CENG442 TAKE-HOME FINAL EXAM

TEXT SUMMARIZATION SYSTEM FOR NEWS ARTICLES

SELÇUK ÜSTÜN 20050111072

1. Introduction

Text summarization is a crucial task in Natural Language Processing (NLP) that aims to generate concise summaries from lengthy documents while retaining essential information. This project develops a text summarization system for news articles using Sequence-to-Sequence (Seq2Seq) models with attention mechanisms. The system is implemented using TensorFlow, leveraging an Encoder-Decoder architecture to produce meaningful and coherent summaries. This report details the dataset, methodology, experimental results, and performance analysis of the developed system.

2. Dataset and Preprocessing

2.1 Dataset Description

The dataset used in this project contains news articles and their corresponding summaries provided in Parquet format. The dataset comprises diverse articles from various categories, ensuring variability for robust model training.

2.2 Data Preparation

1. **Loading the Dataset:**

```
import pandas as pd
df = pd.read_parquet('file_path.parquet')
```

2. **Splitting the Dataset:**
    - **Training Set:** 80%
    - **Validation Set:** 10%
    - **Test Set:** 10%

```
from sklearn.model_selection import train_test_split
train_data, temp_data = train_test_split(df, test_size=0.2, random_state=42)
val_data, test_data = train_test_split(temp_data, test_size=0.5, random_state=42)
```

3. **Preprocessing Steps:**
    - Lowercasing and cleaning the text.
    - Adding special tokens <sos> (start-of-sequence) and <eos> (end-of-sequence) to summaries.
    - Tokenizing and padding sequences.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer(num_words=30000, oov_token='<unk>')
tokenizer.fit_on_texts(train_texts + train_summaries)
X_train = pad_sequences(tokenizer.texts_to_sequences(train_texts), maxlen=100, padding='post')
Y_train = pad_sequences(tokenizer.texts_to_sequences(train_summaries), maxlen=30, padding='post')
```

3. Methodology

3.1 Model Architecture

The model utilizes an Encoder-Decoder architecture enhanced with an Additive Attention mechanism. The components are:

- Encoder:
    - LSTM layer with 512 units and a 50% dropout rate.
- Decoder:
    - LSTM layer with 512 units, initialized with the Encoder's final state.

- Attention Mechanism:
    - Additive Attention to enhance focus on relevant parts of the input sequence.

```
from tensorflow.keras.layers import AdditiveAttention

attention = AdditiveAttention(name="attention_layer")
attention_output = attention([decoder_lstm, encoder_lstm])

from tensorflow.keras.layers import Dense

decoder_dense = Dense(
    30000,
    activation='softmax',
    name="output_layer"
)(attention_output)

from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

model = Model([encoder_input, decoder_input], decoder_dense)

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

3.2 Pretrained Embeddings

The model uses GloVe (Global Vectors for Word Representation) with 100-dimensional embeddings to initialize the embedding layers.

```
embedding_matrix = np.zeros((30000, 100))
for word, i in tokenizer.word_index.items():
    if i < 30000:
        vector = embeddings_index.get(word)
        if vector is not None:
            embedding_matrix[i] = vector

print(f"Embedding matrix shape: {embedding_matrix.shape}")
```

4. Experiments and Training

4.1 Hyperparameters

- Learning Rate: 0.0001
- Batch Size: 32
- Epochs: 15 (with early stopping)
- Optimizer: Adam
- Loss Function: Sparse Categorical Crossentropy

4.2 Training Process

The model was trained on the training set and validated using the validation set. Early stopping was employed to prevent overfitting.

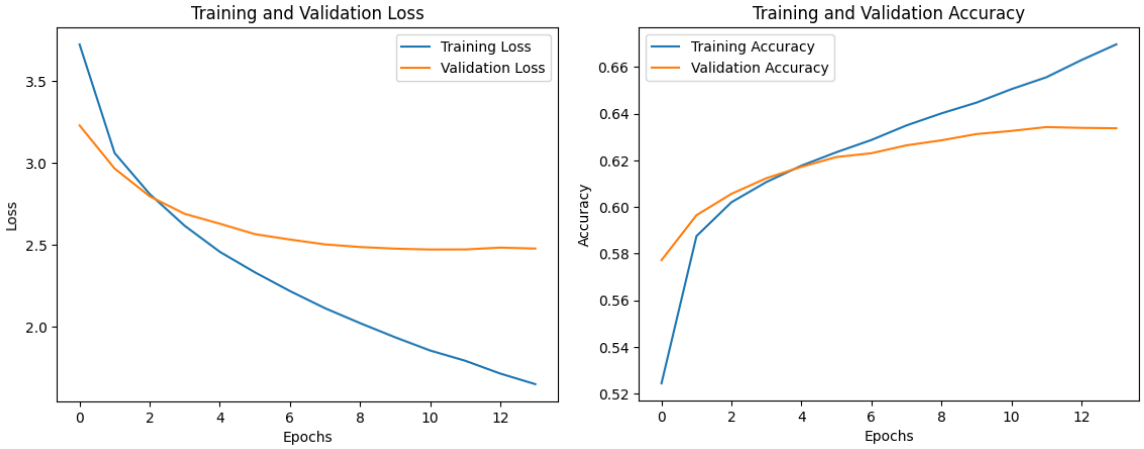
```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

history = model.fit(
    [X_train, Y_train[:, :-1]],
    Y_train[:, 1:],
    validation_data=([X_val, Y_val[:, :-1]], Y_val[:, 1:]),
    batch_size=32,
    epochs=15,
    callbacks=[early_stopping]
)
```

4.3 Training Results

Loss and Accuracy

Training and validation losses and accuracies are shown in the graphs below:



5. Evaluation and Analysis

5.1 ROUGE Metrics

The model was evaluated using the ROUGE metric, which measures the overlap between the generated summaries and reference summaries.

```
from rouge import Rouge
rouge = Rouge()
scores = rouge.get_scores(generated_summaries, reference_summaries, avg=True)
print(scores)
```

Results:

- **ROUGE-1:**
    - Recall: 0.3461
    - Precision: 0.4586
    - F1-Score: 0.3884

- **ROUGE-2:**
    - Recall: 0.0849
    - Precision: 0.0760
    - F1-Score: 0.0795
- **ROUGE-L:**
    - Recall: 0.3352
    - Precision: 0.4446
    - F1-Score: 0.3762

5.2 Observations

- The ROUGE-1 scores indicate decent performance for word-level matching.
- ROUGE-2 and ROUGE-L scores suggest room for improvement in capturing semantic structures and fluency.

6. Conclusion and Recommendations

6.1 Performance Analysis

The model successfully generates summaries with reasonable overlap compared to reference summaries. However, the low ROUGE-2 scores highlight the need for better semantic understanding and structural alignment.

6.2 Future Improvements

1. **Model Enhancements:**
    - Experiment with Transformer-based architectures (e.g., BERT, GPT).
    - Incorporate advanced attention mechanisms such as multi-head attention.
2. **Data Augmentation:**

- Increase dataset diversity to improve generalization.

3. **Hyperparameter Tuning:**
    - Optimize learning rate, batch size, and LSTM units.
4. **Post-processing:**
    - Implement grammar correction and coherence adjustments for generated summaries.