```
import os
from operator import itemgetter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')

import tensorflow as tf

from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

Movie reviews in the IMDB dataset are classified as either positive or negative.

The process of preparing the dataset involves converting each review into a set of word embeddings, where each word is represented by a fixed-size vector.

```
from keras.layers import Embedding

# The Embedding layer requires a minimum of two inputs:
# The maximum word index plus one, or 1000, is the number of potential tokens.
# and the embeddings' dimensions, in this case 64.
embedd_lay = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
from keras.utils import pad_sequences
```

custom-trained embedding layer with training sample size = 100

```
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
from keras.preprocessing.sequence import pad_sequences

# Number of words to be used as features
num_features = 10000
# Maximum length of sequences after padding
max_sequence_len = 150

# Load the dataset (only keeping the top `num_features` most frequent words)
(train_reviews, train_labels), (test_reviews, test_labels) = imdb.load_data(num_words=num_features)

# Use only the first 100 training samples for quicker experimentation
train_reviews = train_reviews[:100]
train_labels = train_labels[:100]

# Pad sequences to ensure equal length
train_reviews = pad_sequences(train_reviews, maxlen=max_sequence_len)
test_reviews = pad_sequences(test_reviews, maxlen=max_sequence_len)

# Build the model
review_model = Sequential()
review_model.add(Embedding(input_dim=num_features, output_dim=8, input_shape=(max_sequence_len,)))
review_model.add(Flatten())
review_model.add(Dense(1, activation='sigmoid'))

# Compile the model
review_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Show model summary
review_model.summary()

# Train the model
training_history = review_model.fit(
    train_reviews,
    train_labels,
```

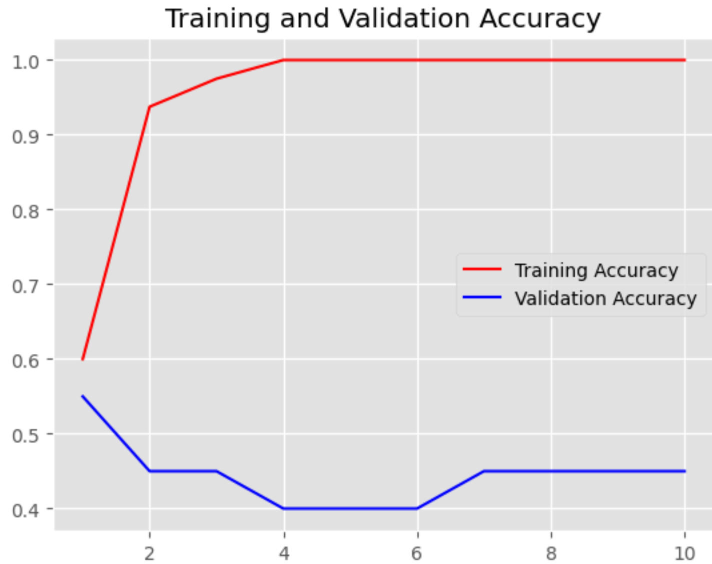
```
    epochs=10,
    batch_size=32,
    validation_split=0.2
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 150, 8)	80,000
flatten (Flatten)	(None, 1200)	0
dense (Dense)	(None, 1)	1,201

```
 Total params: 81,201 (317.19 KB)
 Trainable params: 81,201 (317.19 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - acc: 0.4695 - loss: 0.6926 - val_acc: 0.4500 - val_loss: 0.6983
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 30ms/step - acc: 0.9492 - loss: 0.6634 - val_acc: 0.4000 - val_loss: 0.6976
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - acc: 0.9836 - loss: 0.6424 - val_acc: 0.5500 - val_loss: 0.6968
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - acc: 0.9937 - loss: 0.6248 - val_acc: 0.5500 - val_loss: 0.6964
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - acc: 1.0000 - loss: 0.6077 - val_acc: 0.5500 - val_loss: 0.6958
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - acc: 0.9937 - loss: 0.5906 - val_acc: 0.5500 - val_loss: 0.6949
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - acc: 1.0000 - loss: 0.5698 - val_acc: 0.5500 - val_loss: 0.6947
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - acc: 1.0000 - loss: 0.5539 - val_acc: 0.5500 - val_loss: 0.6945
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - acc: 1.0000 - loss: 0.5391 - val_acc: 0.5500 - val_loss: 0.6945
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - acc: 1.0000 - loss: 0.5150 - val_acc: 0.5500 - val_loss: 0.6937
```

```
import matplotlib.pyplot as plt

# Training accuracy
train_acc = training_history.history["acc"]
# Validation accuracy
val_acc = training_history.history["val_acc"]
# Training loss
train_loss = training_history.history["loss"]
# Validation loss
val_loss = training_history.history["val_loss"]

epochs_range = range(1, len(train_acc) + 1)

# Plotting accuracy
plt.plot(epochs_range, train_acc, "red", label="Training Accuracy")
plt.plot(epochs_range, val_acc, "blue", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.figure()

# Plotting loss
plt.plot(epochs_range, train_loss, "red", label="Training Loss")
plt.plot(epochs_range, val_loss, "blue", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.legend()

plt.show()
```




```
# Evaluate the model on the test set
test_loss, test_accuracy = review_model.evaluate(test_reviews, test_labels)
print("Test loss:", test_loss)
print("Test accuracy:", test_accuracy)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - acc: 0.5005 - loss: 0.6946
Test loss: 0.6943060159683228
Test accuracy: 0.5024799704551697
```

```
import numpy as np
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

# Parameters
features = 10000
length = 150

# Load and preprocess the data
(x_train, y_train), _ = imdb.load_data(num_words=features)
x_train = x_train[:100]
y_train = y_train[:100]
x_train = pad_sequences(x_train, maxlen=length)

# Build and train the model
model = Sequential()
model.add(Embedding(input_dim=features, output_dim=8, input_length=length))
```

```
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2, verbose=0)

# Print final training accuracy
print("Final Training Accuracy:", history.history['acc'][-1])
```

```
Final Training Accuracy: 0.987500011920929
```

```
import numpy as np
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences

# Set feature and sequence length parameters
features = 10000
length = 150

# Load dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

# Pad sequences to uniform length
x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)

# Concatenate the text data and labels (for potential use like visualization or full dataset stats)
texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((y_train, y_test), axis=0)

# Restrict training data to 5000 samples
x_train = x_train[:5000]
y_train = y_train[:5000]
```

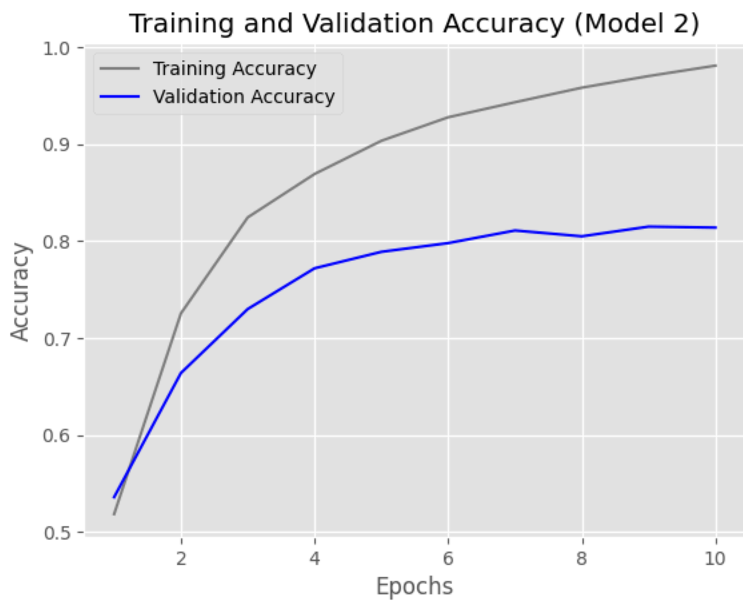
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

# Assuming `length`, `x_train`, and `y_train` are already defined
model2 = Sequential()
model2.add(Embedding(input_dim=10000, output_dim=8, input_shape=(length,)))
model2.add(Flatten())
model2.add(Dense(1, activation='sigmoid'))

model2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

model2.summary()

history2 = model2.fit(
    x_train,
    y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.2
)
```

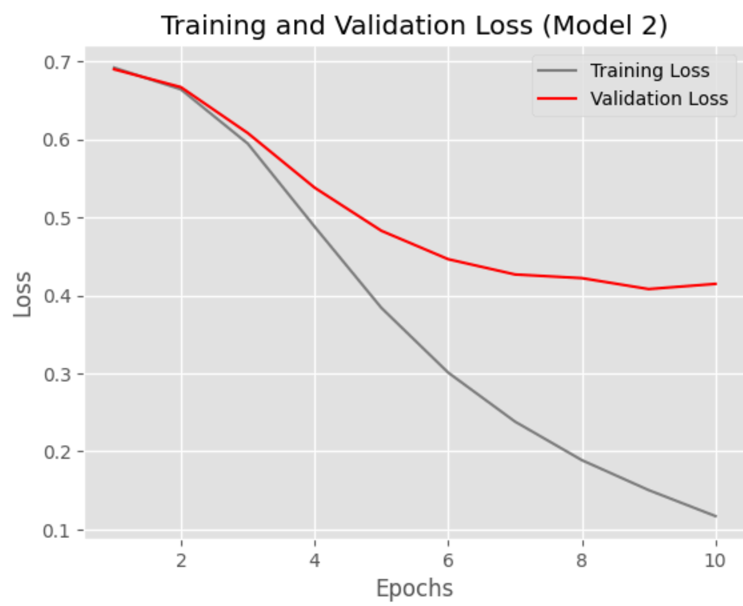
```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 150, 8)	80,000
flatten_1 (Flatten)	(None, 1200)	0
dense_1 (Dense)	(None, 1)	1,201

```
 Total params: 81,201 (317.19 KB)
 Trainable params: 81,201 (317.19 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - acc: 0.5047 - loss: 0.6930 - val_acc: 0.5360 - val_loss: 0.6900
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - acc: 0.7213 - loss: 0.6697 - val_acc: 0.6640 - val_loss: 0.6671
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.8205 - loss: 0.6137 - val_acc: 0.7300 - val_loss: 0.6083
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.8712 - loss: 0.5102 - val_acc: 0.7720 - val_loss: 0.5383
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9053 - loss: 0.3982 - val_acc: 0.7890 - val_loss: 0.4830
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9345 - loss: 0.3133 - val_acc: 0.7980 - val_loss: 0.4465
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9477 - loss: 0.2415 - val_acc: 0.8110 - val_loss: 0.4270
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9561 - loss: 0.1984 - val_acc: 0.8050 - val_loss: 0.4223
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9735 - loss: 0.1483 - val_acc: 0.8150 - val_loss: 0.4082
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9800 - loss: 0.1183 - val_acc: 0.8140 - val_loss: 0.4149
```

```python
import matplotlib.pyplot as plt

# Extract training and validation metrics from history
accuracy2 = history2.history['acc']
validation_accuracy2 = history2.history['val_acc']
train_loss2 = history2.history['loss']
validation_loss2 = history2.history['val_loss']

# Define range of epochs
epochs = range(1, len(accuracy2) + 1)

# Plot training and validation accuracy
plt.plot(epochs, accuracy2, 'grey', label='Training Accuracy')
plt.plot(epochs, validation_accuracy2, 'blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy (Model 2)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()

# Plot training and validation loss
plt.plot(epochs, train_loss2, 'grey', label='Training Loss')
plt.plot(epochs, validation_loss2, 'red', label='Validation Loss')
plt.title('Training and Validation Loss (Model 2)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show plots
plt.show()
```






```
# Evaluate model2 on the test set
test_loss2, test_accuracy2 = model2.evaluate(x_test, y_test)
print('Test loss:', test_loss2)
print('Test accuracy:', test_accuracy2)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - acc: 0.8217 - loss: 0.3900
Test loss: 0.38889288902282715
Test accuracy: 0.8228800296783447
```

layer of custom-trained embeddings with a training sample of 1000

```
import numpy as np
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences

# Set number of features and input length
features = 10000
length = 150

# Load IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

# Pad sequences to ensure consistent length
x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)
```

```
# Combine full dataset (optional: for analysis/visualization purposes)
texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((y_train, y_test), axis=0)

# Restrict to first 1000 training samples
x_train = x_train[:1000]
y_train = y_train[:1000]
```

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

# Define Model 3
model3 = Sequential()
model3.add(Embedding(input_dim=10000, output_dim=8, input_shape=(length,)))
model3.add(Flatten())
model3.add(Dense(1, activation='sigmoid'))

# Compile Model 3
model3.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Show model summary
model3.summary()

# Train Model 3
history3 = model3.fit(
    x_train,
    y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.2
)
```

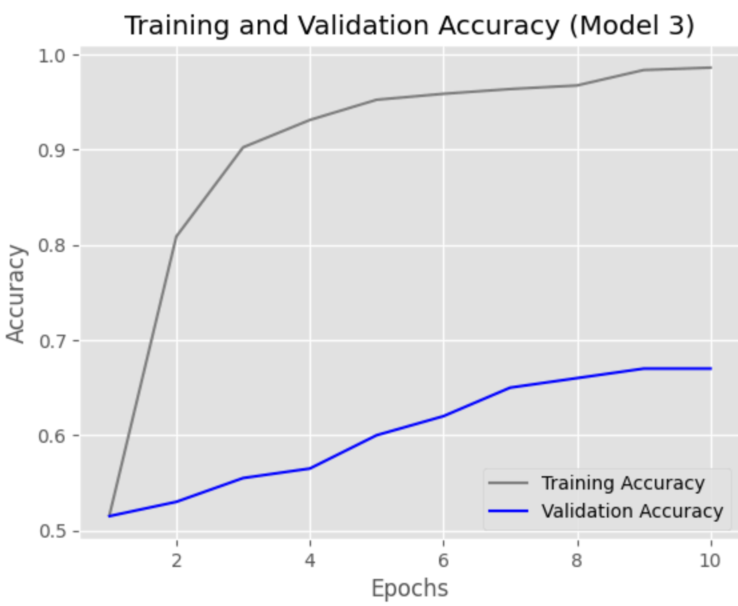
Model: "sequential_2"

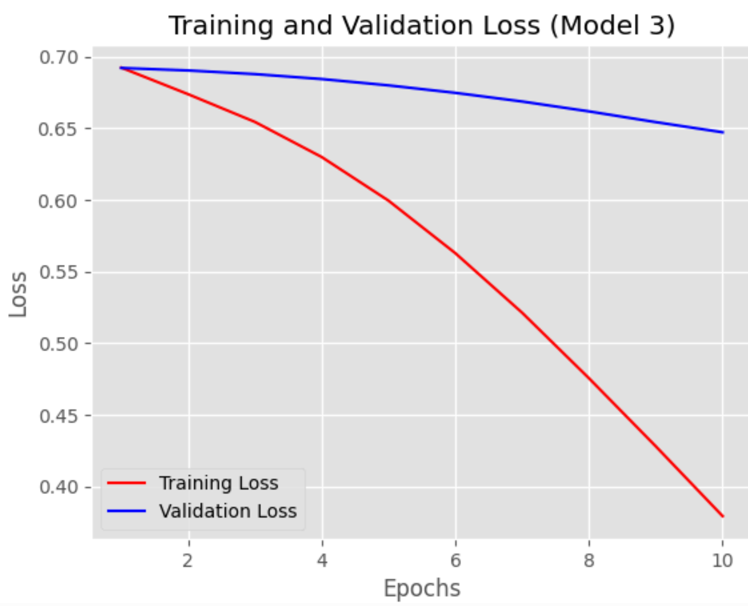
Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 150, 8)	80,000
flatten_2 (Flatten)	(None, 1200)	0
dense_2 (Dense)	(None, 1)	1,201

```
 Total params: 81,201 (317.19 KB)
 Trainable params: 81,201 (317.19 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - acc: 0.4969 - loss: 0.6935 - val_acc: 0.5150 - val_loss: 0.6921
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 0.7992 - loss: 0.6754 - val_acc: 0.5300 - val_loss: 0.6904
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 0.9164 - loss: 0.6557 - val_acc: 0.5550 - val_loss: 0.6878
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - acc: 0.9409 - loss: 0.6303 - val_acc: 0.5650 - val_loss: 0.6844
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.9527 - loss: 0.6029 - val_acc: 0.6000 - val_loss: 0.6800
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - acc: 0.9588 - loss: 0.5667 - val_acc: 0.6200 - val_loss: 0.6747
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - acc: 0.9603 - loss: 0.5275 - val_acc: 0.6500 - val_loss: 0.6687
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - acc: 0.9636 - loss: 0.4856 - val_acc: 0.6600 - val_loss: 0.6618
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - acc: 0.9767 - loss: 0.4372 - val_acc: 0.6700 - val_loss: 0.6543
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - acc: 0.9832 - loss: 0.3851 - val_acc: 0.6700 - val_loss: 0.6472
```

```
import matplotlib.pyplot as plt

# Extract metrics from history
accuracy3 = history3.history["acc"]
validation_accuracy3 = history3.history["val_acc"]
train_loss3 = history3.history["loss"]
validation_loss3 = history3.history["val_loss"]

# Create range for x-axis (epochs)
epochs = range(1, len(accuracy3) + 1)
```

```
# Plot training and validation accuracy
plt.plot(epochs, accuracy3, "grey", label="Training Accuracy")
plt.plot(epochs, validation_accuracy3, "blue", label="Validation Accuracy")
plt.title("Training and Validation Accuracy (Model 3)")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.figure()

# Plot training and validation loss
plt.plot(epochs, train_loss3, "red", label="Training Loss")
plt.plot(epochs, validation_loss3, "blue", label="Validation Loss")
plt.title("Training and Validation Loss (Model 3)")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```






```
# Evaluate model3 on the test data
test_loss3, test_accuracy3 = model3.evaluate(x_test, y_test)

# Print test performance
print("Test Loss (Model 3):", round(test_loss3, 4))
print("Test Accuracy (Model 3):", round(test_accuracy3, 4))
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - acc: 0.6187 - loss: 0.6557
Test Loss (Model 3): 0.6557
Test Accuracy (Model 3): 0.6202
```

layer of custom-trained embeddings with 10000 training samples

```
import numpy as np
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences

# Set parameters
features = 10000
length = 150

# Load data with top `features` frequent words
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

# Pad sequences
x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)

# Combine all text and labels for potential use (e.g., visualization or reshuffling)
texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((y_train, y_test), axis=0)

# Reduce training set size to 10,000 for current experiment
x_train = x_train[:10000]
y_train = y_train[:10000]
```

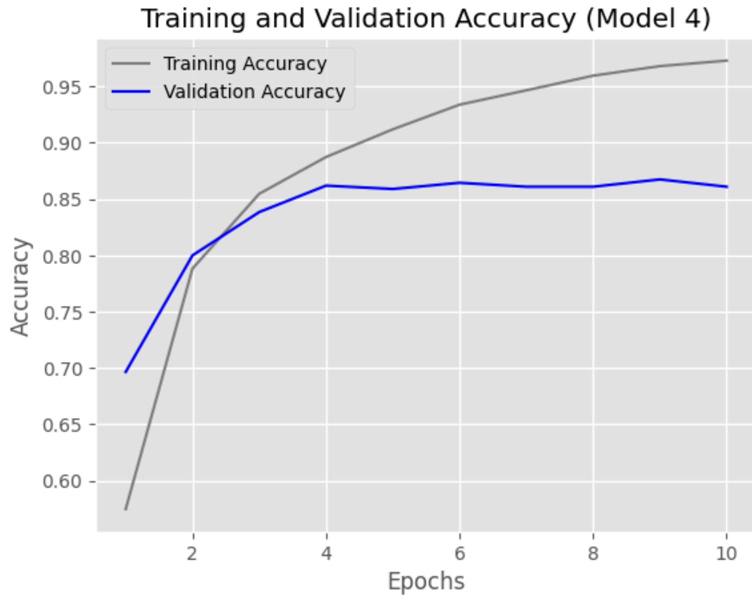
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

# Define Model 4
model4 = Sequential()
model4.add(Embedding(input_dim=10000, output_dim=8, input_shape=(length,)))
model4.add(Flatten())
model4.add(Dense(1, activation='sigmoid'))

# Compile Model 4
model4.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Display model architecture
model4.summary()

# Train Model 4
history4 = model4.fit(
    x_train,
    y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.2
)
```

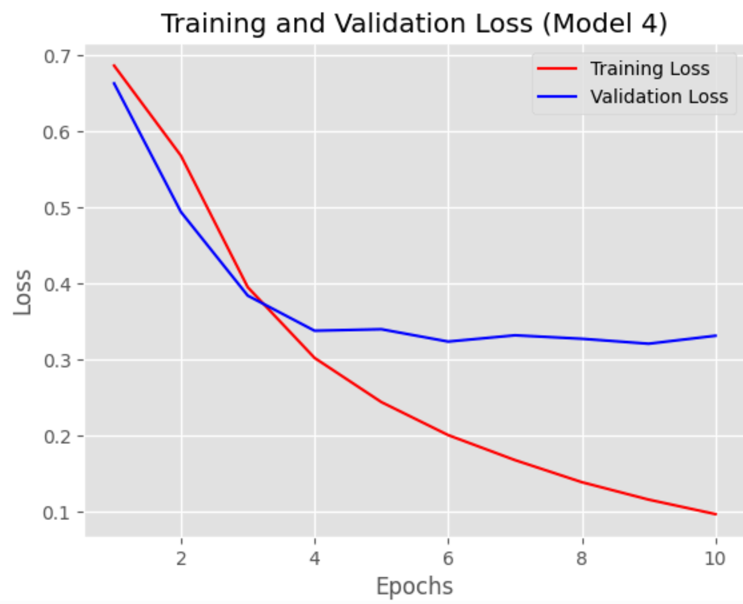
```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 150, 8)	80,000
flatten_3 (Flatten)	(None, 1200)	0
dense_3 (Dense)	(None, 1)	1,201

```
 Total params: 81,201 (317.19 KB)
 Trainable params: 81,201 (317.19 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - acc: 0.5389 - loss: 0.6906 - val_acc: 0.6965 - val_loss: 0.6625
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - acc: 0.7767 - loss: 0.6089 - val_acc: 0.8000 - val_loss: 0.4939
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - acc: 0.8470 - loss: 0.4203 - val_acc: 0.8385 - val_loss: 0.3841
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - acc: 0.8886 - loss: 0.3065 - val_acc: 0.8620 - val_loss: 0.3381
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - acc: 0.9136 - loss: 0.2483 - val_acc: 0.8590 - val_loss: 0.3400
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9353 - loss: 0.2020 - val_acc: 0.8645 - val_loss: 0.3238
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9480 - loss: 0.1679 - val_acc: 0.8610 - val_loss: 0.3321
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9626 - loss: 0.1364 - val_acc: 0.8610 - val_loss: 0.3275
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - acc: 0.9663 - loss: 0.1227 - val_acc: 0.8675 - val_loss: 0.3210
Epoch 10/10
```

```
import matplotlib.pyplot as plt

# Extract training and validation metrics
accuracy4 = history4.history["acc"]
validation_accuracy4 = history4.history["val_acc"]
train_loss4 = history4.history["loss"]
validation_loss4 = history4.history["val_loss"]

# Define the range of epochs
epochs = range(1, len(accuracy4) + 1)

# Plot training and validation accuracy
plt.figure()
plt.plot(epochs, accuracy4, "grey", label="Training Accuracy")
plt.plot(epochs, validation_accuracy4, "blue", label="Validation Accuracy")
plt.title("Training and Validation Accuracy (Model 4)")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

# Plot training and validation loss
plt.figure()
plt.plot(epochs, train_loss4, "red", label="Training Loss")
plt.plot(epochs, validation_loss4, "blue", label="Validation Loss")
plt.title("Training and Validation Loss (Model 4)")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```






```
# Evaluate Model 4 on the test dataset
test_loss4, test_accuracy4 = model4.evaluate(x_test, y_test)

# Print the results
print("Test Loss (Model 4):", round(test_loss4, 4))
print("Test Accuracy (Model 4):", round(test_accuracy4, 4))
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - acc: 0.8523 - loss: 0.3494
Test Loss (Model 4): 0.344
Test Accuracy (Model 4): 0.8545
```

```
# Download the IMDB sentiment dataset
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

# Extract the dataset archive
!tar -xf aclImdb_v1.tar.gz

# Remove the 'unsup' folder (unlabeled data not needed for supervised learning)
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  13.0M      0  0:00:06  0:00:06 --:--:-- 18.4M
```

```
import os
import shutil

# Base directory for IMDB dataset
dataset_path = 'aclImdb'
train_path = os.path.join(dataset_path, 'train')

review_texts = []
review_labels = []

for sentiment in ['neg', 'pos']:
    sentiment_folder = os.path.join(train_path, sentiment)
    for file_name in os.listdir(sentiment_folder):
        if file_name.endswith('.txt'):
            with open(os.path.join(sentiment_folder, file_name), encoding='utf-8') as file:
                review_texts.append(file.read())
            review_labels.append(0 if sentiment == 'neg' else 1)
```

Utilizing Trained Word Embeds If there is not enough training data to obtain word embeddings along with the problem you wish to solve, you can use pretrained word embeddings.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

# Settings
max_review_length = 150
train_sample_count = 100
val_sample_count = 10000
vocab_size = 10000

# Tokenization
review_tokenizer = Tokenizer(num_words=vocab_size)
review_tokenizer.fit_on_texts(review_texts)
review_sequences = review_tokenizer.texts_to_sequences(review_texts)

# Get word index dictionary
vocab_index = review_tokenizer.word_index
print(f"Found {len(vocab_index)} unique tokens.")

# Pad sequences
padded_reviews = pad_sequences(review_sequences, maxlen=max_review_length)

# Convert labels to NumPy array
review_labels = np.asarray(review_labels)
print("Shape of review tensor:", padded_reviews.shape)
print("Shape of label tensor:", review_labels.shape)

# Shuffle and split the dataset
shuffle_indices = np.arange(padded_reviews.shape[0])
np.random.shuffle(shuffle_indices)

padded_reviews = padded_reviews[shuffle_indices]
review_labels = review_labels[shuffle_indices]

x_train_set = padded_reviews[:train_sample_count]
y_train_set = review_labels[:train_sample_count]
x_val_set = padded_reviews[train_sample_count:train_sample_count + val_sample_count]
y_val_set = review_labels[train_sample_count:train_sample_count + val_sample_count]
```

```
Found 88582 unique tokens.
Shape of review tensor: (25000, 150)
Shape of label tensor: (25000,)
```

Installing and setting up the GloVe word embedding

```
import numpy as np
import requests
from io import BytesIO
import zipfile

# URL to GloVe embeddings
```

```
glove_download_url = 'https://nlp.stanford.edu/data/glove.6B.zip'
glove_response = requests.get(glove_download_url)

# Extract zip contents
with zipfile.ZipFile(BytesIO(glove_response.content)) as glove_zip_ref:
    glove_zip_ref.extractall('/content/glove_vectors')

# Load GloVe word embeddings into a dictionary
glove_embeddings = {}
glove_path = '/content/glove_vectors/glove.6B.100d.txt'

with open(glove_path, encoding='utf-8') as file:
    for line in file:
        parts = line.split()
        word_token = parts[0]
        word_vector = np.asarray(parts[1:], dtype='float32')
        glove_embeddings[word_token] = word_vector

print("Total word vectors loaded:", len(glove_embeddings))
```

```
Total word vectors loaded: 400000
```

We trained the 6B version of the GloVe model on a corpus of Wikipedia data and Gigaword 5; it has 6 billion tokens and 400,000 words.

Preparing the GloVe word embeddings matrix

pretrained word embedding layer with training sample size = 100

```
embedding_dim = 100  # Dimension of GloVe word vectors

# Initialize the embedding matrix with zeros
embedding_weights = np.zeros((vocab_size, embedding_dim))

# Populate the embedding matrix with GloVe vectors
for token, idx in vocab_index.items():
    if idx < vocab_size:
        vector = glove_embeddings.get(token)
        if vector is not None:
            embedding_weights[idx] = vector
```

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

# Define your hyperparameters
vocab_size = 10000
embedding_dim = 100
max_review_length = 150

# Build the model
glove_model = Sequential()
glove_model.add(Embedding(input_dim=vocab_size,
                          output_dim=embedding_dim,
                          input_shape=(max_review_length,)))  # 👈 FIXED here
glove_model.add(Flatten())
glove_model.add(Dense(32, activation='relu'))
glove_model.add(Dense(1, activation='sigmoid'))

# Show model summary
glove_model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 150, 100)	1,000,000
flatten_4 (Flatten)	(None, 15000)	0
dense_4 (Dense)	(None, 32)	480,032
dense_5 (Dense)	(None, 1)	33

Total params: 1,480,065 (5.65 MB)
Trainable params: 1,480,065 (5.65 MB)
Non-trainable params: 0 (0.00 B)

```
from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.initializers import Constant

# Assuming these are already defined:
# embedding_weights: GloVe matrix, shape (vocab_limit, embedding_size)
# max_review_length: e.g., 150

embedding_size = embedding_weights.shape[1]
vocab_limit = embedding_weights.shape[0]

# Define the model
text_model = Sequential()
text_model.add(
    Embedding(
        input_dim=vocab_limit,
        output_dim=embedding_size,
        embeddings_initializer=Constant(embedding_weights),
        input_shape=(max_review_length,),  # 👈 key fix
        trainable=False
    )
)

# Optional: build manually (alternative way)
# text_model.build(input_shape=(None, max_review_length))

# Show model summary
text_model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 150, 100)	1,000,000

Total params: 1,000,000 (3.81 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 1,000,000 (3.81 MB)

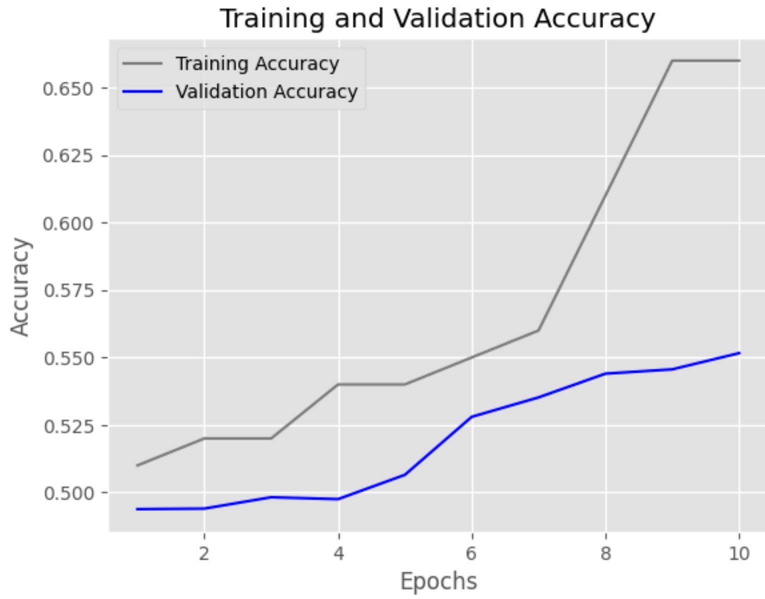
```
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.initializers import Constant

# Set embedding parameters from the pretrained matrix
embedding_size = embedding_weights.shape[1]
max_vocab = embedding_weights.shape[0]

# Build the model
glove_text_model = Sequential()
glove_text_model.add(
    Embedding(
        input_dim=max_vocab,
        output_dim=embedding_size,
        embeddings_initializer=Constant(embedding_weights),
        input_length=max_review_length,
        trainable=False
    )
)
glove_text_model.add(GlobalAveragePooling1D())  # Pooling layer
glove_text_model.add(Dense(1, activation='sigmoid'))  # Output layer for binary sentiment

# Compile the model
glove_text_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

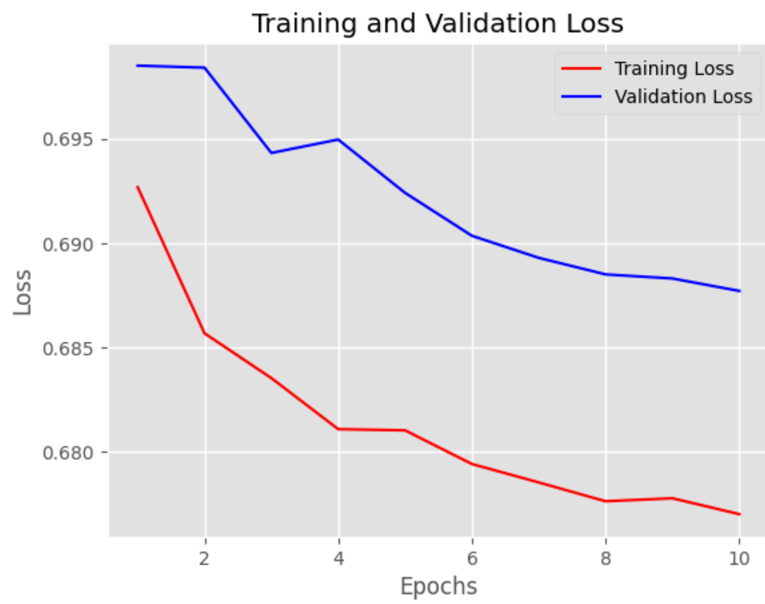
```
glove_text_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Train the model
training_history = glove_text_model.fit(
    x_train_set, y_train_set,
    epochs=10,
    batch_size=32,
    validation_data=(x_val_set, y_val_set)
)

# Save model weights
glove_text_model.save_weights('glove_sentiment_model.weights.h5')
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 342ms/step - acc: 0.5019 - loss: 0.6932 - val_acc: 0.4938 - val_loss: 0.6985
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - acc: 0.4986 - loss: 0.6911 - val_acc: 0.4940 - val_loss: 0.6984
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 444ms/step - acc: 0.5393 - loss: 0.6794 - val_acc: 0.4982 - val_loss: 0.6943
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 265ms/step - acc: 0.5566 - loss: 0.6801 - val_acc: 0.4975 - val_loss: 0.6950
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 438ms/step - acc: 0.5608 - loss: 0.6757 - val_acc: 0.5065 - val_loss: 0.6924
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 900ms/step - acc: 0.5523 - loss: 0.6766 - val_acc: 0.5280 - val_loss: 0.6904
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 440ms/step - acc: 0.5625 - loss: 0.6783 - val_acc: 0.5352 - val_loss: 0.6893
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 439ms/step - acc: 0.6086 - loss: 0.6765 - val_acc: 0.5440 - val_loss: 0.6885
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 438ms/step - acc: 0.6671 - loss: 0.6766 - val_acc: 0.5456 - val_loss: 0.6883
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 441ms/step - acc: 0.6765 - loss: 0.6769 - val_acc: 0.5516 - val_loss: 0.6877
```

The Embeddig layer receives pre-trained word embedding. Setting this to False when calling the Embedding layer guarantees that it cannot be trained. Setting trainable = True will allow the optimization procedure to alter the word embedding settings. To keep students from forgetting what they already "know," it is advisable to avoid updating pretrained parts while they are still receiving instruction.

```
import matplotlib.pyplot as plt

# Extract metrics from training history
train_acc = training_history.history['acc']
val_acc = training_history.history['val_acc']
loss_train = training_history.history['loss']
loss_val = training_history.history['val_loss']

# Epochs for x-axis
epoch_range = range(1, len(train_acc) + 1)

# Plot training vs validation accuracy
plt.plot(epoch_range, train_acc, 'grey', label='Training Accuracy')
plt.plot(epoch_range, val_acc, 'blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.figure()

# Plot training vs validation loss
plt.plot(epoch_range, loss_train, 'red', label='Training Loss')
plt.plot(epoch_range, loss_val, 'blue', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```






```
# Evaluate the GloVe-based model on the test dataset
final_loss, final_accuracy = glove_text_model.evaluate(x_test, y_test)

# Display test performance
print("Test Loss:", round(final_loss, 4))
print("Test Accuracy:", round(final_accuracy, 4))
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - acc: 0.5045 - loss: 0.6932
Test Loss: 0.6934
Test Accuracy: 0.501
```

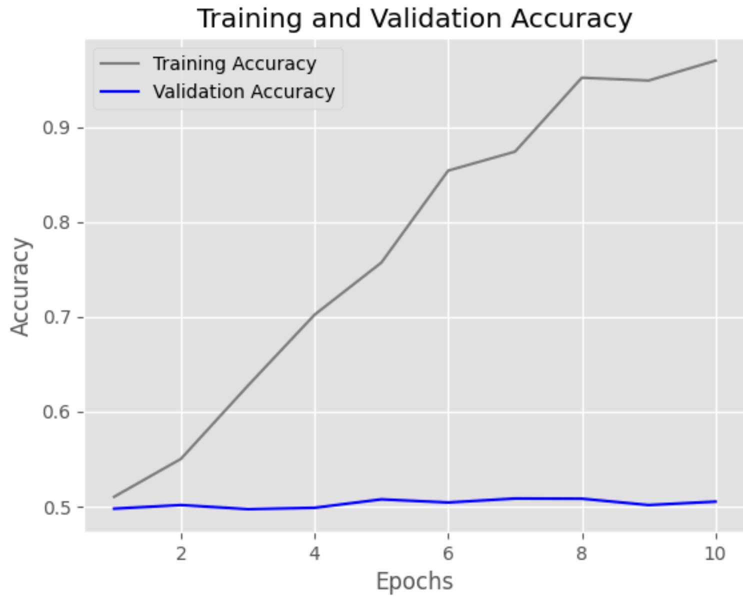
pretrained word embedding layer with training sample size = 5000

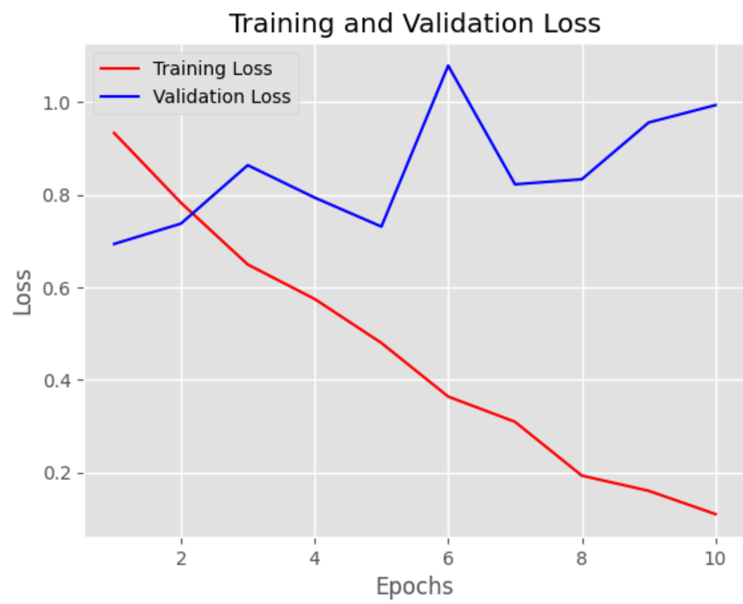
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

# Define model with renamed variables
glove_dense_model = Sequential()
glove_dense_model.add(
    Embedding(
        input_dim=vocab_size,
        output_dim=embedding_dim,
        input_length=max_review_length
    )
```

```
)
glove_dense_model.add(Flatten())
glove_dense_model.add(Dense(32, activation='relu'))
glove_dense_model.add(Dense(1, activation='sigmoid'))

# Explicitly build model to load pretrained weights
glove_dense_model.build(input_shape=(None, max_review_length))

# Load pretrained GloVe weights into the embedding layer
glove_dense_model.layers[0].set_weights([embedding_weights])
glove_dense_model.layers[0].trainable = False  # Freeze embedding layer

# Compile the model
glove_dense_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Train the model
history_dense_glove = glove_dense_model.fit(
    x_train_set, y_train_set,
    epochs=10,
    batch_size=32,
    validation_data=(x_val_set, y_val_set)
)

# Save the trained model weights
glove_dense_model.save_weights('glove_dense_frozen.weights.h5')
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 520ms/step - acc: 0.5021 - loss: 2.9554 - val_acc: 0.5057 - val_loss: 2.5293
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 445ms/step - acc: 0.5417 - loss: 1.9877 - val_acc: 0.4958 - val_loss: 2.1258
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 444ms/step - acc: 0.8379 - loss: 0.4990 - val_acc: 0.5012 - val_loss: 1.2312
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 360ms/step - acc: 0.8848 - loss: 0.2374 - val_acc: 0.5058 - val_loss: 2.5900
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 870ms/step - acc: 0.7685 - loss: 0.4407 - val_acc: 0.5124 - val_loss: 1.0004
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 448ms/step - acc: 1.0000 - loss: 0.0470 - val_acc: 0.5509 - val_loss: 0.7803
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 349ms/step - acc: 1.0000 - loss: 0.0243 - val_acc: 0.5596 - val_loss: 0.7630
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 345ms/step - acc: 1.0000 - loss: 0.0152 - val_acc: 0.5601 - val_loss: 0.7674
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 447ms/step - acc: 1.0000 - loss: 0.0118 - val_acc: 0.5607 - val_loss: 0.7739
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 442ms/step - acc: 1.0000 - loss: 0.0087 - val_acc: 0.5564 - val_loss: 0.7853
```

```
# Plot Accuracy
plt.plot(epochs_range, train_acc, 'grey', label='Training Accuracy')
plt.plot(epochs_range, val_acc, 'blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.figure()

# Plot Loss
plt.plot(epochs_range, train_loss, 'red', label='Training Loss')
plt.plot(epochs_range, val_loss, 'blue', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.show()
```






```
# Evaluate the GloVe-based dense model on the test set
test_loss_dense, test_accuracy_dense = glove_dense_model.evaluate(x_test, y_test)

# Print test performance
print("Test Loss:", round(test_loss_dense, 4))
print("Test Accuracy:", round(test_accuracy_dense, 4))
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - acc: 0.4977 - loss: 0.8272
Test Loss: 0.8202
Test Accuracy: 0.5059
```

pretrained word embedding layer with training sample size = 1000

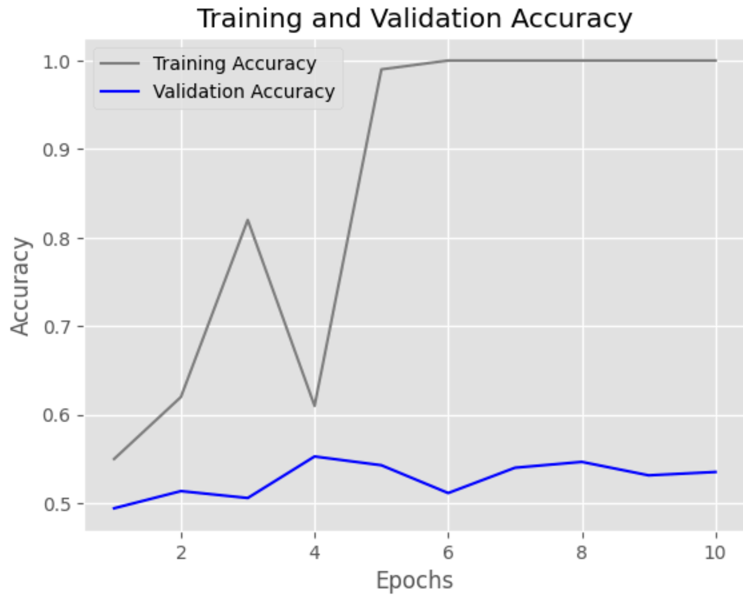
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

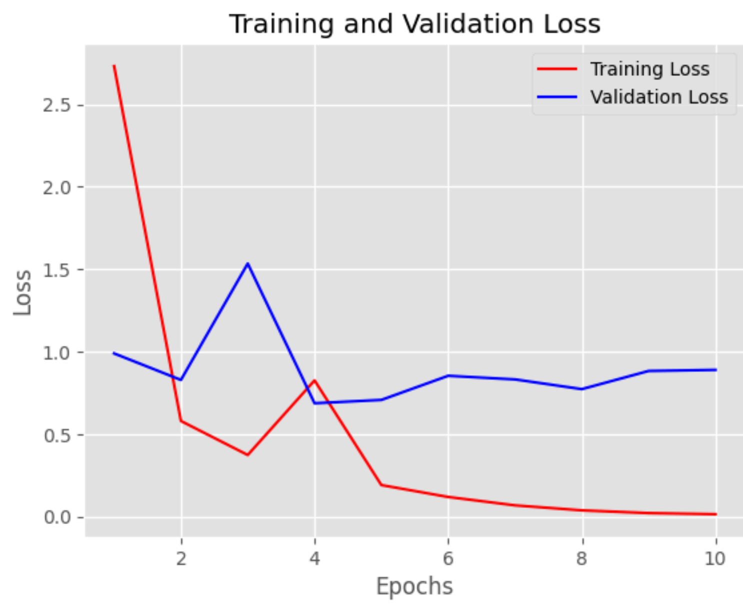
```
# Force embedding layer to initialize its weights before loading GloVe
glove_dense_model.layers[0].build(input_shape=(None, max_review_length))
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
```

```
from tensorflow.keras.initializers import Constant
import matplotlib.pyplot as plt

# Define the model
glove_dense_model = Sequential()
glove_dense_model.add(
    Embedding(
        input_dim=vocab_size,
        output_dim=embedding_dim,
        input_length=max_review_length
    )
)
glove_dense_model.add(Flatten())
glove_dense_model.add(Dense(32, activation='relu'))
glove_dense_model.add(Dense(1, activation='sigmoid'))

# Force the embedding layer to build its weights
glove_dense_model.layers[0].build(input_shape=(None, max_review_length))

# Set pretrained GloVe embeddings
glove_dense_model.layers[0].set_weights([embedding_weights])
glove_dense_model.layers[0].trainable = False

# Compile the model
glove_dense_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Fit the model
history_glove_dense = glove_dense_model.fit(
    x_train_set, y_train_set,
    epochs=10,
    batch_size=32,
    validation_data=(x_val_set, y_val_set)
)

# Save the model weights
glove_dense_model.save_weights('glove_dense_frozen.weights.h5')

# --- Plot Training & Validation Results ---

train_acc = history_glove_dense.history['acc']
val_acc = history_glove_dense.history['val_acc']
train_loss = history_glove_dense.history['loss']
val_loss = history_glove_dense.history['val_loss']
epoch_range = range(1, len(train_acc) + 1)

# Accuracy Plot
plt.plot(epoch_range, train_acc, 'grey', label='Training Accuracy')
plt.plot(epoch_range, val_acc, 'blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()

# Loss Plot
plt.plot(epoch_range, train_loss, 'red', label='Training Loss')
plt.plot(epoch_range, val_loss, 'blue', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 537ms/step - acc: 0.5315 - loss: 2.2784 - val_acc: 0.4943 - val_loss: 0.9892
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 449ms/step - acc: 0.6188 - loss: 0.5857 - val_acc: 0.5137 - val_loss: 0.8285
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 368ms/step - acc: 0.7843 - loss: 0.3966 - val_acc: 0.5059 - val_loss: 1.5342
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 443ms/step - acc: 0.6430 - loss: 0.7358 - val_acc: 0.5528 - val_loss: 0.6870
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 443ms/step - acc: 0.9845 - loss: 0.2182 - val_acc: 0.5429 - val_loss: 0.7073
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 442ms/step - acc: 1.0000 - loss: 0.1193 - val_acc: 0.5115 - val_loss: 0.8536
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - acc: 1.0000 - loss: 0.0697 - val_acc: 0.5401 - val_loss: 0.8317
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 444ms/step - acc: 1.0000 - loss: 0.0372 - val_acc: 0.5467 - val_loss: 0.7727
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 442ms/step - acc: 1.0000 - loss: 0.0198 - val_acc: 0.5315 - val_loss: 0.8829
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 882ms/step - acc: 1.0000 - loss: 0.0152 - val_acc: 0.5353 - val_loss: 0.8896
```






```
# Evaluate the GloVe-based dense model on the test set
final_test_loss, final_test_accuracy = glove_dense_model.evaluate(x_test, y_test)

# Print test performance
print("Test Loss:", round(final_test_loss, 4))
print("Test Accuracy:", round(final_test_accuracy, 4))
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - acc: 0.4953 - loss: 0.9962
Test Loss: 0.9783
Test Accuracy: 0.5032
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.initializers import Constant

# Parameters
max_len = 150
train_samples = 1000
val_samples = 10000
vocab_limit = 10000
embedding_dim = 100

# Tokenization
tokenizer_glove = Tokenizer(num_words=vocab_limit)
tokenizer_glove.fit_on_texts(review_texts)
sequence_data = tokenizer_glove.texts_to_sequences(review_texts)
word_idx = tokenizer_glove.word_index
print("Found %s unique tokens." % len(word_idx))

# Padding
padded_data = pad_sequences(sequence_data, maxlen=max_len)
label_array = np.asarray(review_labels)
print("Shape of data tensor:", padded_data.shape)
print("Shape of label tensor:", label_array.shape)

# Shuffle and split
indices = np.arange(padded_data.shape[0])
np.random.shuffle(indices)
padded_data = padded_data[indices]
label_array = label_array[indices]

x_train_embed = padded_data[:train_samples]
y_train_embed = label_array[:train_samples]
x_val_embed = padded_data[train_samples:train_samples + val_samples]
y_val_embed = label_array[train_samples:train_samples + val_samples]

# Build embedding matrix
embedding_matrix_final = np.zeros((vocab_limit, embedding_dim))
for word, idx in word_idx.items():
    if idx < vocab_limit:
        vec = glove_embeddings.get(word)
        if vec is not None:
            embedding_matrix_final[idx] = vec

# Define the model
glove_flat_model = Sequential()
glove_flat_model.add(Embedding(
    input_dim=vocab_limit,
    output_dim=embedding_dim,
    embeddings_initializer=Constant(embedding_matrix_final),
    input_shape=(max_len,),
    trainable=False
))
glove_flat_model.add(Flatten())
glove_flat_model.add(Dense(32, activation='relu'))
glove_flat_model.add(Dense(1, activation='sigmoid'))

# Show model summary
glove_flat_model.summary()

# Compile the model
glove_flat_model.compile(optimizer='rmsprop',
                         loss='binary_crossentropy',
                         metrics=['acc'])

# Train the model
history_flat = glove_flat_model.fit(
    x_train_embed, y_train_embed,
    epochs=10,
    batch_size=32
```

```
    batch_size=32,
    validation_data=(x_val_embed, y_val_embed)
)

# Save weights
glove_flat_model.save_weights('glove_flat_model.weights.h5')

# --- Plot Training & Validation Metrics ---

train_acc = history_flat.history['acc']
val_acc = history_flat.history['val_acc']
train_loss = history_flat.history['loss']
val_loss = history_flat.history['val_loss']
epochs_range = range(1, len(train_acc) + 1)

# Accuracy Plot
plt.plot(epochs_range, train_acc, 'grey', label='Training Accuracy')
plt.plot(epochs_range, val_acc, 'blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()

# Loss Plot
plt.plot(epochs_range, train_loss, 'red', label='Training Loss')
plt.plot(epochs_range, val_loss, 'blue', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
```