TransformerEncoder layer

`TransformerEncoder` class [source]

```
keras_nlp.layers.TransformerEncoder(
    intermediate_dim,
    num_heads,
    dropout=0,
    activation="relu",
    layer_norm_epsilon=1e-05,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    name=None,
    **kwargs
)
```

Transformer encoder.

This class follows the architecture of the transformer encoder layer in the paper Attention is All You Need. Users can instantiate multiple instances of this class to stack up an encoder.

This layer will correctly compute an attention mask from an implicit Keras padding mask (for example, by passing `mask_zero=True` to a `keras.layers.Embedding` layer). See the Masking and Padding guide for more details.

Arguments

- **intermediate_dim**: int, the hidden size of feedforward network.
- **num_heads**: int, the number of heads in the `keras.layers.MultiHeadAttention` layer.
- **dropout**: float, defaults to 0. the dropout value, shared by `keras.layers.MultiHeadAttention` and feedforward network.
- **activation**: string or `keras.activations`, defaults to "relu". the activation function of feedforward network.
- **layer_norm_epsilon**: float, defaults to 1e-5. The epsilon value in layer normalization components.
- **kernel_initializer**: string or `keras.initializers` initializer, defaults to "glorot_uniform". The kernel initializer for the dense and multiheaded attention layers.
- **bias_initializer**: string or `keras.initializers` initializer, defaults to "zeros". The bias initializer for the dense and multiheaded attention layers.
- **name**: string, defaults to None. The name of the layer.
- ****kwargs**: other keyword arguments.

Examples

```
# Create a single transformer encoder layer.
encoder = keras_nlp.layers.TransformerEncoder(
    intermediate_dim=64, num_heads=8)

# Create a simple model containing the encoder.
input = keras.Input(shape=[10, 64])
output = encoder(input)
model = keras.Model(inputs=input, outputs=output)

# Call encoder on the inputs.
input_data = tf.random.uniform(shape=[2, 10, 64])
output = model(input_data)
```

References

- Vaswani et al., 2017

`call` method [source]

```
TransformerEncoder.call(inputs, padding_mask=None, attention_mask=None)
```

Forward pass of the TransformerEncoder.

Arguments

- **inputs**: a Tensor. The input data to TransformerEncoder, should be of shape [batch_size, sequence_length, feature_dim].
- **padding_mask**: a boolean Tensor. It indicates if the token should be masked because the token is introduced due to padding. `padding_mask` should have shape [batch_size, sequence_length]. False means the certain certain is masked out.
- **attention_mask**: a boolean Tensor. Customized mask used to mask out certain tokens. `attention_mask` should have shape [batch_size, sequence_length, sequence_length].

Returns

A Tensor of the same shape as the `inputs`.
