Untitled

```r
library(ISLR2)
Gitters <- na.omit(Hitters)
n <- nrow(Gitters)
set.seed(13)
ntest <- trunc(n / 3)
testid <- sample(1:n, ntest)
library(glmnet)
```

```
Loading required package: Matrix
```

```
Loaded glmnet 4.1-8
```

```r
library(torch)
```

```
Warning: package 'torch' was built under R version 4.3.3
```

```r
library(luz) # high-level interface for torch
library(torchvision) # for datasets and image transformation
```

```
Warning: package 'torchvision' was built under R version 4.3.3
```

```r
library(torchdatasets) # for datasets we are going to use
```

```
Warning: package 'torchdatasets' was built under R version 4.3.3
```

```r
library(zeallot)
torch_manual_seed(13)
library(ggplot2)
```

###########labs #10.9.1 A Single Layer Network on the Hitters Data

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

```
[1] 254.6687
```

fit the lasso

```r
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] 253.1761
```

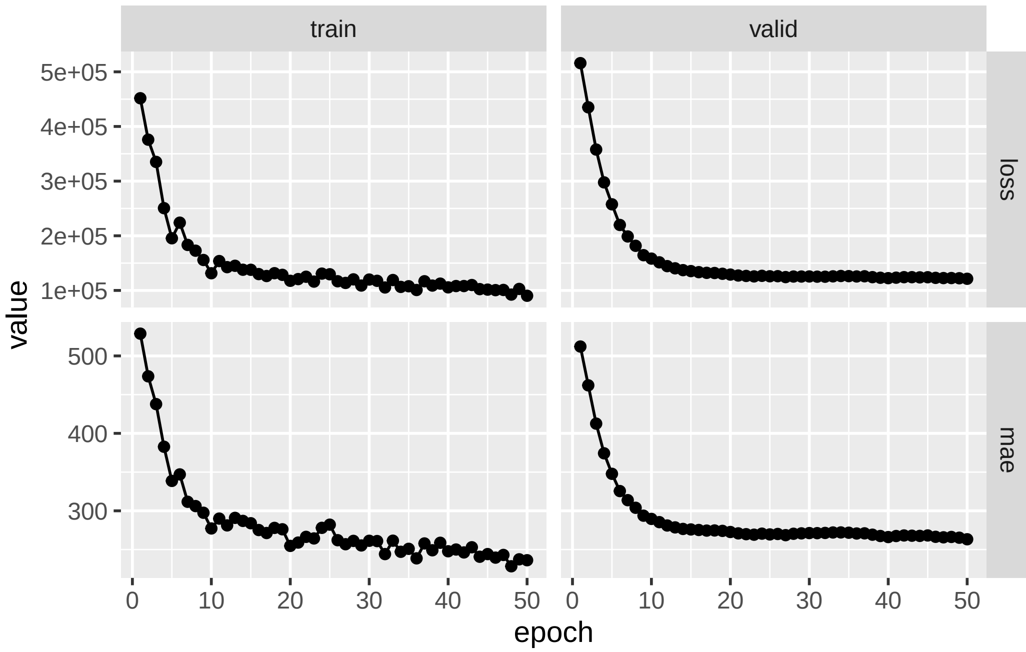
#to fit the neural network, we frst set up a model structure that describes the network.

algorithm tracks the mean absolute error on the training data, and on validation data if it is supplied.

```r
modnn <- nn_module(
  initialize = function(input_size) {
    self$hidden <- nn_linear(input_size, 50)
    self$activation <- nn_relu()
    self$dropout <- nn_dropout(0.2)
    self$output <- nn_linear(50, 1)
  },
```

```r
  forward = function(x) {
    x %>%
      self$hidden() %>%
      self$activation() %>%
      self$dropout() %>%
      self$output()
  }
)

x <- model.matrix(Salary ~ . - 1, data = Gitters) %>% scale()

modnn <- modnn %>%
  setup(
    loss = nn_mse_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_mae())
  ) %>%
  set_hparams(input_size = ncol(x))
```

```r
fitted <- modnn %>%
  fit(
    data = list(x[-testid, ], matrix(y[-testid], ncol = 1)),
    valid_data = list(x[testid, ], matrix(y[testid], ncol = 1)),
    epochs = 50
  )

plot(fitted)
```




```
npred <- predict(fitted, x[testid, ])
npred_array <- as_array(npred)
mae <- mean(abs(y[testid] - npred_array))
print(mae)
```

```
[1] 263.2895
```

‘

#10.9.2 A Multilayer Network on the MNIST Digit Data

```
library(ISLR2)
train_ds <- mnist_dataset(root = ".", train = TRUE, download = TRUE)
test_ds <- mnist_dataset(root = ".", train = FALSE, download = TRUE)

str(train_ds[1])
```

```
List of 2
 $ x: int [1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
 $ y: int 6
```

```r
str(test_ds[2])
```

```
List of 2
 $ x: int [1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
 $ y: int 3
```

```r
length(train_ds)
```

```
[1] 60000
```

```r
length(test_ds)
```

```
[1] 10000
```

```r
transform <- function(x) {
  x %>%
    torch_tensor() %>%
    torch_flatten() %>%
    torch_div(255)
}
train_ds <- mnist_dataset(
  root = ".",
  train = TRUE,
  download = TRUE,
  transform = transform
)
test_ds <- mnist_dataset(
  root = ".",
  train = FALSE,
  download = TRUE,
  transform = transform
)
```

```r
modelnn <- nn_module(
  initialize = function() {
    self$linear1 <- nn_linear(in_features = 28*28, out_features = 256)
    self$linear2 <- nn_linear(in_features = 256, out_features = 128)
```

```r
    self$linear3 <- nn_linear(in_features = 128, out_features = 10)

    self$drop1 <- nn_dropout(p = 0.4)
    self$drop2 <- nn_dropout(p = 0.3)

    self$activation <- nn_relu()
  },
  forward = function(x) {
    x %>%

      self$linear1() %>%
      self$activation() %>%
      self$drop1() %>%

      self$linear2() %>%
      self$activation() %>%
      self$drop2() %>%

      self$linear3()
  }
)

print(modelnn())
```

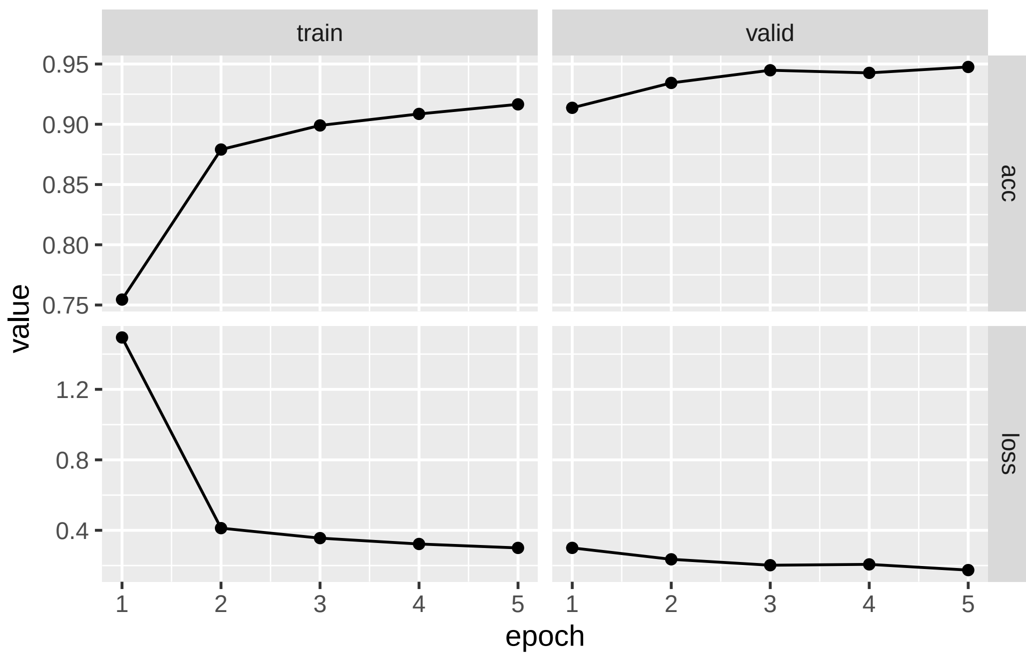
```
An `nn_module` containing 235,146 parameters.

-- Modules ---------------------------------------------------------------------
* linear1: <nn_linear> #200,960 parameters
* linear2: <nn_linear> #32,896 parameters
* linear3: <nn_linear> #1,290 parameters
* drop1: <nn_dropout> #0 parameters
* drop2: <nn_dropout> #0 parameters
* activation: <nn_relu> #0 parameters
```

```r
modelnn <- modelnn %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_accuracy())
  )
```

```r
system.time(
   fitted <- modelnn %>%
      fit(
        data = train_ds,
        epochs = 5,
        valid_data = 0.2,
        dataloader_options = list(batch_size = 256),
        verbose = FALSE
      )
 )
```

```
  user  system elapsed
34.556   1.980  36.751
```

```r
plot(fitted)
```




```r
accuracy <- function(pred, truth) {
   mean(pred == truth) }

# gets the true classes from all observations in test_ds.
```

```r
truth <- sapply(seq_along(test_ds), function(x) test_ds[x][[2]])

fitted %>%
  predict(test_ds) %>%
  torch_argmax(dim = 2) %>%  # the predicted class is the one with higher 'logit'.
  as_array() %>% # we convert to an R object
  accuracy(truth)
```

```
[1] 0.9532
```

```r
modellr <- nn_module(
  initialize = function() {
    self$linear <- nn_linear(784, 10)
  },
  forward = function(x) {
    self$linear(x)
  }
)
print(modellr())
```

```
An `nn_module` containing 7,850 parameters.

-- Modules ---------------------------------------------------------------------
* linear: <nn_linear> #7,850 parameters
```

```r
fit_modellr <- modellr %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_accuracy())
  ) %>%
  fit(
    data = train_ds,
    epochs = 5,
    valid_data = 0.2,
    dataloader_options = list(batch_size = 128)
  )

fit_modellr %>%
```

```r
    predict(test_ds) %>%
    torch_argmax(dim = 2) %>%  # the predicted class is the one with higher 'logit'.
    as_array() %>% # we convert to an R object
    accuracy(truth)
```

```
[1] 0.918
```

```r
  # alternatively one can use the `evaluate` function to get the results
  # on the test_ds
  evaluate(fit_modellr, test_ds)
```

```
A `luz_module_evaluation`
-- Results ---------------------------------------------------------------------
loss: 0.3
acc: 0.918
```

#10.9.3 Convolutional Neural Networks

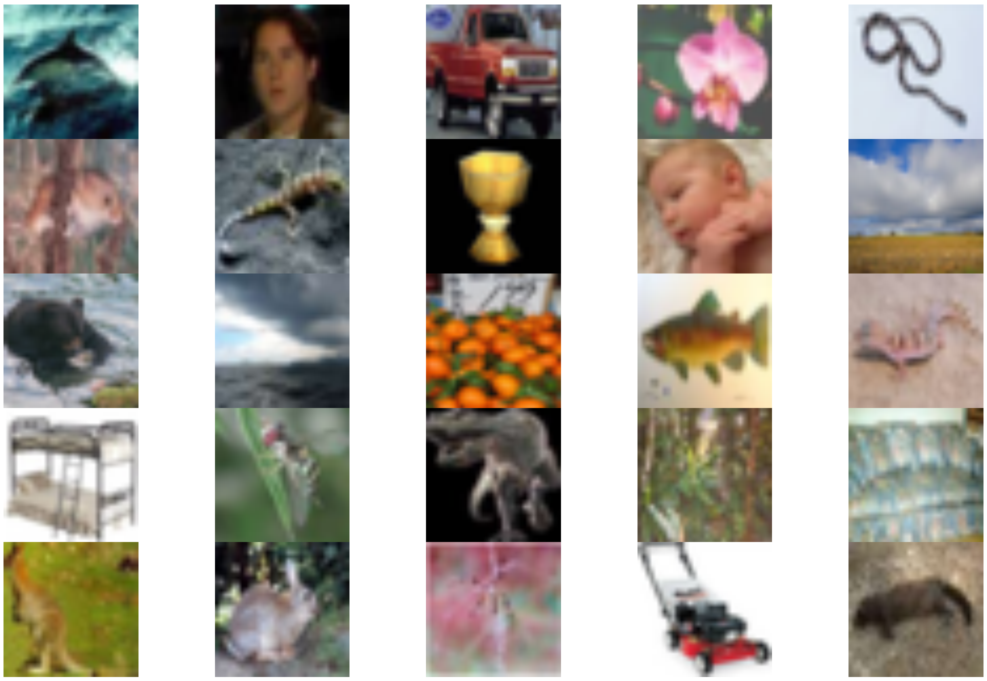
```r
  transform <- function(x) {
    transform_to_tensor(x)
  }

  train_ds <- cifar100_dataset(
    root = "./",
    train = TRUE,
    download = TRUE,
    transform = transform
  )

  test_ds <- cifar100_dataset(
    root = "./",
    train = FALSE,
    transform = transform
  )

  str(train_ds[1])
```

```
List of 2
 $ x:Float [1:3, 1:32, 1:32]
 $ y: int 20
```

```r
length(train_ds)
```

```
[1] 50000
```

```r
par(mar = c(0, 0, 0, 0), mfrow = c(5, 5))
index <- sample(seq(50000), 25)
for (i in index) plot(as.raster(as.array(train_ds[i][[1]]$permute(c(2,3,1)))))
```



```r
conv_block <- nn_module(
  initialize = function(in_channels, out_channels) {
    self$conv <- nn_conv2d(
      in_channels = in_channels,
      out_channels = out_channels,
      kernel_size = c(3,3),
      padding = "same"
    )
    self$relu <- nn_relu()
    self$pool <- nn_max_pool2d(kernel_size = c(2,2))
  },
  forward = function(x) {
```

```r
      x %>%
        self$conv() %>%
        self$relu() %>%
        self$pool()
    }
)

model <- nn_module(
  initialize = function() {
    self$conv <- nn_sequential(
      conv_block(3, 32),
      conv_block(32, 64),
      conv_block(64, 128),
      conv_block(128, 256)
    )
    self$output <- nn_sequential(
      nn_dropout(0.5),
      nn_linear(2*2*256, 512),
      nn_relu(),
      nn_linear(512, 100)
    )
  },
  forward = function(x) {
    x %>%
      self$conv() %>%
      torch_flatten(start_dim = 2) %>%
      self$output()
  }
)
model()
```

```
An `nn_module` containing 964,516 parameters.

-- Modules ---------------------------------------------------------------------
* conv: <nn_sequential> #388,416 parameters
* output: <nn_sequential> #576,100 parameters
```

```r
fitted <- model %>%
  setup(
    loss = nn_cross_entropy_loss(),
    optimizer = optim_rmsprop,
```

```r
    metrics = list(luz_metric_accuracy())
  ) %>%
  set_opt_hparams(lr = 0.001) %>%
  fit(
    train_ds,
    epochs = 10, #30,
    valid_data = 0.2,
    dataloader_options = list(batch_size = 128)
  )

print(fitted)
```

```
A `luz_module_fitted`
-- Time ------------------------------------------------------------------------
* Total time: 3m 17.9s
* Avg time per training epoch: 17.1s

-- Results ---------------------------------------------------------------------
Metrics observed in the last epoch.

i Training:
loss: 2.351
acc: 0.3824

-- Model -----------------------------------------------------------------------
An `nn_module` containing 964,516 parameters.

-- Modules ---------------------------------------------------------------------
* conv: <nn_sequential> #388,416 parameters
* output: <nn_sequential> #576,100 parameters
```

```r
evaluate(fitted, test_ds)
```

```
A `luz_module_evaluation`
-- Results ---------------------------------------------------------------------
loss: 2.4013
acc: 0.3816
```

#10.9.4 Using Pretrained CNN Models

```r
img_dir <- "book_images"
image_names <- list.files(img_dir)
num_images <- length(image_names)
x <- torch_empty(num_images, 3, 224, 224)
for (i in 1:num_images) {
   img_path <- file.path(img_dir, image_names[i])
   img <- img_path %>%
     base_loader() %>%
     transform_to_tensor() %>%
     transform_resize(c(224, 224)) %>%
     # normalize with imagenet mean and stds.
     transform_normalize(
       mean = c(0.485, 0.456, 0.406),
       std = c(0.229, 0.224, 0.225)
     )
   x[i,,, ] <- img
}
```

```r
model <- torchvision::model_resnet18(pretrained = TRUE)
model$eval() # put the model in evaluation mode
preds <- model(x)

mapping <- jsonlite::read_json("https://s3.amazonaws.com/deep-learning-models/image-models
  sapply(function(x) x[[2]])

top3 <- torch_topk(preds, dim = 2, k = 3)

top3_prob <- top3[[1]] %>%
  nnf_softmax(dim = 2) %>%
  torch_unbind() %>%
  lapply(as.numeric)

top3_class <- top3[[2]] %>%
  torch_unbind() %>%
  lapply(function(x) mapping[as.integer(x)])

result <- purrr::map2(top3_prob, top3_class, function(pr, cl) {
  names(pr) <- cl
  pr
})
names(result) <- image_names
```

```r
print(result)
```

```
$Cape_Weaver.jpg
hummingbird    lorikeet   bee_eater
  0.3633293   0.3577291   0.2789416

$Hawk_cropped.jpg
     kite       jay    magpie
0.6157786 0.2311880 0.1530334

$Hawk_Fountain.jpg
        eel       agama common_newt
  0.5391121   0.2527187   0.2081692

$Lhasa_Apso.jpg
          Lhasa Tibetan_terrier        Shih-Tzu
     0.79760498      0.12012957      0.08226541

$Sleeping_Cat.jpg
       Saint_Bernard          guinea_pig Bernese_mountain_dog
           0.3946666           0.3426994            0.2626340
```

#10.9.5 IMDb Document Classifcation

```r
max_features <- 10000
imdb_train <- imdb_dataset(
  root = ".",
  download = TRUE,
  num_words = max_features
)
imdb_test <- imdb_dataset(
  root = ".",
  download = TRUE,
  num_words = max_features
)

imdb_train[1]$x[1:12]
```

```
 [1]    2  261  297   14   20   23    4 6253 1307   13   70   65
```

```r
word_index <- imdb_train$vocabulary
decode_review <- function(text, word_index) {
   word <- names(word_index)
   idx <- unlist(word_index, use.names = FALSE)
   word <- c("<PAD>", "<START>", "<UNK>", word)
   words <- word[text]
   paste(words, collapse = " ")
}
decode_review(imdb_train[1]$x[1:12], word_index)
```

```
[1] "<START> having watched this movie on the scifi channel i can only"
```

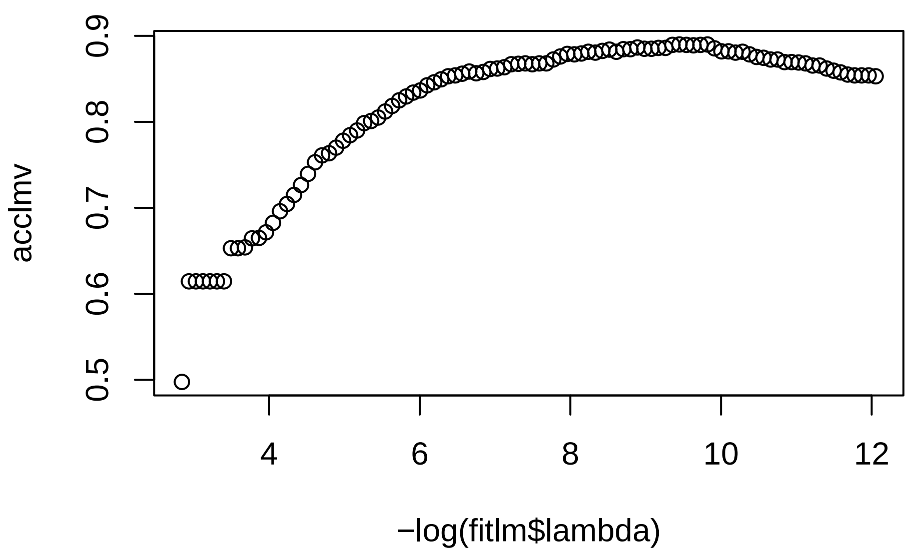
```r
library(Matrix)
one_hot <- function(sequences, dimension) {
   seqlen <- sapply(sequences, length)
   n <- length(seqlen)
   rowind <- rep(1:n, seqlen)
   colind <- unlist(sequences)
   sparseMatrix(i = rowind, j = colind,
      dims = c(n, dimension))
}

# collect all values into a list
train <- seq_along(imdb_train) %>%
  lapply(function(i) imdb_train[i]) %>%
  purrr::transpose()
test <- seq_along(imdb_test) %>%
  lapply(function(i) imdb_test[i]) %>%
  purrr::transpose()

# num_words + padding + start + oov token = 10000 + 3
x_train_1h <- one_hot(train$x, 10000 + 3)
x_test_1h <- one_hot(test$x, 10000 + 3)
dim(x_train_1h)
```

```
[1] 25000 10003
```

```r
nnzero(x_train_1h) / (25000 * (10000 + 3))
```

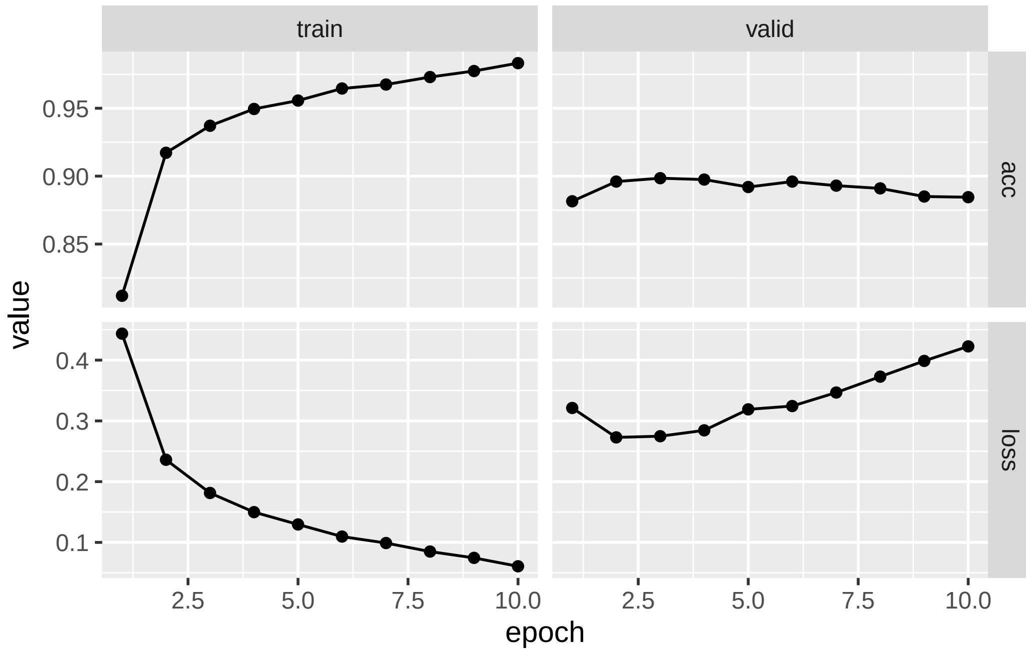
```
[1] 0.01316756
```

```
set.seed(3)
ival <- sample(seq(along = train$y), 2000)
itrain <- seq_along(train$y)[-ival]

library(glmnet)
y_train <- unlist(train$y)

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

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




```
model <- nn_module(
  initialize = function(input_size = 10000 + 3) {
    self$dense1 <- nn_linear(input_size, 16)
    self$relu <- nn_relu()
```

```r
    self$dense2 <- nn_linear(16, 16)
    self$output <- nn_linear(16, 1)
  },
  forward = function(x) {
    x %>%
      self$dense1() %>%
      self$relu() %>%
      self$dense2() %>%
      self$relu() %>%
      self$output() %>%
      torch_flatten(start_dim = 1)
  }
)
model <- model %>%
  setup(
    loss = nn_bce_with_logits_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_binary_accuracy_with_logits())
  ) %>%
  set_opt_hparams(lr = 0.001)

fitted <- model %>%
  fit(
    # we transform the training and validation data into torch tensors
    list(
      torch_tensor(as.matrix(x_train_1h[itrain,]), dtype = torch_float()),
      torch_tensor(unlist(train$y[itrain]))
    ),
    valid_data = list(
      torch_tensor(as.matrix(x_train_1h[ival, ]), dtype = torch_float()),
      torch_tensor(unlist(train$y[ival]))
    ),
    dataloader_options = list(batch_size = 512),
    epochs = 10
  )

plot(fitted)
```




```
fitted <- model %>%
  fit(
    list(
      torch_tensor(as.matrix(x_train_1h[itrain,]), dtype = torch_float()),
      torch_tensor(unlist(train$y[itrain]))
    ),
    valid_data = list(
      torch_tensor(as.matrix(x_test_1h), dtype = torch_float()),
      torch_tensor(unlist(test$y))
    ),
    dataloader_options = list(batch_size = 512),
    epochs = 10
  )
```

#10.9.6 Recurrent Neural Networks ## Sequential Models for Document Classification

```
wc <- sapply(seq_along(imdb_train), function(i) length(imdb_train[i]$x))
median(wc)
```

```
[1] 178
```

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

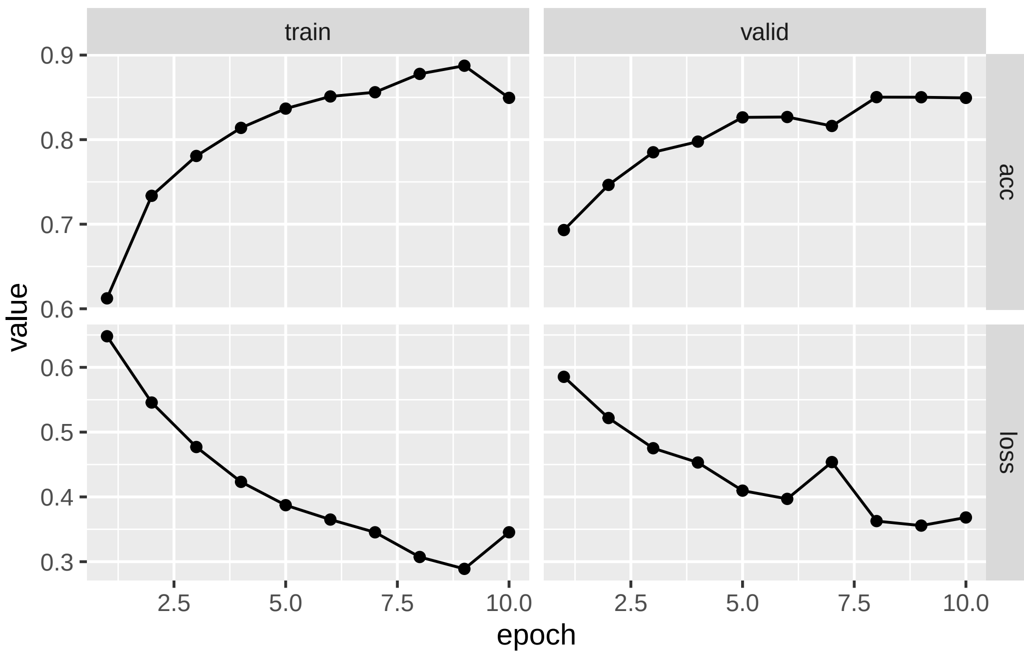
```
[1] 0.916
```

```r
maxlen <- 500
num_words <- 10000
imdb_train <- imdb_dataset(root = ".", split = "train", num_words = num_words,
                           maxlen = maxlen)
imdb_test <- imdb_dataset(root = ".", split = "test", num_words = num_words,
                          maxlen = maxlen)

vocab <- c(rep(NA, imdb_train$index_from - 1), imdb_train$get_vocabulary())
tail(names(vocab)[imdb_train[1]$x])
```

```
[1] "compensate" "you"        "the"        "rental"     ""
[6] "d"
```

```r
model <- nn_module(
  initialize = function() {
    self$embedding <- nn_embedding(10000 + 3, 32)
    self$lstm <- nn_lstm(input_size = 32, hidden_size = 32, batch_first = TRUE)
    self$dense <- nn_linear(32, 1)
  },
  forward = function(x) {
    c(output, c(hn, cn)) %<-% (x %>%
      self$embedding() %>%
      self$lstm())
    output[,-1,] %>%  # get the last output
      self$dense() %>%
      torch_flatten(start_dim = 1)
  }
)

model <- model %>%
  setup(
    loss = nn_bce_with_logits_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_binary_accuracy_with_logits())
  ) %>%
```

```
  set_opt_hparams(lr = 0.001)

fitted <- model %>% fit(
  imdb_train,
  epochs = 10,
  dataloader_options = list(batch_size = 128),
  valid_data = imdb_test
)
plot(fitted)
```




```
predy <- torch_sigmoid(predict(fitted, imdb_test)) > 0.5
evaluate(fitted, imdb_test, dataloader_options = list(batch_size = 512))
```

```
A `luz_module_evaluation`
-- Results ---------------------------------------------------------------------
loss: 0.3685
acc: 0.8494
```

Time Series Prediction

```r
library(ISLR2)
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
```

```
[1] 0.413223
```

```r
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
```

```r
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
```

```r
model <- nn_module(
  initialize = function() {
    self$rnn <- nn_rnn(3, 12, batch_first = TRUE)
    self$dense <- nn_linear(12, 1)
    self$dropout <- nn_dropout(0.2)
  },
  forward = function(x) {
    c(output, ...) %<-% (x %>%
      self$rnn())
    output[,-1,] %>%
      self$dropout() %>%
      self$dense() %>%
      torch_flatten(start_dim = 1)
  }
)

model <- model %>%
  setup(
    optimizer = optim_rmsprop,
    loss = nn_mse_loss()
  ) %>%
  set_opt_hparams(lr = 0.001)
```

```r
fitted <- model %>% fit(
    list(xrnn[istrain,, ], arframe[istrain, "log_volume"]),
    epochs = 75, #epochs = 200,
    dataloader_options = list(batch_size = 64),
    valid_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )
kpred <- as.numeric(predict(fitted, xrnn[!istrain,, ]))
```

```r
1 - mean((kpred - arframe[!istrain, "log_volume"])^2) / V0
```

```
[1] 0.4131635
```

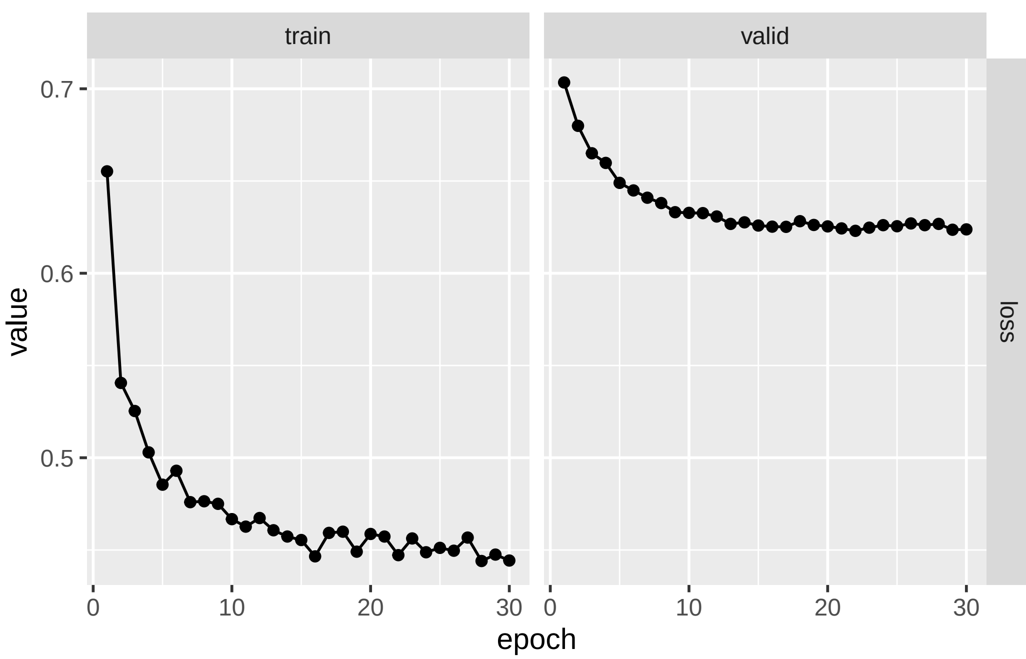
```r
model <- nn_module(
  initialize = function() {
    self$dense <- nn_linear(15, 1)
  },
  forward = function(x) {
    x %>%
      torch_flatten(start_dim = 2) %>%
      self$dense()
  }
)
```

```r
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"
```

```r
arnnd <- nn_module(
  initialize = function() {
    self$dense <- nn_linear(15, 32)
    self$dropout <- nn_dropout(0.5)
    self$activation <- nn_relu()
    self$output <- nn_linear(32, 1)

  },
  forward = function(x) {
    x %>%
      torch_flatten(start_dim = 2) %>%
      self$dense() %>%
      self$activation() %>%
```

```
        self$dropout() %>%
        self$output() %>%
        torch_flatten(start_dim = 1)
    }
)
arnnd <- arnnd %>%
  setup(
    optimizer = optim_rmsprop,
    loss = nn_mse_loss()
  ) %>%
  set_opt_hparams(lr = 0.001)

fitted <- arnnd %>% fit(
    list(xrnn[istrain,, ], arframe[istrain, "log_volume"]),
    epochs = 30, #epochs = 200,
    dataloader_options = list(batch_size = 64),
    valid_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )
plot(fitted)
```

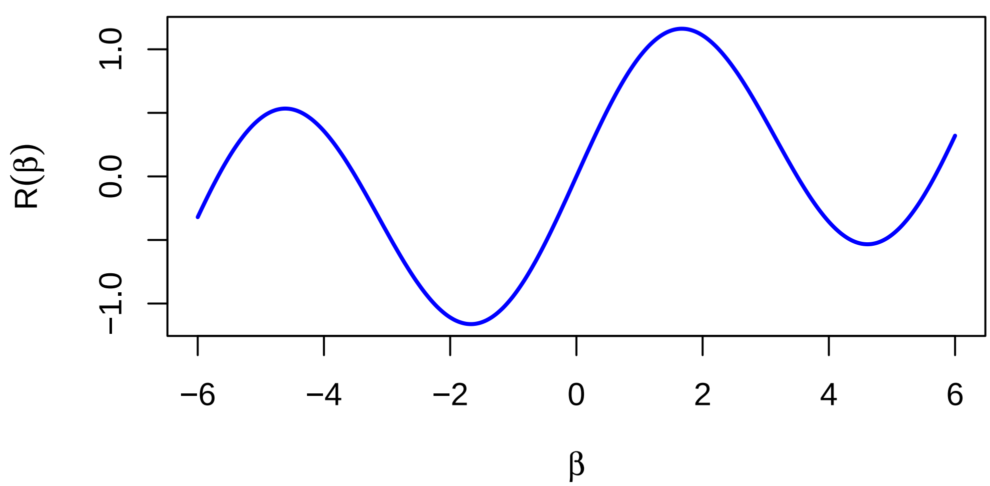


```r
npred <- as.numeric(predict(fitted, xrnn[!istrain, ,]))
1 - mean((arframe[!istrain, "log_volume"] - npred)^2) / V0
```

```
[1] 0.428227
```

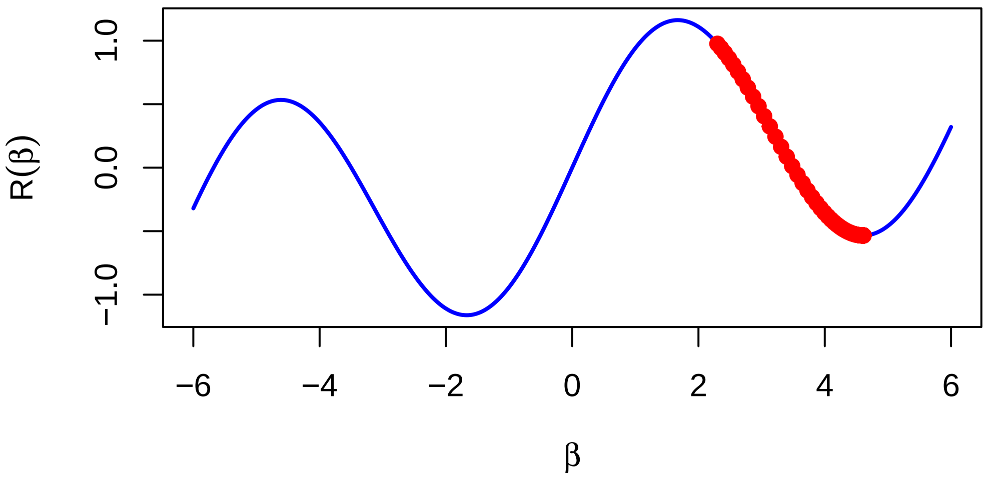
#6.

```r
R_beta <- function(beta) {
  sin(beta) + beta / 10
}

b <- seq(-6, 6, length.out = 400)

R_b <- R_beta(b)

plot(b, R_b, type = "l", col = "blue", lwd = 2,
     xlab = expression(beta), ylab = expression(R(beta)))
```




```r
dR_beta <- function(beta) {
  cos(beta) + 0.1
```

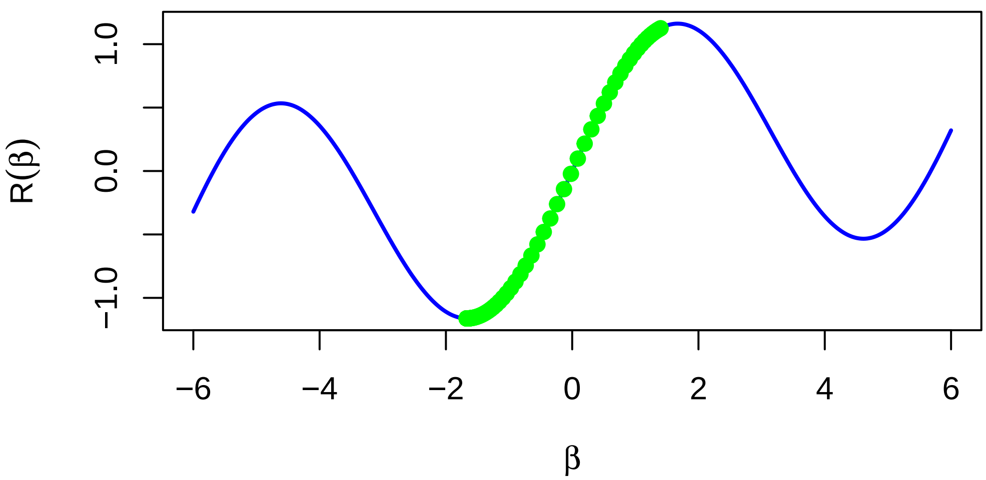
```
}

beta_0 <- 2.3
learning_rate <- 0.1
num_iterations <- 100

beta_values <- numeric(num_iterations)
beta_values[1] <- beta_0

for (t in 2:num_iterations) {
  beta_values[t] <- beta_values[t-1] - learning_rate * dR_beta(beta_values[t-1])
}

plot(b, R_b, type = "l", col = "blue", lwd = 2, xlab = expression(beta), ylab = expression
points(beta_values, R_beta(beta_values), col = "red", pch = 19)
```




```
cat("Final   value after gradient descent:", beta_values[num_iterations], "\n")
```

```
Final   value after gradient descent: 4.612013
```

```
beta_1 <- 1.4
beta_values <- numeric(num_iterations)
beta_values[1] <- beta_1

for (t in 2:num_iterations) {
  beta_values[t] <- beta_values[t-1] - learning_rate * dR_beta(beta_values[t-1])
}

plot(b, R_b, type = "l", col = "blue", lwd = 2, xlab = expression(beta), ylab = expression
points(beta_values, R_beta(beta_values), col = "green", pch = 19)
```




```
cat("Final  value after gradient descent:", beta_values[num_iterations], "\n")
```

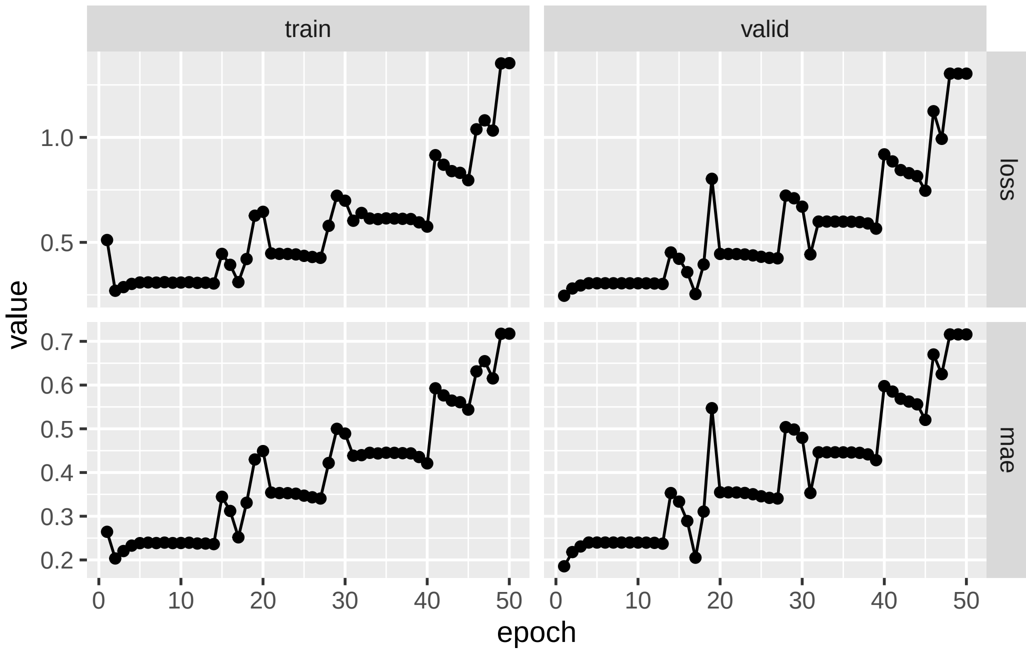
```
Final  value after gradient descent: -1.670396
```

#7.Fit a neural network to the Default data.

```
Default$default <- ifelse(Default$default == "Yes", 1, 0)

n <- nrow(Default)
```

```r
set.seed(13)
ntest <- trunc(n / 5)
testid <- sample(1:n, ntest)

modnn <- nn_module(
  initialize = function(input_size) {
    self$hidden <- nn_linear(input_size, 10)
    self$activation <- nn_relu()
    self$dropout <- nn_dropout(0.3)
    self$output <- nn_linear(10, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden() %>%
      self$activation() %>%
      self$dropout() %>%
      self$output()
  }
)

x <- model.matrix(default ~ . - 1, data = Default) %>% scale()
y <- Default$default

modnn <- modnn %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_mae())
  ) %>%
  set_hparams(input_size = ncol(x))
```

```r
fitted <- modnn %>%
  fit(
    data = list(x[-testid, ], matrix(y[-testid], ncol = 1)),
    valid_data = list(x[testid, ], matrix(y[testid], ncol = 1)),
    epochs = 50
  )

plot(fitted)
```




```
npred <- predict(fitted, x[testid, ])
npred_array <- as_array(npred)
predictions <- ifelse(npred_array > 0.5, 1, 0)
accuracy <- mean(predictions == y[testid])

print(paste("Accuracy:", accuracy))
```

```
[1] "Accuracy: 0.0355"
```

#8. From your collection of personal photographs, pick 10 images of animals

```
img_dir <- "animal_images"
image_names <- list.files(img_dir)
num_images <- length(image_names)
x <- torch_empty(num_images, 3, 224, 224)
for (i in 1:num_images) {
   img_path <- file.path(img_dir, image_names[i])
   img <- img_path %>%
     base_loader() %>%
     transform_to_tensor() %>%
     transform_resize(c(224, 224)) %>%
```

```r
      # normalize with imagenet mean and stds.
      transform_normalize(
        mean = c(0.485, 0.456, 0.406),
        std = c(0.229, 0.224, 0.225)
      )
    x[i,,, ] <- img
}
```

```r
model <- torchvision::model_resnet18(pretrained = TRUE)
model$eval() # put the model in evaluation mode
preds <- model(x)

mapping <- jsonlite::read_json("https://s3.amazonaws.com/deep-learning-models/image-models
  sapply(function(x) x[[2]])

top5 <- torch_topk(preds, dim = 2, k = 5)

top5_prob <- top5[[1]] %>%
  nnf_softmax(dim = 2) %>%
  torch_unbind() %>%
  lapply(as.numeric)

top5_class <- top5[[2]] %>%
  torch_unbind() %>%
  lapply(function(x) mapping[as.integer(x)])

result <- purrr::map2(top5_prob, top5_class, function(pr, cl) {
  names(pr) <- cl
  pr
})
names(result) <- image_names
print(result)
```

```
$IMG_4037.JPG
Labrador_retriever   golden_retriever     cocker_spaniel             beagle
       0.648640752        0.321777314        0.015168818        0.008276871
            kuvasz
       0.006136307

$IMG_4191.JPG
          carton        guinea_pig         Band_Aid crossword_puzzle
```

```
      0.2440549        0.2263593        0.1907472        0.1806872
    plastic_bag
      0.1581516

$IMG_4588.JPG
   Shih-Tzu       Lhasa Maltese_dog    Pekinese  toy_poodle
 0.55186415  0.16366476  0.12603915  0.07993906  0.07849292

$IMG_4591.JPG
     otter     marmot     weasel       mink     beaver
0.36287266 0.34405339 0.14921607 0.08073228 0.06312557

$IMG_4592.JPG
           chickadee               puffer red-backed_sandpiper
          0.40517688           0.32740453           0.09740868
               quail             sea_slug
          0.08738206           0.08262786

$IMG_4593.JPG
brown_bear      bison      teddy   mongoose       chow
0.65225726 0.19173110 0.07481807 0.04515351 0.03604012

$IMG_4594.JPG
           dhole          cheetah            dingo  standard_poodle
       0.2736522        0.2573565        0.2295044        0.1486792
miniature_poodle
       0.0908076

$IMG_4595.JPG
                Maltese_dog                  Persian_cat
                 0.54989856                   0.22765583
                   Shih-Tzu West_Highland_white_terrier
                 0.09817947                   0.06886514
       Old_English_sheepdog
                 0.05540101

$IMG_4596.JPG
     llama     Angora     kuvasz guinea_pig    wallaby
 0.2987915  0.1975946  0.1930207  0.1647692  0.1458240

$IMG_4597.JPG
     Angora Persian_cat Siamese_cat wood_rabbit     hamster
 0.71234006  0.16844420  0.06128484  0.02935519  0.02857562
```

#9. Fit a lag-5 autoregressive model to the NYSE data

```
library(ISLR2)
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)]
V0 <- var(arframe[!istrain, "log_volume"])
```

```
NYSE$month <- format(as.Date(NYSE$date), "%m")
NYSE$month <- factor(NYSE$month, levels = sprintf("%02d", 1:12))
arframed <- data.frame(day = NYSE[-(1:5), "day_of_week"], month = NYSE[-(1:5), "month"], a

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

```
[1] 0.4629872
```

#0.4629872

#10.

```
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 <- nn_module(
  initialize = function() {
    self$rnn <- nn_rnn(3, 12, batch_first = TRUE)
    self$dense <- nn_linear(12, 1)
    self$dropout <- nn_dropout(0.2)
  },
  forward = function(x) {
    c(output, ...) %<-% (x %>%
      self$rnn())
    output[,-1,] %>%
      self$dropout() %>%
      self$dense() %>%
      torch_flatten(start_dim = 1)
  }
)

model <- model %>%
  setup(
    optimizer = optim_rmsprop,
    loss = nn_mse_loss()
  ) %>%
  set_opt_hparams(lr = 0.001)
```

```
fitted <- model %>% fit(
    list(xrnn[istrain,, ], arframe[istrain, "log_volume"]),
    epochs = 200, #epochs = 200,
    dataloader_options = list(batch_size = 64),
    valid_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )
kpred <- as.numeric(predict(fitted, xrnn[!istrain,, ]))
1 - mean((kpred - arframe[!istrain, "log_volume"])^2) / V0
```

```
[1] 0.4085854
```

#11.

```
model <- nn_module(
  initialize = function() {
    self$dense <- nn_linear(15, 1)
  },
  forward = function(x) {
    x %>%
      torch_flatten(start_dim = 2) %>%
      self$dense()
  }
)
```

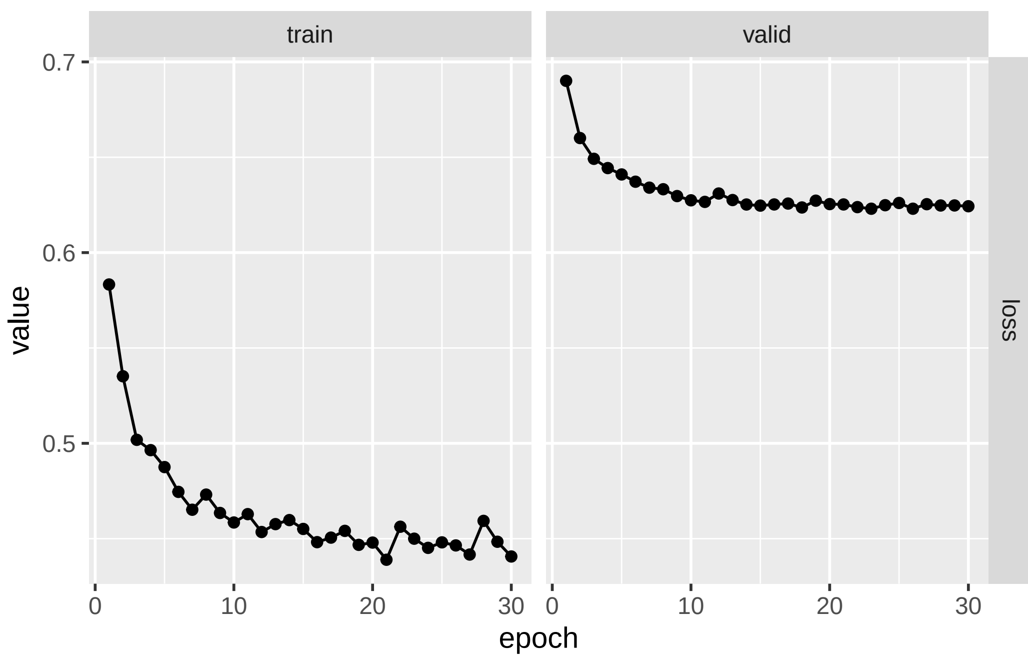
```
x <- model.matrix(log_volume ~ . - 1, data = arframed)
colnames(x)
```

```
 [1] "dayfri"             "daymon"             "daythur"
 [4] "daytues"            "daywed"             "month02"
 [7] "month03"            "month04"            "month05"
[10] "month06"            "month07"            "month08"
[13] "month09"            "month10"            "month11"
[16] "month12"            "L1.DJ_return"       "L1.log_volume"
[19] "L1.log_volatility" "L2.DJ_return"       "L2.log_volume"
[22] "L2.log_volatility" "L3.DJ_return"       "L3.log_volume"
[25] "L3.log_volatility" "L4.DJ_return"       "L4.log_volume"
[28] "L4.log_volatility" "L5.DJ_return"       "L5.log_volume"
[31] "L5.log_volatility"
```

```
arnnd <- nn_module(
  initialize = function() {
    self$dense <- nn_linear(15, 32)
    self$dropout <- nn_dropout(0.5)
    self$activation <- nn_relu()
    self$output <- nn_linear(32, 1)

  },
  forward = function(x) {
    x %>%
      torch_flatten(start_dim = 2) %>%
      self$dense() %>%
      self$activation() %>%
```

```
        self$dropout() %>%
        self$output() %>%
        torch_flatten(start_dim = 1)
    }
)
arnnd <- arnnd %>%
  setup(
    optimizer = optim_rmsprop,
    loss = nn_mse_loss()
  ) %>%
  set_opt_hparams(lr = 0.001)
```

```
fitted <- arnnd %>% fit(
    list(xrnn[istrain,, ], arframe[istrain, "log_volume"]),
    epochs = 30,
    dataloader_options = list(batch_size = 64),
    valid_data =
      list(xrnn[!istrain,, ], arframe[!istrain, "log_volume"])
  )
plot(fitted)
```



```r
npred <- as.numeric(predict(fitted, xrnn[!istrain, ,]))
1 - mean((arframe[!istrain, "log_volume"] - npred)^2) / V0
```

```
[1] 0.4277066
```

#13.

```r
dict_sizes <- c(1000, 3000, 5000, 10000)

accuracy <- function(pred, truth) {
   mean(pred == truth) }

# Loop through each dictionary size
for (max_features in dict_sizes) {

  cat("\n\nTesting with dictionary size:", max_features, "\n\n")

  imdb_train <- imdb_dataset(
    root = ".",
    download = TRUE,
    num_words = max_features
  )

  imdb_test <- imdb_dataset(
    root = ".",
    download = TRUE,
    num_words = max_features
  )


  word_index <- imdb_train$vocabulary

  decode_review <- function(text, word_index) {
    word <- names(word_index)
    idx <- unlist(word_index, use.names = FALSE)
    word <- c("<PAD>", "<START>", "<UNK>", word)
    words <- word[text]
    paste(words, collapse = " ")
  }

  print(decode_review(imdb_train[1]$x[1:12], word_index))
```

```
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))
}

train <- seq_along(imdb_train) %>%
  lapply(function(i) imdb_train[i]) %>%
  purrr::transpose()

test <- seq_along(imdb_test) %>%
  lapply(function(i) imdb_test[i]) %>%
  purrr::transpose()

# One-hot encoding (adjust the size according to max_features)
x_train_1h <- one_hot(train$x, max_features + 3)  # Padding and special tokens
x_test_1h <- one_hot(test$x, max_features + 3)

cat("Train data dimensions: ", dim(x_train_1h), "\n")
cat("Non-zero elements in train set: ", nnzero(x_train_1h) / (25000 * (max_features + 3)


set.seed(3)
ival <- sample(seq(along = train$y), 2000)  # Validation set indices
itrain <- seq_along(train$y)[-ival]  # Training set indices

y_train <- unlist(train$y)

fitlm <- glmnet(x_train_1h[itrain, ], unlist(y_train[itrain]),
                family = "binomial", standardize = FALSE)

classlmv <- predict(fitlm, x_train_1h[ival, ]) > 0
acclmv <- apply(classlmv, 2, accuracy,  unlist(y_train[ival]) > 0)



model <- nn_module(
  initialize = function(input_size = max_features + 3) {
    self$dense1 <- nn_linear(input_size, 16)
```

```
    self$relu <- nn_relu()
    self$dense2 <- nn_linear(16, 16)
    self$output <- nn_linear(16, 1)
  },
  forward = function(x) {
    x %>%
      self$dense1() %>%
      self$relu() %>%
      self$dense2() %>%
      self$relu() %>%
      self$output() %>%
      torch_flatten(start_dim = 1)
  }
)

model <- model %>%
  setup(
    loss = nn_bce_with_logits_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_binary_accuracy_with_logits())
  ) %>%
  set_opt_hparams(lr = 0.001)

fitted <- model %>%
  fit(
    list(
      torch_tensor(as.matrix(x_train_1h[itrain, ]), dtype = torch_float()),
      torch_tensor(unlist(train$y[itrain]))
    ),
    valid_data = list(
      torch_tensor(as.matrix(x_train_1h[ival, ]), dtype = torch_float()),
      torch_tensor(unlist(train$y[ival]))
    ),
    dataloader_options = list(batch_size = 512),
    epochs = 10
  )


fitted <- model %>%
  fit(
    list(
```

```
        torch_tensor(as.matrix(x_train_1h[itrain, ]), dtype = torch_float()),
        torch_tensor(unlist(train$y[itrain]))
      ),
      valid_data = list(
        torch_tensor(as.matrix(x_test_1h), dtype = torch_float()),
        torch_tensor(unlist(test$y))
      ),
      dataloader_options = list(batch_size = 512),
      epochs = 10
    )

  cat("\nFinished testing with dictionary size:", max_features, "\n\n")
}
```

```
Testing with dictionary size: 1000

[1] "<START> this does give away some of the plot by the way"
Train data dimensions:  25000 1003
Non-zero elements in train set:  0.09439083

Finished testing with dictionary size: 1000



Testing with dictionary size: 3000

[1] "<START> i saw this movie about a year ago and found it"
Train data dimensions:  25000 3003
Non-zero elements in train set:  0.03836119

Finished testing with dictionary size: 3000



Testing with dictionary size: 5000

[1] "<START> i saw this movie about a year ago and found it"
Train data dimensions:  25000 5003
Non-zero elements in train set:  0.02459676
```

```
Finished testing with dictionary size: 5000



Testing with dictionary size: 10000

[1] "<START> i saw this movie about a year ago and found it"
Train data dimensions:  25000 10003
Non-zero elements in train set:  0.01316756

Finished testing with dictionary size: 10000
```