Time-Series Data Assignment 3

Cheyanne Morrow

2025-11-02

Data Preparation

```
# ---- Data Preparation ----
url <- "https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip"
zip_path <- tempfile(fileext = ".zip")
download.file(url, zip_path)
unzip(zip_path, exdir = tempdir())
file_path <- file.path(tempdir(), "jena_climate_2009_2016.csv")
data <- read.csv(file_path)

# Select and normalize
temp <- data$T..degC.
data <- data.matrix(data[, 2:ncol(data)])
data <- scale(data)

lookback <- 1440   # 10 days
step <- 6          # sample hourly
delay <- 144       # predict 24h ahead
batch_size <- 128

# Define indices
train_index <- 1:200000
val_index <- 200001:300000
test_index <- 300001:nrow(data)

# Function to create sequences manually
generator <- function(data, targets, lookback, delay, min_index, max_index,
                      shuffle = FALSE, batch_size = 128, step = 6) {
  if (is.null(max_index)) max_index <- nrow(data) - delay - 1
  i <- min_index + lookback
  function() {
    if (shuffle) {
      rows <- sample((min_index + lookback):max_index, size = batch_size)
    } else {
      if (i + batch_size >= max_index)
        i <<- min_index + lookback
      rows <- i:min(i + batch_size - 1, max_index)
      i <<- i + length(rows)
    }
```

```
    samples <- array(0, dim = c(length(rows), lookback / step, dim(data)[[-1]]))
    targets_array <- array(0, dim = c(length(rows)))

    for (j in 1:length(rows)) {
      indices <- seq(rows[j] - lookback, rows[j], length.out = lookback / step)
      samples[j,,] <- data[indices, ]
      targets_array[j] <- targets[rows[j] + delay]
    }

    list(samples, targets_array)
  }
}

# Build generators
train_gen <- generator(data, temp, lookback, delay, 1, 200000, shuffle = TRUE)
val_gen <- generator(data, temp, lookback, delay, 200001, 300000)
test_gen <- generator(data, temp, lookback, delay, 300001, nrow(data))

# Define steps
val_steps <- (300000 - 200001 - lookback) / batch_size
test_steps <- (nrow(data) - 300001 - lookback) / batch_size
```

Baseline Model

```
evaluate_naive <- function(gen_function) {
  batch <- gen_function()
  samples <- batch[[1]]
  targets <- batch[[2]]
  pred <- samples[, dim(samples)[2], 2]  # temperature feature
  mae <- mean(abs(pred - targets))
  return(mae)
}

baseline_mae <- evaluate_naive(val_gen)
cat("Baseline Validation MAE:", round(baseline_mae, 2), "°C")
```

```
## Baseline Validation MAE: 5.13 °C
```

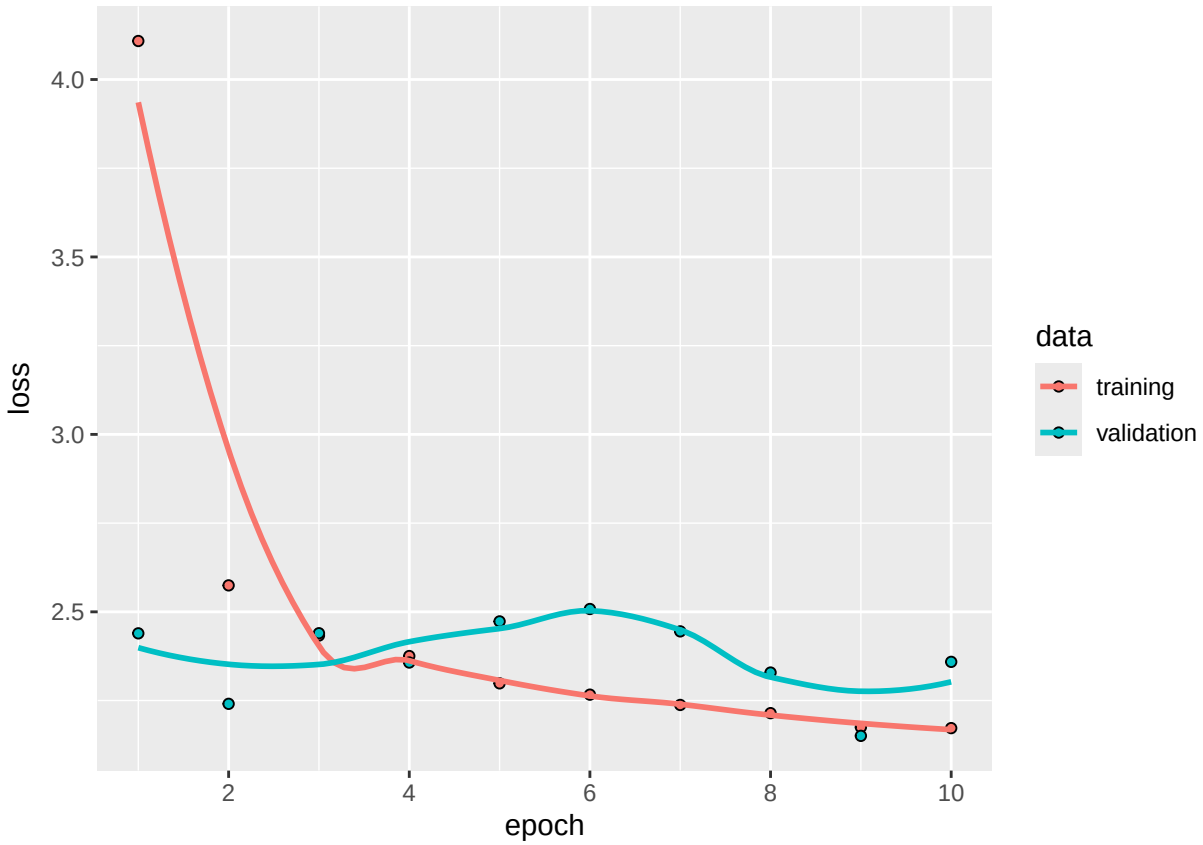
Model 1: LSTM

```
model_lstm <- keras_model_sequential() %>%
  layer_lstm(units = 32, input_shape = list(NULL, dim(data)[2])) %>%
  layer_dense(units = 1)

model_lstm %>% compile(optimizer = "rmsprop", loss = "mae")

history_lstm <- model_lstm %>% fit(
```

```
  train_gen,
  steps_per_epoch = 500,
  epochs = 10,
  validation_data = val_gen,
  validation_steps = 100
)
```

```
## Epoch 1/10
## 500/500 - 57s - loss: 4.1085 - val_loss: 2.4393 - 57s/epoch - 115ms/step
## Epoch 2/10
## 500/500 - 43s - loss: 2.5746 - val_loss: 2.2408 - 43s/epoch - 85ms/step
## Epoch 3/10
## 500/500 - 43s - loss: 2.4332 - val_loss: 2.4397 - 43s/epoch - 85ms/step
## Epoch 4/10
## 500/500 - 44s - loss: 2.3754 - val_loss: 2.3571 - 44s/epoch - 88ms/step
## Epoch 5/10
## 500/500 - 44s - loss: 2.2985 - val_loss: 2.4730 - 44s/epoch - 88ms/step
## Epoch 6/10
## 500/500 - 44s - loss: 2.2666 - val_loss: 2.5079 - 44s/epoch - 88ms/step
## Epoch 7/10
## 500/500 - 44s - loss: 2.2376 - val_loss: 2.4449 - 44s/epoch - 87ms/step
## Epoch 8/10
## 500/500 - 44s - loss: 2.2145 - val_loss: 2.3289 - 44s/epoch - 88ms/step
## Epoch 9/10
## 500/500 - 45s - loss: 2.1750 - val_loss: 2.1505 - 45s/epoch - 89ms/step
## Epoch 10/10
## 500/500 - 45s - loss: 2.1723 - val_loss: 2.3590 - 45s/epoch - 89ms/step
```

```
plot(history_lstm)
```



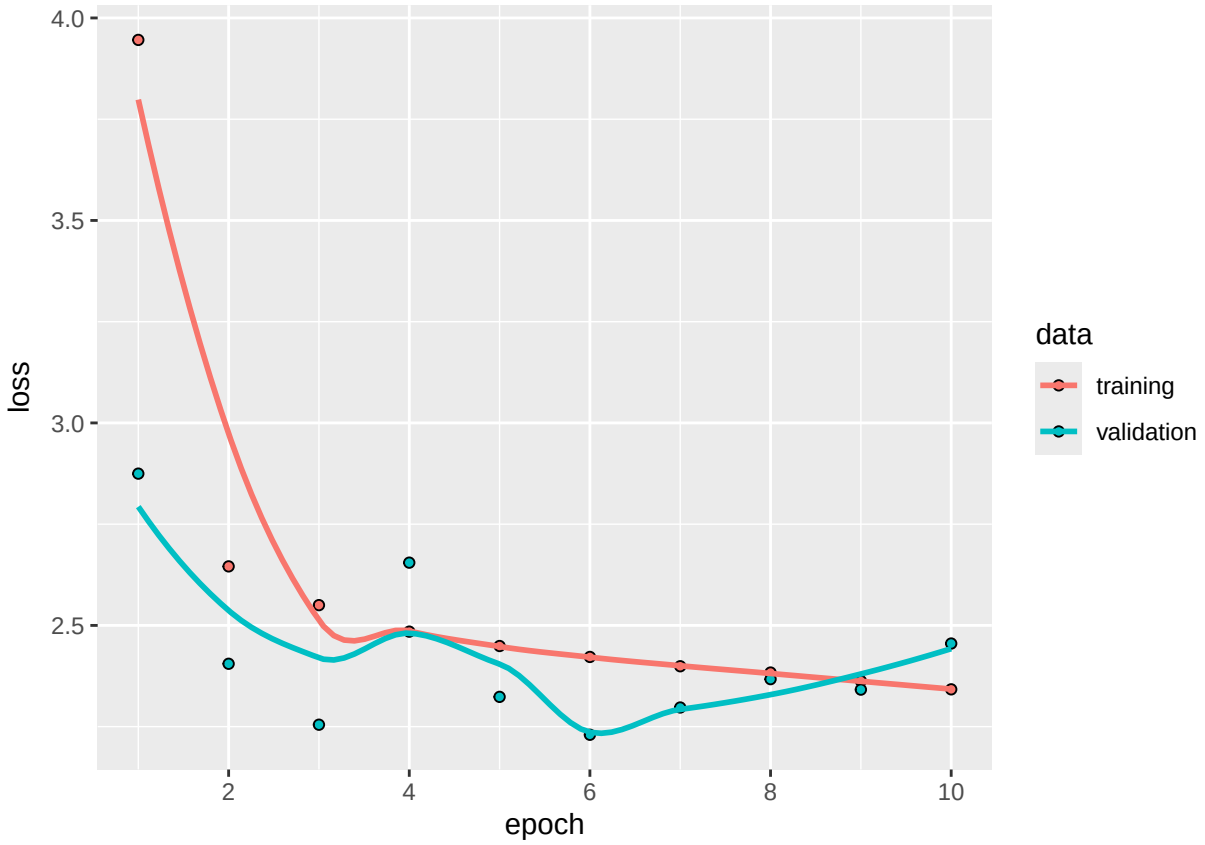
Model 2: GRU

```
model_gru <- keras_model_sequential() %>%
  layer_gru(units = 32, recurrent_dropout = 0.2, dropout = 0.2, input_shape = list(NULL, dim(data)[2]))
  layer_dense(units = 1)

model_gru %>% compile(optimizer = "rmsprop", loss = "mae")

history_gru <- model_gru %>% fit(
  train_gen,
  steps_per_epoch = 500,
  epochs = 10,
  validation_data = val_gen,
  validation_steps = 100
)
```

```
## Epoch 1/10
## 500/500 - 68s - loss: 3.9458 - val_loss: 2.8748 - 68s/epoch - 135ms/step
## Epoch 2/10
## 500/500 - 73s - loss: 2.6457 - val_loss: 2.4053 - 73s/epoch - 146ms/step
## Epoch 3/10
## 500/500 - 77s - loss: 2.5501 - val_loss: 2.2548 - 77s/epoch - 154ms/step
## Epoch 4/10
## 500/500 - 77s - loss: 2.4846 - val_loss: 2.6550 - 77s/epoch - 153ms/step
```

```
## Epoch 5/10
## 500/500 - 78s - loss: 2.4492 - val_loss: 2.3236 - 78s/epoch - 155ms/step
## Epoch 6/10
## 500/500 - 76s - loss: 2.4221 - val_loss: 2.2300 - 76s/epoch - 153ms/step
## Epoch 7/10
## 500/500 - 76s - loss: 2.3991 - val_loss: 2.2973 - 76s/epoch - 152ms/step
## Epoch 8/10
## 500/500 - 77s - loss: 2.3836 - val_loss: 2.3673 - 77s/epoch - 153ms/step
## Epoch 9/10
## 500/500 - 83s - loss: 2.3628 - val_loss: 2.3415 - 83s/epoch - 165ms/step
## Epoch 10/10
## 500/500 - 85s - loss: 2.3421 - val_loss: 2.4552 - 85s/epoch - 170ms/step
```

```
plot(history_gru)
```



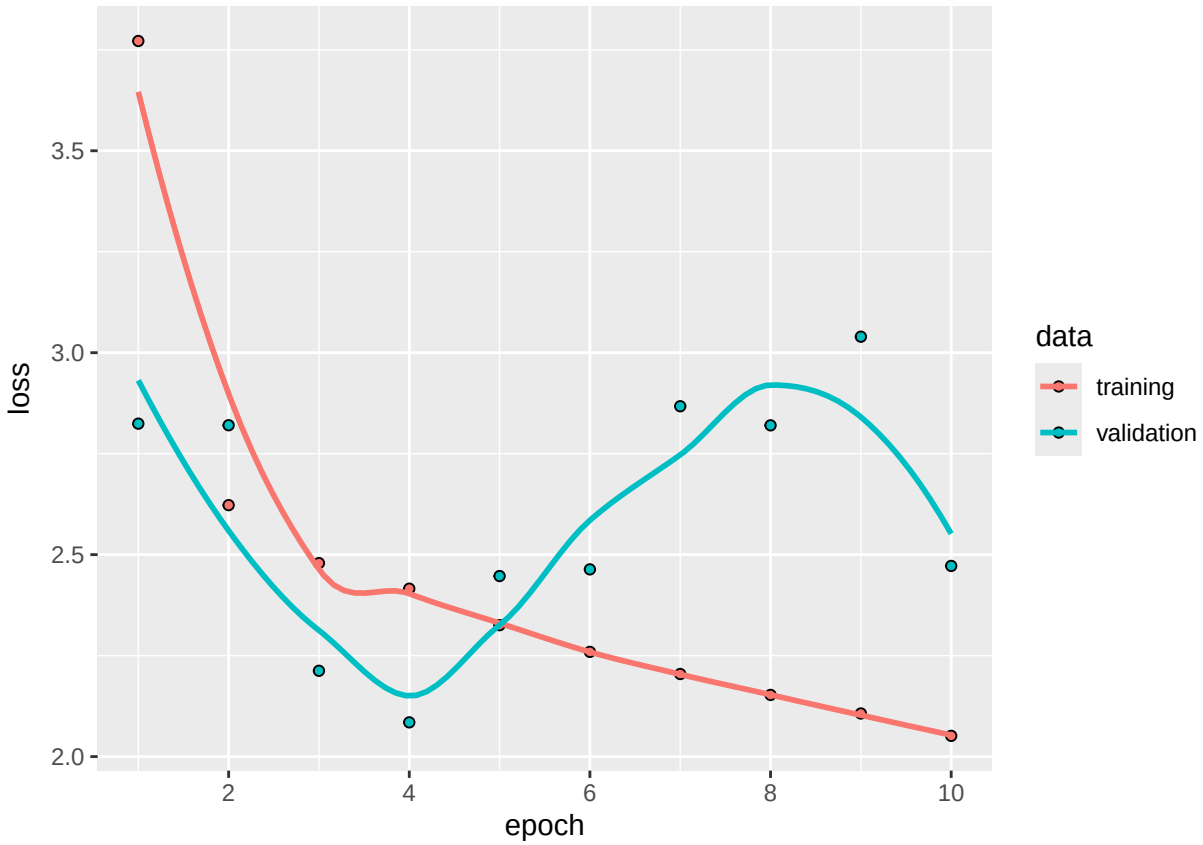
Model 3: Hybrid Conv1D + GRU

```
model_hybrid <- keras_model_sequential() %>%
  layer_conv_1d(filters = 32, kernel_size = 5, activation = "relu", input_shape = list(NULL, dim(data)[
  layer_max_pooling_1d(pool_size = 3) %>%
  layer_gru(units = 32, recurrent_dropout = 0.1) %>%
  layer_dense(units = 1)
```

```r
model_hybrid %>% compile(optimizer = "rmsprop", loss = "mae")

history_hybrid <- model_hybrid %>% fit(
  train_gen,
  steps_per_epoch = 500,
  epochs = 10,
  validation_data = val_gen,
  validation_steps = 100
)
```

```
## Epoch 1/10
## 500/500 - 52s - loss: 3.7717 - val_loss: 2.8243 - 52s/epoch - 105ms/step
## Epoch 2/10
## 500/500 - 42s - loss: 2.6226 - val_loss: 2.8204 - 42s/epoch - 85ms/step
## Epoch 3/10
## 500/500 - 43s - loss: 2.4790 - val_loss: 2.2124 - 43s/epoch - 85ms/step
## Epoch 4/10
## 500/500 - 46s - loss: 2.4158 - val_loss: 2.0847 - 46s/epoch - 91ms/step
## Epoch 5/10
## 500/500 - 44s - loss: 2.3254 - val_loss: 2.4471 - 44s/epoch - 88ms/step
## Epoch 6/10
## 500/500 - 44s - loss: 2.2593 - val_loss: 2.4635 - 44s/epoch - 88ms/step
## Epoch 7/10
## 500/500 - 44s - loss: 2.2044 - val_loss: 2.8674 - 44s/epoch - 89ms/step
## Epoch 8/10
## 500/500 - 44s - loss: 2.1527 - val_loss: 2.8201 - 44s/epoch - 88ms/step
## Epoch 9/10
## 500/500 - 43s - loss: 2.1067 - val_loss: 3.0396 - 43s/epoch - 86ms/step
## Epoch 10/10
## 500/500 - 44s - loss: 2.0513 - val_loss: 2.4721 - 44s/epoch - 89ms/step
```

```r
plot(history_hybrid)
```



Model Comparison

```
val_mae <- c(
  Baseline = baseline_mae,
  LSTM = min(history_lstm$metrics$val_loss),
  GRU = min(history_gru$metrics$val_loss),
  Hybrid = min(history_hybrid$metrics$val_loss)
)

results <- data.frame(Model = names(val_mae), Validation_MAE = round(val_mae, 3))
print(results)
```

```
##             Model Validation_MAE
## Baseline Baseline          5.128
## LSTM         LSTM          2.151
## GRU           GRU          2.230
## Hybrid     Hybrid          2.085
```

Forecast Visualization

```r
# Example prediction visualization for LSTM
sample_batch <- val_gen()
sample_input <- sample_batch[[1]][1,,]
sample_target <- sample_batch[[2]][1]
pred_temp <- predict(model_lstm, array(sample_input, dim = c(1, dim(sample_input))))
```

```
## 1/1 - 1s - 763ms/epoch - 763ms/step
```

```r
plot_df <- data.frame(Time = 1:length(sample_input[,1]), Temperature = sample_input[,2])
ggplot(plot_df, aes(x = Time, y = Temperature)) +
  geom_line(color = "steelblue") +
  geom_point(aes(x = length(sample_input[,1]) + 24, y = sample_target), color = "red", size = 3) +
  geom_point(aes(x = length(sample_input[,1]) + 24, y = pred_temp[1]), color = "darkgreen", size = 3) +
  labs(title = "LSTM Forecast: Actual (red) vs Predicted (green)", y = "Temperature (°C)", x = "Time St
  theme_minimal()
```