Sequence-to-Sequence Learning Part 2

Bahdanau attention implementation untuk mengatasi fixed context vector limitation Seq2Seq vanilla. Visualisasi attention weights untuk model interpretability. [1]

Bahdanau Attention Mechanism

Attention Scores Calculation:

```
score(s_t, h_i) = v_a^T * tanh(W_a * [s_t; h_i])
α_ti = softmax(score(s_t, h_i))
context_vector = Σ α_ti * h_i
```

TensorFlow Implementation:

```
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, decoder_state, encoder_outputs): # decoder_state shape:
    (batch, dec_units)
    # encoder_outputs shape: (batch, seq_len, enc_units)

    # Expand dimensions untuk broadcasting decoder_state =
    tf.expand_dims(decoder_state, 1)

    # Score calculation
    score = self.V(tf.nn.tanh(
        self.W1(decoder_state) + self.W2(encoder_outputs)
    ))

    attention_weights = tf.nn.softmax(score, axis=1) context_vector = attention_weights *
    encoder_outputs context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

Attention-integrated Decoder:

```
class AttentionDecoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units):
        super().__init__()
        self.embedding = Embedding(vocab_size, embedding_dim)
        self.gru = GRU(dec_units, return_sequences=True, return_state=True)
        self.attention = BahdanauAttention(dec_units)
        self.dense = Dense(vocab_size)

    def call(self, inputs, encoder_outputs, state=None):
        x = self.embedding(inputs)
        x, state = self.gru(x, initial_state=state)

        context, attention_weights = self.attention(state, encoder_outputs)
        x = tf.concat([tf.expand_dims(context, 1), x], axis=-1)

        logits = self.dense(x)
        return logits, state, attention_weights
```

Attention Visualization

```
def plot_attention(attention_weights, sentence, prediction):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention_weights, cmap='viridis')


    ax.set_xticklabels([''] + list(sentence.split()) + ['<EOS>'])
    ax.set_yticklabels([''] + list(prediction.split()))

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
    plt.show()
```

Training dengan Attention

Loss calculation tetap sama, tetapi decoder menerima encoder outputs:

```
@tf.function
def train_attention_step(en_batch, de_batch, encoder, decoder, optimizer):
    enc_output, enc_state = encoder(en_batch)

    dec_input = tf.expand_dims([de_tokenizer.word_index['<start>']] * en_batch.shape[^0],
    dec_state = enc_state

    loss = 0
    with tf.GradientTape() as tape:
        for t in range(1, de_batch.shape[^1]):
            predictions, dec_state, _ = decoder(
                dec_input, enc_output, dec_state
            )
            loss += loss_function(de_batch[:, t], predictions[:, -1:, :])
            dec_input = de_batch[:, t:t+1]

    # Gradient application
    variables = encoder.trainable_variables + decoder.trainable_variables
    gradients = tape.gradient(loss, variables)
    optimizer.apply_gradients(zip(gradients, variables))
    return loss
```

Performance Improvement

Attention mechanism meningkatkan BLEU score 5 10 points dengan context vector yang dinamis per timestep.

Kesimpulan

Bahdanau attention solves fixed context limitation dan memberikan interpretable visualizations untuk machine translation.[1]