MT Exercise 4
Layer Normalization for Transformer Models

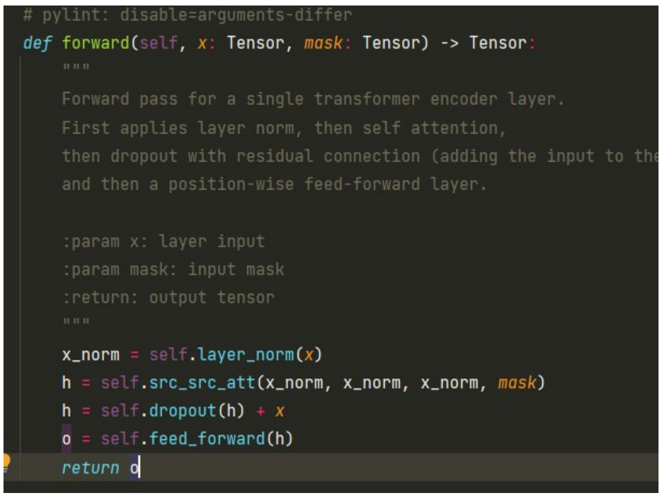
Link to 'ex4' repository: https://github.com/sabrinabraendle/mt-exercise-4

Link to 'joeynmt' repository: https://github.com/sabrinabraendle/joeynmt

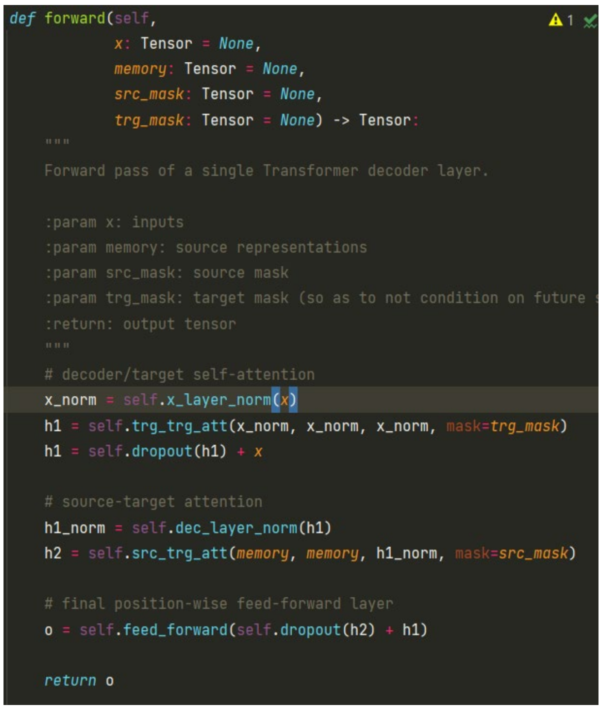
Instances of LayerNorm in JoeyNMT

1. Python script: transformer_layers.py

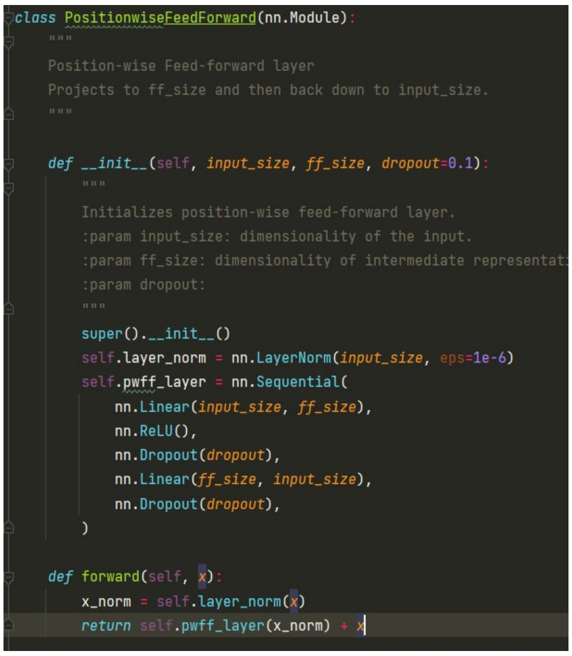
In this script, the layer normalization instance corresponds to pre-norm. The sublayers it modifies are the first one of the encoder and the first and second one of the decoder. The relevant code is the following:



```python
    # pylint: disable=arguments-differ
    def forward(self, x: Tensor, mask: Tensor) -> Tensor:
        """
        Forward pass for a single transformer encoder layer.
        First applies layer norm, then self attention,
        then dropout with residual connection (adding the input to the
        and then a position-wise feed-forward layer.

        :param x: layer input
        :param mask: input mask
        :return: output tensor
        """
        x_norm = self.layer_norm(x)
        h = self.src_src_att(x_norm, x_norm, x_norm, mask)
        h = self.dropout(h) + x
        o = self.feed_forward(h)
        return o
```



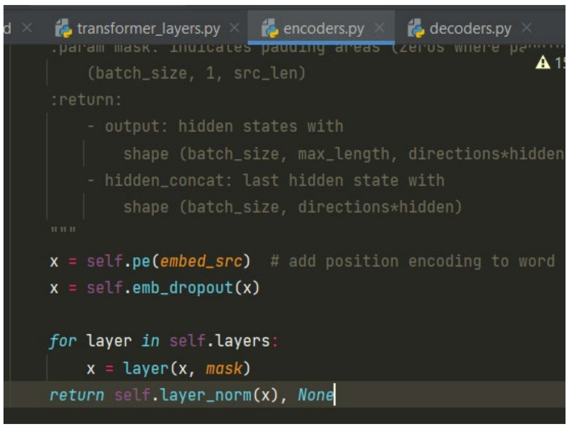
```python
def forward(self,
            x: Tensor = None,
            memory: Tensor = None,
            src_mask: Tensor = None,
            trg_mask: Tensor = None) -> Tensor:
    """
    Forward pass of a single Transformer decoder layer.

    :param x: inputs
    :param memory: source representations
    :param src_mask: source mask
    :param trg_mask: target mask (so as to not condition on future s
    :return: output tensor
    """
    # decoder/target self-attention
    x_norm = self.x_layer_norm(x)
    h1 = self.trg_trg_att(x_norm, x_norm, x_norm, mask=trg_mask)
    h1 = self.dropout(h1) + x

    # source-target attention
    h1_norm = self.dec_layer_norm(h1)
    h2 = self.src_trg_att(memory, memory, h1_norm, mask=src_mask)

    # final position-wise feed-forward layer
    o = self.feed_forward(self.dropout(h2) + h1)

    return o
```



```python
class PositionwiseFeedForward(nn.Module):
    """
    Position-wise Feed-forward layer
    Projects to ff_size and then back down to input_size.
    """

    def __init__(self, input_size, ff_size, dropout=0.1):
        """
        Initializes position-wise feed-forward layer.
        :param input_size: dimensionality of the input.
        :param ff_size: dimensionality of intermediate representat
        :param dropout:
        """
        super().__init__()
        self.layer_norm = nn.LayerNorm(input_size, eps=1e-6)
        self.pwff_layer = nn.Sequential(
            nn.Linear(input_size, ff_size),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(ff_size, input_size),
            nn.Dropout(dropout),
        )

    def forward(self, x):
        x_norm = self.layer_norm(x)
        return self.pwff_layer(x_norm) + x
```

2. Python script: encoders.py

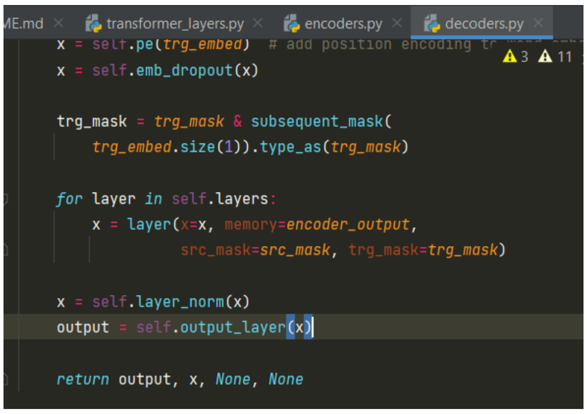
In this script, the layer normalization instance corresponds to post-norm. The sublayer it modifies is the last one of the encoder. The relevant code is the following:



```
    :param mask: indicates padding areas (zeros where pa
        (batch_size, 1, src_len)
    :return:
        - output: hidden states with
            shape (batch_size, max_length, directions*hidden
        - hidden_concat: last hidden state with
            shape (batch_size, directions*hidden)
    """
    x = self.pe(embed_src)  # add position encoding to word
    x = self.emb_dropout(x)

    for layer in self.layers:
        x = layer(x, mask)
    return self.layer_norm(x), None
```

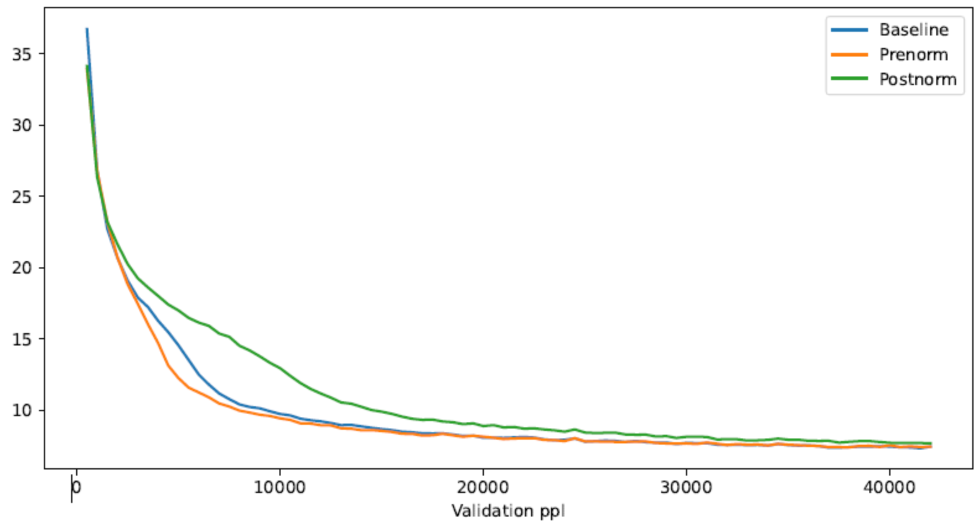
3. Python script: decoders.py

In this script, the layer normalization instance corresponds to post-norm as well. The sublayer it modifies is the last one of the decoder. The relevant code is the following:



```
    x = self.pe(trg_embed)  # add position encoding to
    x = self.emb_dropout(x)

    trg_mask = trg_mask & subsequent_mask(
        trg_embed.size(1)).type_as(trg_mask)

    for layer in self.layers:
        x = layer(x=x, memory=encoder_output,
                  src_mask=src_mask, trg_mask=trg_mask)

    x = self.layer_norm(x)
    output = self.output_layer(x)

    return output, x, None, None
```

Discussion



In the prenorm setting, the validation perplexity could be reduced slightly faster than the baseline around step 5000. But the perplexity can be reduced about more or less the same amount with further steps during the training. This makes sense as the pre-norm Transformer benefits more from the increase in encoder depth.

In the postnorm setting, a worse performance can be seen around step 500, while the perplexity values approach the ones of the baseline with further steps. It is possible that the performance is not significantly better because of the vanishing gradient problem.

Wang et al. also adapted some other hyperparameters according to the setting, i.e. prenorm or postnorm. It is therefore possible that the results in their experiments could be optimized as in our settings, the hyperparameters were kept the same. Additionally, the Transformer-Base model was updated for 100k steps, whereas our model was only trained during 42000 steps.