Theoretical Questions - Chapter 16

1. Why do people use encoder-decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?

Encoder-decoder RNNs take the whole sentence into account. Plain s-t-s RNNs do word per word which can result in a wrong translation

2. How can you deal with variable-length input sequences? What about variable length output sequences?

- Input: add padding to the input sequences that are shorter, make sure the RNN ignores the padding tokens or use ragged tensors
- Output: Make the model output and EOS token

3. What is beam search, and why would you use it?

- Beam search is taking the top k (beam width) predictions, then predicting the next word for each of these first k predictions and taking the top k prediction again. This goes on until there is an eos.
- You would use this rather than gready decoding as this will give you more random results and more correct results because words don't always have the same order in other languages and just picking the highest probability could result in wrong translations for example

4. What is an attention mechanism? How does it help?

- Attention mechanism: The decoder can access all states of the encoder instead of just the output state
- The model can process longer input sequences + model is easier to debug and interpret because you can see which part of the sequence the model was paying attention to

5. What is the most important layer in the transformer architecture? What is its purpose?

- Multi-head attention layer
- Purpose: to allow model to identify which words are most aligned with each other and then improve each word's representation using these contextual clues

6. Explain the three different types of attention mechanisms as used in thetransformer architecture from the seminal paper “Attention is all you need”

- Self-attention: (attention in encoder) attention where Q, K and V are all the same
- Masked Self-attention: (attention in decoder) self-attention where the inputs are shifted 1 position so that the decoder can't look into the future
- Cross-attention: (attention in decoder) where the encoder states (K and V) are combined with the decoders Q

7. Consider dot-product attention mechanism. For simplicity, we don't consider a possible scaling factor

Suppose we have a sequence of length 3 andthat the embedding dimension is 4. The embeddings are:

```
[ 0   2/3   2/3   1/3]
[1/2  1/2   1/2   1/2]
[1/2  1/2  -1/2  -1/2]
```

Apply self-attention to this sequence. What is the output of the attention mechanism?

$Self-attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$

- Q, K, V = [zie arr hierboven]
- d_k = 4

$softmax(\frac{QK^T}{\sqrt{d_k}})$:

```
[0.40353829, 0.37127315, 0.22518855]
[0.36414687, 0.39579271, 0.24006041]
[0.25780466, 0.28020893, 0.46198642]
```

x V:

```
[ 0.29823085,  0.56725638,  0.34206783,  0.20755507]
[ 0.31792656,  0.56069115,  0.32063073,  0.19924844]
[ 0.37109767,  0.54296744,  0.08098103, -0.00495386]
```

8. Take a look at the following image of a transformer architecture

Assume

- sequences length N= 128
- embedding size d_model= 512
- h = 8
- d_k = d_v = d_model / h = 64
- The image contains multiple ∈-symbols without a dimension specified. What is the dimension of each of these matrices
    - W^V_h: (512, 64)
    - V_h: (128, 64)
    - W^Q_h: (512, 64)
    - Q_h: (128, 64)

    - $head_h$: (128, 64)
    - $Z = concat(head_1, ..., head_h)$: (128, 512) (128, 8*64)
    - $MultiHead(Q, K, V) = Z . W^O$: Only the multihead part here -> (128, 64)
    - W^O: (512, 512)

- What is the purpose of Attention?

    - capturing contextual information / relationship between elements in sequence: attention allows model to weigh importance of different parts of input sequence
    - long-range dependencies: RNNs struggle with long sequences -> transformers don't because of attention
    - parallelization: computations in parallel because the sequence order is encoded and does not have to be kept in order during computation
    - flexibility: model learns which parts of sequence are relevant for given task (attention weights)

- Why is it called multi-head attention?

    - because there are multiple linear transformations happening (amount of linear transformations = number of heads)

- What is the purpose of the different heads?

    - finding different patterns / relationships for the sequence in each head

- Why do we use a masked multi-head attention matrix on the decoder side?

    - because we don't want to have the decoder look into the future, so we add a mask

- Why do they use Outputs(shifted right)at the decoder side?

    - for teacher forcing, that way the model can see what the correct target was after having predicted it which speeds up training

- Now it says Layer 1 to 6. Is that always the case?

    - No, that can depend from architecture to architecture

- We talk about V,K and Q (at the bottom of the orange box at the left). Are theses different matrices?

    - no because the box represents the multi-head attention in the encoder, which uses self-attention and in self-attention, Q, K and V are the same

- Why do we talk about Values, Keys and Queries? What is the meaning behind that?

    - queries:
        - elements in input sequence that seek info or attention from other elements
        - queries determine what info to retrieve from rest of sequence
    - keys:
        - all elements in input sequence
        - keys play crucial role in determining relevance / similarity between elements
    - dot product between queries and keys = attention scores
    - values:

        - associated info or content for each element of input sequence
        - values weighed by attention scores to produce final output
        - values capture info relevant to queries

- Which matrices contain trainable parameters?
    - Q: Parameters in the Query Matrix (W_q)
    - K: Parameters in the Key Matrix (W_k)
    - V: Parameters in the Value Matrix (W_v)
- What is the purpose of the positional embedding?
    - because the sequence is not computed in order (but in parallel) we need to encode the order of the sequence to not lose it
- Will the positional embedding be different for another sentence consisting of 128 words?
    - yes
- Why is it important that the output from multi-head attention has the same shape as the input?
    - Consistent shapes across layers ensure smooth information flow through multiple blocks, aiding effective learning of hierarchical patterns.
- At inference time, we want to translate the sentence 'i love football very much' to 'amo mucho el futbol'
    - How often is the encoder executed? 1
    - How often is the decoder executed? You may assume that the translation produced by the decoder is correct: 1
- Feed Forward consists of 2 Dense Layers. Does the number of units in both Dense Layers matter? Can you choose this freely?
    - The first dense layer should have more or equal amounts of units than/as the output_dim and the second should have units = output_dim
- Is it possible to use Dropout in the Feed Forward?
    - Yes
- Can the consecutive words on the decoder side be calculated in parallel or are they calculated one after the other
    - *during training? parallel
    - *at inference time? one after the other
- In the Masked Multi-head attention matrix, why do we use $-\infty$ and not just 0 before applying softmax?
    - the $-\infty$ will become 0 after applying softmax and thus have an attention weight of zero
    - 0 after softmax won't be equal to 0 and thus receive a non-zero attention weight

9. To determine whether an Imdb review is positive or negative, you will use

- An encoder -> this
- A decoder
- A transformer with encoder and decoder

10. To do the classification of emails (spam / not spam), you will use

- An encoder -> this
- A decoder
- A transformer with encoder and decoder

11. We have a dna sequence of length 10_000

```
dna_sequence = "cgttacgatagatatagatttta...cgcgttagaa"
text_vec_layer = tf.keras.layers.TextVectorization(split="character",
standardize="lower")
text_vec_layer.adapt([dna_sequence])
encoded = text_vec_layer([dna_sequence])[0]

# yields
# <tf.Tensor: shape=(...,), dtype=int64,numpy=array([5, 2, 3, 3, 4, 5, 2, 4, 3, 4,
2, 4, 3, 4, 3, 4, 2, 4, 3, ... ])
```

- What is the purpose of .adapt?
  - to let the text_vec_layer learn the vocabulary
- What is the shape of encoded?
  - (10_000,)
- Explain the value of encoded
  - every number represents a unique character of the sequence (c=5, g=2, t=3, a=4, 1=[UNK], 0=padding)
- What is the value of text_vec_layer.vocabulary_size()
  - 6
- What is the value of text_vec_layer.get_vocabulary()
  - ['', '[UNK]', c, g, t, a]
- Does it make sense to do encoded -=2, like we did with the shakespeare text, what will become the new values of encoded
  - yes, because we won't need the padding and the unknown characters in the sequence
  - 3, 0, 1, 1, 2, 3,... (everything -2)

12. The encoder-decoder network is depicted in the book on page 542. Giveone important difference with the Transformer architecture from 2017 onbook page 610, which also has an encoder and decoder

Ik zie geen foto op P. 610

13. Why are positional encodings used with a transformer while this is not thecase with an RNN?

- Because a transformer computes the whole sequence in parallel, which would result in the loss of order without the positional encoding. RNNs process it sequentially hence why they don't need positional encoding

14. Explain the difference between greedy decoding and beam search using the following tree. Use a beam width equal to 3

Assume the sentence definitely starts with the word “The”.

The numbers represent the conditional probabilities of the different words (given the previous words).

The = (100%) = 1

- route 1: 1 * 0.4 = 0.4
  - route 1.1: 0.4 * 0.05 = 0.02
  - route 1.2: 0.4 * 0.05 = 0.02
  - route 1.3: 0.4 * 0.9 = 0.36 --> you would continue with this one
- route 2: 1 * 0.5 = 0.5
  - route 2.1: 0.5 * 0.4 = 0.2
  - route 2.2: 0.5 * 0.3 = 0.15 --> you would continue with this one
  - route 2.3: 0.5 * 0.3 = 0.15 --> you would continue with this one
- route 3: 1 * 0.1 = 0.1
  - route 3.1: 0.1 * 0.3 = 0.03
  - route 3.2: 0.1 * 0.5 = 0.05
  - route 3.3: 0.1 * 0.2 = 0.02