Natural Language Understanding, Generation, and Machine Translation (2023–24)

School of Informatics, University of Edinburgh
Alexandra Birch and Shay Cohen

Tutorial 2: Transformers (Week 6)

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 Lectures 7 & 8 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 Transformer's Efficiency

Question 1:
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 self-attention network has a lower complexity than other networks.

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.

For (a), we want to consider a problem of scale for NLP tasks. According to this paper, a self-attention network has approximate complexity of $O(n^2.d)$, so we are quadratically proportional to sequence length and linearly proportional to dimensionality. The opposite is true for RNNs with $O(n.d^2)$

As a case study, we can set d=1024 and compare between these models. For most NLP tasks here we can approximate that $n << d$ and therefore the Transformer has lower complexity than the RNN. It might be helpful to name and discuss a few tasks where this is true. Conversely, there might be some cases where the inverse is true (document translation?). In terms of increasing n, the additional complexity in a Transformer makes sense as we need an additional self attention computation against all prior sequence elements. For the RNN, we only need to compute the RNN cell with the new state and cumulative history. The intended insight for students is

to think in terms of practical values and make a measured decision concerning which network is best and when.

However for (b), we now want to add in the practical other constraints that exist when running experiments. You could discuss that really this complexity is only a part of choosing models as well as performance and constraints such as GPU Memory, data availability, tunable hyperparameters etc. A typical Transformer is much larger than an RNN and space/speed constraints might arise before complexity can even be considered. This is an opportunity for students to share their own ideas.

Question 2:
One of the greatest advantages of the Transformer model is that it is possible to parallelise its computation, and therefore train a model on much larger training datasets (for example, for pre-training) with less compute.

a. Consider an encoder-decoder RNN model, following up on the question before. Can the computation of the encoder be easily parallelised with respect to the number of tokens (meaning, can you break the input and the computation into chunks such that they run in parallel)? Explain why or why not.

b. Consider the Transformer encoder layer. Explain why its computation can be parallelised over tokens *per layer*. Can computation be easily parallelised across layers?

For (a), consider that an RNN encoder needs to compute the activation of step i, before being able to compute the activation at step $i + 1$. Therefore, RNN computation is not easily parallelisable. New RNN methods, such as state-space models try to overcome some of these issues by building the RNN in such a way that it has certain properties that allow more efficient compute ("associative scan"). For (b), consider that the computation of one Transformer layer activations can be done in parallel. More specifically, the computation of the activation for token i requires integrating information from the *previous* layer of all other tokens, but not from the *current* layer. This also means that the computation cannot be easily parallelisable across layers.

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 becayse 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:

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

For encoder input, [A, B, C], into encoder f(X), producing [X, Y, Z] outputs - the output of [B, C, A] into f(X), will be [Y, Z, X].

More generally, equivariance for some permutation function $r \longrightarrow r(f(x)) = f(r(x))$.

Note that this is slightly different to how the slides portray the problem and they state "the input is position invariant" which may confuse some. The inputs themselves have no sequence information without position embeddings (See Q3(d)) and the attention **for a single output state** is invariant to where the other words in the sequence are. The full Transformer is equivariant for the reason above.

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

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

Extending the previous result, the max-pooling of [X, Y, Z] will be the same as the max-pooling of [Y, Z, X]. It doesn't matter what order the inputs are in so the model is invariant to permutations.

More generally, invariance for some permutation function $r \longrightarrow f(x) = f(r(x))$.

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 embeddings provide exactly what their name states – they imbue the model input with positional information so that each vector is unique representing both semantic information and a token's position in the sequence. Therefore, a sentence such "I can can a can" won't end up with three identical representations for "can" and each representation is contextual on both adjacent words and word order.