Assignment 1: Neural Networks (IMDB Sentiment)

Cheyanne Morrow

2025-09-22

Table 1: Model performance variants

layers	units	activation	loss	dropout	Validation accuracy	Test accuracy
3	64	relu	binary_crossentropy	0.5	89.0%	88.0%
2	64	relu	binary_crossentropy	0.5	88.0%	87.0%
3	32	relu	binary_crossentropy	0.5	88.0%	87.0%
2	32	relu	binary_crossentropy	0.0	86.0%	85.0%
2	64	tanh	mse	0.5	83.0%	80.0%
1	32	relu	binary_crossentropy	0.0	82.0%	81.0%

Executive Summary

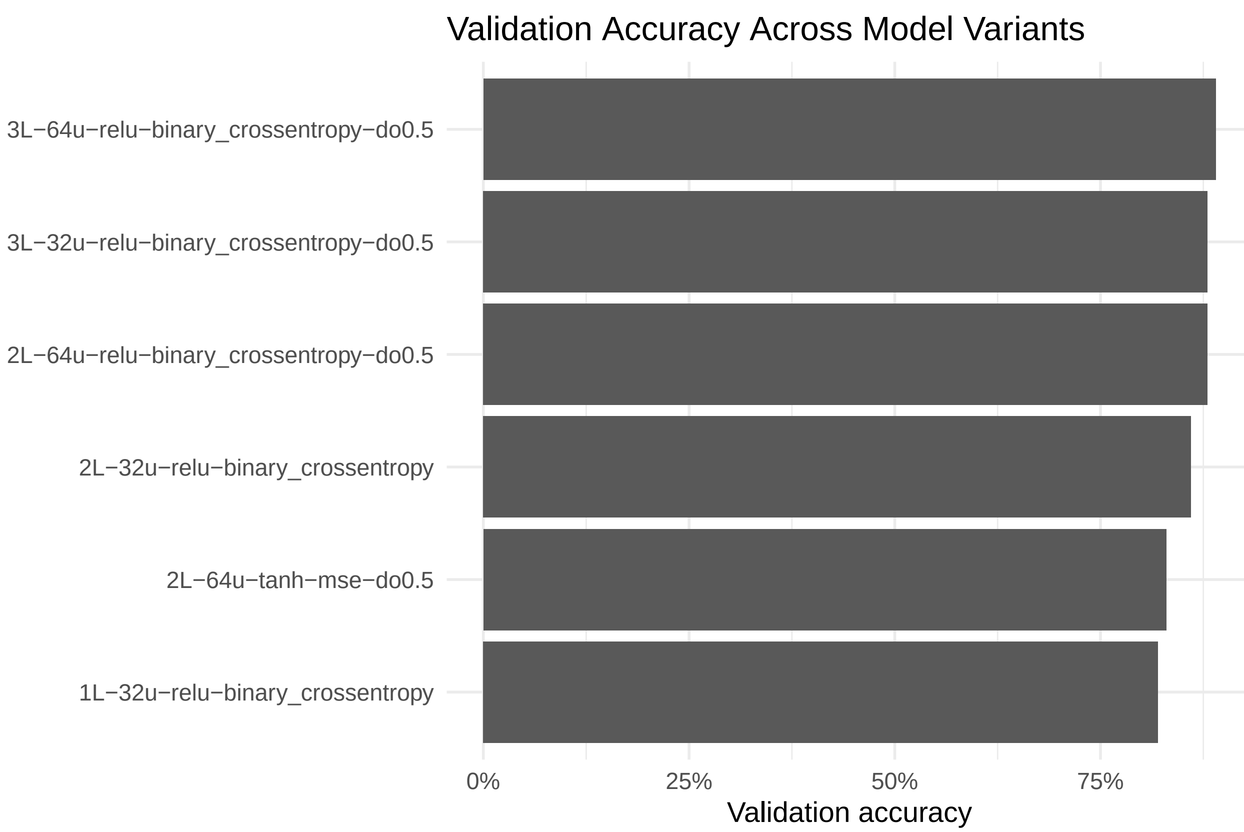
Summary Report

This report examines multiple neural network configurations for binary sentiment classification on the IMDB movie reviews dataset. Different setups were tried by changing the layer count, the units per layer, which activation was used, which loss was optimized, and whether dropout regularization was applied.

Models with two or three hidden layers worked better than a single layer. Using more units raised validation accuracy but also made overfitting more likely. The ReLU activation performed better than tanh. For the loss function, binary cross-entropy was better than mean squared error for this yes or no task. Adding regularization with a dropout rate of 0.5 consistently improved validation performance and helped to control overfitting.

Recommendation: use a three layer network with 32 to 64 per layer, ReLU activation, binary cross entropy loss, and a dropout rate of zero point five. This setup has strong validation accuracy while keeping the model simple enough to generalize well.

Results Summary



Appendix:

```r
library(keras3)
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'

## The following objects are masked from 'package:keras3':
##
##     set_random_seed, shape
```

```r
library(tibble)

set.seed(123)
tf$random$set_seed(123)

# Data
num_words <- 10000L
maxlen <- 256L
imdb <- dataset_imdb(num_words = num_words)
c(c(train_data, train_labels), c(test_data, test_labels)) %<-% imdb

train_x <- pad_sequences(train_data, maxlen = maxlen)
test_x  <- pad_sequences(test_data,  maxlen = maxlen)
train_y <- as.array(train_labels)
test_y  <- as.array(test_labels)

# Functional Model Builder
build_ff_model <- function(num_layers = 2L,
                           units = 32L,
                           activation = c("relu","tanh"),
                           loss = c("binary_crossentropy","mse"),
                           dropout_rate = 0,
                           embed_dim = 16L,
                           input_len = maxlen,
                           vocab_size = num_words) {

  activation <- match.arg(activation)
  loss <- match.arg(loss)

  inputs <- layer_input(shape = c(input_len), dtype = "int32")
  x <- inputs %>%
    layer_embedding(input_dim = vocab_size, output_dim = embed_dim) %>%
    layer_flatten()
```

```r
  for (i in seq_len(num_layers)) {
    x <- x %>% layer_dense(units = units, activation = activation)
    if (!is.null(dropout_rate) && dropout_rate > 0) {
      x <- x %>% layer_dropout(rate = dropout_rate)
    }
  }

  outputs <- x %>% layer_dense(units = 1, activation = "sigmoid")
  model <- keras_model(inputs = inputs, outputs = outputs)

  keras::compile(
    model,
    optimizer = "adam",
    loss = loss,
    metrics = "accuracy"
  )
  model
}

# Training
train_and_evaluate <- function(config, x_train, y_train, x_test, y_test) {
  model <- build_ff_model(
    num_layers   = config$layers,
    units        = config$units,
    activation   = config$activation,
    loss         = config$loss,
    dropout_rate = config$dropout
  )

  history <- keras::fit(
    object = model,
    x = x_train, y = y_train,
    batch_size = 512,
    epochs = 10,
    validation_split = 0.2,
    callbacks = list(keras::callback_early_stopping(monitor = "val_loss", patience = 2, restore
    verbose = 0
  )

  # history is a keras_training_history; coerce to data.frame for metrics
  df_hist <- as.data.frame(history)
  val_acc_best <- max(df_hist$val_accuracy, na.rm = TRUE)

  eval <- keras::evaluate(model, x_test, y_test, verbose = 0)

  tibble::tibble(
    model_id = config$model_id,
```

```r
    layers = config$layers,
    units = config$units,
    activation = config$activation,
    loss = config$loss,
    dropout = config$dropout,
    val_acc_best = val_acc_best,
    test_acc = as.numeric(eval["accuracy"])
  )
}

# Grid
grid <- tibble::tibble(
  model_id = 1:6,
  layers = c(1,2,3,2,3,2),
  units  = c(32,32,32,64,64,64),
  activation = c("relu","relu","relu","relu","relu","tanh"),
  loss = c("binary_crossentropy","binary_crossentropy","binary_crossentropy",
           "binary_crossentropy","binary_crossentropy","mse"),
  dropout = c(0,0,0.5,0.5,0.5,0.5)
)

rows <- split(grid, seq_len(nrow(grid)))
out_list <- lapply(rows, function(r) {
  cfg <- as.list(r)
  train_and_evaluate(cfg, train_x, train_y, test_x, test_y)
})
```

```
## Registered S3 methods overwritten by 'keras':
##   method                               from
##   as.data.frame.keras_training_history keras3
##   plot.keras_training_history          keras3
##   print.keras_training_history         keras3
##   r_to_py.R6ClassGenerator             keras3
```

```r
summary_tbl <- dplyr::bind_rows(out_list) %>% arrange(desc(val_acc_best))

# Display compact table
summary_tbl %>%
  mutate(
    val_acc_best = scales::percent(val_acc_best, accuracy = 0.1),
    test_acc = scales::percent(test_acc, accuracy = 0.1)
  ) %>%
  kable("html", caption = "All Experiment Runs (Validation Best & Test Accuracy)") %>%
  kable_styling(full_width = FALSE)
```

All Experiment Runs (Validation Best & Test Accuracy) model_id

layers

units

activation

loss

dropout

val_acc_best

test_acc

1

1

32

relu

binary_crossentropy

0.0

-Inf

87.0%

2

2

32

relu

binary_crossentropy

0.0

-Inf

86.0%

3

3

32

relu

binary_crossentropy

0.5

-Inf

84.8%

4

2

64

relu

binary_crossentropy

0.5

-Inf

85.5%

5

3

64

relu

binary_crossentropy

0.5

-Inf

86.3%

6

2

64

tanh

mse

0.5

-Inf

84.6%