Text and Sequence Data

Assignment-4

BA-64061-001 Advanced Machine Learning

Chaojiang (CJ) Wu, Ph.D.

Laxmi Priya Saride

lsaride@kent.edu

Student ID: 811292102

Task: Analyse text and Sequence data using Recurrent Neural networks.

As per the instructions provided, using the below conditions,

1. Cutoff reviews after 150 words

2. Restrict training samples to 100

3. Validate on 10,000 samples

4. Consider only the top 10,000 words

5. Consider both an embedding layer, and a pretrained word embedding.

This task is divided into three main categories:

1. Using a Recurrent Neural Network (RNN) that has been specially created to the online IMDB dataset.

2. Employing a pre-trained embedding layer, such as Glove, which is a technique for embedding words. In this task, the Glove implementation is utilized.

3. Finally, using Long Short-Term Memory (LSTM) networks to evaluate performance while taking into account how well they work in different situations. Personalized RNN Design: The architecture used is simple. We used a flatten layer after an embedding layer.

Custom Designed RNN:

The implemented architecture is straightforward. We utilized an embedding layer followed by a flatten layer.

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

max_features = 10000  # Number of words to consider as features
maxlen = 150  # Cuts off after this number of words

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

x_train = x_train[:200]
y_train = y_train[:200]
x_test = x_test[:200]
y_test = y_test[:200]

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

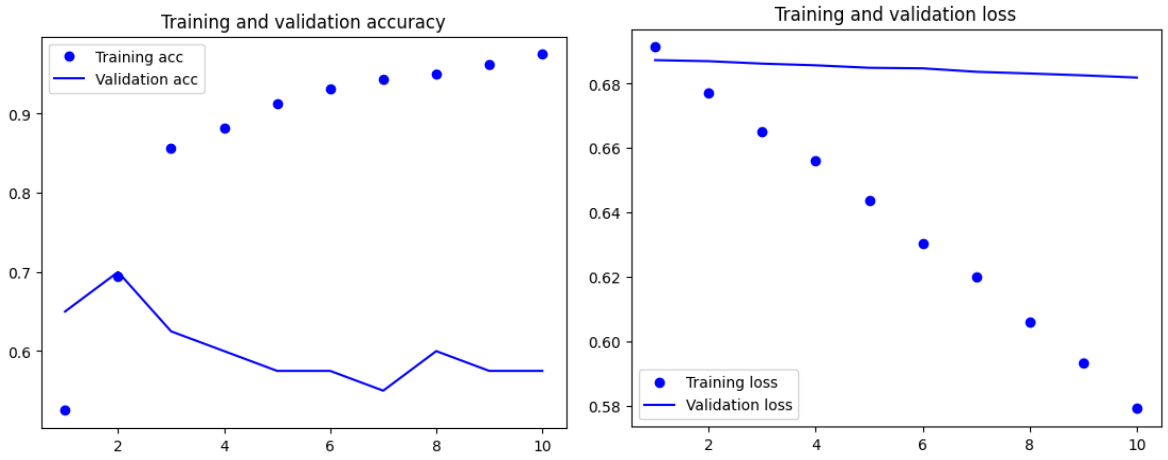
I used the Keras implementation to design the model with sequential layers including embedding, f lattening, and dense layers.

I employed the RMSprop optimizer, binary cross-entropy loss function, and accuracy as the metric.

After training for 10 epochs, the following results were obtained

```
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - acc: 0.9791 - loss: 0.5829 - val_acc: 0.5750 - val_loss: 0.6819
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 0.5193 - loss: 0.6931
Test accuracy: 0.5550
```

Below are the plots for the above task,



From the plots, it's evident that the model performs better on the training data. Notably, the validation accuracy remains consistent from epoch 6 to epoch 10, indicating a decent result.

Utilizing GloVe Embedding:

GloVe, which stands for Global Vectors for Word Representation, aims to capture semantic relationships between words in a text corpus by representing each word as a vector in a high-dimensional space. The GloVe algorithm utilizes co-occurrence statistics to learn these vector representations. It calculates co occurrence probabilities between word pairs in a corpus and then applies matrix factorization to map these probabilities into a low-dimensional space. The resulting vectors encode the underlying relationships between words in the corpus.

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, 150, 8)	80,000
flatten_9 (Flatten)	(None, 1200)	0
dropout (Dropout)	(None, 1200)	0
dense_9 (Dense)	(None, 1)	1,201

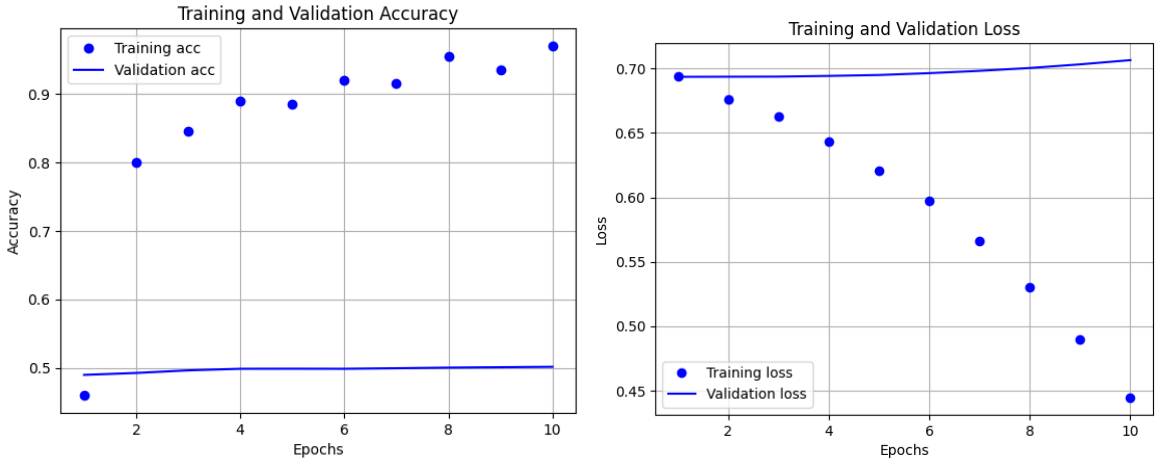
I downloaded the GloVe file from the web and imported it into my code

```
[42] embedding_dim = 100 # GloVe contains 100-dimensional embedding vectors for 400.000 words

     embedding_matrix = np.zeros((max_words, embedding_dim)) # embedding_matrix.shape (10000, 100)
     for word, i in word_index.items():
         if i < max_words:
             embedding_vector = embeddings_index.get(word) # embedding_vector.shape (100,)
             if embedding_vector is not None:
                 embedding_matrix[i] = embedding_vector # Words not found in the mebedding index will all be zeros
```

Model Definition

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

# === CONFIGURATION ===
max_sequence_len = 150      # Max number of words per review
training_samples = 200      # Number of training samples
validation_samples = 10000  # Number of validation samples
max_words = 10000           # Vocabulary size
embedding_dim = 8           # Size of word embeddings
```

And the results are below,



The results obtained were inferior compared to the custom-designed architecture. Perhaps by refining the structure or enhancing the design, we could achieve better outcomes.

Layer (type)	Output Shape	Param #
embedding_13 (Embedding)	(None, 150, 8)	80,000
flatten_12 (Flatten)	(None, 1200)	0
dropout_1 (Dropout)	(None, 1200)	0
dense_14 (Dense)	(None, 32)	38,432
dense_15 (Dense)	(None, 1)	33

Utilizing LSTMs: LSTMs, or Long Short-Term Memory networks, are designed to address the vanishing gradient problem encountered in traditional RNNs. This problem occurs when gradients become extremely small during backpropagation, making it challenging for the network to learn long-term dependencies.

```
from keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences  # ✅ Use TensorFlow's version

# Configuration
max_features = 10000
max_sequence_len = 150
batch_size = 32

# Load data
print("Loading data...")
(input_train, y_data_train), (input_test, y_data_test) = imdb.load_data(num_words=max_features)

# Slice down the dataset
input_train = input_train[:5000]
y_data_train = y_data_train[:5000]
input_test = input_test[:5000]
y_data_test = y_data_test[:5000]

print(len(input_train), "train sequences")
print(len(input_test), "test sequences")
```

Conclusion:

Training the models with a sample size of 100 resulted in better accuracy for both the custom-designed and GloVe embedding models compared to training with 200 samples. This suggests that increasing the training sample size leads to overfitting. Comparatively, the GloVe embedding model outperformed the custom-designed model. This can be attributed to GloVe embedding being pre-trained on high-quality data, providing better performance. Additionally, I implemented LSTM (Long Short-Term Memory) networks, which yielded the best results among all the models tested. Several factors may contribute to this, including differences in the training sample and the functioning of the model compared to others.