Natural Language Understanding, Generation, and Machine Translation (2021–22)

School of Informatics, University of Edinburgh
Tom Sherborne

Tutorial 4: Transformers (Week 8)

Questions on this worksheet that ask for an expression or calculation generally have one correct answer, and are designed to concretely explore aspect of these models just beyond what we discussed in lecture, by asking you about some implications of model design. These questions are not intended to be difficult, but notice that they are not primarily about recalling information—they require you to engage with the material and pay attention to the details. You should expect that some of the *easier* exam questions may be of this form.

We also include some more open-ended questions that you will need to think about. It is intended to help you see how modelling choices impact the number of parameters and how this impacts your design decisions when applying the Transformer model to a task. Most of these questions are answerable from combined understanding of the lecture as well as your experience of Coursework 1. For a complete understanding, we advise reviewing Attention is all you Need, the paper which proposed the Transformer model, before beginning this tutorial.

1 Modelling a Transformer

a.
self-attention block
W_q = d^2
W_k = d^2
W_v = d^2
W_u = d^2
b_q = d
b_k = d
b_v = d
b_u = d

feedforward block
W_ff1 = d*f
b_ff1 = f
W_ff2 = f*d
b_ff2 = d

b.
self-attention block
W_q = h*d^2
W_k = h*d^2
W_v = h*d^2
W_u = d^2
b_q = d
b_k = d
b_v = d
b_u = d
feedforward block
W_ff1 = d*f
b_ff1 = f
W_ff2 = f*d
b_ff2 = d

A Transformer encoding layer consists of a multi-head self attention network and a feed-forward network with additional normalisation and skip-connection features. A Transformer decoder layer includes an additional multi-head attention network to incorporate information from the encoder during decoding.

Question 1:
Consider a single Transformer encoder layer using narrow self-attention. The self-attention projection size is 1024 (e.g. the size of $W_{\{q,k,v\}}$), the feed-forward projection size is 4096 and each layer has 16 heads.

a. Calculate the number of weights in this single layer that will need to be learned? You can ignore normalisation parameters. 4*d^2+4*d+2*f*d+f+d

We now change the encoder to use *wide* self-attention with other parameters unchanged.

4*h*d^2+ 3*h*d+d+2*f*d+f+d

b. Calculate the number of weights in this new layer? Ignore normalisation parameters again.

narrow self attention: d/h
wide self attention: d
union matrix needs focus on the output projection, so always d

Now we consider an encoder-decoder Transformer model for English sentence compression. The encoder and decoder each have six layers, as described above, and we also define a vocabulary of 64,000 words with corresponding embeddings. Assume the Transformer layers use *narrow* self-attention, the embedding dimensionality is equal to the self-attention projection size and normalisation parameters can be similarly ignored.

c. Calculate the number of weights in this full model. You will need to consider the encoder and decoder layers as well as additional input and output parameters required for this task.

(1)two vocabulary for both input and output. [2*V*d]
(2)encoder [(4*d^2+4*d+2*f*d+f+d)*6]
(3)decoder[(2*(4*d^2+ 3d)+d+ 2*f*d+f+d)]
(4)linear [d*V + V]

The input and output vocabularies for this task are equal as we are encoding and decoding English. Therefore, we can use the same embedding matrix for both the encoder and decoder, referred to as *tying* the embedding matrices, and learn one embedding matrix used for both encoder and decoder.

d. What advantage does *tying* these embedding matrices together have in terms of the learned word representation?

embedding matrix is way bigger than self attention matrix but actually they do nothing in the model.

e. What is the percentage change in the number of weights to be learned from this change?

about 70%

Question 2:
We now want to compare theoretical complexities between different models used in NLP tasks. Inspect Table 1 in Attention is all you Need with a focus on the "Complexity per Layer" column wherein n is the sequence length and d is the representation dimension, the same parameter as the self-attention projection size from Q1.

a. Consider the complexity bounds and your own knowledge of how NLP tasks are constructed – describe when a Transformer network might have lower complexity than other networks.

Transformer O(n^2*d) RNN O(nd^2)

b. Other than complexity – describe other constraining factors to be considered when planning experiments using neural networks for NLP. There is no right answer here, state your own ideas.

The limitation of training data.

2 Considering Permutations

The Transformer self-attention routine operates on *sets* of inputs without respect for sequence ordering. This model will produce identical outputs with varying combinations of inputs because the attention states between any two words will be the same regardless of word position. This gives the Transformer some interesting mathematical properties we want you to think about here.

Question 3:

equivariant: [ABC]->[xyz] ; [CBA]->[zyx], maintains the order

a. For a transformer encoder model without positional embeddings – explain why the model is permutation **equivariant** but not **invariant**.

Consider this model with an additional max pooling layer on the output to combine the outputs to produce one output vector.

max pooling always return the same intermidiated result. (invariant)

c. Explain why the whole model is now permutation **invariant** and not **equivariant**.

These properties are not desirable for sequence modelling in natural language processing. For this reason, we augment the inputs with **positional embeddings** to provide additional information to the input.

d. What additional information is provided with this addition and how does it help sequence modelling? Explain your answer in relation to how the properties described above are affected.

positional embedding allows contextual embedding e.g. I can can a can.

Final discussion question: With this additional augmentation – what is the special desirable property the model now achieves in a sequence-to-sequence framework?