##   class_name class_description      score
## 1  n04523525             vault 0.30266422
## 2  n04553703         washbasin 0.10002303
## 3  n04005630            prison 0.06567439
## 4  n04040759          radiator 0.04754867
## 5  n03874599           padlock 0.02866268
##
## $Stingray.jpg
##   class_name class_description       score
## 1  n01496331      electric_ray 0.817473173
## 2  n01498041          stingray 0.154839367
## 3  n01924916          flatworm 0.020372352
## 4  n07734744          mushroom 0.002621449
## 5  n01950731          sea_slug 0.002227621
```

Interestingly, the model does decently well for semi- obvious, clear pictures such as the Stingray, monkey, and dog images. However, it does really poorly when the animal is not obvious such as the coyote, parrot, and fish.