Task 6: Add Basic Attention Mechanism (Optional - Bonus)

* Modify your decoder to include attention on encoder outputs
* Visualize attention weights (e.g., with heatmaps)

----

1. Define the Attention Layer

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

    def call(self, query, values):
        query_with_time_axis = tf.expand_dims(query, 1)
        score = self.V(tf.nn.tanh(self.W1(query_with_time_axis) + self.W2(values)))
        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
```

----

2. Modify the Decoder to Use Attention

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units):
        super().__init__()
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
                   self.lstm  =  tf.keras.layers.LSTM(dec_units,  return_sequences=True,
return_state=True)
        self.fc = tf.keras.layers.Dense(vocab_size)
        self.attention = BahdanauAttention(self.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, state_h, state_c = self.lstm(x)
        output = tf.reshape(output, (-1, output.shape[2]))
        x = self.fc(output)
        return x, state_h, state_c, attention_weights
```

----

3. Visualize Attention Weights (Heatmap)

```
import matplotlib.pyplot as plt
import numpy as np

def plot_attention(attention, input_sentence, predicted_sentence):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(1, 1, 1)
```

```
attention = np.array(attention)
cax = ax.matshow(attention, cmap='viridis')
fig.colorbar(cax)
ax.set_xticklabels([''] + input_sentence, rotation=90)
ax.set_yticklabels([''] + predicted_sentence)
ax.xaxis.set_major_locator(plt.MultipleLocator(1))
ax.yaxis.set_major_locator(plt.MultipleLocator(1))
plt.show()
```