This code defines a custom Attention layer as a subclass of the tf.keras.Model class in TensorFlow. The Attention layer takes in two inputs:

encode\_outs: A tensor representing the output of an encoder model, which typically encodes a source sequence in a sequence-to-sequence model.

pre\_state: A tensor representing the previous state of the decoder model in a sequence-to-sequence model.

The Attention layer calculates an attention score for each encoded input token, based on how well it matches the previous decoder state. The attention scores are then used to weight the encoded inputs to produce a context vector, which is then used as input to the decoder model.

Here is a breakdown of each part of the code:

super(Attention, self).\_\_init\_\_(): This line calls the constructor of the tf.keras.Model class to initialize the Attention object.

self.W\_out\_encode = tf.keras.layers.Dense(hidden\_unit): This creates a fully connected layer that takes in the encoder outputs and maps them to a hidden state of size hidden\_unit.

self.W\_state = tf.keras.layers.Dense(hidden\_unit): This creates another fully connected layer that takes in the decoder state and maps it to a hidden state of size hidden\_unit.

self.V = tf.keras.layers.Dense(1): This creates a fully connected layer that takes in the concatenated hidden states of the W\_out\_encode and W\_state layers and maps them to a single scalar value representing the attention score.

def call(self, encode\_outs, pre\_state): This defines the call method of the Attention layer, which takes in two input tensors: encode\_outs and pre\_state.

pre\_state = tf.expand\_dims(pre\_state, axis=1): This adds an extra dimension to the pre\_state tensor, turning it into a tensor of shape (batch\_size, 1, hidden\_unit). This is necessary to allow for broadcasting during the attention calculation.

pre\_state = self.W\_state(pre\_state): This applies the W\_state layer to the pre\_state tensor, resulting in a tensor of shape (batch\_size, 1, hidden\_unit).

encode\_outs = self.W\_out\_encode(encode\_outs): This applies the W\_out\_encode layer to the encode\_outs tensor, resulting in a tensor of shape (batch\_size, seq\_len, hidden\_unit).

score = self.V(tf.nn.tanh(pre\_state + encode\_outs)): This concatenates the pre\_state tensor with the encode\_outs tensor along the sequence length dimension, applies the tanh activation function to the result, and then applies the V layer to obtain a tensor of shape (batch\_size, seq\_len, 1) representing the attention scores.

score = tf.nn.softmax(score, axis=1): This applies the softmax function to the score tensor along the sequence length dimension to obtain a tensor of shape (batch\_size, seq\_len, 1) representing the attention weights.

context\_vector = score\*encode\_outs: This multiplies the attention weights with the encoded output tensor element-wise, resulting in a tensor of shape (batch\_size, seq\_len, hidden\_unit) representing the weighted encoded inputs.

context\_vector = tf.reduce\_sum(context\_vector, axis=1): This sums the weighted encoded inputs along the sequence length dimension to obtain a tensor of shape (batch\_size, hidden\_unit) representing the context vector.

return context\_vector, score: This returns the context vector and the attention scores.