AML Lab 3

> Attention Mechanisms in Neural Networks: Implement an attention mechanism for a language task like machine translation or text summarization, showing how attention improves performance over baseline models.

✅ Step 1: Install and Import Required Libraries

```
# ✅ Step 1: Install and Import Required Libraries
!pip install -q tensorflow tensorflow-datasets

import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Model
```

✅ Step 2: Load a text summarization dataset

```
# ✅ Step 2: Load a text summarization dataset
# We'll use a small sample of the CNN/DailyMail dataset
data, info = tfds.load('cnn_dailymail', split='train[:1%]', with_info=True)
tokenizer = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    (sample['article'].numpy() + sample['highlights'].numpy() for sample in data),
    target_vocab_size=2**13
)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/cnn_dailymail/3.4.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tensc
Dl Completed...: 100%  5/5 [12:24<00:00,  7.60s/ url]
Dl Size...:  557/? [12:24<00:00, 12.19 MiB/s]
Extraction completed...: 100%  312085/312085 [12:24<00:00, 1177.12 file/s]
Dataset cnn_dailymail downloaded and prepared to /root/tensorflow_datasets/cnn_dailymail/3.4.0. Subsequent calls will reuse this da
```

✅ Step 3: Preprocess the data

```
# ✅ Step 3: Preprocess the data

MAX_LEN = 100

def encode(article, summary):
    article_tokens = tokenizer.encode(article.numpy())[:MAX_LEN]
    summary_tokens = tokenizer.encode(summary.numpy())[:MAX_LEN]
    return article_tokens, summary_tokens

def tf_encode(article, summary):
    result_article, result_summary = tf.py_function(encode, [article, summary], [tf.int64, tf.int64])
    result_article.set_shape([None])
    result_summary.set_shape([None])
    return result_article, result_summary

train_dataset = data.map(lambda x: tf_encode(x['article'], x['highlights']))
train_dataset = train_dataset.shuffle(1000).padded_batch(64, padded_shapes=([None], [None]))
```

✅ Step 4: Define the Encoder, Attention, and Decoder

```
# ✅ Step 4: Define the Encoder, Attention, and Decoder

class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units):
        super(Encoder, self).__init__()
        self.enc_units = enc_units
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.lstm = LSTM(enc_units, return_sequences=True, return_state=True)
```

```
    def call(self, x):
        x = self.embedding(x)
        output, state_h, state_c = self.lstm(x)
        return output, state_h, state_c

class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)

    def call(self, query, values):
        query_with_time_axis = tf.expand_dims(query, 1)
        score = self.V(tf.nn.tanh(self.W1(values) + self.W2(query_with_time_axis)))
        attention_weights = tf.nn.softmax(score, axis=1)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights

class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units):
        super(Decoder, self).__init__()
        self.dec_units = dec_units
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.lstm = LSTM(dec_units, return_sequences=True, return_state=True)
        self.fc = Dense(vocab_size)
        self.attention = BahdanauAttention(dec_units)

    def call(self, x, hidden, enc_output):
        context_vector, attention_weights = self.attention(hidden, enc_output)
        x = self.embedding(x)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
        output, _, _ = self.lstm(x)
        output = tf.reshape(output, (-1, output.shape[2]))
        x = self.fc(output)
        return x, attention_weights
```

✅ Step 5: Initialize and Train

```
# ✅ Step 5: Initialize and Train

vocab_size = tokenizer.vocab_size
embedding_dim = 64
units = 12

encoder = Encoder(vocab_size, embedding_dim, units)
decoder = Decoder(vocab_size, embedding_dim, units)

optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')

@tf.function
def train_step(inp, targ):
    loss = 0
    with tf.GradientTape() as tape:
        enc_output, enc_hidden_h, enc_hidden_c = encoder(inp)
        dec_hidden = enc_hidden_h
        dec_input = tf.expand_dims([tokenizer.vocab_size - 2] * targ.shape[0], 1)  # <start>

        for t in range(1, targ.shape[1]):
            predictions, _ = decoder(dec_input, dec_hidden, enc_output)
            loss += loss_object(targ[:, t], predictions)
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = tf.reduce_mean(loss)
    variables = encoder.trainable_variables + decoder.trainable_variables
    gradients = tape.gradient(loss, variables)
    optimizer.apply_gradients(zip(gradients, variables))
    return batch_loss

for epoch in range(2):  # Small run for demo
    total_loss = 0
    for (batch, (inp, targ)) in enumerate(train_dataset.take(100)):
        batch_loss = train_step(inp, targ)
        total_loss += batch_loss
        if batch % 10 == 0:
            print(f'Epoch {epoch+1} Batch {batch} Loss {batch_loss.numpy():.4f}')
```

```
    print(f'Epoch {epoch+1} Loss {total_loss / 100:.4f}')
```

```
Epoch 1 Batch 0 Loss 892.5539
Epoch 1 Batch 10 Loss 889.0922
Epoch 1 Batch 20 Loss 882.3152
Epoch 1 Batch 30 Loss 871.0068
Epoch 1 Batch 40 Loss 858.6083
Epoch 1 Loss 395.1544
Epoch 2 Batch 0 Loss 853.8019
Epoch 2 Batch 10 Loss 837.3614
Epoch 2 Batch 20 Loss 827.8582
Epoch 2 Batch 30 Loss 810.7155
Epoch 2 Batch 40 Loss 793.9666
Epoch 2 Loss 369.5564
```

✅ Step 6: Summarization Demo (Inference)

```
# ✅ Step 6: Summarization Demo (Inference)
def summarize(article):
    inputs = tokenizer.encode(article)
    inputs = tf.expand_dims(inputs, 0)
    enc_output, enc_hidden_h, enc_hidden_c = encoder(inputs)
    dec_hidden = enc_hidden_h
    dec_input = tf.expand_dims([tokenizer.vocab_size - 2], 0)  # <start>

    result = ''
    for t in range(20):  # max summary length
        predictions, attention_weights = decoder(dec_input, dec_hidden, enc_output)
        predicted_id = tf.argmax(predictions[0]).numpy()
        if predicted_id == tokenizer.vocab_size - 1:  # <end>
            break
        result += tokenizer.decode([predicted_id]) + ' '
        dec_input = tf.expand_dims([predicted_id], 0)

    return result.strip()
```

✅ Step 7: Try the model

sample_article = next(iter(data))['article'].numpy().decode() print("Original Article:\n", sample_article[:300], "\n") print("Generated Summary:\n", summarize(sample_article))

```
# ✅ Step 7: Try the model
sample_article = next(iter(data))['article'].numpy().decode()
print("Original Article:\n", sample_article[:300], "\n")
print("Generated Summary:\n", summarize(sample_article))
```

```
Original Article:
 By. Associated Press. PUBLISHED:. 14:11 EST, 25 October 2013. |. UPDATED:. 15:36 EST, 25 October 2013. The bishop of the Fargo Cath

Generated Summary:
 and and and and and and and and and and and and and and and and and and and
```