**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 and encoder state an energy score is computed:
* The energies are normalized using softmax to produce attention weights
* The context vector for each decoder step is the weighted sum of encoder outputs:

​

**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 — 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 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.

A graph of a number of numbers

AI-generated content may be incorrect.

Figure 1 Attention patterns visualized for an input English text

A graph of a graph

AI-generated content may be incorrect.

Figure 2 Attention patterns visualized for an input English text