Training Day-79 Report:

1. Sentiment Analysis Hands-On

Sentiment Analysis is a common NLP task where the goal is to determine the sentiment (e.g., positive, negative, neutral) of a given text.

Steps for Sentiment Analysis:

1. **Data Preparation**: Use a dataset like IMDb Movie Reviews or any labeled sentiment dataset.
2. **Preprocessing**:
   - Tokenization.
   - Padding sequences to a fixed length.
   - Encoding labels.
3. **Building the Model**: Use an Embedding layer followed by LSTM/GRU or CNN for feature extraction.
4. **Training**: Train the model to classify sentiments.
5. **Evaluation**: Test the model on unseen data.

Implementation in TensorFlow:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Load IMDb dataset
vocab_size = 10000
max_len = 200
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Pad sequences to ensure uniform input length
x_train = pad_sequences(x_train, maxlen=max_len)
```

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

# Build the sentiment analysis model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=max_len),  # Embedding layer
    LSTM(64, return_sequences=False),                                # LSTM layer
    Dropout(0.5),                                            # Dropout for regularization
    Dense(1, activation='sigmoid')                               # Output layer for binary classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=3, batch_size=64, validation_data=(x_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy}")

# Make predictions
sample_review = "The movie was fantastic and very engaging!"
encoded_review = [1] + [word for word in sample_review.split() if word in imdb.get_word_index()]
padded_review = pad_sequences([encoded_review], maxlen=max_len)
prediction = model.predict(padded_review)
print(f"Sentiment: {'Positive' if prediction[0][0] > 0.5 else 'Negative'}")
```

2. Seq-to-Seq Model

Sequence-to-Sequence (Seq2Seq) models are used for tasks like machine translation, summarization, and chatbot applications. These models take a sequence as input and generate another sequence as output.

Seq-to-Seq Architecture:

1. **Encoder**:
   - Encodes the input sequence into a fixed-length context vector.
   - Typically uses RNNs, LSTMs, or GRUs.
2. **Decoder**:
   - Takes the context vector as input and generates the output sequence.
3. **Attention Mechanism**:
   - Enhances performance by allowing the decoder to focus on specific parts of the input sequence at each time step.

Seq-to-Seq Implementation for Machine Translation:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.models import Model
import numpy as np

# Sample data
input_texts = ["hello", "how are you", "good morning"]
target_texts = ["hola", "cómo estás", "buenos días"]

# Tokenize data
input_tokenizer = tf.keras.preprocessing.text.Tokenizer()
target_tokenizer = tf.keras.preprocessing.text.Tokenizer()

input_tokenizer.fit_on_texts(input_texts)
target_tokenizer.fit_on_texts(target_texts)

input_sequences = input_tokenizer.texts_to_sequences(input_texts)
```

```
target_sequences = target_tokenizer.texts_to_sequences(target_texts)

input_data = tf.keras.preprocessing.sequence.pad_sequences(input_sequences,
padding='post')
target_data = tf.keras.preprocessing.sequence.pad_sequences(target_sequences,
padding='post')

# Define model parameters
num_encoder_tokens = len(input_tokenizer.word_index) + 1
num_decoder_tokens = len(target_tokenizer.word_index) + 1
latent_dim = 256

# Encoder
encoder_inputs = Input(shape=(None,))
encoder_embedding = Dense(64, activation='relu')(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)

# Decoder
decoder_inputs = Input(shape=(None,))
decoder_embedding = Dense(64, activation='relu')(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=[state_h, state_c])
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')

# Training
```

```
model.fit([input_data, target_data[:, :-1]], target_data[:, 1:], batch_size=64, epochs=50)

# Inference
def translate(input_seq):
    states = encoder_lstm.predict(input_seq)
    translated_seq = decoder_lstm.predict(states)
    return translated_seq

print("Translation:", translate(input_data[0:1]))
```

Summary:

1. **Sentiment Analysis**:
   - Used LSTM for sequence-based binary classification.
   - Applied text preprocessing, tokenization, and padding.
2. **Seq-to-Seq**:
   - Demonstrated architecture for sequence translation tasks.
   - Encoder-decoder structure handles input-to-output sequence transformation.