tf.nn.dynamic_rnn

```
dynamic_rnn(
    cell,
    inputs,
    sequence_length=None,
    initial_state=None,
    dtype=None,
    parallel_iterations=None,
    swap_memory=False,
    time_major=False,
    scope=None
)
```

Defined in `tensorflow/python/ops/rnn.py`.

See the guide: Neural Network > Recurrent Neural Networks

Creates a recurrent neural network specified by RNNCell `cell`.

Performs fully dynamic unrolling of `inputs`.

Example:

```
# create a BasicRNNCell
rnn_cell = tf.nn.rnn_cell.BasicRNNCell(hidden_size)

# 'outputs' is a tensor of shape [batch_size, max_time, cell_state_size]

# defining initial state
initial_state = rnn_cell.zero_state(batch_size, dtype=tf.float32)

# 'state' is a tensor of shape [batch_size, cell_state_size]
outputs, state = tf.nn.dynamic_rnn(rnn_cell, input_data,
                                   initial_state=initial_state,
                                   dtype=tf.float32)
```

```
# create 2 LSTMCells
rnn_layers = [tf.nn.rnn_cell.LSTMCell(size) for size in [128, 256]]

# create a RNN cell composed sequentially of a number of RNNCells
multi_rnn_cell = tf.nn.rnn_cell.MultiRNNCell(rnn_layers)

# 'outputs' is a tensor of shape [batch_size, max_time, 256]
# 'state' is a N-tuple where N is the number of LSTMCells containing a
# tf.contrib.rnn.LSTMStateTuple for each cell
outputs, state = tf.nn.dynamic_rnn(cell=multi_rnn_cell,
                                   inputs=data,
                                   dtype=tf.float32)
```

Args:

- `cell`: An instance of RNNCell.
- `inputs`: The RNN inputs. If `time_major == False` (default), this must be a `Tensor` of shape: `[batch_size,`

`max_time, ...]`, or a nested tuple of such elements. If `time_major == True`, this must be a `Tensor` of shape: `[max_time, batch_size, ...]`, or a nested tuple of such elements. This may also be a (possibly nested) tuple of Tensors satisfying this property. The first two dimensions must match across all the inputs, but otherwise the ranks and other shape components may differ. In this case, input to `cell` at each time-step will replicate the structure of these tuples, except for the time dimension (from which the time is taken). The input to `cell` at each time step will be a `Tensor` or (possibly nested) tuple of Tensors each with dimensions `[batch_size, ...]`.

- `sequence_length`: (optional) An int32/int64 vector sized `[batch_size]`. Used to copy-through state and zero-out outputs when past a batch element's sequence length. So it's more for correctness than performance.
- `initial_state`: (optional) An initial state for the RNN. If `cell.state_size` is an integer, this must be a `Tensor` of appropriate type and shape `[batch_size, cell.state_size]`. If `cell.state_size` is a tuple, this should be a tuple of tensors having shapes `[batch_size, s] for s in cell.state_size`.
- `dtype`: (optional) The data type for the initial state and expected output. Required if initial_state is not provided or RNN state has a heterogeneous dtype.
- `parallel_iterations`: (Default: 32). The number of iterations to run in parallel. Those operations which do not have any temporal dependency and can be run in parallel, will be. This parameter trades off time for space. Values >> 1 use more memory but take less time, while smaller values use less memory but computations take longer.
- `swap_memory`: Transparently swap the tensors produced in forward inference but needed for back prop from GPU to CPU. This allows training RNNs which would typically not fit on a single GPU, with very minimal (or no) performance penalty.
- `time_major`: The shape format of the `inputs` and `outputs` Tensors. If true, these `Tensors` must be shaped `[max_time, batch_size, depth]`. If false, these `Tensors` must be shaped `[batch_size, max_time, depth]`. Using `time_major = True` is a bit more efficient because it avoids transposes at the beginning and end of the RNN calculation. However, most TensorFlow data is batch-major, so by default this function accepts input and emits output in batch-major form.
- `scope`: VariableScope for the created subgraph; defaults to "rnn".

Returns:

A pair (outputs, state) where:

- `outputs`: The RNN output `Tensor`.

  If time_major == False (default), this will be a `Tensor` shaped: `[batch_size, max_time, cell.output_size]`.

  If time_major == True, this will be a `Tensor` shaped: `[max_time, batch_size, cell.output_size]`.

  Note, if `cell.output_size` is a (possibly nested) tuple of integers or `TensorShape` objects, then `outputs` will be a tuple having the same structure as `cell.output_size`, containing Tensors having shapes corresponding to the shape data in `cell.output_size`.
- `state`: The final state. If `cell.state_size` is an int, this will be shaped `[batch_size, cell.state_size]`. If it is a `TensorShape`, this will be shaped `[batch_size] + cell.state_size`. If it is a (possibly nested) tuple of ints or `TensorShape`, this will be a tuple having the corresponding shapes. If cells are `LSTMCells` `state` will be a tuple containing a `LSTMStateTuple` for each cell.

Raises:

- `TypeError`: If `cell` is not an instance of RNNCell.
- `ValueError`: If inputs is None or an empty list.

Stay Connected

Blog

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**