Xun Xiao 21-738-885

Jingmiao Li 22-748-651

Task 1

1. Where and how are pre- and post-norm implemented:

1) Layer normalization position **assignment**: with the parameter layer_norm in the *.yaml files in the configs directory.
2) **Implemention**: in transformer_layers.py in the following classes: PositionwiseFeedForward TransformerEncoderLayer TransformerDecoderLayer These classes define different layers of the Transformer model where layer normalization is applied either before or after certain operations.

2. Default:

1) The default behavior is to apply layer normalization after the operations, which is referred to as "post-norm".
2) The layer_norm parameter in the constructor of each class is set to "post" by default.
3) The code asserts that layer_norm_position is either "pre" or "post", ensuring that only these two options are allowed.

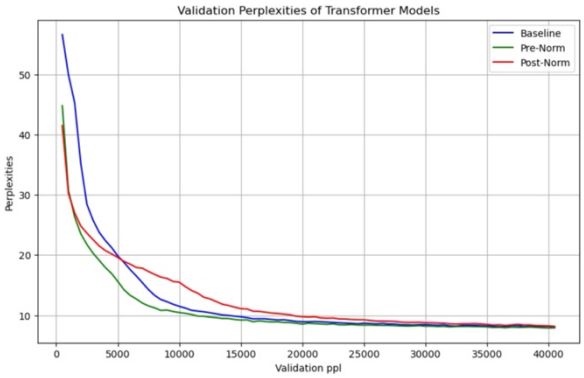
3. Difference and how to control:

Users can control the position of layer normalization through the layer_norm parameter in configs files.

1) When `layer_norm` is set to "pre", layer normalization is applied before the main operations.
2) When `layer_norm` is set to "post" (the default), layer normalization is applied after the main operations.

Task 2

1. Difference in Training Progress and Reason:



The chart shows variations in the training progress among the three models. Each model converges at a different rate, and the validation perplexity fluctuates differently over time.

This difference could be due to the different positions of layer normalization:

1) **Pre-Norm:**
   If layer normalization is applied before the activation function in each layer, it normalizes the inputs to the activation function. This can help stabilize the gradients and activations, making the optimization process smoother and potentially leading to faster convergence during training. Howerver, pre-normalization might suffer from vanishing gradients in deeper networks.
2) **Post-Norm:**
   If layer normalization is applied after the activation function in each layer, it normalizes the outputs of the activation function. While still effective in stabilizing the activations, post-normalization might have different effects on the gradients compared to pre-normalization. It could introduce gradient noise during training.This could influence how quickly the model learns and converges during training.
3) **Baseline:**
   If no layer normalization is applied as in the baseline model, the model may experience more unstable gradients and activations, especially in deeper networks. This instability could lead to slower convergence during training, as the optimization process may encounter difficulties in effectively updating the model parameters.
   ***Note:** In the transformer_layers.py file, layer_norm is “post” by default as mentioned in task 1. However, since the baseline and the post-norm models performed differently, we assume that no normalization is applied to the baseline.

2. Comparison with Wang et al. 2019:

Our setup differs from Wang et al. (2019) in several aspects:

1) Data size: in their paper, they use 4.5M pre- processed data while our data is much smaller.
2) Layers of encoder and decoder: They use 20-layer encoder and 6-layer decoder while we only use 4-layer encoder and 1-layer decoder.
3) Dropout: They use the attention dropout P= 0.1 and feed-forward dropout P = 0.1 while we use 0 for dropout.
4) Batch size: They use 8192 for the batch size while we use 2048 for the batch size.