Machine Translation Exercise 04

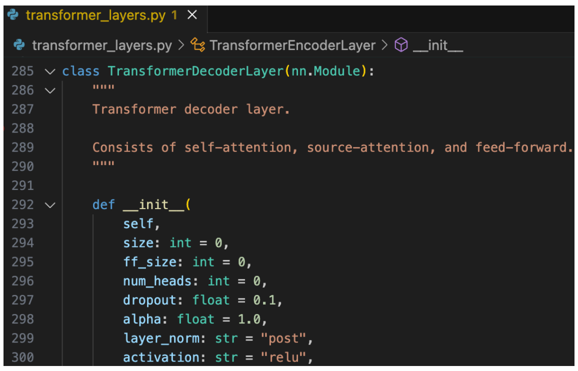
GitHub repository link: https://github.com/k-horn/mt-exercise-4

Part 1: Understanding Code: LayerNorm in JoeyNMT

Where and how are pre- and post-norm implemented?

Pre- and post-normalisation is explicitly implemented in decoders.py, encoders.py and transformer_layers.py. Both normalisations are implemented by default variables in the JoeyNMT code. To change the layer normalisation from ‘pre’ to ‘post’ or vice versa, a user doesn’t have to change the python scripts but has to make the adjustments in the yaml-configuration file which is further explained below.

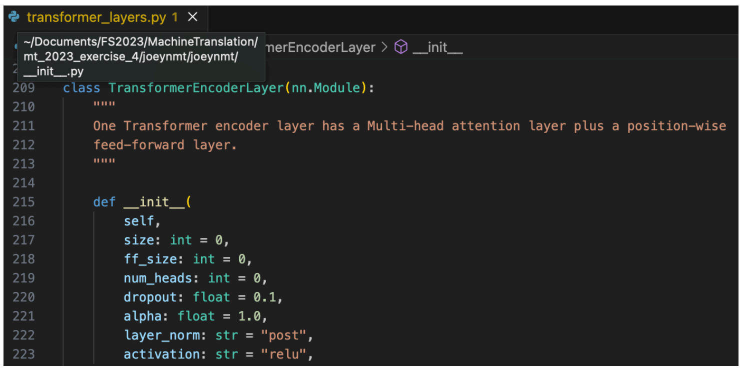
What is the default behaviour?



```python
class TransformerDecoderLayer(nn.Module):
    """
    Transformer decoder layer.

    Consists of self-attention, source-attention, and feed-forward.
    """

    def __init__(
        self,
        size: int = 0,
        ff_size: int = 0,
        num_heads: int = 0,
        dropout: float = 0.1,
        alpha: float = 1.0,
        layer_norm: str = "post",
        activation: str = "relu",
```


transformer_layers.py 1
~/Documents/FS2023/MachineTranslation/
mt_2023_exercise_4/joeynmt/joeynmt/
__init__.py
merEncoderLayer > __init__

```python
class TransformerEncoderLayer(nn.Module):
    """
    One Transformer encoder layer has a Multi-head attention layer plus a position-wise
    feed-forward layer.
    """

    def __init__(
        self,
        size: int = 0,
        ff_size: int = 0,
        num_heads: int = 0,
        dropout: float = 0.1,
        alpha: float = 1.0,
        layer_norm: str = "post",
        activation: str = "relu",
```

Looking at the python script, the default behaviour can be found in transformer_layers.py where layer_norm is set to “post” if nothing else is indicated by the user. The classes *PositionalEncoding, TransformerEncoderLayer & TransformerDecoderLayer* is then imported into encoders.py and decoders.py. Under decoders.py in line 528 you can see that the default setting is also set to “post”. On the other hand, encoders.py has the default setting set to “pre” (*see line 207*). We believe that the JoeyNMT uses a mix of both pre- and post-normalisation because encoders.py accesses transformer_layers.py and overwrites layer_norm from “post” to “pre”.

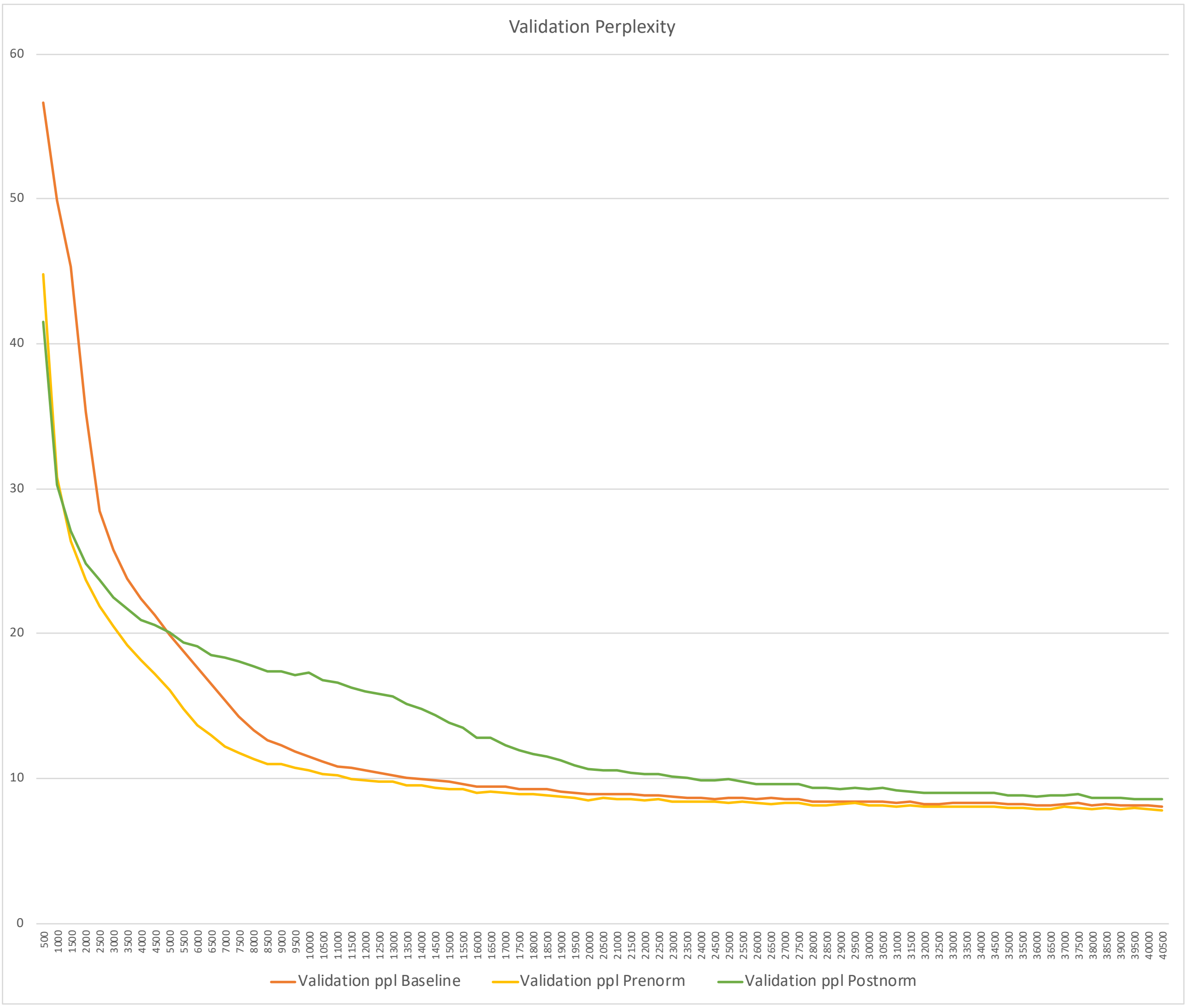
What are the specific differences between the two options and how do you control them?

Both options allow a user to to control where the layer normalisation is done in JoeyNMT. Through the pre-normalisation the inputs are first normalised to make the training process more efficient by stabilising the inputs to the different layers at hand.
In post-normalisation, layer normalisation is done to the outputs, that means after the operation to improve the generalisation of the translation done by JoeyNMT.
The configuration file 'config.yaml' lets the user control the normalisation options by adding and/or changing the variable 'layer_norm' under 'model' (encoder/decoder) to *layer_norm: "pre"* or *layer_norm: "post"*.

Part 2: Implementing Pre- and Post-Normalisation

Important: Table is in ValidationPPL_Ex04.xlsx

Line Chart:



Given that there is a difference in the training progress for the three models, can you think of a reason for it?

The line chart shows that the validation perplexity (ppl) for pre-normalisation is slightly lower than the baseline and that the validation perplexity (ppl) for post-normalisation is higher than the baseline. Looking at the pre-normalisation, the graph shows that its perplexity gets lower way faster than the other perplexities. It is also notable that the starting validation perplexity for the baseline model is at a higher position than our trained models. Overall, the pre-normalisation model seems to be the most efficient.

We believe that the pre-normalisation of the inputs leads to a more homogeneous data for the model to train on. Therefore, the model can focus learning the important representations from the data. Also, as mentioned above, the model can stabilise the training process and faster optimise its parameters.

In what way does our setup differ from Wang et. al. 2019? How could that have influenced our results?

Wang et. al. 2019 also comes to the conclusions that pre-normalisation is more efficient than post-normalisation. Both setups differ in the data set as the paper always works with at least 1 million sentence pairs while we worked with a dataset of 100K segments. Because we are working with a low-resource setting in comparison to the research paper, our results can not be compared to the ones of Wang et. al. 2019.