Introduction to Neural Networks Assignment 07
Movie Rating Regression (But how bad was it really?...)

Mangesh Sakordekar

Aim of this project is to use IMDB dataset to predict the actual rating a movie

- Read and process IMDB data.
- Extract the ratings from the file names.
- Split data into training, validation and test data sets.
- Create the text to integer map using the training data set.
- Vectorize the reviews.
- Train neural net model to predict movie rating.
- Test the trained model using test data.

Reading the Data

The data was read by crawling through the directories using the os module and stored in variable. The code for this section can be seen in the screenshot below.

```
import os

data_dir = 'aclImdb'

def load_data(directory):
    reviews = []
    ratings = []
    for category in ['pos', 'neg']:
        path = os.path.join(directory, category)
        for filename in os.listdir(path):
            if filename.endswith('.txt'):
                with open(os.path.join(path, filename), 'r', encoding='utf-8') as file:
                    review = file.read()
                    reviews.append(review)
                    # Extract the rating from the filename
                    rating = int(filename.split('_')[-1].split('.')[0])
                    ratings.append(rating)
    return reviews, ratings

train_dir = os.path.join(data_dir, 'train')
test_dir = os.path.join(data_dir, 'test')
val_dir = os.path.join(data_dir, 'val')

# Load train and test data
train_reviews, train_ratings = load_data(train_dir)
test_reviews, test_ratings = load_data(test_dir)
val_reviews, val_ratings = load_data(test_dir)
```

This code reads the files in each of the folders in the aclImdb directory (train, test and validation) and stores them in a list. It also extracts the rating associated with each review from the file name and stores them in a separate list.

Vectorization

To create the map, code from chapter 11 notebook was used. Once the mapping was generated, the train, test and validation reviews were vectorized and stored.

```
batch_size = 32
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=batch_size
)
text_only_train_ds = train_ds.map(lambda x, y: x)
```

```
Found 20000 files belonging to 2 classes.
```

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

```
train_reviews_vectorized = text_vectorization(train_reviews)
test_reviews_vectorized = text_vectorization(test_reviews)
val_reviews_vectorized = text_vectorization(val_reviews)
```

Data Preprocessing

Before the data was used to train the model, it was converted into numpy arrays.

```
#Convert the data to numpy arrays
train_data = np.array(train_reviews_vectorized)
train_labels = np.array(train_ratings).astype(float)
test_data = np.array(test_reviews_vectorized)
test_labels = np.array(test_ratings).astype(float)
val_data = np.array(val_reviews_vectorized)
val_labels = np.array(val_ratings).astype(float)
```

Model and Training

Model used in this assignment is the model from chapter 11 part 2 notebook which uses an embedding layer. The model was modified to fit for linear regression and to reduce the runtimes. The activation function for the output layer was was changed to ”selu”. The loss function used was mean squared error and the metrics was set to mean absolute error. The model was trained for 15 epochs and best model was saved.

```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(
    input_dim=max_tokens, output_dim=64, mask_zero=True)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.4)(x)
outputs = layers.Dense(1, activation="selu")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["mae"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings.keras",
                                    save_best_only=True)
]
history = model.fit(train_data, train_labels, validation_data=(val_data, val_labels),
                    epochs=15, callbacks=callbacks)
model = keras.models.load_model("embeddings.keras")
print(f"Test MAE: {model.evaluate(test_data, test_labels)[1]:.2f}")
```

```
Model: "model_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_15 (InputLayer)       [(None, None)]            0

 embedding_14 (Embedding)    (None, None, 64)          1280000

 bidirectional_14 (Bidirecti  (None, 64)               24832
 onal)

 dropout_39 (Dropout)        (None, 64)                0

 dense_59 (Dense)            (None, 1)                 65

=================================================================
Total params: 1,304,897
Trainable params: 1,304,897
Non-trainable params: 0
```

Through multiple runs of training the model, the training loss achieved was around 0.9849 and mean absolute error was 0.7305. Unfortunately, I was not able to achieve a validation loss below 1. The lowest achieved was 4.5657 and the mean absolute error was 1.5144. The test MAE was 1.58 for this particular run. Training the model took approximately 50 minutes.

```
Epoch 1/15
625/625 [==============================] - 207s 320ms/step - loss: 8.6084 - mae: 2.3803 - val_loss: 5.4367 - val_mae: 1.8100
Epoch 2/15
625/625 [==============================] - 208s 334ms/step - loss: 4.9825 - mae: 1.7073 - val_loss: 5.5625 - val_mae: 1.7775
Epoch 3/15
625/625 [==============================] - 201s 322ms/step - loss: 4.0256 - mae: 1.5093 - val_loss: 4.3889 - val_mae: 1.5849
Epoch 4/15
625/625 [==============================] - 201s 322ms/step - loss: 3.4720 - mae: 1.3982 - val_loss: 4.4755 - val_mae: 1.6088
Epoch 5/15
625/625 [==============================] - 206s 330ms/step - loss: 3.0395 - mae: 1.3029 - val_loss: 4.9778 - val_mae: 1.6467
Epoch 6/15
625/625 [==============================] - 202s 324ms/step - loss: 2.6389 - mae: 1.1993 - val_loss: 4.5657 - val_mae: 1.5144
Epoch 7/15
625/625 [==============================] - 210s 336ms/step - loss: 2.2438 - mae: 1.1058 - val_loss: 5.2762 - val_mae: 1.6770
Epoch 8/15
625/625 [==============================] - 203s 325ms/step - loss: 2.0151 - mae: 1.0439 - val_loss: 4.8575 - val_mae: 1.5601
Epoch 9/15
625/625 [==============================] - 205s 328ms/step - loss: 1.7885 - mae: 0.9800 - val_loss: 4.8912 - val_mae: 1.5561
Epoch 10/15
625/625 [==============================] - 202s 323ms/step - loss: 1.6029 - mae: 0.9282 - val_loss: 4.7474 - val_mae: 1.5736
Epoch 11/15
625/625 [==============================] - 208s 333ms/step - loss: 1.4310 - mae: 0.8777 - val_loss: 4.8815 - val_mae: 1.5455
Epoch 12/15
625/625 [==============================] - 204s 327ms/step - loss: 1.2865 - mae: 0.8302 - val_loss: 4.9545 - val_mae: 1.5993
Epoch 13/15
625/625 [==============================] - 211s 338ms/step - loss: 1.1697 - mae: 0.7906 - val_loss: 5.2523 - val_mae: 1.6436
Epoch 14/15
625/625 [==============================] - 203s 324ms/step - loss: 1.0624 - mae: 0.7564 - val_loss: 5.0303 - val_mae: 1.6319
Epoch 15/15
625/625 [==============================] - 208s 333ms/step - loss: 0.9849 - mae: 0.7305 - val_loss: 5.1729 - val_mae: 1.6132
782/782 [==============================] - 76s 90ms/step - loss: 4.3889 - mae: 1.5849
Test MAE: 1.58
```

Testing the Model

To see how the model was predicting, 10 random samples form the test data were selected and given to the model for predicting. The results can be seen below.

```
random_ind = [random.randint(0, 25000) for _ in range(10)]
for i in random_ind:
    print("predicted rating: " + str(model.predict(np.reshape(test_data[i, :], (1,600,1)), verbose=0)[0][0])
          + "\tactual rating: " + str(test_labels[i]))
```

```
predicted rating: 5.858356      actual rating: 9.0
predicted rating: 5.400948      actual rating: 7.0
predicted rating: 8.505113      actual rating: 7.0
predicted rating: 3.4114602     actual rating: 4.0
predicted rating: 8.419339      actual rating: 8.0
predicted rating: 8.477797      actual rating: 10.0
predicted rating: 3.3568032     actual rating: 3.0
predicted rating: 8.123314      actual rating: 10.0
predicted rating: 9.6205435     actual rating: 9.0
predicted rating: 1.3601909     actual rating: 1.0
```