Weekly Summary Template

Brady Miller

Table of contents

Thursday, April 6 1

Thursday, April 6

❗ TIL

Include a *very brief* summary of what you learnt in this class here.
Today, I learnt the following concepts in class:

1. Luz Hyperparameters
2. Luz setup and fit process
3. Luz validation and metrics that we can use

```
packages <- c(
  "dplyr",
  "readr",
  "tidyr",
  "purrr",
  "stringr",
  "corrplot",
  "car",
  "caret",
  "torch",
  "nnet",
  "broom",
  "torch",
  "torchvision",
  "e1071",
```

```r
  "glmnet",
  "nnet",
  "rpart",
  "ISLR2",
  'luz',
  'torchvision'
)

renv::install(packages)
```

```
Installing dplyr [1.1.1] ...
    OK [linked cache in 2.4 milliseconds]
Installing readr [2.1.4] ...
    OK [linked cache in 2.7 milliseconds]
Installing purrr [1.0.1] ...
    OK [linked cache in 2.6 milliseconds]
Installing stringr [1.5.0] ...
    OK [linked cache in 3.1 milliseconds]
Installing tidyr [1.3.0] ...
    OK [linked cache in 3.4 milliseconds]
Installing corrplot [0.92] ...
    OK [linked cache in 2.4 milliseconds]
Installing nnet [7.3-18] ...
    OK [linked cache in 3.1 milliseconds]
Installing broom [1.0.4] ...
    OK [linked cache in 3.1 milliseconds]
Installing car [3.1-2] ...
    OK [linked cache in 2.8 milliseconds]
Installing e1071 [1.7-13] ...
    OK [linked cache in 2.4 milliseconds]
Installing rpart [4.1.19] ...
    OK [linked cache in 3.2 milliseconds]
Installing caret [6.0-94] ...
    OK [linked cache in 5 milliseconds]
Installing torch [0.9.1] ...
    OK [linked cache in 2.3 milliseconds]
Installing torchvision [0.5.0] ...
    OK [linked cache in 2.4 milliseconds]
Installing glmnet [4.1-7] ...
    OK [linked cache in 2.5 milliseconds]
Installing ISLR2 [1.3-2] ...
    OK [linked cache in 2.5 milliseconds]
```

```
Installing luz [0.3.1] ...
    OK [linked cache in 2.4 milliseconds]
```

```r
sapply(packages, require, character.only=T)
```

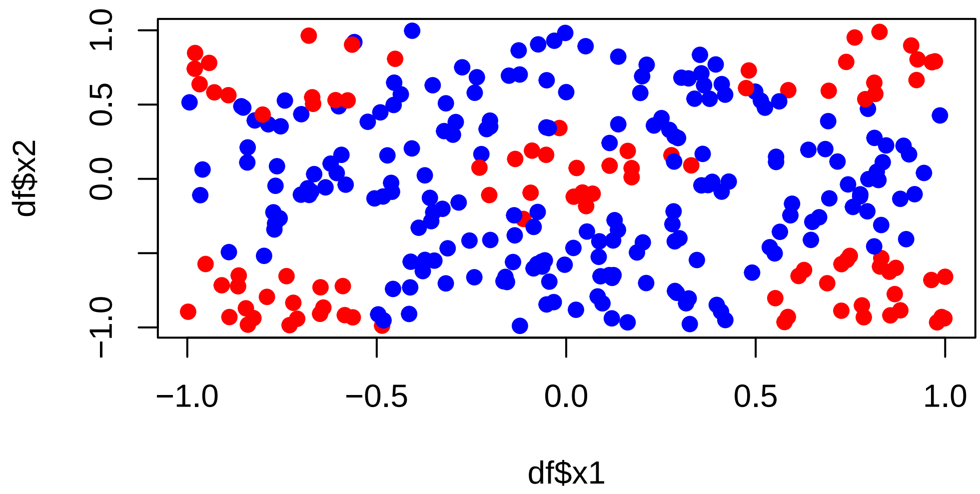
```
Loading required package: dplyr

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

Loading required package: readr

Loading required package: tidyr

Loading required package: purrr

Loading required package: stringr

Loading required package: corrplot

corrplot 0.92 loaded

Loading required package: car

Loading required package: carData

Attaching package: 'car'

The following object is masked from 'package:purrr':

    some
```

```
The following object is masked from 'package:dplyr':

    recode

Loading required package: caret

Loading required package: ggplot2

Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift

Loading required package: torch

Loading required package: nnet

Loading required package: broom

Loading required package: torchvision

Loading required package: e1071

Loading required package: glmnet

Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-7
```

```
Loading required package: rpart

Loading required package: ISLR2

Loading required package: luz

      dplyr       readr       tidyr       purrr     stringr    corrplot 
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE 
        car       caret       torch        nnet       broom       torch 
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE 
torchvision       e1071      glmnet        nnet       rpart       ISLR2 
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE 
        luz torchvision 
       TRUE        TRUE 
```

Creating a dataset that we can use with luz and neural network to make predictions with

```r
ex <- \(x) ifelse(
  ((abs(x[1]) + 0.05 * rnorm(1) > 0.50 & abs(x[2]) + 0.05 * rnorm(1) > 0.50)) |
  ((abs(x[1]) + 0.05 * rnorm(1) < 0.25 & abs(x[2]) + 0.05 * rnorm(1) < 0.25)),1,0
)

n <- 300
X <- t(replicate(n, 2 * runif(2) - 1))
y <- apply(X, 1, ex) %>% as.factor()
col <- ifelse(y == 0, 'blue', 'red')
df <- data.frame(y = y, x1 = X[,1], x2 = X[,2])
model <- glm(y ~ x1 + x2, df, family = binomial())
plot(df$x1, df$x2, col = col, pch = 19)
```




```
xnew <- cbind(
  rep(seq(-1.1, 1.1, length.out = 50), 50),
  rep(seq(-1.1, 1.1, length.out = 50), each = 50)
)

df_new = data.frame(x1 = xnew[,1], x2 = xnew[,2])
```

Luz Hyperparameters

We can use hyperparameters to specify the inputs for the neural network so we can create our own inputs/dimensions for the hidden layers

```
nn_model <- nn_module(
  initialize = function(p, q1, q2, q3) {
    self$f <- nn_linear(p,q1)
    self$g <- nn_linear(q1, q2)
    self$h <- nn_linear(q2, q3)
    self$i <- nn_linear(q3, 1)
    self$a <- nn_relu()
    self$s <- nn_sigmoid()
  },
```

```
  forward = function(x) {
    x %>%
      self$f() %>%
      self$a() %>%
      self$g() %>%
      self$a() %>%
      self$h() %>%
      self$a() %>%
      self$i() %>%
      self$s()
  }
)
```

Luz Setup & Fit

In this next section, I while show how to use luz to setup and fit the nn model

```
nn_model %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam
  )
```

```
<luz_module_generator>
```

*Setup function takes in 2 mandatory arguments –> the loss function and the optimizer (used for minmimizing loss) 1. optimizer could be optim_adam, optim_rmsprop, …

The code above is equivalent to…

```
 F <- hh2_module()
 optimizer <- optim_adam(F$parameters, lr = 0.05)
 epochs <- 1000

 for (i in 1:epochs){
    loss <- nn_mse_loss()(F(X_tensor), y_tensor)
    optimizer$zero_grad()
    loss$backward()
    optimizer$step()
    }
```

Other things we may want to specify

1. Epochs
2. Gradient descent step
3. x,y as tensors
4. Learning rate
5. what p & q are

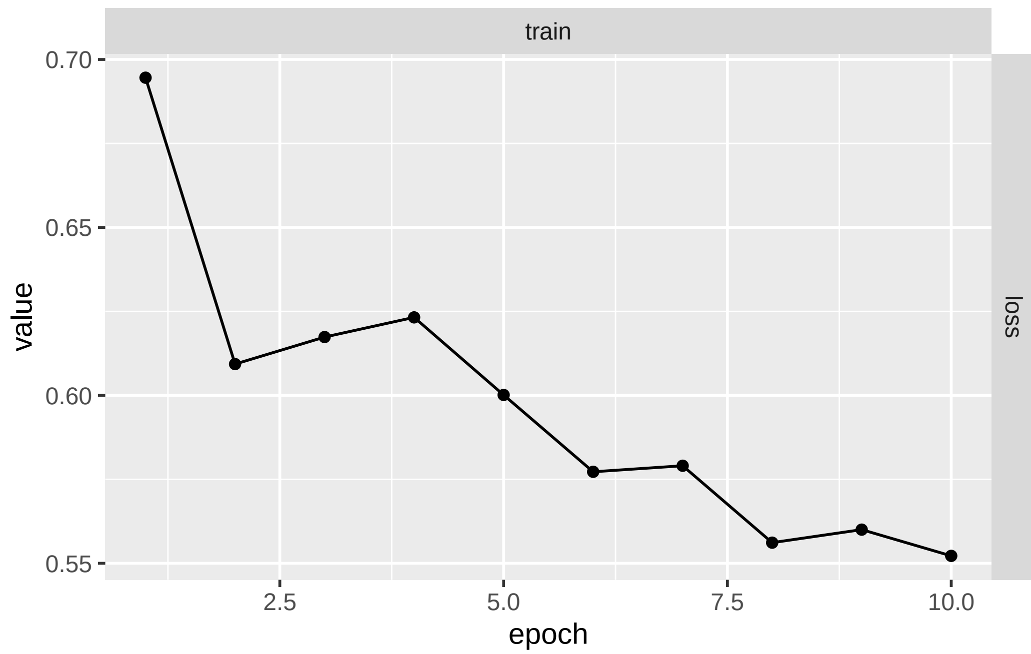
These things listed above are now added/specified in the code below

```
fit_nn <- nn_model %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(p = 2, q1 = 5, q2 = 7, q3 = 5) %>%
  set_opt_hparams(lr = 0.02) %>%
  # Fit the neural network
  # Have to change formatting b/c torch can only read matrices, not data frames
  fit(
    data = list(
      as.matrix(df[,-1]),
      as.numeric(df[,1]) - 1
    ),
    epochs = 10,
    verbose = TRUE
  )
```

```
Epoch 1/10
Train metrics: Loss: 0.6946
Epoch 2/10
Train metrics: Loss: 0.6093
Epoch 3/10
Train metrics: Loss: 0.6174
Epoch 4/10
Train metrics: Loss: 0.6232
Epoch 5/10
Train metrics: Loss: 0.6001
Epoch 6/10
Train metrics: Loss: 0.5772
Epoch 7/10
Train metrics: Loss: 0.579
Epoch 8/10
Train metrics: Loss: 0.5561
```

```
Epoch 9/10
Train metrics: Loss: 0.56
Epoch 10/10
Train metrics: Loss: 0.5522
```

```
# plots change in loss for the epochs specified
plot(fit_nn)
```



Based on the plot, we can see that loss does decrease a little bit, but it jumps around, both increasing and decreasing after the initial decrease.

The output of the Luz allows you to use the predict function

```
predict(fit_nn, xnew)
```

```
torch_tensor
 0.7034
 0.6928
 0.6820
 0.6710
 0.6599
```

```
 0.6485
 0.6370
 0.6253
 0.6135
 0.5735
 0.5256
 0.4772
 0.4293
 0.3827
 0.3381
 0.2962
 0.2575
 0.2264
 0.2026
 0.1806
 0.1698
 0.1639
 0.1653
 0.1666
 0.1679
 0.1693
 0.1706
 0.1720
 0.1734
 0.1747
... [the output was truncated (use n=-1 to disable)]
[ CPUFloatType{2500,1} ]
```

```r
predict(fit_nn, cbind(rnorm(10), rnorm(10))) %>% as.array
```

```
           [,1]
 [1,] 0.5580807
 [2,] 0.4615246
 [3,] 0.5967558
 [4,] 0.1521935
 [5,] 0.5666237
 [6,] 0.6143605
 [7,] 0.6944618
 [8,] 0.4637559
 [9,] 0.1690233
[10,] 0.4269857
```

Luz Validation Data and Metrics

Randomly selecting the indicies of 23 rows in the data frame without replacement

```
test_ind <- sample(1:nrow(df), 23, replace = FALSE)
```

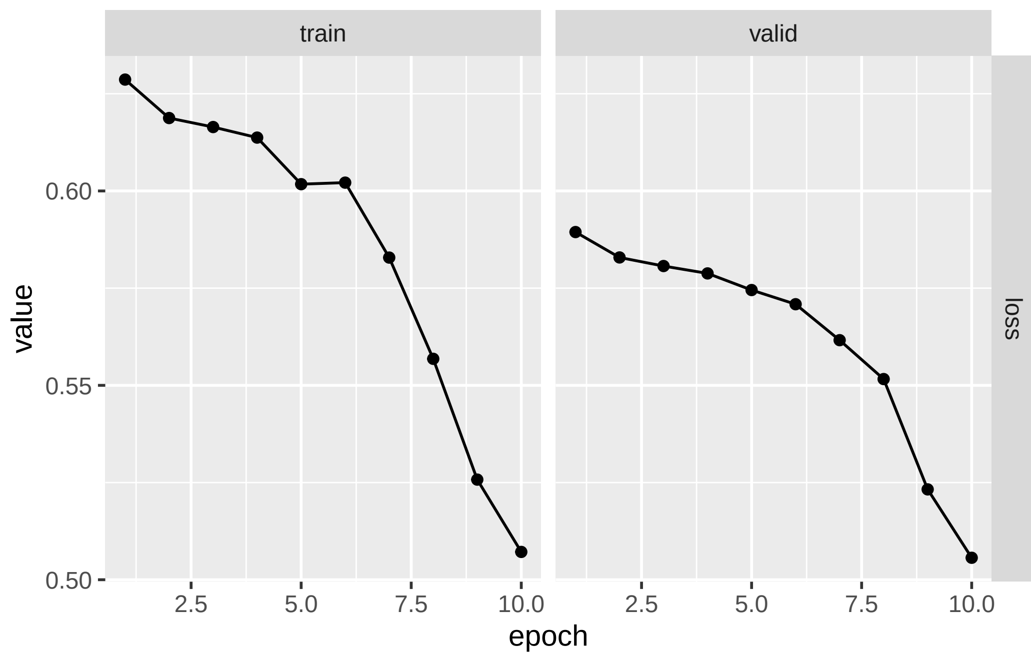
Using the luz code we created to validate the data

```
fit_nn <- nn_model %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam
  ) %>%
  set_hparams(p = 2, q1 = 5, q2 = 7, q3 = 5) %>%
  set_opt_hparams(lr = 0.02) %>%
  fit(
    data = list(
      as.matrix(df[-test_ind,-1]),
      as.numeric(df[-test_ind,1]) - 1
    ),
    valid_data = list(
      as.matrix(df[+test_ind, -1]),
      as.numeric(df[+test_ind,1]) - 1
    ),
    epochs = 10,
    verbose = TRUE
  )
```

```
Epoch 1/10
Train metrics: Loss: 0.6286
Valid metrics: Loss: 0.5894
Epoch 2/10
Train metrics: Loss: 0.6188
Valid metrics: Loss: 0.5829
Epoch 3/10
Train metrics: Loss: 0.6164
Valid metrics: Loss: 0.5807
Epoch 4/10
Train metrics: Loss: 0.6137
Valid metrics: Loss: 0.5788
Epoch 5/10
Train metrics: Loss: 0.6017
```

```
Valid metrics: Loss: 0.5745
Epoch 6/10
Train metrics: Loss: 0.6021
Valid metrics: Loss: 0.5709
Epoch 7/10
Train metrics: Loss: 0.5829
Valid metrics: Loss: 0.5616
Epoch 8/10
Train metrics: Loss: 0.5568
Valid metrics: Loss: 0.5516
Epoch 9/10
Train metrics: Loss: 0.5258
Valid metrics: Loss: 0.5232
Epoch 10/10
Train metrics: Loss: 0.5072
Valid metrics: Loss: 0.5057
```

```r
plot(fit_nn)
```



From this plot we can see that the loss generally decreases of for the training data and has a greater loss value than the test data throughout the epochs, but the test data switches between increasing and decreasing as the number of epochs increases.

- Luz has built in metrics(ex. accuracy, mse, ...) some of which are shown below

```
predicted <- torch_randn(100)
expected <- torch_randn(100)
metric <- luz_metric_binary_accuracy()

metric <-  metric$new()
metric$update(predicted, expected)
metric$compute()
```

```
[1] 0
```

This time we are specifiying the metrics we want to include, in the fitting of the neural network model

```
fit_nn <- nn_model %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam,
    # specifying metrics we want to use
    metrics = list(
      luz_metric_binary_accuracy(),
      luz_metric_binary_auroc()
    )
  ) %>%
  set_hparams(p = 2, q1 = 5, q2 = 7, q3 = 5) %>%
  set_opt_hparams(lr = 0.02) %>%
  fit(
    data = list(
      as.matrix(df[-test_ind,-1]),
      as.numeric(df[-test_ind,1]) - 1
    ),
    valid_data = list(
      as.matrix(df[+test_ind, -1]),
      as.numeric(df[+test_ind,1]) - 1
    ),
    epochs = 50,
    verbose = TRUE
  )
```

```
Epoch 1/50
```

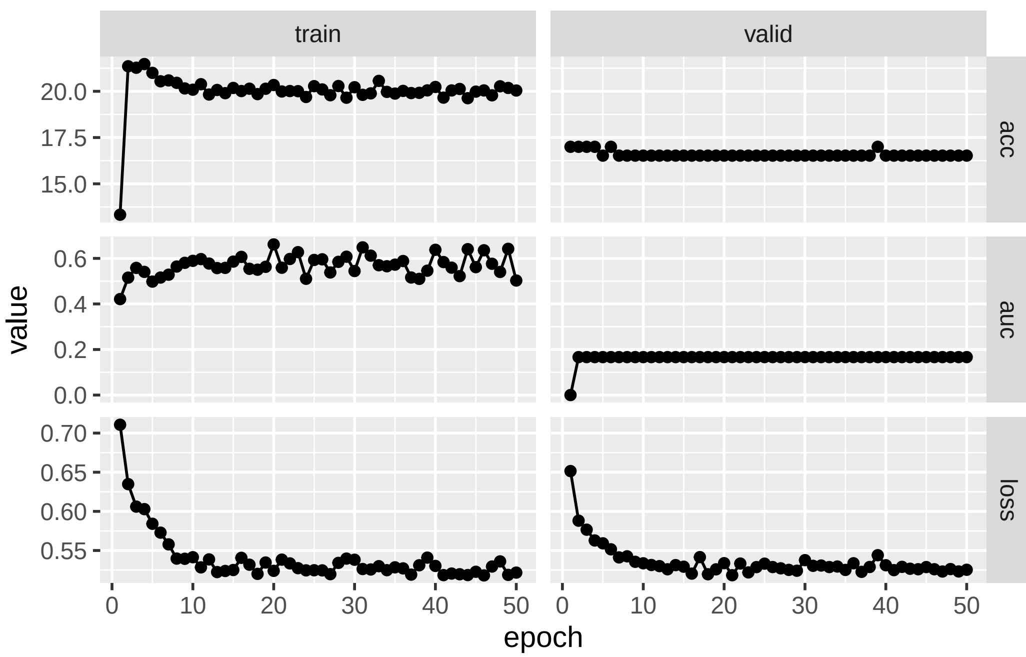
```
Train metrics: Loss: 0.7107 - Acc: 13.3285 - AUC: 0.4212
Valid metrics: Loss: 0.6515 - Acc: 17 - AUC: 0
Epoch 2/50
Train metrics: Loss: 0.6348 - Acc: 21.3538 - AUC: 0.5155
Valid metrics: Loss: 0.5881 - Acc: 17 - AUC: 0.1667
Epoch 3/50
Train metrics: Loss: 0.606 - Acc: 21.2744 - AUC: 0.5583
Valid metrics: Loss: 0.5766 - Acc: 17 - AUC: 0.1667
Epoch 4/50
Train metrics: Loss: 0.6029 - Acc: 21.4729 - AUC: 0.5408
Valid metrics: Loss: 0.5628 - Acc: 17 - AUC: 0.1667
Epoch 5/50
Train metrics: Loss: 0.5841 - Acc: 20.9964 - AUC: 0.4981
Valid metrics: Loss: 0.5592 - Acc: 16.5217 - AUC: 0.1667
Epoch 6/50
Train metrics: Loss: 0.5729 - Acc: 20.5415 - AUC: 0.5159
Valid metrics: Loss: 0.5515 - Acc: 17 - AUC: 0.1667
Epoch 7/50
Train metrics: Loss: 0.5578 - Acc: 20.5884 - AUC: 0.5284
Valid metrics: Loss: 0.5412 - Acc: 16.5217 - AUC: 0.1667
Epoch 8/50
Train metrics: Loss: 0.5397 - Acc: 20.4585 - AUC: 0.5642
Valid metrics: Loss: 0.5426 - Acc: 16.5217 - AUC: 0.1667
Epoch 9/50
Train metrics: Loss: 0.5394 - Acc: 20.1552 - AUC: 0.5805
Valid metrics: Loss: 0.5356 - Acc: 16.5217 - AUC: 0.1667
Epoch 10/50
Train metrics: Loss: 0.5414 - Acc: 20.0866 - AUC: 0.5892
Valid metrics: Loss: 0.5337 - Acc: 16.5217 - AUC: 0.1667
Epoch 11/50
Train metrics: Loss: 0.5285 - Acc: 20.3827 - AUC: 0.5969
Valid metrics: Loss: 0.5315 - Acc: 16.5217 - AUC: 0.1667
Epoch 12/50
Train metrics: Loss: 0.5386 - Acc: 19.8267 - AUC: 0.5773
Valid metrics: Loss: 0.53 - Acc: 16.5217 - AUC: 0.1667
Epoch 13/50
Train metrics: Loss: 0.5225 - Acc: 20.0722 - AUC: 0.5574
Valid metrics: Loss: 0.526 - Acc: 16.5217 - AUC: 0.1667
Epoch 14/50
Train metrics: Loss: 0.5239 - Acc: 19.8989 - AUC: 0.5583
Valid metrics: Loss: 0.5313 - Acc: 16.5217 - AUC: 0.1667
Epoch 15/50
Train metrics: Loss: 0.5251 - Acc: 20.1805 - AUC: 0.5858
```

```
Valid metrics: Loss: 0.5293 - Acc: 16.5217 - AUC: 0.1667
Epoch 16/50
Train metrics: Loss: 0.5406 - Acc: 20.0144 - AUC: 0.6066
Valid metrics: Loss: 0.5205 - Acc: 16.5217 - AUC: 0.1667
Epoch 17/50
Train metrics: Loss: 0.5319 - Acc: 20.1408 - AUC: 0.5543
Valid metrics: Loss: 0.5416 - Acc: 16.5217 - AUC: 0.1667
Epoch 18/50
Train metrics: Loss: 0.5201 - Acc: 19.8484 - AUC: 0.5503
Valid metrics: Loss: 0.5197 - Acc: 16.5217 - AUC: 0.1667
Epoch 19/50
Train metrics: Loss: 0.5346 - Acc: 20.1372 - AUC: 0.563
Valid metrics: Loss: 0.5259 - Acc: 16.5217 - AUC: 0.1667
Epoch 20/50
Train metrics: Loss: 0.524 - Acc: 20.3394 - AUC: 0.6615
Valid metrics: Loss: 0.5338 - Acc: 16.5217 - AUC: 0.1667
Epoch 21/50
Train metrics: Loss: 0.5383 - Acc: 19.9892 - AUC: 0.5589
Valid metrics: Loss: 0.5184 - Acc: 16.5217 - AUC: 0.1667
Epoch 22/50
Train metrics: Loss: 0.5335 - Acc: 20.0144 - AUC: 0.5975
Valid metrics: Loss: 0.5333 - Acc: 16.5217 - AUC: 0.1667
Epoch 23/50
Train metrics: Loss: 0.5274 - Acc: 20.0072 - AUC: 0.6273
Valid metrics: Loss: 0.522 - Acc: 16.5217 - AUC: 0.1667
Epoch 24/50
Train metrics: Loss: 0.5246 - Acc: 19.6931 - AUC: 0.5105
Valid metrics: Loss: 0.5288 - Acc: 16.5217 - AUC: 0.1667
Epoch 25/50
Train metrics: Loss: 0.5248 - Acc: 20.278 - AUC: 0.5932
Valid metrics: Loss: 0.5332 - Acc: 16.5217 - AUC: 0.1667
Epoch 26/50
Train metrics: Loss: 0.5246 - Acc: 20.1011 - AUC: 0.596
Valid metrics: Loss: 0.5289 - Acc: 16.5217 - AUC: 0.1667
Epoch 27/50
Train metrics: Loss: 0.5198 - Acc: 19.787 - AUC: 0.5382
Valid metrics: Loss: 0.5272 - Acc: 16.5217 - AUC: 0.1667
Epoch 28/50
Train metrics: Loss: 0.5342 - Acc: 20.2852 - AUC: 0.5848
Valid metrics: Loss: 0.5253 - Acc: 16.5217 - AUC: 0.1667
Epoch 29/50
Train metrics: Loss: 0.5397 - Acc: 19.6534 - AUC: 0.6074
Valid metrics: Loss: 0.5242 - Acc: 16.5217 - AUC: 0.1667
```

```
Epoch 30/50
Train metrics: Loss: 0.5382 - Acc: 20.2202 - AUC: 0.5444
Valid metrics: Loss: 0.5377 - Acc: 16.5217 - AUC: 0.1667
Epoch 31/50
Train metrics: Loss: 0.5263 - Acc: 19.8087 - AUC: 0.6484
Valid metrics: Loss: 0.5304 - Acc: 16.5217 - AUC: 0.1667
Epoch 32/50
Train metrics: Loss: 0.5258 - Acc: 19.8881 - AUC: 0.612
Valid metrics: Loss: 0.5308 - Acc: 16.5217 - AUC: 0.1667
Epoch 33/50
Train metrics: Loss: 0.53 - Acc: 20.5632 - AUC: 0.5701
Valid metrics: Loss: 0.5289 - Acc: 16.5217 - AUC: 0.1667
Epoch 34/50
Train metrics: Loss: 0.5249 - Acc: 19.9675 - AUC: 0.5655
Valid metrics: Loss: 0.5295 - Acc: 16.5217 - AUC: 0.1667
Epoch 35/50
Train metrics: Loss: 0.5285 - Acc: 19.8773 - AUC: 0.5726
Valid metrics: Loss: 0.5252 - Acc: 16.5217 - AUC: 0.1667
Epoch 36/50
Train metrics: Loss: 0.5272 - Acc: 20.0217 - AUC: 0.5885
Valid metrics: Loss: 0.5337 - Acc: 16.5217 - AUC: 0.1667
Epoch 37/50
Train metrics: Loss: 0.5192 - Acc: 19.9061 - AUC: 0.5162
Valid metrics: Loss: 0.5227 - Acc: 16.5217 - AUC: 0.1667
Epoch 38/50
Train metrics: Loss: 0.5311 - Acc: 19.917 - AUC: 0.51
Valid metrics: Loss: 0.5289 - Acc: 16.5217 - AUC: 0.1667
Epoch 39/50
Train metrics: Loss: 0.5409 - Acc: 20.0505 - AUC: 0.5459
Valid metrics: Loss: 0.544 - Acc: 17 - AUC: 0.1667
Epoch 40/50
Train metrics: Loss: 0.5303 - Acc: 20.2347 - AUC: 0.6377
Valid metrics: Loss: 0.5312 - Acc: 16.5217 - AUC: 0.1667
Epoch 41/50
Train metrics: Loss: 0.5186 - Acc: 19.657 - AUC: 0.5838
Valid metrics: Loss: 0.5249 - Acc: 16.5217 - AUC: 0.1667
Epoch 42/50
Train metrics: Loss: 0.5205 - Acc: 20.0469 - AUC: 0.5593
Valid metrics: Loss: 0.5291 - Acc: 16.5217 - AUC: 0.1667
Epoch 43/50
Train metrics: Loss: 0.5197 - Acc: 20.1264 - AUC: 0.5222
Valid metrics: Loss: 0.5267 - Acc: 16.5217 - AUC: 0.1667
Epoch 44/50
```

```
Train metrics: Loss: 0.5186 - Acc: 19.6282 - AUC: 0.6405
Valid metrics: Loss: 0.5261 - Acc: 16.5217 - AUC: 0.1667
Epoch 45/50
Train metrics: Loss: 0.5228 - Acc: 19.9819 - AUC: 0.5615
Valid metrics: Loss: 0.5288 - Acc: 16.5217 - AUC: 0.1667
Epoch 46/50
Train metrics: Loss: 0.5182 - Acc: 20.0469 - AUC: 0.6356
Valid metrics: Loss: 0.5261 - Acc: 16.5217 - AUC: 0.1667
Epoch 47/50
Train metrics: Loss: 0.5296 - Acc: 19.7834 - AUC: 0.5764
Valid metrics: Loss: 0.5232 - Acc: 16.5217 - AUC: 0.1667
Epoch 48/50
Train metrics: Loss: 0.5361 - Acc: 20.2708 - AUC: 0.5404
Valid metrics: Loss: 0.5263 - Acc: 16.5217 - AUC: 0.1667
Epoch 49/50
Train metrics: Loss: 0.5188 - Acc: 20.1805 - AUC: 0.6419
Valid metrics: Loss: 0.5232 - Acc: 16.5217 - AUC: 0.1667
Epoch 50/50
Train metrics: Loss: 0.5218 - Acc: 20.0433 - AUC: 0.5028
Valid metrics: Loss: 0.5254 - Acc: 16.5217 - AUC: 0.1667
```

```r
plot(fit_nn)
```



Based on the graphs, we can see that the accuracy on the test and train data are both low (with test accuracy ending up lower). Also, both a relatively steady at the beginning but decrease as the number of epochs gets higher. For the AUC value, both the train and test values end around the same value, but the train AUC value is more steady towards the end of the epochs while the test AUC jumps around. Finally, for the loss, both the test and train data decrease a significant amount of value, once again with the training loss having a more steady/consistent decrease while the test loss flucuates more.