Weekly Summary Template

Brady Miller

Table of contents

Tuesday, April 11

> **!** TIL
>
> Include a *very brief* summary of what you learnt in this class here.
> Today, I learnt the following concepts in class:
>
> 1. Application of neural networks to breast cancer & titanic datasets
> 2. What are DataLoaders and its strengths
> 3. How to use DataLoaders to determine the value of a handwritten number

Tuesday April 11

```
packages <- c(
    # Old packages
    "ISLR2",
    "dplyr",
    "tidyr",
    "readr",
    "purrr",
    "repr",
    "tidyverse",
    "kableExtra",
```

```
    "IRdisplay",
    # NEW
    "torch",
    "torchvision",
    "luz"
)

# renv::install(packages)
sapply(packages, require, character.only=TRUE)
```

```
Loading required package: ISLR2
```

```
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: tidyr
```

```
Loading required package: readr
```

```
Loading required package: purrr
```

```
Loading required package: repr
```

```
Loading required package: tidyverse
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
Loading required package: kableExtra


Attaching package: 'kableExtra'


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

    group_rows


Loading required package: IRdisplay

Loading required package: torch

Loading required package: torchvision

Loading required package: luz
```

```
      ISLR2       dplyr       tidyr       readr       purrr        repr
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
  tidyverse  kableExtra   IRdisplay       torch torchvision         luz
       TRUE        TRUE        TRUE        TRUE        TRUE        TRUE
```

Application of neural networks to breast cancer & Titanic datasets

```
url <- "https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv"

df <- read_csv(url) %>%
    mutate_if(\(x) is.character(x), as.factor) %>%
    mutate(y = Survived) %>%
    select(-c(Name, Survived)) %>%
    (\(x) {
```

```
        names(x) <- tolower(names(x))
        x
    })
```

```
Rows: 887 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): Name, Sex
dbl (6): Survived, Pclass, Age, Siblings/Spouses Aboard, Parents/Children Ab...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsi

# col_names <- c("id", "diagnosis", paste0("feat", 1:30))

# df <- read_csv(
#         url, col_names, col_types = cols()
#     ) %>%
#     select(-id) %>%
#     mutate(y = ifelse(diagnosis == "M", 1, 0)) %>%
#     select(-diagnosis)


# df %>% head
```

```
k <- 5

test_ind <- sample(
    1:nrow(df),
    floor(nrow(df) / k),
    replace=FALSE
)

df_train <- df[-test_ind, ]
df_test  <- df[test_ind, ]

nrow(df_train) + nrow(df_test) == nrow(df)
```

```
[1] TRUE
```

```r
fit_glm <- glm(
    y ~ .,
    df_train %>% mutate_at("y", factor),
    family = binomial()
)

glm_test <- predict(
    fit_glm,
    df_test,
    output = "response"
)

glm_preds <- ifelse(glm_test > 0.5, 1, 0)
table(glm_preds, df_test$y)
```

```
glm_preds   0   1
        0 108  25
        1   5  39
```

```r
NNet <- nn_module(
  initialize = function(p, q1, q2, q3) {
    self$hidden1 <- nn_linear(p, q1)
    self$hidden2 <- nn_linear(q1, q2)
    self$hidden3 <- nn_linear(q2, q3)
    self$output <- nn_linear(q3, 1)
    self$activation <- nn_relu()
    self$sigmoid <- nn_sigmoid()
  },

  forward = function(x) {
    x %>%
      self$hidden1() %>% self$activation() %>%
      self$hidden2() %>% self$activation() %>%
      self$hidden3() %>% self$activation() %>%
      self$output() %>% self$sigmoid()
  }
)
```

```r
# fitting a model without an intercept - when its zero, everything else is zero
M <- model.matrix(y ~ 0 + ., data = df_train)
```

```r
# model.matrix(y ~ ., data = df_train) [ ,-1]

fit_nn <- NNet %>%
    #
    # Setup the model
    #
    setup(
        loss = nn_bce_loss(),
        optimizer = optim_adam,
        metrics = list(
            luz_metric_accuracy()
        )
    ) %>%
    #
    # Set the hyperparameters
    #
    set_hparams(p=ncol(M), q1=256, q2=128, q3=64) %>%
    set_opt_hparams(lr=0.005) %>%
    #
    # Fit the model
    #
    fit(
        data = list(
            model.matrix(y ~ 0 + ., data = df_train),
            df_train %>% select(y) %>% as.matrix
        ),
        valid_data = list(
            model.matrix(y ~ 0 + ., data = df_test),
            df_test %>% select(y) %>% as.matrix
        ),
        epochs = 50,
        verbose = TRUE
    )
```

```
Epoch 1/50
Train metrics: Loss: 0.6671 - Acc: 12.4563
Valid metrics: Loss: 0.6743 - Acc: 11.1469
Epoch 2/50
Train metrics: Loss: 0.6391 - Acc: 12.4197
Valid metrics: Loss: 0.6032 - Acc: 11.1469
Epoch 3/50
Train metrics: Loss: 0.6081 - Acc: 12.4563
```

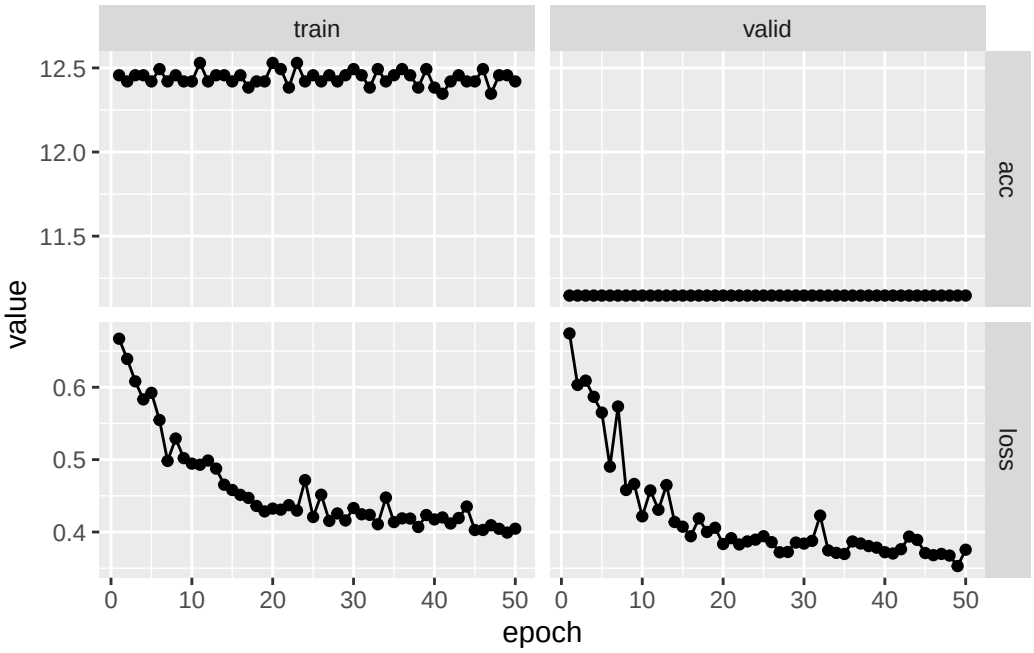
```
Valid metrics: Loss: 0.6091 - Acc: 11.1469
Epoch 4/50
Train metrics: Loss: 0.5832 - Acc: 12.4563
Valid metrics: Loss: 0.5868 - Acc: 11.1469
Epoch 5/50
Train metrics: Loss: 0.5923 - Acc: 12.4197
Valid metrics: Loss: 0.565 - Acc: 11.1469
Epoch 6/50
Train metrics: Loss: 0.5547 - Acc: 12.493
Valid metrics: Loss: 0.4903 - Acc: 11.1469
Epoch 7/50
Train metrics: Loss: 0.4982 - Acc: 12.4197
Valid metrics: Loss: 0.5734 - Acc: 11.1469
Epoch 8/50
Train metrics: Loss: 0.5292 - Acc: 12.4563
Valid metrics: Loss: 0.4581 - Acc: 11.1469
Epoch 9/50
Train metrics: Loss: 0.5021 - Acc: 12.4197
Valid metrics: Loss: 0.4664 - Acc: 11.1469
Epoch 10/50
Train metrics: Loss: 0.4944 - Acc: 12.4197
Valid metrics: Loss: 0.4216 - Acc: 11.1469
Epoch 11/50
Train metrics: Loss: 0.493 - Acc: 12.5296
Valid metrics: Loss: 0.4574 - Acc: 11.1469
Epoch 12/50
Train metrics: Loss: 0.4988 - Acc: 12.4197
Valid metrics: Loss: 0.4307 - Acc: 11.1469
Epoch 13/50
Train metrics: Loss: 0.4876 - Acc: 12.4563
Valid metrics: Loss: 0.4649 - Acc: 11.1469
Epoch 14/50
Train metrics: Loss: 0.4653 - Acc: 12.4563
Valid metrics: Loss: 0.4138 - Acc: 11.1469
Epoch 15/50
Train metrics: Loss: 0.4581 - Acc: 12.4197
Valid metrics: Loss: 0.4073 - Acc: 11.1469
Epoch 16/50
Train metrics: Loss: 0.4513 - Acc: 12.4563
Valid metrics: Loss: 0.3942 - Acc: 11.1469
Epoch 17/50
Train metrics: Loss: 0.4471 - Acc: 12.3831
Valid metrics: Loss: 0.4188 - Acc: 11.1469
```

```
Epoch 18/50
Train metrics: Loss: 0.4359 - Acc: 12.4197
Valid metrics: Loss: 0.4001 - Acc: 11.1469
Epoch 19/50
Train metrics: Loss: 0.4285 - Acc: 12.4197
Valid metrics: Loss: 0.4059 - Acc: 11.1469
Epoch 20/50
Train metrics: Loss: 0.4324 - Acc: 12.5296
Valid metrics: Loss: 0.3834 - Acc: 11.1469
Epoch 21/50
Train metrics: Loss: 0.4309 - Acc: 12.493
Valid metrics: Loss: 0.3915 - Acc: 11.1469
Epoch 22/50
Train metrics: Loss: 0.4371 - Acc: 12.3831
Valid metrics: Loss: 0.3828 - Acc: 11.1469
Epoch 23/50
Train metrics: Loss: 0.4296 - Acc: 12.5296
Valid metrics: Loss: 0.387 - Acc: 11.1469
Epoch 24/50
Train metrics: Loss: 0.4717 - Acc: 12.4197
Valid metrics: Loss: 0.3895 - Acc: 11.1469
Epoch 25/50
Train metrics: Loss: 0.4207 - Acc: 12.4563
Valid metrics: Loss: 0.3942 - Acc: 11.1469
Epoch 26/50
Train metrics: Loss: 0.4515 - Acc: 12.4197
Valid metrics: Loss: 0.386 - Acc: 11.1469
Epoch 27/50
Train metrics: Loss: 0.4154 - Acc: 12.4563
Valid metrics: Loss: 0.372 - Acc: 11.1469
Epoch 28/50
Train metrics: Loss: 0.4257 - Acc: 12.4197
Valid metrics: Loss: 0.3725 - Acc: 11.1469
Epoch 29/50
Train metrics: Loss: 0.416 - Acc: 12.4563
Valid metrics: Loss: 0.3854 - Acc: 11.1469
Epoch 30/50
Train metrics: Loss: 0.4332 - Acc: 12.493
Valid metrics: Loss: 0.384 - Acc: 11.1469
Epoch 31/50
Train metrics: Loss: 0.4245 - Acc: 12.4563
Valid metrics: Loss: 0.3877 - Acc: 11.1469
Epoch 32/50
```

```
Train metrics: Loss: 0.4237 - Acc: 12.3831
Valid metrics: Loss: 0.4227 - Acc: 11.1469
Epoch 33/50
Train metrics: Loss: 0.4107 - Acc: 12.493
Valid metrics: Loss: 0.3746 - Acc: 11.1469
Epoch 34/50
Train metrics: Loss: 0.4476 - Acc: 12.4197
Valid metrics: Loss: 0.3711 - Acc: 11.1469
Epoch 35/50
Train metrics: Loss: 0.4137 - Acc: 12.4563
Valid metrics: Loss: 0.3697 - Acc: 11.1469
Epoch 36/50
Train metrics: Loss: 0.4189 - Acc: 12.493
Valid metrics: Loss: 0.3869 - Acc: 11.1469
Epoch 37/50
Train metrics: Loss: 0.4186 - Acc: 12.4563
Valid metrics: Loss: 0.3841 - Acc: 11.1469
Epoch 38/50
Train metrics: Loss: 0.407 - Acc: 12.3831
Valid metrics: Loss: 0.3806 - Acc: 11.1469
Epoch 39/50
Train metrics: Loss: 0.4235 - Acc: 12.493
Valid metrics: Loss: 0.3784 - Acc: 11.1469
Epoch 40/50
Train metrics: Loss: 0.4174 - Acc: 12.3831
Valid metrics: Loss: 0.3721 - Acc: 11.1469
Epoch 41/50
Train metrics: Loss: 0.4201 - Acc: 12.3465
Valid metrics: Loss: 0.3703 - Acc: 11.1469
Epoch 42/50
Train metrics: Loss: 0.4118 - Acc: 12.4197
Valid metrics: Loss: 0.3762 - Acc: 11.1469
Epoch 43/50
Train metrics: Loss: 0.4193 - Acc: 12.4563
Valid metrics: Loss: 0.3937 - Acc: 11.1469
Epoch 44/50
Train metrics: Loss: 0.4352 - Acc: 12.4197
Valid metrics: Loss: 0.3891 - Acc: 11.1469
Epoch 45/50
Train metrics: Loss: 0.4028 - Acc: 12.4197
Valid metrics: Loss: 0.3709 - Acc: 11.1469
Epoch 46/50
Train metrics: Loss: 0.4029 - Acc: 12.493
```

```
Valid metrics: Loss: 0.368 - Acc: 11.1469
Epoch 47/50
Train metrics: Loss: 0.4094 - Acc: 12.3465
Valid metrics: Loss: 0.3698 - Acc: 11.1469
Epoch 48/50
Train metrics: Loss: 0.4044 - Acc: 12.4563
Valid metrics: Loss: 0.3675 - Acc: 11.1469
Epoch 49/50
Train metrics: Loss: 0.3992 - Acc: 12.4563
Valid metrics: Loss: 0.3529 - Acc: 11.1469
Epoch 50/50
Train metrics: Loss: 0.4046 - Acc: 12.4197
Valid metrics: Loss: 0.3755 - Acc: 11.1469
```

```r
plot(fit_nn)
```




```r
nn_test <- predict(
    fit_nn,
    model.matrix(y ~ . - 1, data = df_test)
)
# nn_test
```

```r
nn_preds <- ifelse(nn_test > 0.5, 1, 0)

table(nn_preds, df_test$y)
```

```
nn_preds   0   1
       0 108  24
       1   5  40
```

```r
mean(nn_preds == df_test$y)
```

```
[1] 0.8361582
```

```r
table(glm_preds, df_test$y)
```

```
glm_preds   0   1
        0 108  25
        1   5  39
```

```r
mean(glm_preds == df_test$y)
```

```
[1] 0.8305085
```

DataLoaders

- key component in the ML pipeline
- handle loading and preprocessing of data in efficient way for training and evaluating models
- make it easy to work with large datasets (ex. dataset with trillions of bits)
  - Loading data in smaller batches called chunks
- Advantages

1. Efficiency memory management
2. Parallelism
3. Preprocessing
4. Flexibility
5. Standardization

- involves partitioning data into smaller chunks (batches)
- inside one large epoch of gradient descent, it'll break up the data into batches and mini gradient descents (does gradient descent on every batch)
- IS DIFFERENT FROM CROSS VALIDATION
- training data is split up into smaller chunks - data is being cycled through during gradient descent
- if batches are fixed statically (first 100 are batch 1, next 100 are batch 2), this is going to introduce some bias into the gradient descent process, so you want to shuffle the data between every epoch
- DataLoader takes care of these issues

```
transform <- function(x) x %>%
  torch_tensor() %>%
  torch_flatten() %>%
  torch_div(255)


dir <- "./mnist"

train_ds <- mnist_dataset(
    root = dir,
    train = TRUE,
    download = TRUE,
    transform = transform
)
test_ds <- mnist_dataset(
    root = dir,
    train = FALSE,
    download = TRUE,
    transform = transform
)
```

```
typeof(train_ds)
```

```
[1] "environment"
```

```
length(train_ds)
```

```
[1] 60000
```

```r
# 42,000nd  observation is a 28x28 matrix
# one row is a 28x28 matrix
# is an image where a value specifies the intensity of the pixels in the 28x28 image
train_ds$data[42000, ,]
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
 [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [4,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [5,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [6,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [7,]    0    0    0    0    0    0    0    0    0     0     0     6    37
 [8,]    0    0    0    0    0    0    0    0    0     0   125   160   252
 [9,]    0    0    0    0    0    0    0    0    1   109   232   252   252
[10,]    0    0    0    0    0    0    0    0  125   252   252   252   252
[11,]    0    0    0    0    0    0    0    0   62   189   211   252   252
[12,]    0    0    0    0    0    0    0   21  206   252   190   252   168
[13,]    0    0    0    0    0    0   73  253  253   253   253   217     0
[14,]    0    0    0    0    0    0  115  252  252   252   148    30     0
[15,]    0    0    0    0    0    0  217  252  252   252    35     0     0
[16,]    0    0    0    0    0    0  217  252  252   252    35     0     0
[17,]    0    0    0    0    0  110  233  253  253   144     0    79   109
[18,]    0    0    0    0    0  253  252  252  252   237   217   242   252
[19,]    0    0    0    0    0  253  252  252  252   252   252   252   252
[20,]    0    0    0    0    0  170  252  252  252   252   252   252   252
[21,]    0    0    0    0    0    0  218  253  253   253   253   253   253
[22,]    0    0    0    0    0    0   72  231  252   252   252   252   252
[23,]    0    0    0    0    0    0    0   52   71    71    71    71    71
[24,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[25,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[26,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[27,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[28,]    0    0    0    0    0    0    0    0    0     0     0     0     0
      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
 [1,]     0     0     0     0     0     0     0     0     0     0     0     0
 [2,]     0     0     0     0     0     0     0     0     0     0     0     0
 [3,]     0     0     0     0     0     0     0     0     0     0     0     0
 [4,]     0     0     0     0     0     0     0     0     0     0     0     0
 [5,]     0     0     0     0     0     0     0     0     0     0     0     0
 [6,]     0     0     0     0     0     0     0    42   218   134   186     0
 [7,]   182    98    51     0     0     0    27   221   253   252   221    16
```

```
 [8,]   253   252   175   144     0     0    16   190   253   252   252   108
 [9,]   253   252   252   252     0     0     0     0   109   252   236    62
[10,]   253   252   200   179     0     0     0     0   109   252   215    42
[11,]   237    91    20     0     0     0     0    21   212   252   241   221
[12,]    62     0     0     0     0     0    21   206   253   252   252   252
[13,]     0     0     0     0     0    32   212   253   255   253   253   108
[14,]     0     0     0     0     0   115   252   252   253   252   220    15
[15,]     0     0    27   120   182   242   252   252   253   252   112     0
[16,]     0   125   221   252   253   252   252   252   253   128    31     0
[17,]   255   253   253   253   255   253   253   253   208    20     0     0
[18,]   253   252   252   252   253   252   252   210    20     0     0     0
[19,]   253   252   252   252   217   215   112    31     0     0     0     0
[20,]   253   252   252   252     0     0     0     0     0     0     0     0
[21,]   255   253   175    62     0     0     0     0     0     0     0     0
[22,]   119    35    10     0     0     0     0     0     0     0     0     0
[23,]     0     0     0     0     0     0     0     0     0     0     0     0
[24,]     0     0     0     0     0     0     0     0     0     0     0     0
[25,]     0     0     0     0     0     0     0     0     0     0     0     0
[26,]     0     0     0     0     0     0     0     0     0     0     0     0
[27,]     0     0     0     0     0     0     0     0     0     0     0     0
[28,]     0     0     0     0     0     0     0     0     0     0     0     0
      [,26] [,27] [,28]
 [1,]     0     0     0
 [2,]     0     0     0
 [3,]     0     0     0
 [4,]     0     0     0
 [5,]     0     0     0
 [6,]     0     0     0
 [7,]     0     0     0
 [8,]     0     0     0
 [9,]     0     0     0
[10,]     0     0     0
[11,]     0     0     0
[12,]     0     0     0
[13,]     0     0     0
[14,]     0     0     0
[15,]     0     0     0
[16,]     0     0     0
[17,]     0     0     0
[18,]     0     0     0
[19,]     0     0     0
[20,]     0     0     0
[21,]     0     0     0
```

```
[22,]    0    0    0
[23,]    0    0    0
[24,]    0    0    0
[25,]    0    0    0
[26,]    0    0    0
[27,]    0    0    0
[28,]    0    0    0
```

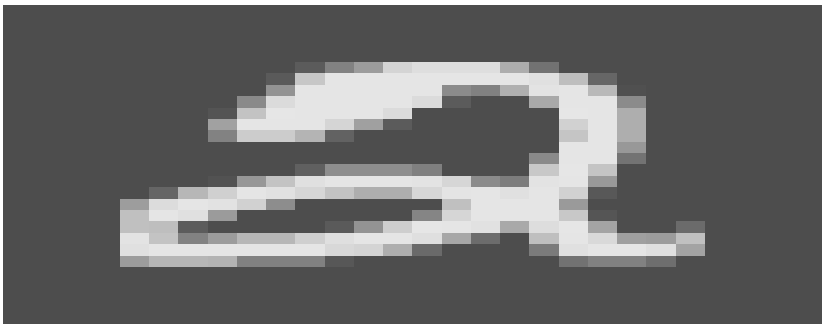
Using the data/DataLoaders to determine the value of a handwritten number

```r
# collection of {x_i, y_i} where i = 1, ranging to 10000
# x_i is 28x28 image of handwritten image
# every y_i {0,1,...,9}
options(repr.plot.width=10, repr.plot.height=10)

i <- sample(1:length(train_ds), 1)
x <- train_ds$data[i, ,] %>% t

image(x[1:28, 28:1], useRaster=TRUE, axes=FALSE, col=gray.colors(1000), main = train_ds$ta
```

2



- data frame with each row being a single image observation and its actual value
- p (# cols) = # of pixels in width^2 so you can have column for each pixel

```
# splitting up batches into size 128
# shuffling allows to negate any bias by shuffling observations for each epoch
train_dl <- dataloader(train_ds, batch_size = 1024, shuffle = TRUE)
test_dl <- dataloader(test_ds, batch_size = 1024)
```

```
NNet_10 <- nn_module(
  initialize = function(p, q1, q2, q3, o) {
    self$hidden1 <- nn_linear(p, q1)
    self$hidden2 <- nn_linear(q1, q2)
    self$hidden3 <- nn_linear(q2, q3)
    self$OUTPUT <- nn_linear(q3, o)
    self$activation <- nn_relu()
  },
  forward = function(x) {
    x %>%
      self$hidden1() %>%
      self$activation() %>%
      self$hidden2() %>%
      self$activation() %>%
      self$hidden3() %>%
      self$activation() %>%
      self$OUTPUT()
  }
)
```

Unable to get this chunk to work as it gives error that says it can't convert argument: input

```
fit_nn <- NNet_10 %>%
    #
    # Setup the model
    #
    setup(
        loss = nn_cross_entropy_loss(),
        optimizer = optim_adam,
        metrics = list(
            luz_metric_accuracy()
        )
    ) %>%
    #
    # Set the hyperparameters
    #
```

```
    set_hparams(p=28*28, q1=256, q2=128, q3=64, o=10) %>%
    #
    # Fit the model
    #
    fit(
        epochs = 10,
        data = train_dl,
        # valid_data = test_dl,
        verbose=TRUE
    )
```

```
Epoch 1/10
Train metrics: Loss: 1.0564 - Acc: 0.6784
Epoch 2/10
Train metrics: Loss: 0.3112 - Acc: 0.9107
Epoch 3/10
Train metrics: Loss: 0.2326 - Acc: 0.9329
Epoch 4/10
Train metrics: Loss: 0.1891 - Acc: 0.9456
Epoch 5/10
Train metrics: Loss: 0.1573 - Acc: 0.9541
Epoch 6/10
Train metrics: Loss: 0.1305 - Acc: 0.9617
Epoch 7/10
Train metrics: Loss: 0.1125 - Acc: 0.9666
Epoch 8/10
Train metrics: Loss: 0.0965 - Acc: 0.9714
Epoch 9/10
Train metrics: Loss: 0.0852 - Acc: 0.9751
Epoch 10/10
Train metrics: Loss: 0.0751 - Acc: 0.9775
```

```
NN10_preds <- fit_nn %>%
  predict(test_ds) %>%
  torch_argmax(dim = 2) %>%
  as_array()
```

```
mean(NN10_preds == test_ds$targets)
```

```
[1] 0.969
```

Table that displays what each digit is predicted as (correct predictions are on the diagonal)

```
table(NN10_preds - 1, test_ds$targets - 1)
```

```
     0    1    2    3    4    5    6    7    8    9
0  968    0    4    0    4    2    3    0    3    4
1    0 1118    3    0    0    0    3    8    0    6
2    0    2  994    3    1    0    0   10    2    2
3    3    1   12  984    1    9    1    8    7   13
4    0    0    3    0  956    1    5    2    3   14
5    4    1    2   10    1  869   16    1    6   14
6    2    3    1    0    6    4  925    0    1    0
7    1    0    6    4    2    0    0  986    2    6
8    2   10    7    9    3    6    5    3  949    9
9    0    0    0    0    8    1    0   10    1  941
```

```
caret::confusionMatrix(
  (NN10_preds - 1) %>% as.factor,
  (test_ds$targets - 1) %>% as.factor
)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  968    0    4    0    4    2    3    0    3    4
         1    0 1118    3    0    0    0    3    8    0    6
         2    0    2  994    3    1    0    0   10    2    2
         3    3    1   12  984    1    9    1    8    7   13
         4    0    0    3    0  956    1    5    2    3   14
         5    4    1    2   10    1  869   16    1    6   14
         6    2    3    1    0    6    4  925    0    1    0
         7    1    0    6    4    2    0    0  986    2    6
         8    2   10    7    9    3    6    5    3  949    9
         9    0    0    0    0    8    1    0   10    1  941

Overall Statistics

               Accuracy : 0.969
                 95% CI : (0.9654, 0.9723)
    No Information Rate : 0.1135
```

```
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9655

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity            0.9878   0.9850   0.9632   0.9743   0.9735   0.9742
Specificity            0.9978   0.9977   0.9978   0.9939   0.9969   0.9940
Pos Pred Value         0.9798   0.9824   0.9803   0.9471   0.9715   0.9405
Neg Pred Value         0.9987   0.9981   0.9958   0.9971   0.9971   0.9975
Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
Detection Rate         0.0968   0.1118   0.0994   0.0984   0.0956   0.0869
Detection Prevalence   0.0988   0.1138   0.1014   0.1039   0.0984   0.0924
Balanced Accuracy      0.9928   0.9914   0.9805   0.9841   0.9852   0.9841
                     Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.9656   0.9591   0.9743   0.9326
Specificity            0.9981   0.9977   0.9940   0.9978
Pos Pred Value         0.9820   0.9791   0.9462   0.9792
Neg Pred Value         0.9964   0.9953   0.9972   0.9925
Prevalence             0.0958   0.1028   0.0974   0.1009
Detection Rate         0.0925   0.0986   0.0949   0.0941
Detection Prevalence   0.0942   0.1007   0.1003   0.0961
Balanced Accuracy      0.9818   0.9784   0.9842   0.9652
```

Thursday, April 13

> **!** TIL
>
> Include a *very brief* summary of what you learnt in this class here.
> Today, I learnt the following concepts in class:
>
> 1. Supervised vs Unsupervised learning
> 2. Principle Component Analysis
> 3. Example of PCA

Supervised vs Unsupervised Learning

Supervised Learning

- when we have access to lablled data (given covariates and the responses)
- given this data our goal had been to predict y using $X_1, X_2, \dots, X_p$ as well as understand how each X_i influences the response y

Unsupervised Learning

- Don't have access to labelled data (only given observations and the covariates for each)
- Don't know ground truth
- Goal is to identify interesting relationships between $X_1, X_2, \dots, X_p$ which can be done through…

1. Dimension reduction

- can we discover subgroups of variables $X_1, X_2, ..., X_p$, which behave similarly?
- subsetting based on how similar they are

1. clustering

- can we discover subgroups of observations 1, 2, …, n which behave similarly
- do for observations
- Unsupervised learning is more 'popular' as it is easier to obtain unlabelled data than labeled data as this requires someone to assign the proper response variable

Principle Component Analysis

- PCA is one of the methods used to tackle the issue of dimension reduction
- objective - given variables$(X_1, X_2, ..., X_p)$, PCA produces a low dimension representation of the dataset (creating lower dimension dataset)
- it compresses data from each row/observation into 2 variables

Step 1

The first principle component Z_1 is the normalized linear combination of the features

$$Z_1 = v_{11}X_1 + v_{21}X_2 + \cdots + v_{p1}X_p$$

such that Z_1 has the largest possible variance and the sum of $v_{p,i}^2 = 1$

Step 2 The first principle component Z_2 is the normalized linear combination of the features

$$Z_2 = v_{12}X_1 + v_{22}X_2 + \cdots + v_{p2}X_p$$

such that V_2 is orthogonal to V_1, Z_2 has the largest possible variance and the sum of $v_{p,2}^2 = 1$

- This step occurs up until the qth principal component Z_q such that Z_q has the largest possible variance V_q is orthogonal to span$(V_1, V_2, \dots, V_{q-1})$ and $v_{p,2}^2 = 1$

Example of PCA

```
data <- tibble(x1 = rnorm(100, mean = 0, sd = 1),
               x2 = x1 + rnorm(100, mean = 0, sd = 0.1)
               )

pca <- princomp(data, cor = TRUE)
summary(pca)
```

```
Importance of components:
                          Comp.1      Comp.2
Standard deviation     1.4125396 0.068788066
Proportion of Variance 0.9976341 0.002365899
Cumulative Proportion  0.9976341 1.000000000
```

```
pca$loadings
```

```
Loadings:
   Comp.1 Comp.2
x1  0.707  0.707
x2  0.707 -0.707

               Comp.1 Comp.2
SS loadings       1.0    1.0
Proportion Var    0.5    0.5
Cumulative Var    0.5    1.0
```