tf.contrib.legacy_seq2seq.attention_decoder

```
attention_decoder(
    decoder_inputs,
    initial_state,
    attention_states,
    cell,
    output_size=None,
    num_heads=1,
    loop_function=None,
    dtype=None,
    scope=None,
    initial_state_attention=False
)
```

Defined in `tensorflow/contrib/legacy_seq2seq/python/ops/seq2seq.py`.

RNN decoder with attention for the sequence-to-sequence model.

In this context "attention" means that, during decoding, the RNN can look up information in the additional tensor attention_states, and it does this by focusing on a few entries from the tensor. This model has proven to yield especially good results in a number of sequence-to-sequence tasks. This implementation is based on http://arxiv.org/abs/1412.7449 (see below for details). It is recommended for complex sequence-to-sequence tasks.

Args:

- `decoder_inputs`: A list of 2D Tensors [batch_size x input_size].
- `initial_state`: 2D Tensor [batch_size x cell.state_size].
- `attention_states`: 3D Tensor [batch_size x attn_length x attn_size].
- `cell`: tf.nn.rnn_cell.RNNCell defining the cell function and size.
- `output_size`: Size of the output vectors; if None, we use cell.output_size.
- `num_heads`: Number of attention heads that read from attention_states.
- `loop_function`: If not None, this function will be applied to i-th output in order to generate i+1-th input, and decoder_inputs will be ignored, except for the first element ("GO" symbol). This can be used for decoding, but also for training to emulate http://arxiv.org/abs/1506.03099. Signature – loop_function(prev, i) = next
  - prev is a 2D Tensor of shape [batch_size x output_size],
  - i is an integer, the step number (when advanced control is needed),
  - next is a 2D Tensor of shape [batch_size x input_size].
- `dtype`: The dtype to use for the RNN initial state (default: tf.float32).
- `scope`: VariableScope for the created subgraph; default: "attention_decoder".
- `initial_state_attention`: If False (default), initial attentions are zero. If True, initialize the attentions from the initial state and attention states – useful when we wish to resume decoding from a previously stored decoder state and attention states.

Returns:

A tuple of the form (outputs, state), where: `outputs`: *A list of the same length as decoder_inputs of 2D Tensors of shape [batch_size x output_size]. These represent the generated outputs. Output i is computed from input i (which is either the i-th element of decoder_inputs or loop_function(output {i-1}, i)) as follows. First, we run the cell on a combination of the input and previous attention masks: cell_output, new_state = cell(linear(input, prev_attn), prev_state). Then, we calculate new attention masks: new_attn = softmax(V^T * tanh(W * attention_states + U * new_state)) and then we calculate the output: output = linear(cell_output, new_attn).* `state`: The state of each decoder cell the final time-step. It is a 2D Tensor of shape [batch_size x cell.state_size].

Raises:

- `ValueError`: when num_heads is not positive, there are no inputs, shapes of attention_states are not set, or input size cannot be inferred from the input.
