AML – Assignment 4 – Thanasit
Recurrent Neural Network for Text Analysis

1. Summary

The One-Hot coding model demonstrates very promising performance, achieving a testing accuracy of 88%. Compared to the Embedded model and the pre-trained GLOVE word embedding model, the One-Hot model shows a significant advantage, with an accuracy that is a solid 6% higher. The adjusted GLOVE embedding model performs equivalently, achieving a testing accuracy of 87.7%, comparable to the One-Hot coding model. The key benefit of using the adjusted GLOVE model is its faster training time. Training the One-Hot model for 10 epochs took approximately 2 hours, whereas the adjusted GLOVE model required only 40 minutes to train for 20 epochs.

Model Comparison Table

Model Name	Model 1 - One_Hot	Model 2 - Embedded Layer	Model 3 - GLOVE Word Embbeded	Model 4 - Adjust GLOVE Word Embedded
Word Length	600	150	150	150
Top Words	20000	10000	10000	20000
Training Size	20000	15000	15000	15000
Validation Size	5000	10000	10000	10000
Testing Size	25000	25000	25000	25000
Model	One Hot	Embedded	Pre-trained Embedded	Pre-trained Embedded
Number of Hidden Layers	1	1	1	2
Highest Number of Nodes	32	32	32	128
validation Loss	0.2915	0.3920	0.3980	0.370
Validation Accuracy	89.48%	84.25%	82.29%	88.5%
Testing Accuracy	88.0%	82.9%	82.5%	87.7%
Epochs	4	4	16	17

2. Data Preparation

In this analysis, the IMDB review dataset is used. The dataset consists of two classes: negative reviews and positive reviews. I created two different datasets, A and B.

A. Word cutoff = 600 words, Training sample = 32, Validation sample = 5,000 samples (2,500 samples for each class), Top words sequence = 20,000 words.

```
batch_size = 32
base_dir = pathlib.Path("aclImdb")
val_dir = base_dir / "val"
train_dir = base_dir / "train"

for category in ("neg", "pos"):
    os.makedirs(val_dir / category, exist_ok=True)
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples = int(0.2 * len(files)) # total of 5000 samples
    val_files = files[-num_val_samples:]
    for fname in val_files:
        shutil.move(train_dir / category / fname,
                    val_dir / category / fname)
```

```
max_length = 600
max_tokens = 20000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
```

B. Word cutoff = 150 words, Training samples = 100, Validation Sample = 10,000 samples (5,000 samples for each class), Top words sequence = 10,000 words.

```
batch_size2 = 100 # Restrict training samples to 100
base_dir2 = pathlib.Path("aclImdb2")
val_dir2 = base_dir2 / "val"
train_dir2 = base_dir2 / "train"

for category in ("neg", "pos"):
    os.makedirs(val_dir2 / category, exist_ok=True)
    files2 = os.listdir(train_dir2 / category)
    random.Random(1337).shuffle(files)
    num_val_samples2 = 5000 # 5000 for each class, in total of 10000 validation samples
    val_files2 = files2[-num_val_samples2:]
    for fname in val_files2:
        shutil.move(train_dir2 / category / fname,
                    val_dir2 / category / fname)
```

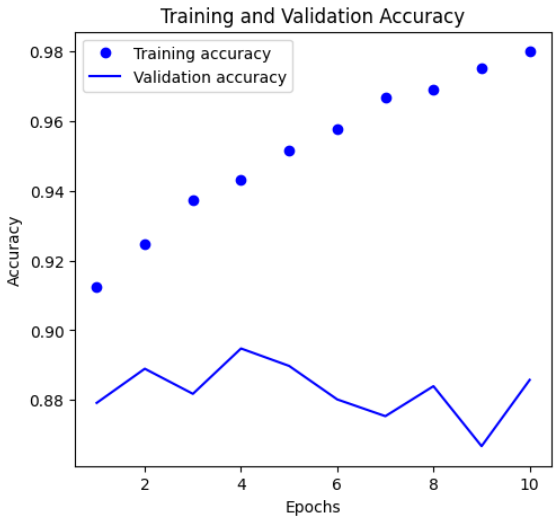
```
max_length2 = 150
max_tokens2 = 10000
text_vectorization2 = layers.TextVectorization(
    max_tokens=max_tokens2,
    output_mode="int",
    output_sequence_length=max_length2,
)
```

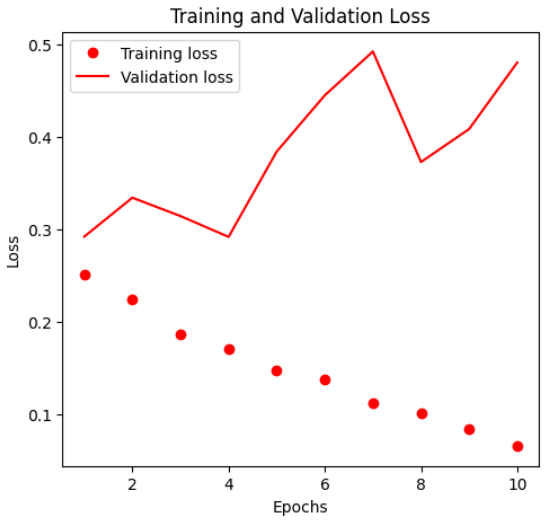
3. Model 1 – One Hot model

The first model utilized the One-hot coding technique to create a text analysis model, using Dataset A. The architecture includes one hidden bidirectional LSTM layer with 32 nodes. The activation function is 'Sigmoid,' and the optimizer is 'RMSprop.'

The model achieved its best performance with a validation loss of 0.2915 and a validation accuracy of 89.48% at epoch 4. The testing accuracy is 88.0%.

```
import tensorflow as tf
inputs = keras.Input(shape=(None,), dtype="int64")

class OneHotEncodingLayer(layers.Layer):
    def __init__(self, depth, **kwargs):
        super().__init__(**kwargs)
        self.depth = depth

    def call(self, inputs):
        return tf.one_hot(inputs, depth=self.depth)

    def compute_output_shape(self, input_shape):
        return input_shape + (self.depth,)

    def get_config(self):
        config = super().get_config()
        config.update({"depth": self.depth})
        return config

embedded = OneHotEncodingLayer(depth=max_tokens)(inputs)

x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```



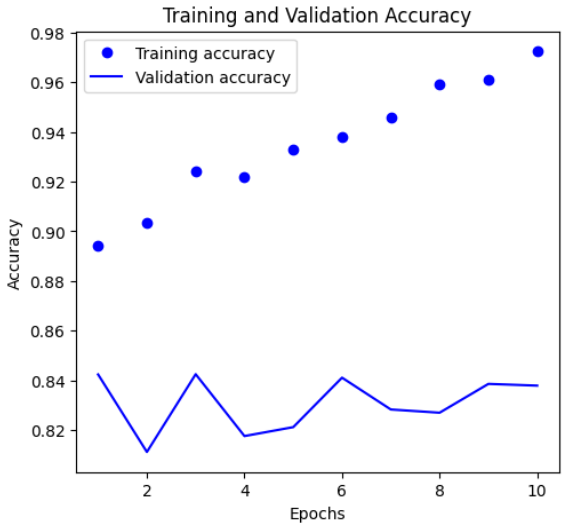


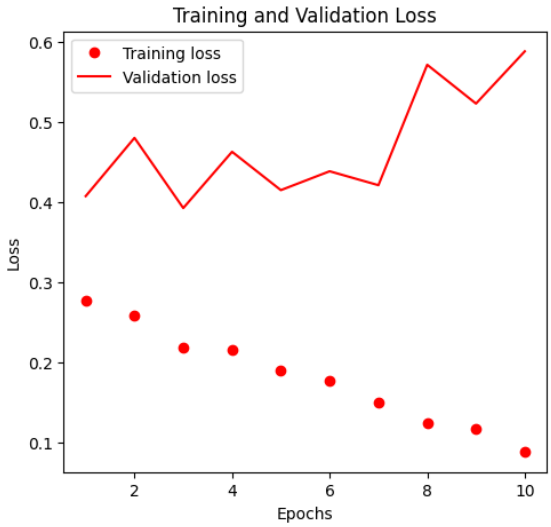
4. Model 2 – Embedded model

Second model used the word embedded technique to create a text analysis model. In this model, dataset B is used. The model configuration is mostly the same as the first model. There is one hidden bidirectional layer with LSTM 32 nodes. Activation function is 'Sigmoid' and optimizer is 'rmsprop'.

The model best performance has validation loss equal to 0.392 and the validation accuracy is 84.25% at epoch 4. The testing accuracy is 82.9%. The embedded model gives a little bit less performance compares to One hot coding model.

```
inputs2 = keras.Input(shape=(None,), dtype="int64")
embedded2 = layers.Embedding(input_dim=max_tokens2, output_dim=100)(inputs2)
x2 = layers.Bidirectional(layers.LSTM(32))(embedded2)
x2 = layers.Dropout(0.5)(x2)
outputs2 = layers.Dense(1, activation="sigmoid")(x2)
model2 = keras.Model(inputs2, outputs2)
model2.compile(optimizer="rmsprop",
               loss="binary_crossentropy",
               metrics=["accuracy"])
model2.summary()
```





5. Model 3 – Pre-trained GLOVE Embedded model

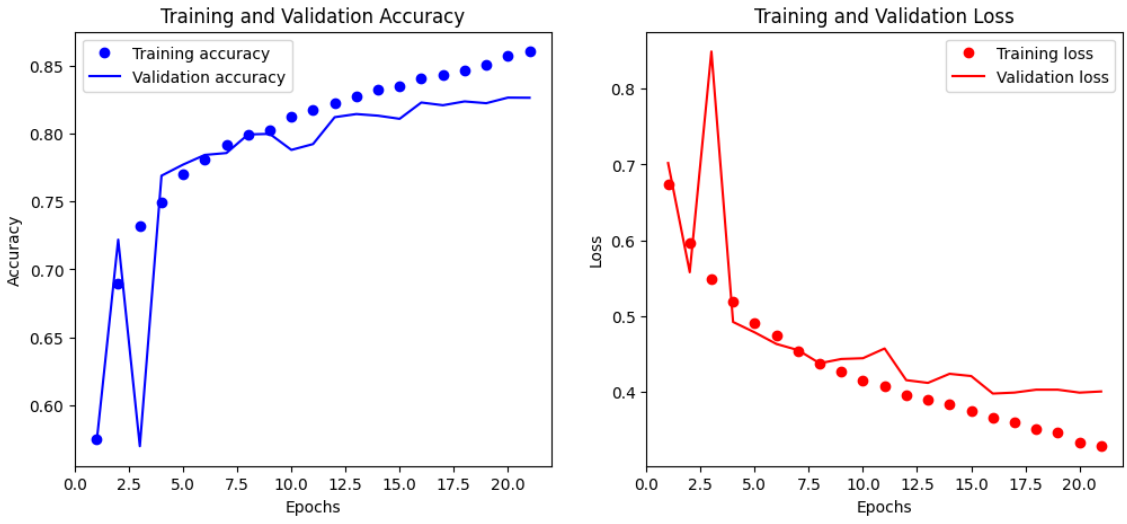
The second model used the word embedding technique to create a text analysis model, utilizing Dataset B. The model configuration is largely the same as the first model, with one hidden bidirectional LSTM layer containing 32 nodes. The activation function is 'Sigmoid,' and the optimizer is 'RMSprop.'

The model achieved its best performance with a validation loss of 0.392 and a validation accuracy of 84.25% at epoch 4. The testing accuracy is 82.9%. The embedded model shows slightly lower performance compared to the One-hot coding model.

```
import numpy as np
path_to_glove_file3 = "glove/glove.6B.100D.txt"

embeddings_index3 = {}
with open(path_to_glove_file3) as f:
    for line in f:
        word3, coefs3 = line.split(maxsplit=1)
        coefs3 = np.fromstring(coefs3, "f", sep=" ")
        embeddings_index3[word3] = coefs3

print(f"Found {len(embeddings_index3)} word vectors.")
```

```
embedding_layer3 = layers.Embedding(
    max_tokens2,
    embedding_dim3,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix3),
    trainable=False,
    mask_zero=True,
)
```

```
inputs3 = keras.Input(shape=(None,), dtype="int64")
embedded3 = embedding_layer3(inputs3)
x3 = layers.Bidirectional(layers.LSTM(32))(embedded3)
x3 = layers.Dropout(0.5)(x3)
outputs3 = layers.Dense(1, activation="sigmoid")(x3)
model3 = keras.Model(inputs3, outputs3)
model3.compile(optimizer="rmsprop",
               loss="binary_crossentropy",
               metrics=["accuracy"])
model3.summary()
```



6. Model 4 – Adjust Pre-trained GLOVE Embedded model

In the final model, I aimed to improve the pre-trained GLOVE model. I decided to use a larger training dataset, Dataset A. Additionally, I adjusted the model configuration by adding one more hidden bidirectional layer, resulting in a total of two hidden layers. The number of LSTM nodes was increased to 64 and 128 nodes. In the output layer, I incorporated L2 regularization with a lambda value of 0.001. Lastly, the optimizer was changed to 'Adam' with a learning rate of 0.001.

The model achieved its best performance with a validation loss of 0.37 and a validation accuracy of 88.5% at epoch 17. The testing accuracy was 87.7%. The adjusted GLOVE embedded model demonstrates significantly better performance compared to the original GLOVE embedded model. It achieves the same level of accuracy as the One-hot coding model but with reduced computational complexity and less time consumption.

```
inputs4 = keras.Input(shape=(None,), dtype="int64")
embedded4 = embedding_layer4(inputs4)
x4 = layers.Bidirectional(layers.LSTM(64, return_sequences=True))(embedded4)
x4 = layers.Bidirectional(layers.LSTM(128))(x4)
x4 = layers.Dropout(0.3)(x4)
x4 = layers.BatchNormalization()(x4)
outputs4 = layers.Dense(1, activation="sigmoid", kernel_regularizer=keras.regularizers.l2(0.001))(x4)
model4 = keras.Model(inputs4, outputs4)
model4.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
               loss="binary_crossentropy",
               metrics=["accuracy"])
model4.summary()
```