Bonus_Lab1_Intro_DL_NN

Sharon Paruwani

2024-11-20

```
library(reticulate)
```

```
Warning: package 'reticulate' was built under R version 4.4.2
```

```
use_condaenv("r-tensorflow", required = TRUE)
```

Exercise 1: Building a Simple Neural Network in R (Keras)

```
library(keras)
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.4.2
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- 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
```

```r
# Create a simple sequential model
library(tensorflow)
use_condaenv("r-tensorflow", required = TRUE)
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'relu', input_shape = c(10)) %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'sigmoid')

model %>% compile(
  optimizer = 'adam',
  loss = 'binary_crossentropy',
  metrics = c('accuracy')
)

summary(model)
```

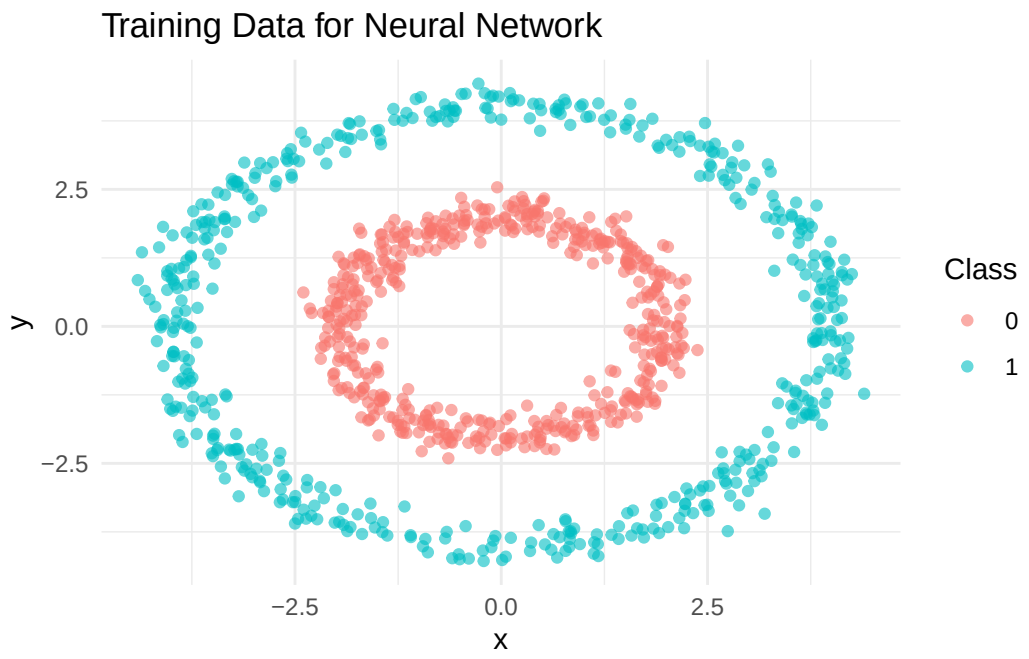
```
Model: "sequential"
_________________________________________________________________________________
 Layer (type)                        Output Shape                      Param #
=================================================================================
 dense_2 (Dense)                     (None, 64)                        704
 dense_1 (Dense)                     (None, 32)                        2080
 dense (Dense)                       (None, 1)                         33
=================================================================================
Total params: 2817 (11.00 KB)
Trainable params: 2817 (11.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________
```

Exercise 2: Building an Intuitive Neural Network in R (Keras)

building an intuitive neural network

```r
library(keras)
library(tidyverse)
library(ggplot2)

# Create synthetic dataset
set.seed(123)
```

```r
n_points <- 1000

# Generate two circular clusters
create_circular_data <- function(n_points) {
  # Create cluster 1
  theta1 <- runif(n_points/2, 0, 2*pi)
  r1 <- rnorm(n_points/2, mean=2, sd=0.2)
  x1 <- r1 * cos(theta1)
  y1 <- r1 * sin(theta1)

  # Create cluster 2
  theta2 <- runif(n_points/2, 0, 2*pi)
  r2 <- rnorm(n_points/2, mean=4, sd=0.2)
  x2 <- r2 * cos(theta2)
  y2 <- r2 * sin(theta2)

  # Combine data
  data.frame(
    x = c(x1, x2),
    y = c(y1, y2),
    label = c(rep(0, n_points/2), rep(1, n_points/2))
  )
}

# Generate and visualize data
df <- create_circular_data(n_points)

# Visualize the data
ggplot(df, aes(x=x, y=y, color=factor(label))) +
  geom_point(alpha=0.6) +
  theme_minimal() +
  labs(title="Training Data for Neural Network",
       color="Class")
```




```r
# Create and compile model
model <- keras_model_sequential() %>%
  layer_dense(units=8, activation='relu', input_shape=c(2)) %>%
  layer_dense(units=4, activation='relu') %>%
  layer_dense(units=1, activation='sigmoid')

model %>% compile(
  optimizer = 'adam',
  loss = 'binary_crossentropy',
  metrics = c('accuracy')
)

# Prepare data for training
x_train <- as.matrix(df[, c("x", "y")])
y_train <- df$label

# Train model
history <- model %>% fit(
  x_train, y_train,
  epochs = 20,
  validation_split = 0.2,
  verbose = 1
)
```

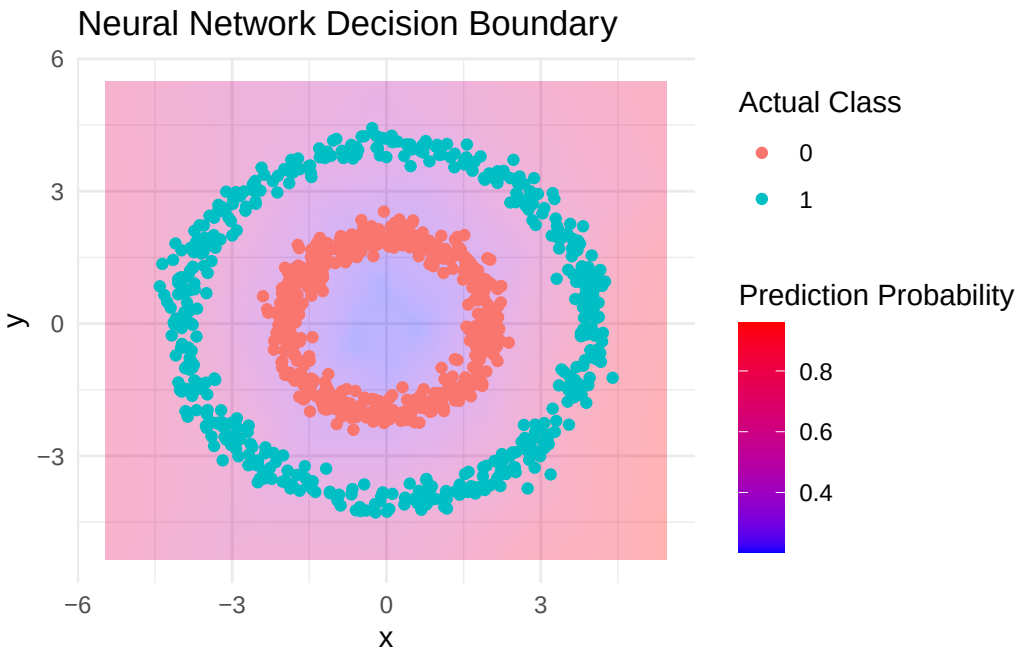
```
Epoch 1/20

 1/25 [>.............................] - ETA: 10s - loss: 1.1323 - accuracy: 0.4688
25/25 [==============================] - 1s 8ms/step - loss: 0.9147 - accuracy: 0.6112 - val
Epoch 2/20

 1/25 [>.............................] - ETA: 0s - loss: 0.9381 - accuracy: 0.5625
13/25 [==============>...............] - ETA: 0s - loss: 0.8463 - accuracy: 0.6106
25/25 [==============================] - 0s 7ms/step - loss: 0.8592 - accuracy: 0.6075 - val
Epoch 3/20

 1/25 [>.............................] - ETA: 0s - loss: 0.8991 - accuracy: 0.5938
25/25 [==============================] - 0s 2ms/step - loss: 0.8212 - accuracy: 0.6050 - val
Epoch 4/20

 1/25 [>.............................] - ETA: 0s - loss: 0.8304 - accuracy: 0.6250
25/25 [==============================] - 0s 2ms/step - loss: 0.7912 - accuracy: 0.6100 - val
Epoch 5/20

 1/25 [>.............................] - ETA: 0s - loss: 0.7564 - accuracy: 0.5625
25/25 [==============================] - 0s 2ms/step - loss: 0.7676 - accuracy: 0.6225 - val
Epoch 6/20

 1/25 [>.............................] - ETA: 0s - loss: 0.8719 - accuracy: 0.6250
25/25 [==============================] - 0s 2ms/step - loss: 0.7476 - accuracy: 0.6388 - val
Epoch 7/20

 1/25 [>.............................] - ETA: 0s - loss: 0.6753 - accuracy: 0.7812
25/25 [==============================] - 0s 2ms/step - loss: 0.7294 - accuracy: 0.6513 - val
Epoch 8/20

 1/25 [>.............................] - ETA: 0s - loss: 0.7326 - accuracy: 0.6562
25/25 [==============================] - 0s 2ms/step - loss: 0.7125 - accuracy: 0.6425 - val
Epoch 9/20

 1/25 [>.............................] - ETA: 0s - loss: 0.7289 - accuracy: 0.6875
25/25 [==============================] - 0s 2ms/step - loss: 0.6961 - accuracy: 0.6538 - val
Epoch 10/20

 1/25 [>.............................] - ETA: 0s - loss: 0.6588 - accuracy: 0.6562
25/25 [==============================] - 0s 2ms/step - loss: 0.6800 - accuracy: 0.6712 - val
Epoch 11/20
```

```
 1/25 [>.............................] - ETA: 0s - loss: 0.7010 - accuracy: 0.6250
25/25 [==============================] - 0s 2ms/step - loss: 0.6601 - accuracy: 0.6938 - val
Epoch 12/20

 1/25 [>.............................] - ETA: 0s - loss: 0.6518 - accuracy: 0.6562
25/25 [==============================] - 0s 2ms/step - loss: 0.6382 - accuracy: 0.7013 - val
Epoch 13/20

 1/25 [>.............................] - ETA: 0s - loss: 0.5885 - accuracy: 0.8125
25/25 [==============================] - 0s 3ms/step - loss: 0.6176 - accuracy: 0.7088 - val
Epoch 14/20

 1/25 [>.............................] - ETA: 0s - loss: 0.5818 - accuracy: 0.7188
25/25 [==============================] - 0s 3ms/step - loss: 0.5972 - accuracy: 0.7350 - val
Epoch 15/20

 1/25 [>.............................] - ETA: 0s - loss: 0.5757 - accuracy: 0.7500
25/25 [==============================] - 0s 3ms/step - loss: 0.5755 - accuracy: 0.7825 - val
Epoch 16/20

 1/25 [>.............................] - ETA: 0s - loss: 0.5557 - accuracy: 0.6875
25/25 [==============================] - 0s 3ms/step - loss: 0.5527 - accuracy: 0.8462 - val
Epoch 17/20

 1/25 [>.............................] - ETA: 0s - loss: 0.5322 - accuracy: 0.8750
25/25 [==============================] - 0s 3ms/step - loss: 0.5292 - accuracy: 0.8625 - val
Epoch 18/20

 1/25 [>.............................] - ETA: 0s - loss: 0.5120 - accuracy: 0.9375
25/25 [==============================] - 0s 3ms/step - loss: 0.5051 - accuracy: 0.8963 - val
Epoch 19/20

 1/25 [>.............................] - ETA: 0s - loss: 0.4482 - accuracy: 0.9688
25/25 [==============================] - 0s 2ms/step - loss: 0.4809 - accuracy: 0.9137 - val
Epoch 20/20

 1/25 [>.............................] - ETA: 0s - loss: 0.4674 - accuracy: 0.9375
25/25 [==============================] - 0s 2ms/step - loss: 0.4559 - accuracy: 0.9262 - val
```

```r
# Visualize decision boundaries
create_decision_boundary <- function(model, df) {
  # Create grid of points
```

```
  x_range <- seq(min(df$x) - 1, max(df$x) + 1, length.out = 100)
  y_range <- seq(min(df$y) - 1, max(df$y) + 1, length.out = 100)
  grid <- expand.grid(x = x_range, y = y_range)

  # Get predictions
  predictions <- predict(model, as.matrix(grid))

  # Create visualization
  grid$pred <- as.vector(predictions)

  ggplot() +
    geom_raster(data=grid, aes(x=x, y=y, fill=pred), alpha=0.3) +
    geom_point(data=df, aes(x=x, y=y, color=factor(label))) +
    scale_fill_gradient(low="blue", high="red") +
    theme_minimal() +
    labs(title="Neural Network Decision Boundary",
         fill="Prediction Probability",
         color="Actual Class")
}

# Visualize the decision boundary
create_decision_boundary(model, df)
```

```
313/313 - 0s - 267ms/epoch - 853us/step
```



Exercise 3: Building a Practical Application of a Neural Network in R (Keras)

practical application of a neural network in R

```r
# Load MNIST dataset
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y

# Preprocess data
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_train <- x_train / 255

# Create model
model <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
```

```
model
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 128)               100480
 dropout (Dropout)           (None, 128)               0
 dense_6 (Dense)             (None, 10)                1290
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Exercise 4: Building and Visualizing a Simple Neural Network in R (Keras)

simple neural network in Python

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Create a similar model in Python
model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(10,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Exercise 5: Building and Visualizing an Intuitive Neural Network in Python

building an intuitive neural network

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import seaborn as sns

# Create synthetic dataset
np.random.seed(123)
n_points = 1000

def create_circular_data(n_points):
    # Create cluster 1
    theta1 = np.random.uniform(0, 2*np.pi, n_points//2)
    r1 = np.random.normal(2, 0.2, n_points//2)
    x1 = r1 * np.cos(theta1)
    y1 = r1 * np.sin(theta1)

    # Create cluster 2
    theta2 = np.random.uniform(0, 2*np.pi, n_points//2)
    r2 = np.random.normal(4, 0.2, n_points//2)
    x2 = r2 * np.cos(theta2)
    y2 = r2 * np.sin(theta2)

    X = np.vstack([np.column_stack((x1, y1)),
                   np.column_stack((x2, y2))])
    y = np.hstack([np.zeros(n_points//2),
                   np.ones(n_points//2)])

    return X, y

# Generate data
X, y = create_circular_data(n_points)

# Visualize data
plt.figure(figsize=(10, 8))
plt.scatter(X[y==0, 0], X[y==0, 1], label='Class 0', alpha=0.6)
```

```
plt.scatter(X[y==1, 0], X[y==1, 1], label='Class 1', alpha=0.6)
plt.title('Training Data for Neural Network')
plt.legend()
plt.show()
```