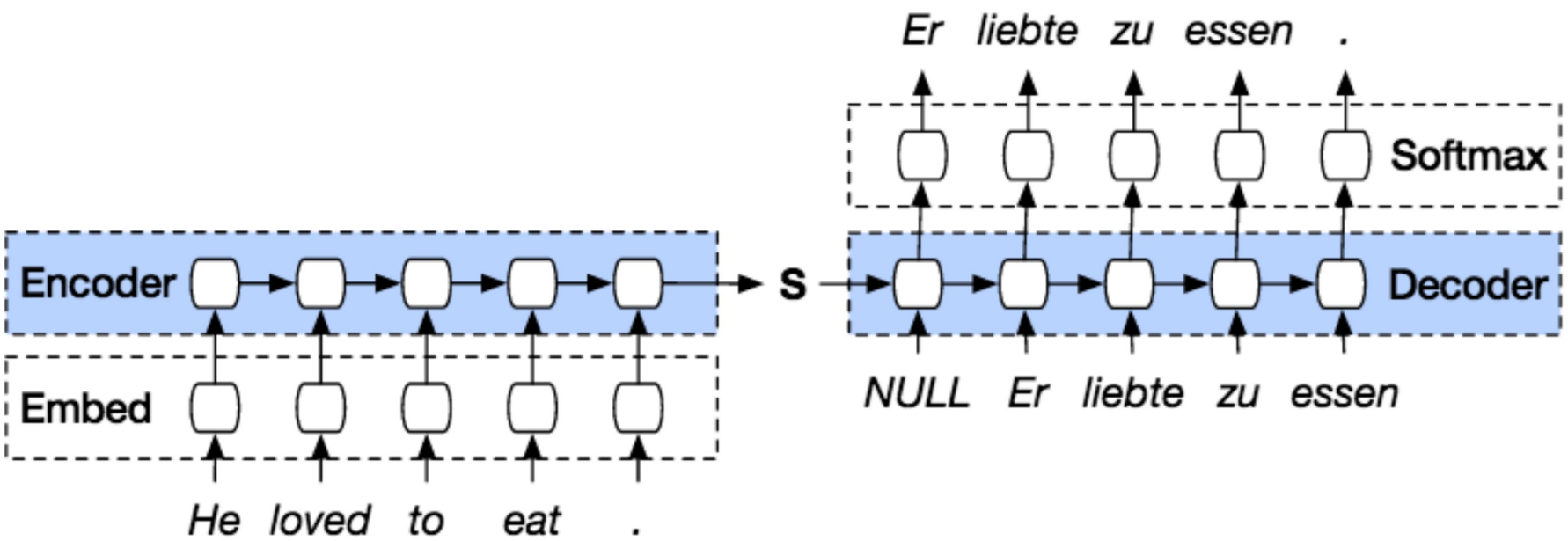
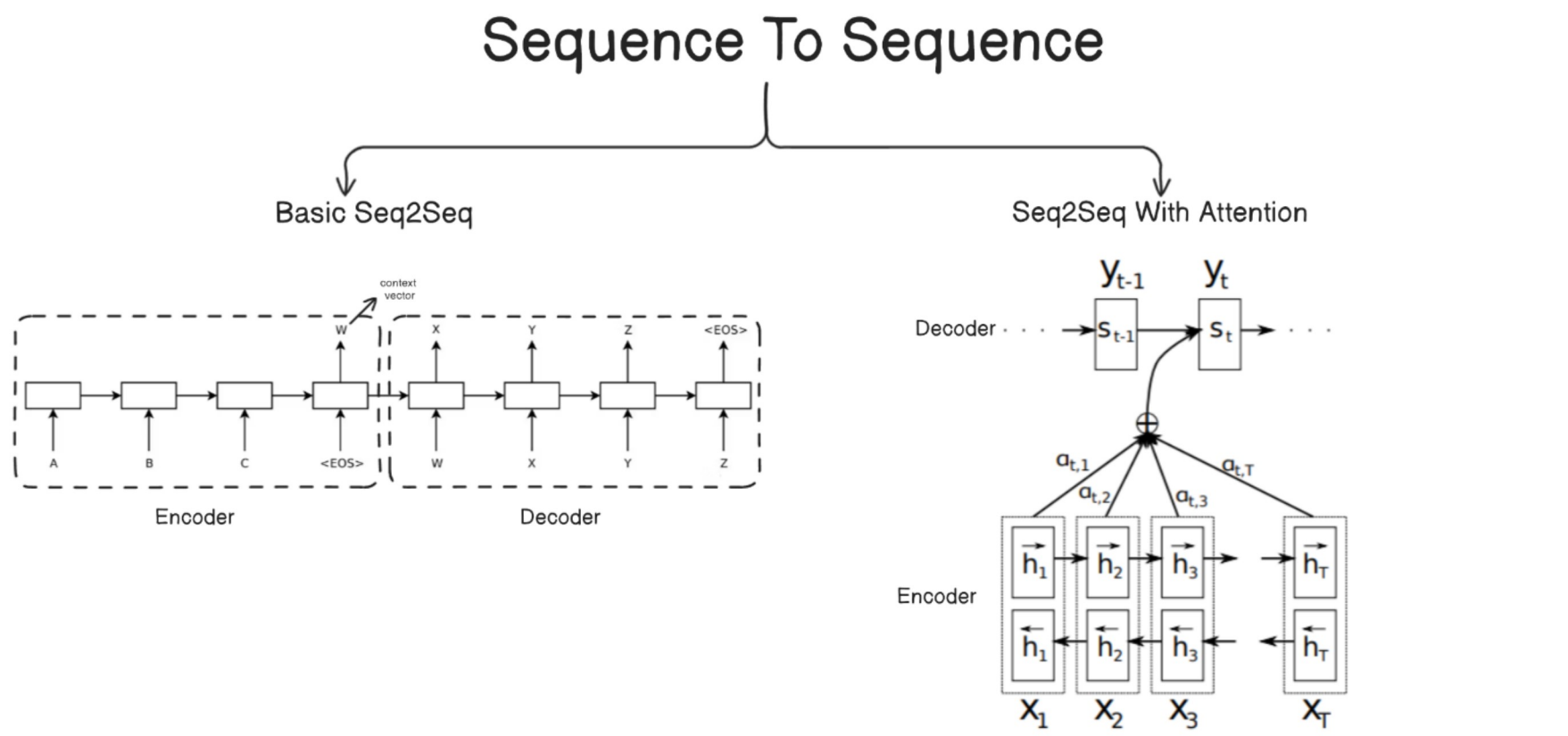
Seq2Seq task with attention

Sequence-to-sequence (Seq2Seq) learning involves training models that convert sequences from one domain to another. A typical example is translating a sentence from one language, like German, into another language, such as English. In this case, the main objective is to translate German sentences into their English equivalents.



An essential enhancement to Seq2Seq models is the **attention mechanism**, which enables the model to focus on specific parts of the input sequence while generating each word of the output. This mechanism simulates the human ability to selectively concentrate on relevant pieces of information. For example, when translating a sentence, attention helps the model pay closer attention to specific words in the input, depending on the word being translated at that moment.



Due to its effectiveness, attention has become a fundamental part of advanced models like **Transformers**, which rely solely on this mechanism to maintain context across sequences.

Seq2Seq models are widely used in natural language processing (NLP) tasks, such as text summarization, speech recognition, and even in modeling biological sequences like DNA. In all of these cases, the input and output are sequences, and the model's job is to generate a new sequence from the given input. Seq2Seq models excel in tasks where structured information needs to be converted into another structured form, making them highly versatile across various domains.

In [13]:
```
eng, deu = preprocess(eng), preprocess(deu)
```

DATA

https://www.kaggle.com/datasets/alincijov/bilingual-sentence-pairs/data Consist of translations of common

sentences used in daily life, here we are using English and Deutch (German) for the Seq2Seq Machine translation model

In [14]:
```
data = pd.DataFrame({'english' : eng, 'deutsch': deu})
```

In [8]:
```
data.head(10)
```

Out[8]:

	english	deutsch
0	go	geh
1	hi	hallo
2	hi	grüß gott
3	run	lauf
4	run	lauf
5	wow	potzdonner
6	wow	donnerwetter
7	fire	feuer
8	help	hilfe
9	help	zu hülf

Out of approximately 220k datapoints, I am considering only the first 100k

In [15]:
```
data = data[:100_000]
```

Data preprocessing

The data processing pipeline looks like :

- **Tokenization**: Breaking down text into smaller units (tokens), such as words or characters.
- **Numerical Encoding**: Assigning numerical representations to each token.
- **Sequence Padding**: Ensuring all sequences have the same length by adding padding tokens.

In [16]:
```
e_tokenizer,d_tokenizer = Tokenizer(), Tokenizer()
e_tokenizer.fit_on_texts(data['english'])
d_tokenizer.fit_on_texts(data['deutsch'])
```

In [18]:
```
e_vocab, d_vocab  = len(e_tokenizer.word_index) + 1, len(d_tokenizer.word_index ) + 1
```

Maximum length of sentences

To find the maximum number of words present in the dataset for both language. This isn't necessary and can be chosen subjectively as per required. However for this case I was experimenting a lot thus chose to let the dataset decide the value for this once.

In [19]:
```
Len = lambda arr: max(len(i.split(" ")) for i in arr)
e_max_len, d_max_len = Len(data['english']), Len(data['deutsch'])
```

Converting text to Numeric sequences

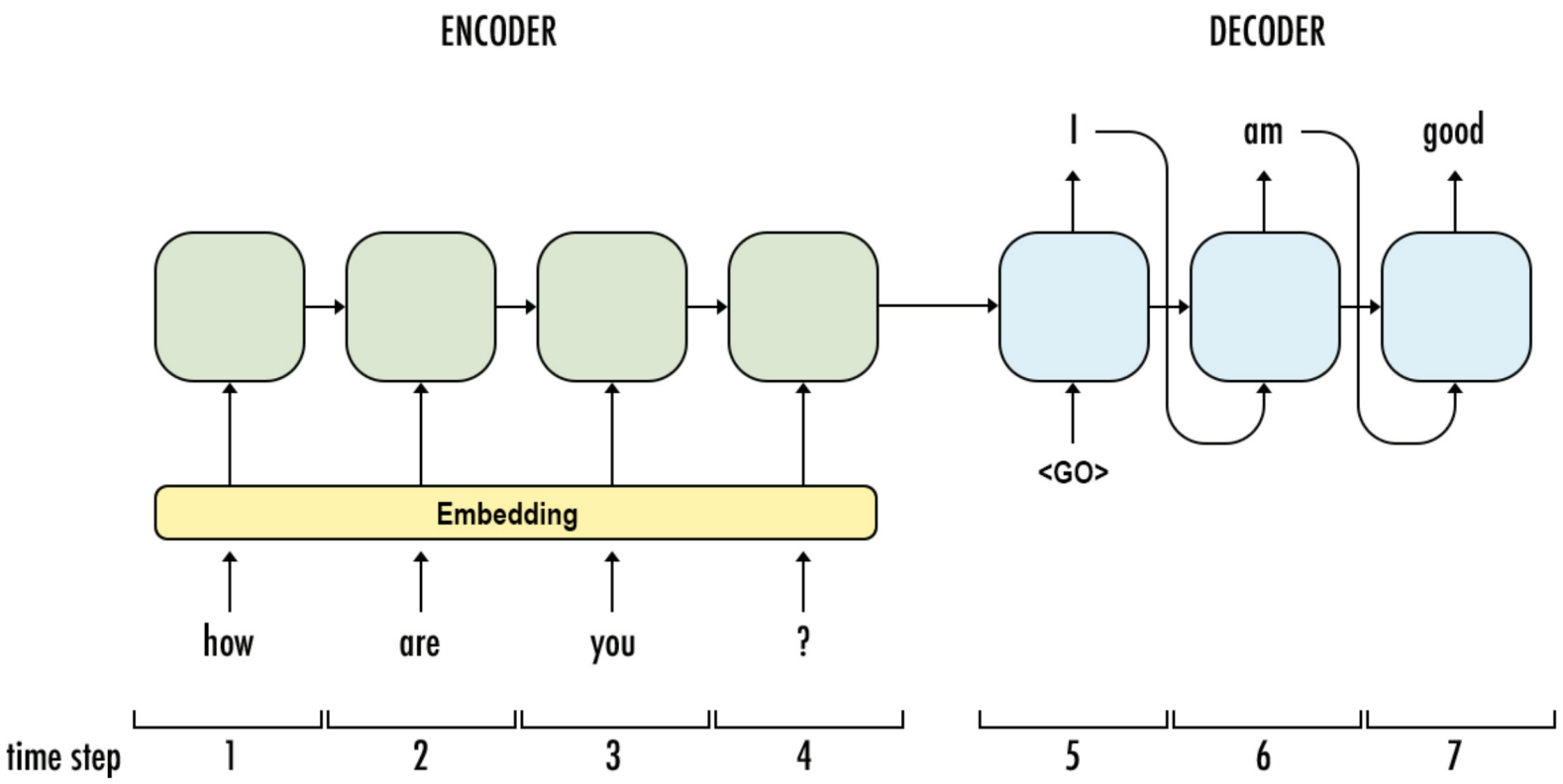
In [20]:
```
def encode(text, tokenizer, max_len):
    text =  tokenizer.texts_to_sequences(text)
    text = pad_sequences(text, max_len, padding = 'post')
    return text

X, y = encode(data['english'],e_tokenizer,e_max_len), encode(data['deutsch'],d_tokenizer, d_max_len)
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Base Seq2Seq Model

Seq2Seq models are constructed using key components like Long Short-Term Memory (LSTM) units, which are specialized types of recurrent neural networks (RNNs) that effectively capture temporal dependencies in data. LSTMs help in processing sequences by remembering important information over longer periods and forgetting irrelevant details, making them useful for handling tasks involving sequences of varying lengths.



I found the following notebook https://www.kaggle.com/code/harshjain123/machine-translation-seq2seq-lstms by Harsh Jain very insightful to understand this process, its will be a great place to understand the data processing pipeline.

In [6]:
```
@keras.saving.register_keras_serializable(package="Custom", name="S2S")
class S2S(Model):
    def __init__(self, in_vocab, out_vocab, in_timesteps, out_timesteps, units, **kwargs):
        super(S2S, self).__init__(**kwargs)

        self.in_vocab = in_vocab
        self.out_vocab = out_vocab
        self.in_timesteps = in_timesteps
        self.out_timesteps = out_timesteps
        self.units = units

        # Define the layers
        self.embed = Embedding(input_dim=in_vocab, output_dim=units, mask_zero=True)
        self.encoder_lstm = LSTM(units)
        self.r_vector = RepeatVector(out_timesteps)
        self.decoder_lstm = LSTM(units, return_sequences=True)
        self.dense = Dense(out_vocab, activation='softmax')

    def call(self, inputs):
        # Define the forward pass
        x = self.embed(inputs)                          # (batch size, in_timesteps, units)
        x = self.encoder_lstm(x)                        # (batch size, units)
        x = self.r_vector(x)                            # (batch size, out_timesteps, units)
        x = self.decoder_lstm(x)                        # (batch size, out_timesteps, units)
        output = self.dense(x)                          # (batch size, out_timesteps, out_vocab)
        return output

    def get_config(self):
        config = super(S2S, self).get_config()
        config.update({
              'in_vocab': self.in_vocab,
              'out_vocab': self.out_vocab,
              'in_timesteps': self.in_timesteps,
```

```
            'out_timesteps': self.out_timesteps,
            'units': self.units
          })
        return config

    @classmethod
    def from_config(cls, config):
        return cls(
            in_vocab=config['in_vocab'],
            out_vocab=config['out_vocab'],
            in_timesteps=config['in_timesteps'],
            out_timesteps=config['out_timesteps'],
            units=config['units']
        )
```

In [22]:
```
seq2seq = S2S(e_vocab, d_vocab, e_max_len,d_max_len, 512)
seq2seq.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
out_shape = (y_train.shape[0], y_train.shape[1], 1)
```

In [31]:
```
seq2seq.summary()
```

Model: "s2s_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	4,724,736
lstm_2 (LSTM)	?	2,099,200
repeat_vector_1 (RepeatVector)	?	0
lstm_3 (LSTM)	?	2,099,200
dense_1 (Dense)	?	8,540,424

Total params: 52,390,682 (199.85 MB)

Trainable params: 17,463,560 (66.62 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 34,927,122 (133.24 MB)

In [18]:
```
seq2seq.fit(x=X_train, y = y_train.reshape(out_shape), epochs = 5, batch_size = 10, validation_batch_s
```

```
Epoch 1/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 187s 23ms/step - loss: 0.2191
Epoch 2/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 185s 23ms/step - loss: 0.2122
Epoch 3/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 185s 23ms/step - loss: 0.2068
Epoch 4/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 186s 23ms/step - loss: 0.2001
Epoch 5/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 185s 23ms/step - loss: 0.1930
```

Out[18]: <keras.src.callbacks.history.History at 0x7afc7b2ca0b0>

(for 25 epochs)

Final Loss : 0.1930

Training Time : 45 minutes 55 seconds

In [19]:
```
seq2seq.save('s2s.keras')
```

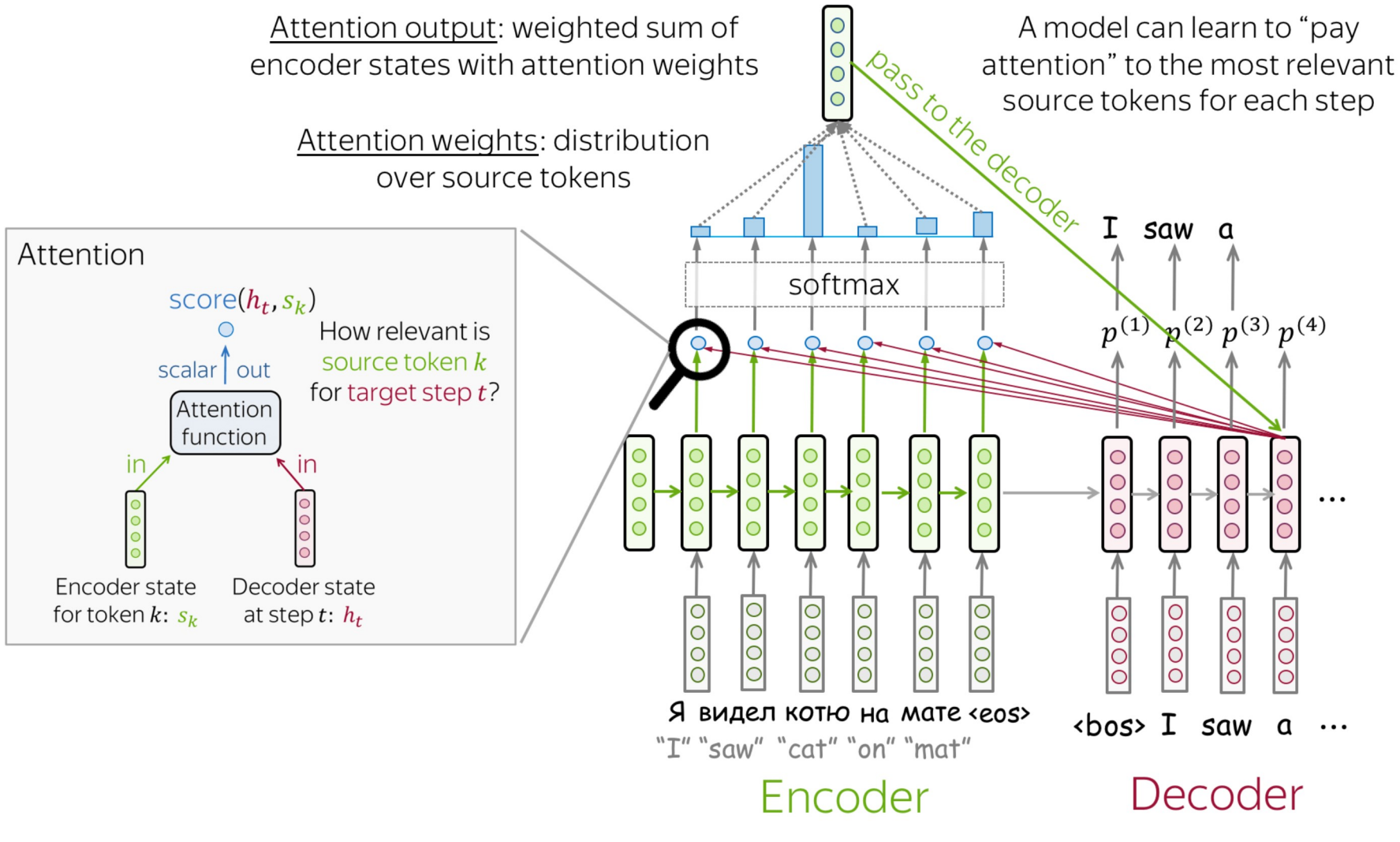
Seq2Seq with Attention:

Traditional Seq2Seq models often struggle with long input sequences. The encoder encodes the entire input sequence into a fixed-size vector, which can lead to information loss, especially for longer sequences.

The attention mechanism addresses this limitation by allowing the decoder to focus on relevant parts of the input sequence at each decoding step. This enables the model to better capture long-range dependencies and produce more accurate outputs.

Implemented using an attention layer that computes a context vector by attending to all encoder outputs. The context vector is combined with the decoder's hidden state at each timestep.

- Encoder LSTM now returns the full sequence of hidden states (encoder_outputs) and the final hidden state (state_h, state_c).
- Attention is applied at each timestep of the decoder to produce a context vector that is combined with the decoder input at that timestep.
- The attention mechanism allows the decoder to focus on different parts of the input sequence when producing each output.



Img sources : https://lena-voita.github.io/resources/lectures/seq2seq/attention/general_scheme-min.png

Bahdanau Attention Layer:

The model calculates alignment scores between the encoder's hidden states and the current decoder's hidden state at each decoding step. The relevance or significance of each encoder hidden state in relation to the current decoding phase is represented by these scores.

- Takes the encoder outputs (sequence of hidden states) and the decoder hidden state to compute the attention weights.
- Computes the context vector, which is a weighted sum of the encoder outputs based on attention weights.

In [27]:
```
class Attention(Layer):

    def __init__(self, units):

        super(Attention, self).__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)
```

```
    def call(self, query, value):

        # query shape == (batch_size, hidden_size) -> decoder hidden state at the current timestep
        # values shape == (batch_size, max_len, hidden_size) -> encoder outputs (all timesteps)

        q_time = tf.expand_dims(query, axis = 1)
        score = self.V(tf.nn.tanh(self.W1(q_time)+self.W2(value)))
        weights = tf.nn.softmax(score, axis=1)
        context = weights * value
        context = tf.reduce_sum(context, axis=1)          # (batch_size, hidden_size)
        return context, weights
```

Call action:

- After encoding, the attention mechanism is applied for each timestep in the decoder.
- The **context vector** and the **decoder input** at each timestep are concatenated and passed to the decoder LSTM.
- The result is a sequence of outputs, one for each decoder timestep.

In [28]:
```
@keras.saving.register_keras_serializable(package="Custom", name="S2SA")
class S2SA(Model):

    def __init__(self, in_vocab, out_vocab, in_len, out_len, units, **kwargs):
        super(S2SA, self).__init__(**kwargs)

        self.in_vocab = in_vocab
        self.out_vocab = out_vocab
        self.in_len = in_len
        self.out_len = out_len
        self.units = units

        self.embed = Embedding(input_dim=in_vocab, output_dim=units, mask_zero = True)
        self.encoder_lstm = LSTM(units, return_sequences=True, return_state=True)
        self.attention = Attention(units)
        self.r_vectors = RepeatVector(out_len)
        self.decoder_lstm = LSTM(units, return_sequences=True, return_state=True)
        self.dense = Dense(out_vocab, activation = 'softmax')


    def call(self, inputs):
        x = self.embed(inputs)
        e_out, e_h_state, e_c_state = self.encoder_lstm(x)       # (batch_size, in_timesteps, units),
        d_in = self.r_vectors(e_h_state)                        # (batch_size, out_timesteps, units)

        d_h_state, d_c_state = e_h_state, e_c_state
        all_dec_out = []


        for t in range(d_in.shape[1]):

            d_at_t = d_in[:, t:t+1, :]                                          # (batch_size, 1, ur

            context_vector,_= self.attention(e_h_state, e_out)                    # (batch_size, uni
                                                                                # TO MATCH THE DIMS
            context_vector = tf.expand_dims(context_vector, axis=1)             # (batch_size, 1, ur

            context_w_inputs = tf.concat([context_vector, d_at_t], axis = -1)     # (batch_size, 1, 2

            d_out,d_h_state, d_c_state = self.decoder_lstm(context_w_inputs, initial_state = [d_h_stat
            d_out = self.dense(d_out)

            all_dec_out.append(d_out)


        d_out = tf.concat(all_dec_out, axis = 1)            # To aggregate outputs across timesteps

        return d_out
```

In [29]:
```
attention_model = S2SA(e_vocab, d_vocab, e_max_len, d_max_len, 512)
attention_model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy')
```

In [32]:
```
attention_model.summary()
```

Model: "s2sa_1"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	?	4,724,736
lstm_6 (LSTM)	?	2,099,200
attention_1 (Attention)	?	525,825
repeat_vector_3 (RepeatVector)	?	0
lstm_7 (LSTM)	?	3,147,776
dense_9 (Dense)	?	8,540,424

Total params: 57,113,885 (217.87 MB)

Trainable params: 19,037,961 (72.62 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 38,075,924 (145.25 MB)

In [22]:
```
attention_model.fit(X_train, y_train.reshape(out_shape), epochs = 5, batch_size=10, validation_batch_s
```

```
Epoch 1/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 652s 80ms/step - loss: 0.1799
Epoch 2/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 641s 80ms/step - loss: 0.1788
Epoch 3/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 643s 80ms/step - loss: 0.1737
Epoch 4/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 643s 80ms/step - loss: 0.1721
Epoch 5/5
8000/8000 ━━━━━━━━━━━━━━━━━━━━ 640s 80ms/step - loss: 0.1697
```

Out[22]:
```
<keras.src.callbacks.history.History at 0x7afe3bc50970>
```

(for 25 epochs)

Final Loss : 0.1697

Training Time : 5 hours, 45 minutes

In [23]:
```
attention_model.save("s2sa.keras")
```

Translation comparisons

Complex translations

In [27]:
```
df_test = test(seq2seq,attention_model, 10,4000)
df_test.head(20)
```

```
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
400/400 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step
```

Out[27]:

	Eng	Deu	Base_Model	Attention_Model
0	she decided to marry tom	sie hat sich entschieden tom zu heiraten	sie entschied sich tom zu heiraten heiraten	sie entschied tom tom zu heiraten
1	do not read while walking	lies nicht im gehen	geht es nicht zeit zu	lesen nicht nicht
2	which ones mine	welche ist meine	welches ist meins	welches ist meins
3	this knife is very sharp	dieses messer ist sehr scharf	dieses zimmer ist sehr scharf	dieses messer ist sehr echt
4	that was just plain stupid	das war einfach nur dumm	das war schlicht und und dumm	das war schlicht und ergreifend dumm
5	theres still a lot left	es gibt noch immer eine menge zu tun	es ist noch viel übrig	es ist noch noch übrig
6	please add up the numbers	bitte addiert die zahlen	bitte addiere die die	bitte addiere die zahlen zahlen
7	he is playing in his room	er spielt in seinem zimmer	er wohnt nach seinem zimmer	er spielt am sein zimmer
8	tom isnt your brother	tom ist nicht dein bruder	tom ist nicht euer bruder	tom ist nicht euer bruder
9	im not going to stop	ich werde nicht aufhören	ich werde nicht aufhören	ich werde nicht aufhören
10	i cant stand kids	ich hasse kinder	ich kann keine nicht	ich kann keine kinder kinder haben
11	i want to study math	ich möchte mathematik studieren	ich will mathematik studieren lernen	ich will golf studieren
12	tom is very arrogant	tom ist ziemlich arrogant	tom ist sehr arrogant	tom ist sehr arrogant
13	i rubbed my feet	ich rieb meine füße	ich lehnte mir meinen die	ich meine meine die
14	can we fish there	kann man hier angeln	können wir da da	können wir fisch lesen
15	do you come here every day	kommst du hier jeden tag her	machst ihr in jeden tag	kommt sie jeden tag uhr
16	go away	verzieh dich	mach die	zisch ab
17	they cheat	sie mogeln	sie betrügen	sie betrügen
18	can you order it for me	können sie ihn mir bestellen	kannst ihr sie mir bestellen	könnt ihr sie mir bestellen
19	tom put on some clothes	tom zog sich an	tom hat sich angezogen	tom hat sich angezogen

Slightly Complex translations

In [35]:
```
df_test[290:310]
```

Out[35]:

	Eng	Deu	Base_Model	Attention_Model
290	wake up	wachen sie auf	wachen sie	wachen sie
291	wake up	wach auf	wachen sie	wachen sie
292	wake up	wachen sie auf	wachen sie	wachen sie
293	wash up	wasch dir die hände	wasch dir die	wasch dir die gesicht
294	wash up	wasch dir das gesicht	wasch dir die	wasch dir die gesicht
295	we lost	wir haben verloren	wir haben verloren	wir haben
296	welcome	willkommen	willkommen	willkommen
297	who ate	wer hat gegessen	wer aß	wer aß
298	who ate	wer aß	wer aß	wer aß
299	who ran	wer rannte	wer rannte	wer rannte
300	who ran	wer ist gerannt	wer rannte	wer rannte
301	who won	wer hat gewonnen	wer hat gewonnen	wer hat gewonnen
302	you run	du läufst	sie laufen	du läufst
303	you run	sie laufen	sie laufen	du läufst
304	you won	du hast gewonnen	du hast gewonnen	du hast gewonnen
305	all rise	alle aufstehen	alle aufstehen	alle aufstehen
306	am i fat	bin ich dick	bin ich dick	bin ich dick
307	ask them	frag sie	frag sie	frag sie
308	back off	komm nicht näher	komm nicht näher	komm nicht näher
309	be a man	sei ein mann	sei ein mann mann	sei ein mann

Simple translations

In [29]:
```
df_test = test1(seq2seq,attention_model, 10,4000)
```

```
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
400/400 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step
```

Out[29]:

	Eng	Deu	Base_Model	Attention_Model
0	go	geh	geh	geh
1	hi	hallo	hallo	grüß gott
2	hi	grüß gott	hallo	grüß gott
3	run	lauf	lauf	lauf
4	run	lauf	lauf	lauf
5	wow	potzdonner	potzdonner	donnerwetter
6	wow	donnerwetter	potzdonner	donnerwetter
7	fire	feuer	feuer	feuer
8	help	hilfe	zu hülf	zu hülf
9	help	zu hülf	zu hülf	zu hülf
10	stop	stopp	stopp	stopp
11	stop	anhalten	stopp	stopp
12	wait	warte	warte	warte
13	wait	warte	warte	warte
14	begin	fang an	fang an	fang an
15	go on	mach weiter	mach weiter	mach weiter
16	hello	hallo	hallo	hallo
17	hello	sers	hallo	hallo
18	hurry	beeil dich	beeil dich	beeil dich
19	hurry	schnell	beeil dich	beeil dich

Amount of exact translations by each model

In [30]:
```
sum(df_test['Deu'] == df_test['Base_Model'])
```

Out[30]: 1861

In [31]:
```
sum(df_test['Deu'] == df_test['Attention_Model'])
```

Out[31]: 2005

Observations and Conclusion

Postivites :

- Translations generated by model equipped with attention were more accurate than with the model without,
- The model without attention was less likely to mistake one or a few words in a sentence, hence better semantic awareness
- After more epochs the model with attention coverges at a much lower loss than the model without, however due to hardware limitations I couldn't show it in the same notebook.
- Model with attention was able to keep up with slightly complex sentences which the base model couldn't in a lot of cases.

Negetives :

- Model with attention takes significanly much time to train approximately times longer (5+ hours as compared to 45 minutes)
- Model with attention takes *4-5 times longer to run* (approx. 21ms) compared to the model without (approx.

2ms) When Inferened on GPU P100.
- Both models didn't perform very well in very complex translations, which can be attributed to limited data and training time.

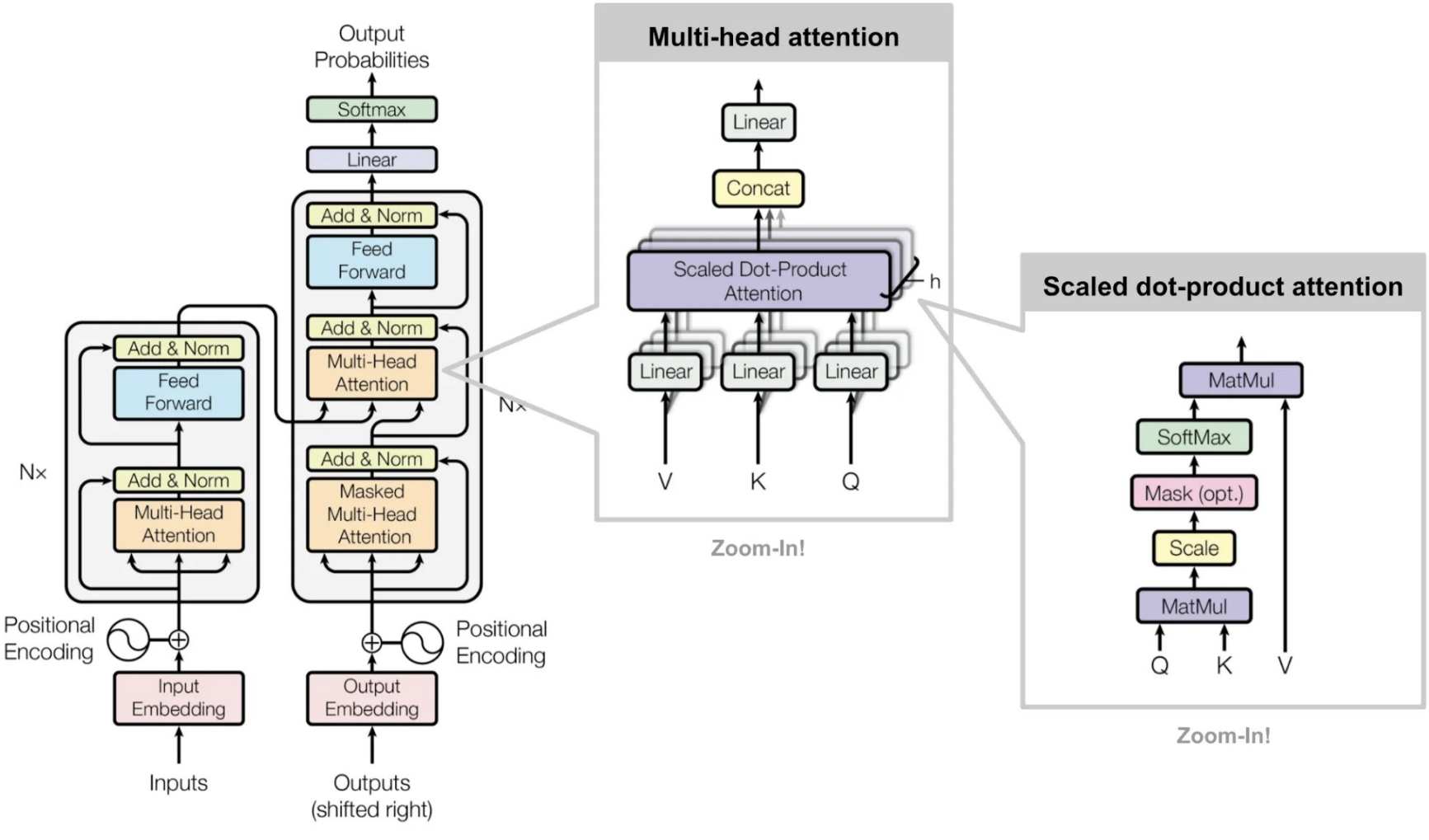
Attention mechanisms have become an essential component of modern sequence-to-sequence models, despite longer training time. By allowing the model to focus on relevant parts of the input sequence, attention helps to improve performance, interpretability, and flexibility.

But why pay 'Attention'?

Since models RNN models which are used sequence modelling need attention despite being specifically designed for the same purpose, it becomes very apparent how powerful attention can be in sequential modelling

Transformer models, which rely entirely on attention mechanisms, have revolutionized the field of natural language processing. They have achieved state-of-the-art results on various tasks, including machine translation, text summarization, and question answering.

Attention in Transformer



I plan on doing transformers in a separate notebook after some time, so stay tuned

Referenes

- https://www.kaggle.com/code/harshjain123/machine-translation-seq2seq-lstms : A very helpful notebook for understanding the data processing pipeline
- https://www.kaggle.com/datasets/alincijov/bilingual-sentence-pairs : Bilingual pair dataset
- https://www.geeksforgeeks.org/seq2seq-model-in-machine-learning : A good Introduction to seq2seq modelling
- https://youtu.be/yInilk6x-OY?si=2e6MOB_DdflA60Ar : A great Lecture on attention

Thank You !!