Sequence-to-Sequence Learning

Vanilla Seq2Seq model untuk English-German machine translation menggunakan bidirectional GRU encoder + GRU decoder dengan teacher forcing.[1]

Machine Translation Dataset Preparation

Multi30k Dataset 29k sentence pairs):

```
def load_translation_data():
    # English-German pairs
    data = tfds.load('multi30k', split='train', as_supervised=True)

    def format_example(en, de):
        en = tf.strings.lower(en.numpy().decode())
        de = tf.strings.lower(de.numpy().decode())
        return en, de

    return data.map(format_example)

# TextVectorization layers
```

```
en_vectorize = TextVectorization(max_tokens=8000, output_sequence_length=20)
de_vectorize = TextVectorization(max_tokens=10000, output_sequence_length=20)
```

Vanilla Seq2Seq Architecture

Encoder (bidirectional GRU

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

    def call(self, tokens, state=None):
        x = self.embedding(tokens)
        output, state = self.gru(x, initial_state=state)
        return output, state
```

Decoder GRU + teacher forcing):

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

    def call(self, tokens, state=None):
        x = self.embedding(tokens)
        output, state = self.gru(x, initial_state=state)
        logits = self.dense(output)
        return logits, state
```

Training Loop dengan teacher forcing:

```
@tf.function
def train_step(en_batch, de_batch, encoder, decoder, optimizer):
    loss = 0
    with tf.GradientTape() as tape:
        # Encoder
        enc_output, enc_state = encoder(en_batch)

        # Initialize decoder dengan <start> token
        dec_input = tf.expand_dims([de_tokenizer.word_index['<start>']] * en_batch.shape[
        dec_state = enc_state

        # Teacher forcing
        for t in range(1, de_batch.shape[^1]):
            dec_output, dec_state = decoder(dec_input, dec_state)
            loss += loss_function(de_batch[:, t], dec_output[:, -1:, :])
            dec_input = de_batch[:, t:t+1]

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

Inference Model No Teacher Forcing)

Separate encoder-decoder untuk generation:

```
def translate(encoder, decoder, sentence):
    enc_tokens = en_vectorize([sentence])
    enc_output, enc_state = encoder(enc_tokens)

    dec_input = tf.expand_dims([de_tokenizer.word_index['<start>']], 0)
    result = []

    for i in range(20):
        dec_output, dec_state = decoder(dec_input, enc_state)
        predicted_id = tf.argmax(dec_output[0, -1:, :], axis=-1)
        result.append(de_tokenizer.index_word[predicted_id])

        if predicted_id == de_tokenizer.word_index['<end>']:
            break

        dec_input = tf.expand_dims([predicted_id], 0)
        enc_state = dec_state

    return ' '.join(result)
```

Kesimpulan

Vanilla Seq2Seq dengan fixed context vector limitation untuk long sequences.