Search Keras documentation...

[MultiHeadAttention layer]
[MultiHeadAttention class]

MultiHeadAttention layer

`MultiHeadAttention` class

[source]

```
tf.keras.layers.MultiHeadAttention(
    num_heads,
    key_dim,
    value_dim=None,
    dropout=0.0,
    use_bias=True,
    output_shape=None,
    attention_axes=None,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

MultiHeadAttention layer.

This is an implementation of multi-headed attention as described in the paper "Attention is all you Need" (Vaswani et al., 2017). If `query`, `key,` `value` are the same, then this is self-attention. Each timestep in `query` attends to the corresponding sequence in `key`, and returns a fixed-width vector.

This layer first projects `query`, `key` and `value`. These are (effectively) a list of tensors of length `num_attention_heads`, where the corresponding shapes are `(batch_size, <query dimensions>, key_dim)`, `(batch_size, <key/value dimensions>, key_dim)`, `(batch_size, <key/value dimensions>, value_dim)`.

Then, the query and key tensors are dot-producted and scaled. These are softmaxed to obtain attention probabilities. The value tensors are then interpolated by these probabilities, then concatenated back to a single tensor.

Finally, the result tensor with the last dimension as value_dim can take an linear projection and return.

When using MultiHeadAttention inside a custom Layer, the custom Layer must implement `build()` and call MultiHeadAttention's `_build_from_signature()`. This enables weights to be restored correctly when the model is loaded. TODO(b/172609172): link to documentation about calling custom build functions when used in a custom Layer.

Examples

Performs 1D cross-attention over two sequence inputs with an attention mask. Returns the additional attention weights over heads.

```
>>> layer = MultiHeadAttention(num_heads=2, key_dim=2)
>>> target = tf.keras.Input(shape=[8, 16])
>>> source = tf.keras.Input(shape=[4, 16])
>>> output_tensor, weights = layer(target, source,
...                                return_attention_scores=True)
>>> print(output_tensor.shape)
(None, 8, 16)
>>> print(weights.shape)
(None, 2, 8, 4)
```

Performs 2D self-attention over a 5D input tensor on axes 2 and 3.

```
>>> layer = MultiHeadAttention(num_heads=2, key_dim=2, attention_axes=(2, 3))
>>> input_tensor = tf.keras.Input(shape=[5, 3, 4, 16])
>>> output_tensor = layer(input_tensor, input_tensor)
>>> print(output_tensor.shape)
(None, 5, 3, 4, 16)
```

MultiHeadAttention layer
MultiHeadAttention class

Arguments

- **num_heads**: Number of attention heads.
- **key_dim**: Size of each attention head for query and key.
- **value_dim**: Size of each attention head for value.
- **dropout**: Dropout probability.
- **use_bias**: Boolean, whether the dense layers use bias vectors/matrices.
- **output_shape**: The expected shape of an output tensor, besides the batch and sequence dims. If not specified, projects back to the key feature dim.
- **attention_axes**: axes over which the attention is applied. `None` means attention over all axes, but batch, heads, and features.
- **kernel_initializer**: Initializer for dense layer kernels.
- **bias_initializer**: Initializer for dense layer biases.
- **kernel_regularizer**: Regularizer for dense layer kernels.
- **bias_regularizer**: Regularizer for dense layer biases.
- **activity_regularizer**: Regularizer for dense layer activity.
- **kernel_constraint**: Constraint for dense layer kernels.
- **bias_constraint**: Constraint for dense layer kernels.

Call arguments

- **query**: Query `Tensor` of shape `(B, T, dim)`.
- **value**: Value `Tensor` of shape `(B, S, dim)`.
- **key**: Optional key `Tensor` of shape `(B, S, dim)`. If not given, will use `value` for both `key` and `value`, which is the most common case.
- **attention_mask**: a boolean mask of shape `(B, T, S)`, that prevents attention to certain positions. The boolean mask specifies which query elements can attend to which key elements, 1 indicates attention and 0 indicates no attention. Broadcasting can happen for the missing batch dimensions and the head dimension.
- **return_attention_scores**: A boolean to indicate whether the output should be `(attention_output, attention_scores)` if `True`, or `attention_output` if `False`. Defaults to `False`.
- **training**: Python boolean indicating whether the layer should behave in training mode (adding dropout) or in inference mode (no dropout). Defaults to either using the training mode of the parent layer/model, or False (inference) if there is no parent layer.

Returns

- **attention_output**: The result of the computation, of shape `(B, T, E)`, where `T` is for target sequence shapes and `E` is the query input last dimension if `output_shape` is `None`. Otherwise, the multi-head outputs are project to the shape specified by `output_shape`.
- **attention_scores**: [Optional] multi-head attention coefficients over attention axes.