Chapter 12

Sequence-to-sequence Learning: Part 2

The earlier translation model used a sequence-to-sequence design with an encoder-decoder architecture and achieved reasonable accuracy. However, it relied heavily on a single context vector passed from the encoder to the decoder, which created a bottleneck. To overcome this limitation and improve translation quality, an attention mechanism is introduced. Attention enables the decoder to consult all encoder outputs at every decoding step, instead of compressing all input information into one vector.

Improving the Model with Attention

The Bahdanau attention mechanism is applied. Unlike earlier models, which relied on the encoder's final hidden state, attention dynamically computes a weighted sum of all encoder outputs at each decoder step. This provides the decoder with the most relevant parts of the input sequence while generating each output word.

Mathematically, attention works as follows:

- For each decoder state s_i and encoder state h_j an energy score e_{ij} is computed:
$$e_{ij} = v^T \tanh\left(W_{s_{i-1}} + Uh_j\right)$$
- The energies are normalized using softmax to produce attention weights a_{ij}
- The context vector c_i for each decoder step is the weighted sum of encoder outputs:
$$c_i = \sum_j a_{ij} h_j$$

Implementing Attention

Since there is no built-in TensorFlow layer for Bahdanau attention, a custom wrapper called DecoderRNNAttentionWrapper is implemented using the Keras subclassing API. It defines:

- Weights W, U, v — trainable parameters shaping how encoder and decoder states interact.
- Step function — calculates energy scores, normalizes them into probabilities, and computes weighted sums of encoder outputs.
- K.rnn() function — iterates over decoder inputs, applying the step function at each timestep to generate attention outputs and attention energy matrices.

The output includes both the new decoder hidden states and the attention weights.

Reproduced code for DecoderRNNAttentionWrapper:

```
import tensorflow as tf

import tensorflow.keras.backend as K
```

```
from tensorflow.keras.layers import GRUCell


class DecoderRNNAttentionWrapper(tf.keras.layers.Layer):
    def __init__(self, cell_fn, units, **kwargs):
        self._cell_fn = cell_fn
        self.units = units
        super(DecoderRNNAttentionWrapper,
self).__init__(**kwargs)

    def build(self, input_shape):
        self.W_a = self.add_weight(
            name='W_a',
            shape=(input_shape[0][2], input_shape[0][2]),
            initializer='uniform',
            trainable=True
        )
        self.U_a = self.add_weight(
            name='U_a',
            shape=(self._cell_fn.units, self._cell_fn.units),
            initializer='uniform',
            trainable=True
        )
        self.V_a = self.add_weight(
            name='V_a',
            shape=(input_shape[0][2], 1),
            initializer='uniform',
            trainable=True
        )
```

```
        super(DecoderRNNAttentionWrapper,
self).build(input_shape)

    def call(self, inputs, initial_state, training=False):
        encoder_outputs, decoder_inputs = inputs

        def _step(inputs, states):
            encoder_full_seq = states[-1]

            W_a_dot_h = K.dot(encoder_outputs, self.W_a)
            U_a_dot_s = K.expand_dims(K.dot(states[0], self.U_a),
1)
            Wh_plus_Us = K.tanh(W_a_dot_h + U_a_dot_s)

            e_i = K.squeeze(K.dot(Wh_plus_Us, self.V_a), axis=-1)
            a_i = K.softmax(e_i)

            c_i = K.sum(encoder_outputs * K.expand_dims(a_i, -1),
axis=1)
            s,   states   =   self._cell_fn(K.concatenate([inputs,
c_i], axis=-1), states)

            return (s, a_i), states

        _, attn_outputs, _ = K.rnn(
            step_function=_step,
            inputs=decoder_inputs,
            initial_states=[initial_state],
            constants=[encoder_outputs]
```

```
        )


        attn_out, attn_energy = attn_outputs
        return attn_out, attn_energy
```

Constructing the Final Attention Model

The encoder remains unchanged: a bidirectional GRU processes English sentences into hidden states. The decoder, however, now incorporates the attention wrapper around a GRUCell.

Key changes:

- The decoder embedding layer converts German tokens into vectors.
- The encoder's final forward and backward states are concatenated to form the decoder's initial state.
- At each decoder step, the GRUCell input is concatenated with the context vector from the attention mechanism.
- The output passes through dense layers before the final softmax, producing probabilities over the German vocabulary.

Both encoder and decoder are tied together into a unified seq2seq model with attention. The system still uses TextVectorization layers for raw string handling.

Training and Results

The model is compiled with Adam optimizer and sparse categorical cross-entropy loss, and trained for five epochs with a batch size of 128. Results show clear improvement:

- After training, validation BLEU increased from near zero to ~0.20.
- Accuracy also improved significantly, surpassing 83% on validation and test sets.

Compared to the previous model, this nearly doubles the BLEU score, confirming the value of attention.

The trained model and vocabularies are saved for reuse. While state-of-the-art English–German models reach BLEU scores above 0.35 on large benchmark datasets, achieving ~0.20 BLEU with this smaller dataset and simpler model is still promising.

Visualizing Attention

Beyond performance gains, attention introduces interpretability. The normalized energy weights a_{ij} reveal how much the model focuses on each input word when generating each output. A visualization function is built to retrieve these weights and display them as heatmaps. English

words appear on the vertical axis and German words on the horizontal axis. Darker or lighter colors indicate stronger or weaker focus.

Figures 1 and 2 illustrate these heatmaps. The patterns generally follow a diagonal, reflecting the word order similarities between English and German. For example:

- When generating “und” (German for “and”), the model pays attention to the English word “and.”
- When producing “maria,” it focuses on “mary.”
- When translating “hast keine nicht,” the model attends to “have no idea.”

This alignment between input and output tokens demonstrates that the model has learned meaningful word correspondences.

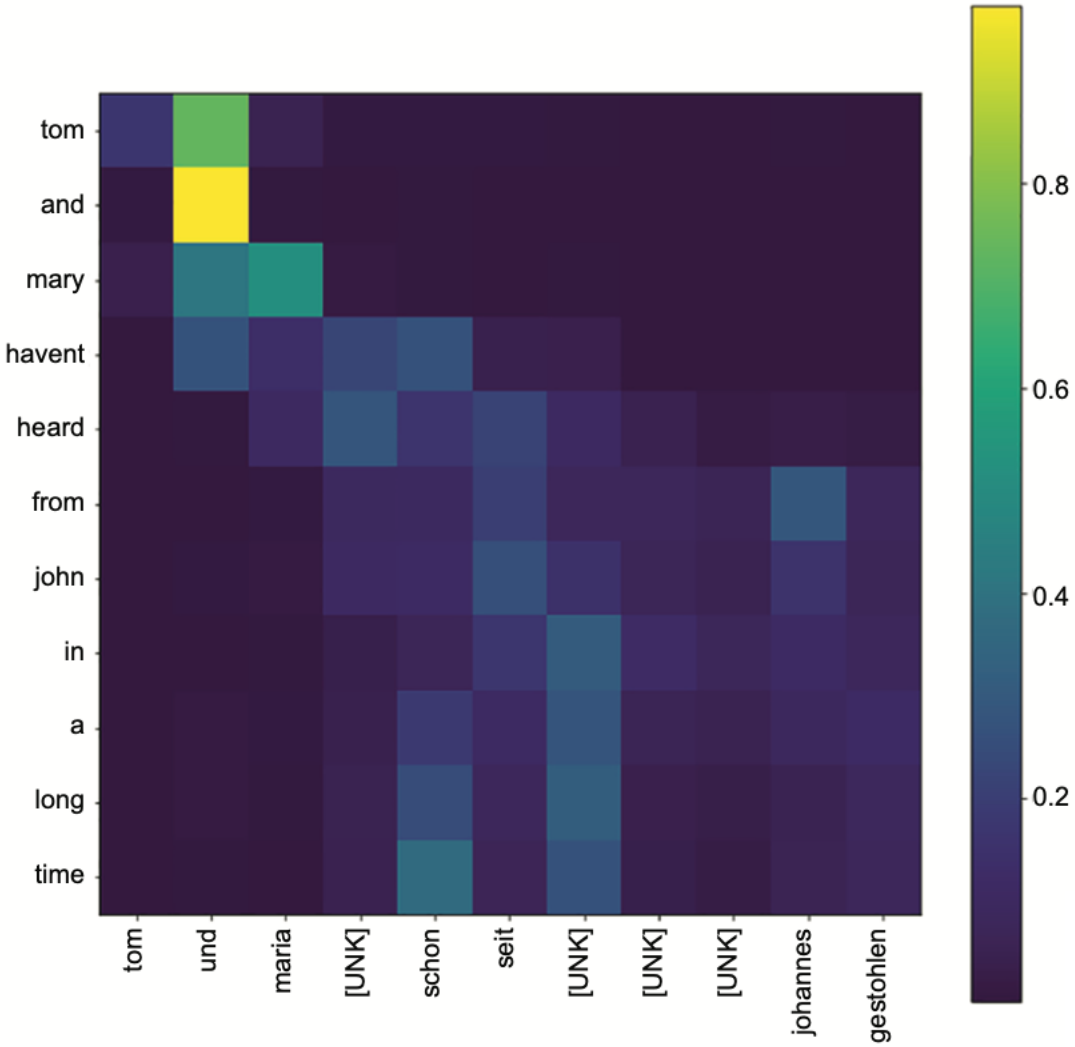

Figure 1 Attention patterns visualized for an input English text

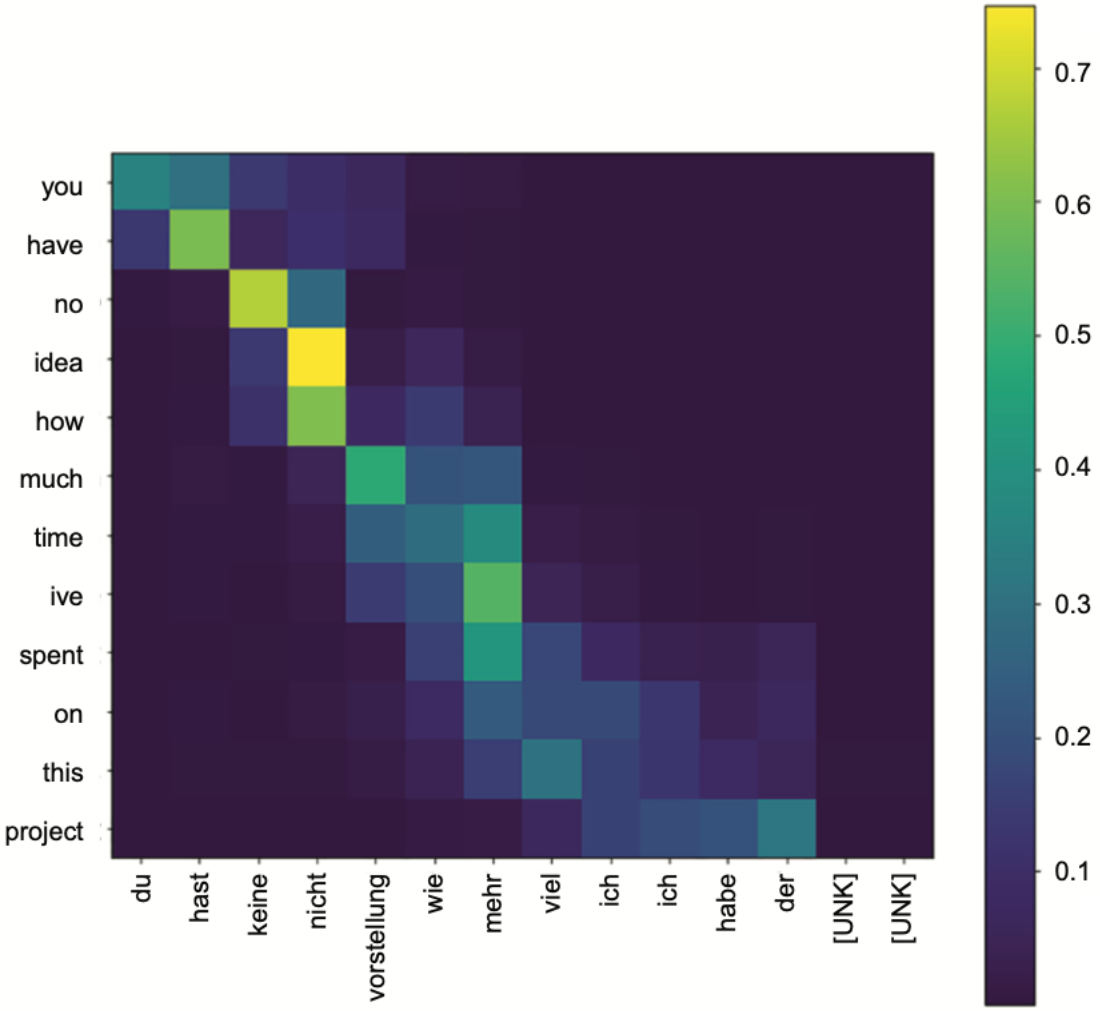

Figure 2 Attention patterns visualized for an input English text