# <Text and Sequence Data Assignment

## Loading Libarary and Data

```
library(keras)

## Warning: package 'keras' was built under R version 4.3.2

library(tensorflow)

## Warning: package 'tensorflow' was built under R version 4.3.2

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.3.3
```

### Loading the IMDB Text data:

```
imdb <- dataset_imdb(num_words = 10000)
```

Data is loaded by containing the top 10000 words of the sample data.

```
train_data <- imdb$train$x
train_labels <- imdb$train$y
#loading test data for validation
test_data <- imdb$test$x
test_labels <- imdb$test$y
```

## Data Preprocessing

Now from the training and Testing/validating data we need to cuttoff the reviews after 150 words.

```
#limiting the training data upto 150 reviews
train_data <- pad_sequences(train_data, maxlen = 150)

# Preprocess the test data
test_data <- pad_sequences(test_data, maxlen = 150)
```

Restricting the training to 100

```
#restricting training samples to 100
train_data <- train_data[1:100, ]
train_labels <- train_labels[1:100]
```

Preparing the Validation data for 10000 samples to evaluate the model.

```
# Limit the number of validation samples to 10000
val_data <- test_data[1:10000, ]
val_labels <- test_labels[1:10000]
```

## Model Building

### Using Embedding layer

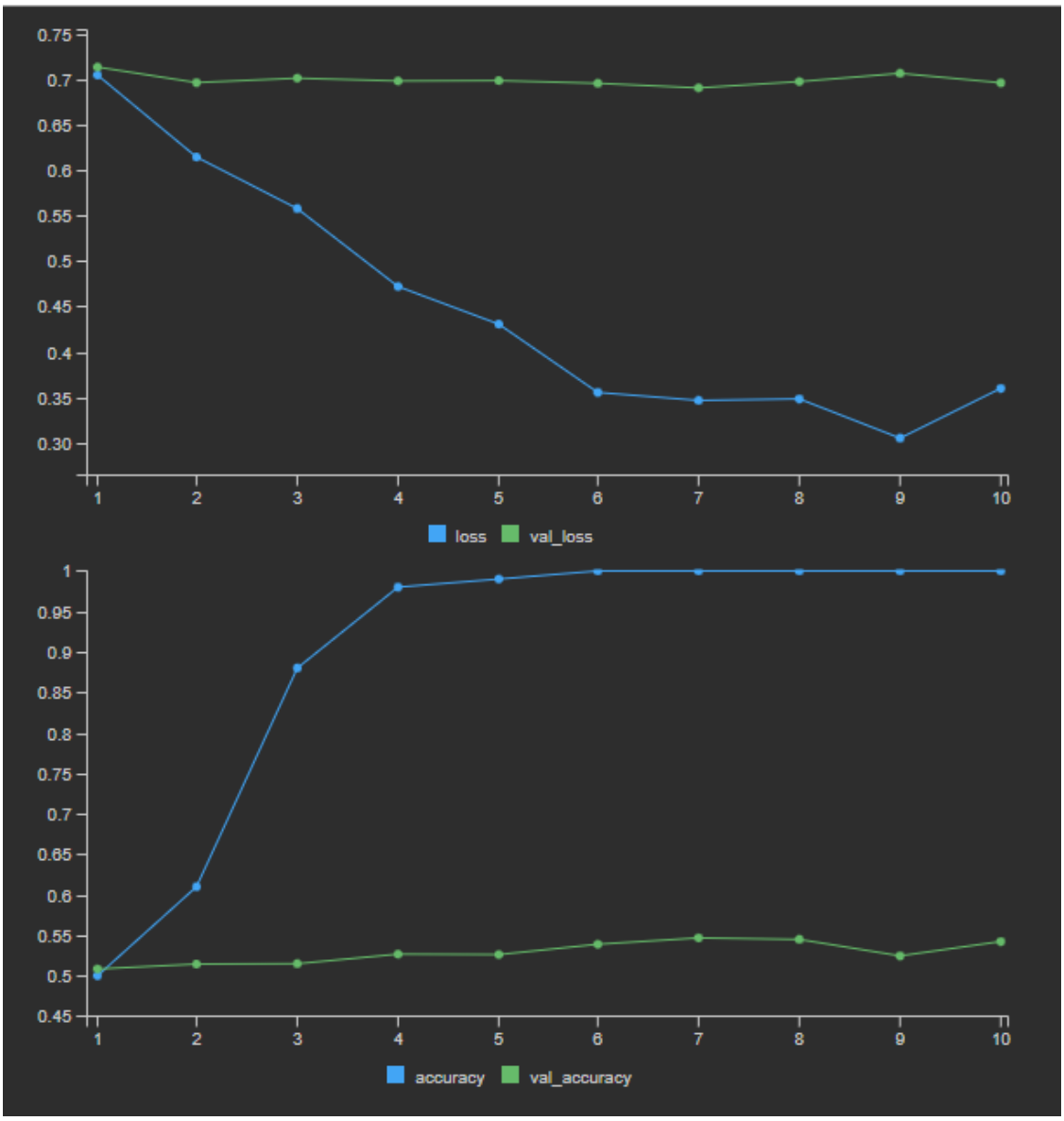
The embedding layer learns the word embeddings as part of the overall model training process. It starts with random word embeddings and then adjusts these embeddings to minimize the loss function of the model.

```
#building the model
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 32) %>%
  layer_simple_rnn(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

Now fitting the model using training data and label and validating the model using validation data.

```
# Train the model
history <- model %>% fit(
  train_data,
  train_labels,
  epochs = 10,
  batch_size = 512,
  validation_data = list(val_data, val_labels)
)

## Epoch 1/10
## 1/1 - 4s - loss: 0.7027 - accuracy: 0.4800 - val_loss: 0.7033 -
val_accuracy: 0.5085 - 4s/epoch - 4s/step
## Epoch 2/10
## 1/1 - 1s - loss: 0.6069 - accuracy: 0.6300 - val_loss: 0.6932 -
val_accuracy: 0.5229 - 1s/epoch - 1s/step
## Epoch 3/10
## 1/1 - 1s - loss: 0.5233 - accuracy: 0.9700 - val_loss: 0.6964 -
val_accuracy: 0.5240 - 1s/epoch - 1s/step
## Epoch 4/10
## 1/1 - 1s - loss: 0.4205 - accuracy: 1.0000 - val_loss: 0.6996 -
val_accuracy: 0.5188 - 1s/epoch - 1s/step
## Epoch 5/10
```

```
## 1/1 - 1s - loss: 0.4077 - accuracy: 1.0000 - val_loss: 0.7000 -
val_accuracy: 0.5241 - 753ms/epoch - 753ms/step
## Epoch 6/10
## 1/1 - 1s - loss: 0.3356 - accuracy: 1.0000 - val_loss: 0.7133 -
val_accuracy: 0.5100 - 787ms/epoch - 787ms/step
## Epoch 7/10
## 1/1 - 1s - loss: 0.3337 - accuracy: 1.0000 - val_loss: 0.7046 -
val_accuracy: 0.5149 - 1s/epoch - 1s/step
## Epoch 8/10
## 1/1 - 1s - loss: 0.3368 - accuracy: 1.0000 - val_loss: 0.7010 -
val_accuracy: 0.5250 - 943ms/epoch - 943ms/step
## Epoch 9/10
## 1/1 - 1s - loss: 0.3338 - accuracy: 1.0000 - val_loss: 0.6985 -
val_accuracy: 0.5291 - 965ms/epoch - 965ms/step
## Epoch 10/10
## 1/1 - 1s - loss: 0.2903 - accuracy: 1.0000 - val_loss: 0.6952 -
val_accuracy: 0.5304 - 987ms/epoch - 987ms/step
```




```
# Get the validation accuracy of the last epoch
val_accs <-
history$metrics$val_accuracy[length(history$metrics$val_accuracy)]

# Print the validation accuracies
print(val_accs)

## [1] 0.5304
```

The accuracy of the last epoch is 0.5423.

Recurrent Neural Network (RNN) model correctly predicted the class of approximately 54.23% of the validation samples.

## Changing the number of Samples for Embedding Layer:

```
# Define the number of training samples to test
num_samples <- c(20, 50, 60, 70, 100)

# Initialize a vector to store the validation accuracies
val_acc <- numeric(length(num_samples))
```

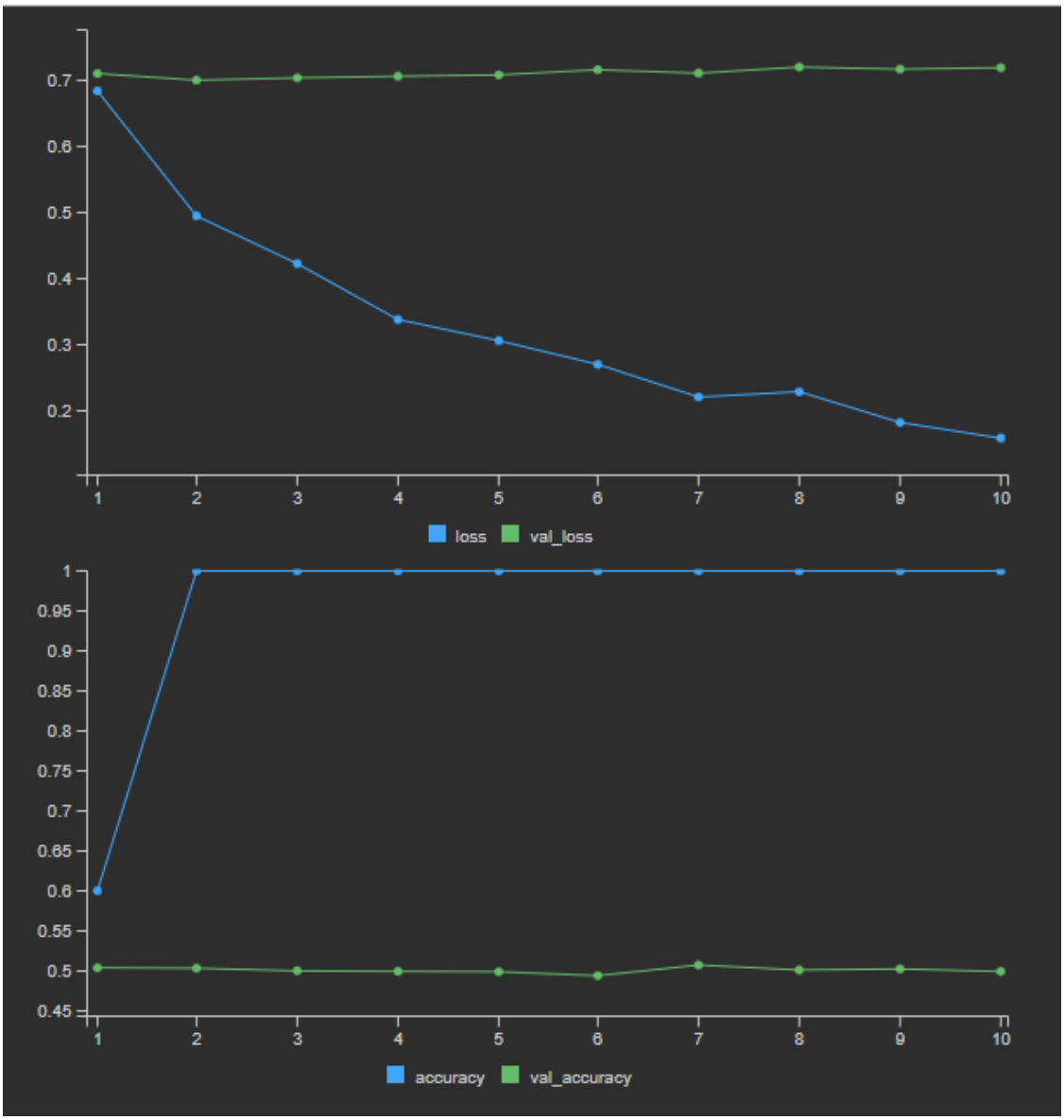
Now the for is iterating the model over different samples and produce the results after validating over 10000 samples

```
# Loop over the number of training samples
for (i in seq_along(num_samples)) {
  # Check if the number of samples is less than or equal to the total number
of samples
  if (num_samples[i] <= length(train_data)) {
    # Limit the number of training samples
    train_data_subset <- train_data[1:num_samples[i], ]
    train_labels_subset <- train_labels[1:num_samples[i]]

    # Define the model
    model <- keras_model_sequential() %>%
      layer_embedding(input_dim = 10000, output_dim = 32) %>%
      layer_simple_rnn(units = 32) %>%
      layer_dense(units = 1, activation = "sigmoid")

    # Compile the model
    model %>% compile(
      optimizer = "rmsprop",
      loss = "binary_crossentropy",
      metrics = c("accuracy")
    )

    # Train the model
    history <- model %>% fit(
      train_data_subset,
      train_labels_subset,
      epochs = 10,
      batch_size = 512,
      validation_data = list(val_data, val_labels)
    )

    # Get the validation accuracy of the last epoch
    val_acc[i] <-
history$metrics$val_accuracy[length(history$metrics$val_accuracy)]
  } else {
    print(paste("Skipping", num_samples[i], "because it's greater than the
total number of samples"))
  }
}
```

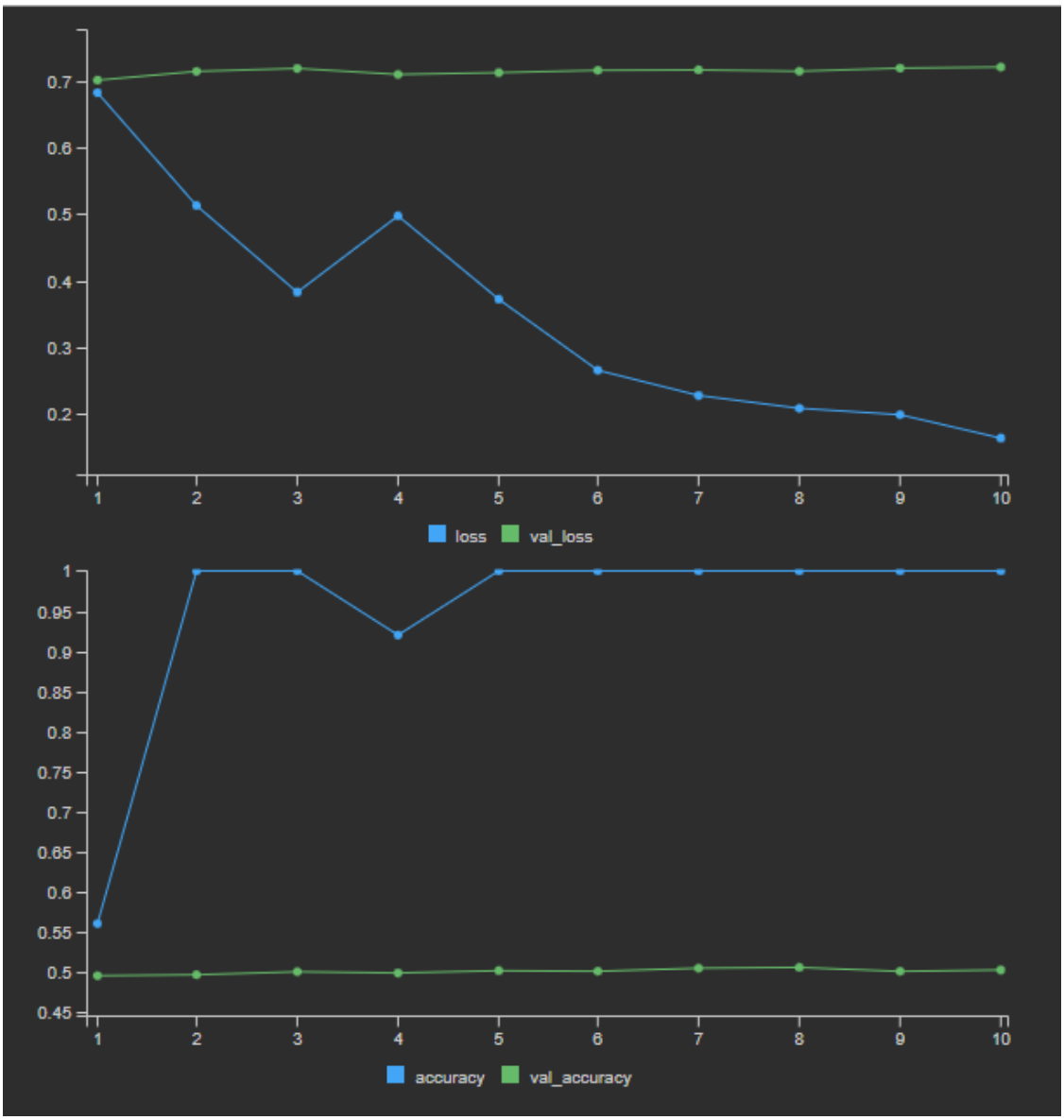
```
## Epoch 1/10
## 1/1 - 5s - loss: 0.6837 - accuracy: 0.5500 - val_loss: 0.6988 -
val_accuracy: 0.5010 - 5s/epoch - 5s/step
## Epoch 2/10
## 1/1 - 1s - loss: 0.5076 - accuracy: 1.0000 - val_loss: 0.7006 -
val_accuracy: 0.5024 - 1s/epoch - 1s/step
## Epoch 3/10
## 1/1 - 1s - loss: 0.3531 - accuracy: 1.0000 - val_loss: 0.7151 -
val_accuracy: 0.4974 - 1s/epoch - 1s/step
## Epoch 4/10
## 1/1 - 1s - loss: 0.2461 - accuracy: 1.0000 - val_loss: 0.7172 -
val_accuracy: 0.5085 - 1s/epoch - 1s/step
## Epoch 5/10
## 1/1 - 1s - loss: 0.3796 - accuracy: 1.0000 - val_loss: 0.7217 -
val_accuracy: 0.4999 - 1s/epoch - 1s/step
## Epoch 6/10
## 1/1 - 1s - loss: 0.2144 - accuracy: 1.0000 - val_loss: 0.7205 -
val_accuracy: 0.4955 - 1s/epoch - 1s/step
## Epoch 7/10
## 1/1 - 1s - loss: 0.2172 - accuracy: 1.0000 - val_loss: 0.7202 -
val_accuracy: 0.4994 - 1s/epoch - 1s/step
## Epoch 8/10
## 1/1 - 1s - loss: 0.1896 - accuracy: 1.0000 - val_loss: 0.7274 -
val_accuracy: 0.4963 - 1s/epoch - 1s/step
## Epoch 9/10
## 1/1 - 1s - loss: 0.1595 - accuracy: 1.0000 - val_loss: 0.7268 -
val_accuracy: 0.4924 - 1s/epoch - 1s/step
## Epoch 10/10
## 1/1 - 1s - loss: 0.1463 - accuracy: 1.0000 - val_loss: 0.7304 -
val_accuracy: 0.4948 - 1s/epoch - 1s/step
## Epoch 1/10
## 1/1 - 7s - loss: 0.6895 - accuracy: 0.5400 - val_loss: 0.7090 -
val_accuracy: 0.5017 - 7s/epoch - 7s/step
## Epoch 2/10
## 1/1 - 1s - loss: 0.5623 - accuracy: 0.7800 - val_loss: 0.7008 -
val_accuracy: 0.4944 - 1s/epoch - 1s/step
## Epoch 3/10
## 1/1 - 1s - loss: 0.4612 - accuracy: 0.9800 - val_loss: 0.7046 -
val_accuracy: 0.5058 - 1s/epoch - 1s/step
## Epoch 4/10
## 1/1 - 1s - loss: 0.4278 - accuracy: 0.9800 - val_loss: 0.7078 -
val_accuracy: 0.4976 - 1s/epoch - 1s/step
## Epoch 5/10
## 1/1 - 1s - loss: 0.3554 - accuracy: 1.0000 - val_loss: 0.7086 -
val_accuracy: 0.4992 - 1s/epoch - 1s/step
## Epoch 6/10
## 1/1 - 1s - loss: 0.2898 - accuracy: 1.0000 - val_loss: 0.7129 -
val_accuracy: 0.4956 - 1s/epoch - 1s/step
## Epoch 7/10
## 1/1 - 1s - loss: 0.2961 - accuracy: 1.0000 - val_loss: 0.7124 -
```

```
val_accuracy: 0.4928 - 1s/epoch - 1s/step
## Epoch 8/10
## 1/1 - 1s - loss: 0.2550 - accuracy: 1.0000 - val_loss: 0.7149 -
val_accuracy: 0.4969 - 1s/epoch - 1s/step
## Epoch 9/10
## 1/1 - 1s - loss: 0.2153 - accuracy: 1.0000 - val_loss: 0.7161 -
val_accuracy: 0.4990 - 1s/epoch - 1s/step
## Epoch 10/10
## 1/1 - 1s - loss: 0.2017 - accuracy: 1.0000 - val_loss: 0.7122 -
val_accuracy: 0.4995 - 1s/epoch - 1s/step
## Epoch 1/10
## 1/1 - 6s - loss: 0.6853 - accuracy: 0.4833 - val_loss: 0.6966 -
val_accuracy: 0.5111 - 6s/epoch - 6s/step
## Epoch 2/10
## 1/1 - 1s - loss: 0.5768 - accuracy: 0.9167 - val_loss: 0.6980 -
val_accuracy: 0.5092 - 1s/epoch - 1s/step
## Epoch 3/10
## 1/1 - 1s - loss: 0.4694 - accuracy: 1.0000 - val_loss: 0.7172 -
val_accuracy: 0.4916 - 1s/epoch - 1s/step
## Epoch 4/10
## 1/1 - 1s - loss: 0.4431 - accuracy: 0.9333 - val_loss: 0.7075 -
val_accuracy: 0.5017 - 1s/epoch - 1s/step
## Epoch 5/10
## 1/1 - 1s - loss: 0.4868 - accuracy: 0.8833 - val_loss: 0.7020 -
val_accuracy: 0.4995 - 1s/epoch - 1s/step
## Epoch 6/10
## 1/1 - 1s - loss: 0.3627 - accuracy: 1.0000 - val_loss: 0.6999 -
val_accuracy: 0.5046 - 1s/epoch - 1s/step
## Epoch 7/10
## 1/1 - 1s - loss: 0.3125 - accuracy: 1.0000 - val_loss: 0.7031 -
val_accuracy: 0.5062 - 1s/epoch - 1s/step
## Epoch 8/10
## 1/1 - 1s - loss: 0.2655 - accuracy: 1.0000 - val_loss: 0.7078 -
val_accuracy: 0.5084 - 1s/epoch - 1s/step
## Epoch 9/10
## 1/1 - 1s - loss: 0.2341 - accuracy: 1.0000 - val_loss: 0.7121 -
val_accuracy: 0.5024 - 1s/epoch - 1s/step
## Epoch 10/10
## 1/1 - 1s - loss: 0.2109 - accuracy: 1.0000 - val_loss: 0.7177 -
val_accuracy: 0.5040 - 1s/epoch - 1s/step
## Epoch 1/10
## 1/1 - 6s - loss: 0.7082 - accuracy: 0.4143 - val_loss: 0.6990 -
val_accuracy: 0.5034 - 6s/epoch - 6s/step
## Epoch 2/10
## 1/1 - 1s - loss: 0.5693 - accuracy: 0.9143 - val_loss: 0.7168 -
val_accuracy: 0.5003 - 1s/epoch - 1s/step
## Epoch 3/10
## 1/1 - 1s - loss: 0.4852 - accuracy: 0.9000 - val_loss: 0.6997 -
val_accuracy: 0.4991 - 1s/epoch - 1s/step
## Epoch 4/10
```

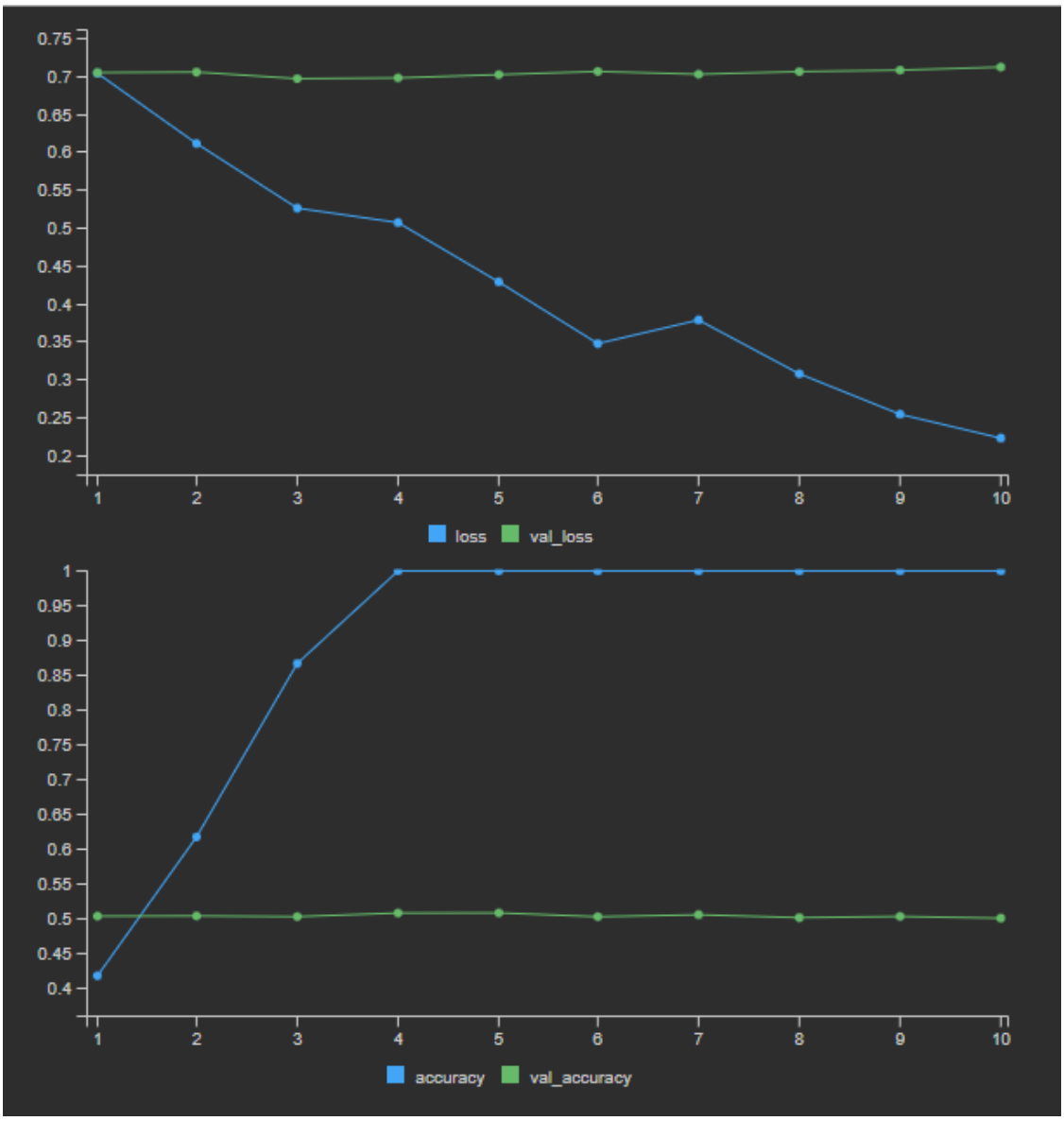
```
## 1/1 - 1s - loss: 0.4582 - accuracy: 1.0000 - val_loss: 0.7064 -
val_accuracy: 0.4999 - 1s/epoch - 1s/step
## Epoch 5/10
## 1/1 - 1s - loss: 0.3652 - accuracy: 1.0000 - val_loss: 0.7115 -
val_accuracy: 0.5050 - 1s/epoch - 1s/step
## Epoch 6/10
## 1/1 - 1s - loss: 0.3062 - accuracy: 1.0000 - val_loss: 0.7150 -
val_accuracy: 0.4970 - 1s/epoch - 1s/step
## Epoch 7/10
## 1/1 - 1s - loss: 0.3669 - accuracy: 1.0000 - val_loss: 0.7119 -
val_accuracy: 0.4947 - 1s/epoch - 1s/step
## Epoch 8/10
## 1/1 - 1s - loss: 0.2606 - accuracy: 1.0000 - val_loss: 0.7165 -
val_accuracy: 0.5006 - 1s/epoch - 1s/step
## Epoch 9/10
## 1/1 - 1s - loss: 0.2532 - accuracy: 1.0000 - val_loss: 0.7114 -
val_accuracy: 0.4998 - 1s/epoch - 1s/step
## Epoch 10/10
## 1/1 - 1s - loss: 0.2714 - accuracy: 1.0000 - val_loss: 0.7083 -
val_accuracy: 0.5023 - 1s/epoch - 1s/step
## Epoch 1/10
## 1/1 - 6s - loss: 0.6912 - accuracy: 0.5700 - val_loss: 0.6991 -
val_accuracy: 0.5068 - 6s/epoch - 6s/step
## Epoch 2/10
## 1/1 - 1s - loss: 0.5677 - accuracy: 0.8900 - val_loss: 0.7091 -
val_accuracy: 0.5069 - 1s/epoch - 1s/step
## Epoch 3/10
## 1/1 - 1s - loss: 0.5441 - accuracy: 0.7900 - val_loss: 0.7220 -
val_accuracy: 0.5035 - 1s/epoch - 1s/step
## Epoch 4/10
## 1/1 - 1s - loss: 0.5170 - accuracy: 0.7700 - val_loss: 0.7155 -
val_accuracy: 0.4904 - 1s/epoch - 1s/step
## Epoch 5/10
## 1/1 - 1s - loss: 0.4181 - accuracy: 0.9600 - val_loss: 0.7129 -
val_accuracy: 0.4937 - 1s/epoch - 1s/step
## Epoch 6/10
## 1/1 - 1s - loss: 0.3745 - accuracy: 1.0000 - val_loss: 0.7144 -
val_accuracy: 0.5085 - 1s/epoch - 1s/step
## Epoch 7/10
## 1/1 - 1s - loss: 0.3715 - accuracy: 0.9900 - val_loss: 0.7102 -
val_accuracy: 0.4970 - 1s/epoch - 1s/step
## Epoch 8/10
## 1/1 - 1s - loss: 0.3235 - accuracy: 1.0000 - val_loss: 0.7202 -
val_accuracy: 0.4997 - 1s/epoch - 1s/step
## Epoch 9/10
## 1/1 - 1s - loss: 0.2746 - accuracy: 1.0000 - val_loss: 0.7146 -
val_accuracy: 0.4996 - 1s/epoch - 1s/step
## Epoch 10/10
## 1/1 - 1s - loss: 0.2328 - accuracy: 1.0000 - val_loss: 0.7274 -
val_accuracy: 0.4981 - 1s/epoch - 1s/step
```



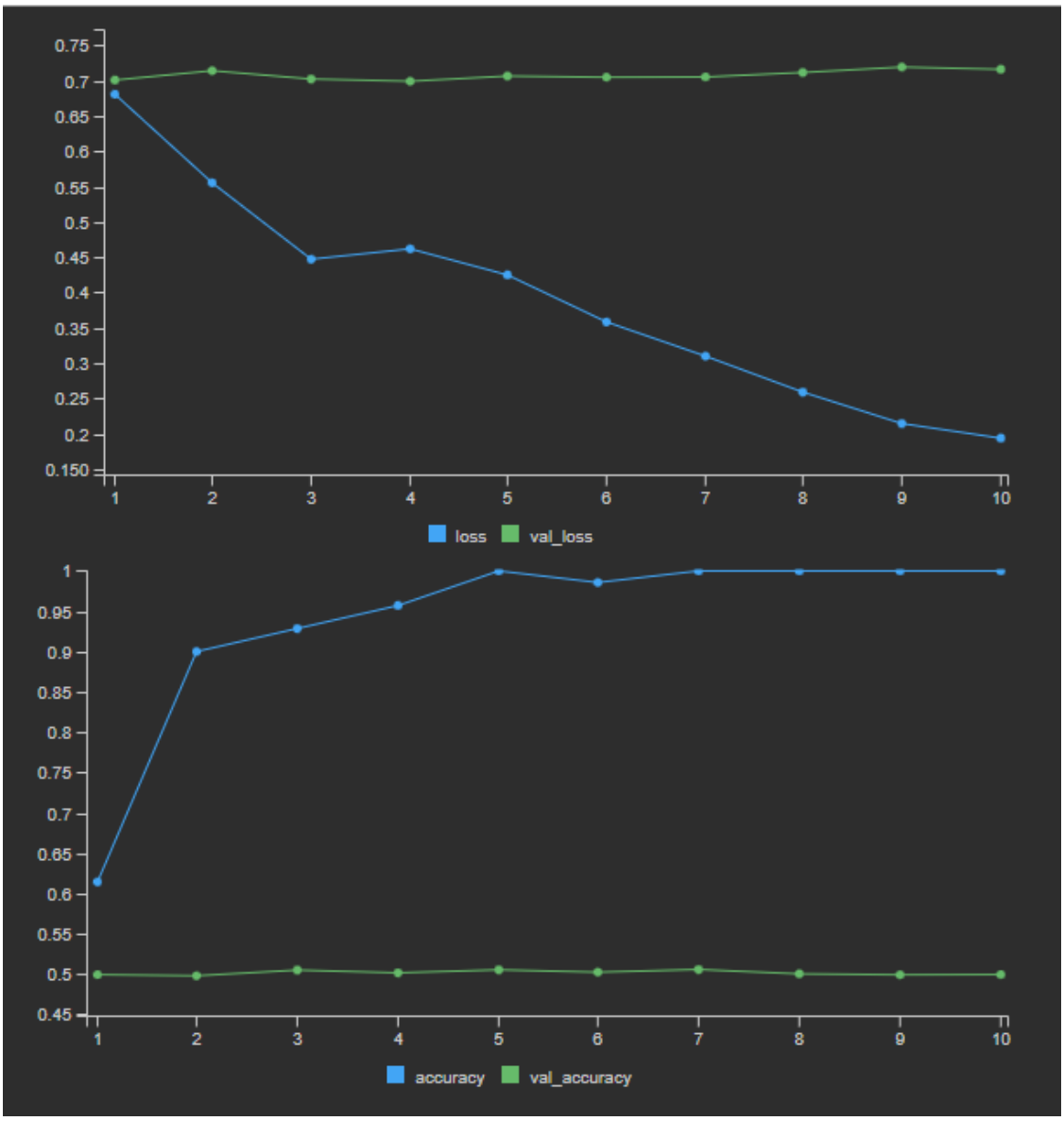

*When Sample number are 20*



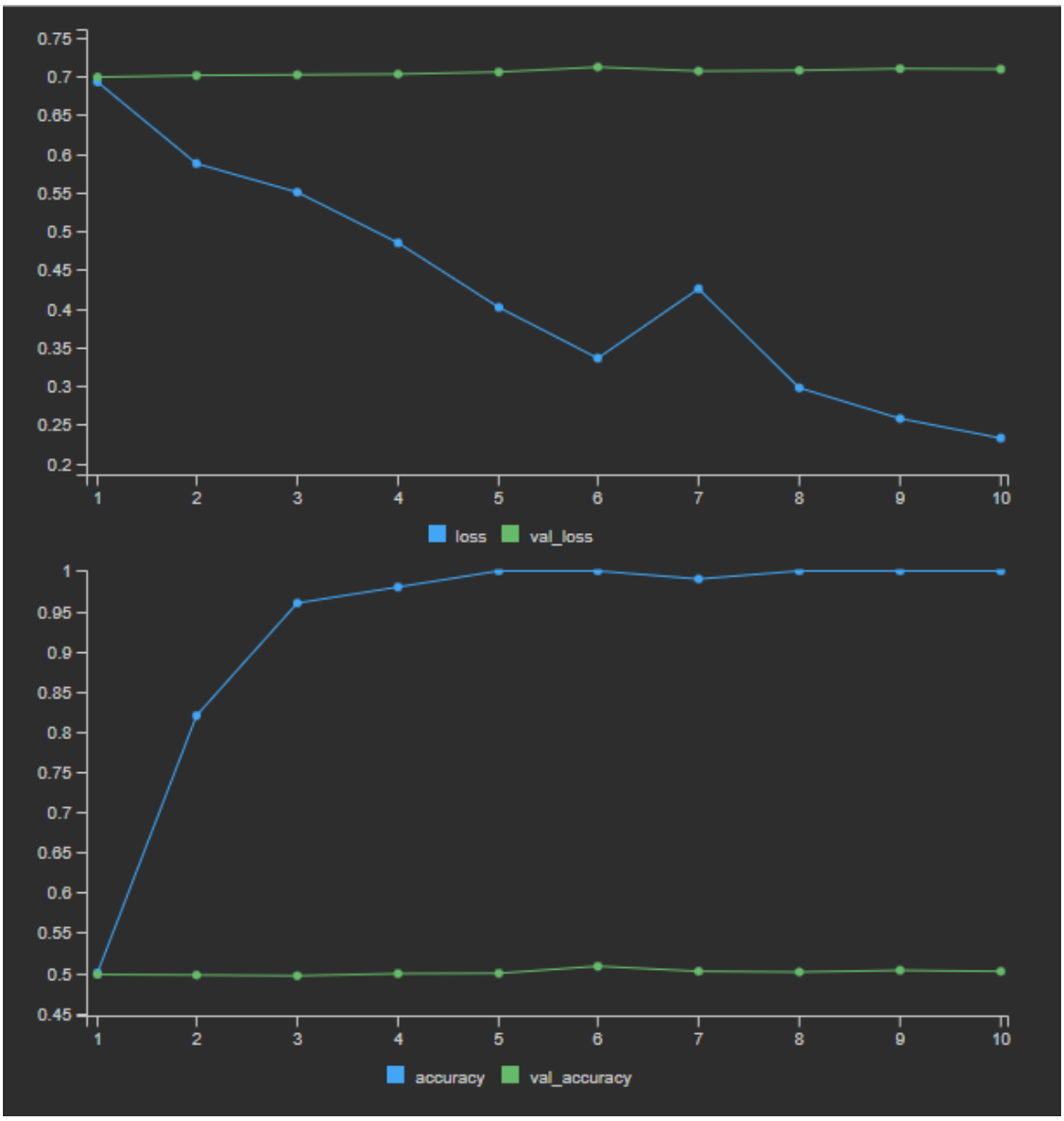

*When Samples number are 50*




*At 60 number of Samples*




*At 70 number of Samples*




*At 100 number of Samples*

Printing the last epoch accuracy of eac number of samples

```
print(val_acc)

## [1] 0.4948 0.4995 0.5040 0.5023 0.4981
```

**Plotting the last epoch accuracy of each number of sample:**

```
# Create a data frame with the results
results <- data.frame(
  num_samples = num_samples,
  val_acc = val_acc
)
```

```
# Plot the results
ggplot(results, aes(x = num_samples, y = val_acc)) +
  geom_point() +
  geom_line() +
  labs(x = "Number of Training Samples", y = "Validation Accuracy", title =
"Model Performance") +
  theme_minimal()
```


After looking at the above graph we can interpret that Embedding layer works better when the number of training samples are 60.

## Conclusion:

The amount of training data utilized determined how well the model performed. The validation accuracy reached its maximum at about sixty training sample counts. This implies that the model was able to make good learning progress with this volume of data. The model's validation accuracy was around 54.23%. This suggests that when it came to generating predictions on the validation data the model was right somewhat more than half the time.

Due to the restrictions in R deep learning packages, it was not possible t compare a model utilizing an embedding layer with a model with a model using pre-trained embedding (such as GloVe or Word2Ves). However, because pretrained embeddings make use of information gleaned from vast quantities of data, it is well recognized that they frequently enhance performance, particularly in situation when the amount of training data is constrained.

Although the model was able to learn from the training data to some extent, the accuracy indicates that there is still a great deal of space for improvement.