tf.contrib.seq2seq.BeamSearchDecoder

Contents

Class `BeamSearchDecoder`

Inherits From: `Decoder`

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

BeamSearch sampling decoder.

NOTE If you are using the `BeamSearchDecoder` with a cell wrapped in `AttentionWrapper`, then you must ensure that:

- The encoder output has been tiled to `beam_width` via `tf.contrib.seq2seq.tile_batch` (NOT `tf.tile`).
- The `batch_size` argument passed to the `zero_state` method of this wrapper is equal to `true_batch_size * beam_width`.
- The initial state created with `zero_state` above contains a `cell_state` value containing properly tiled final state from the encoder.

An example:

```
tiled_encoder_outputs = tf.contrib.seq2seq.tile_batch(
    encoder_outputs, multiplier=beam_width)
tiled_encoder_final_state = tf.conrib.seq2seq.tile_batch(
    encoder_final_state, multiplier=beam_width)
tiled_sequence_length = tf.contrib.seq2seq.tile_batch(
    sequence_length, multiplier=beam_width)
attention_mechanism = MyFavoriteAttentionMechanism(
    num_units=attention_depth,
    memory=tiled_inputs,
    memory_sequence_length=tiled_sequence_length)
attention_cell = AttentionWrapper(cell, attention_mechanism, ...)
decoder_initial_state = attention_cell.zero_state(
    dtype, batch_size=true_batch_size * beam_width)
decoder_initial_state = decoder_initial_state.clone(
    cell_state=tiled_encoder_final_state)
```

Properties

`batch_size`

`output_dtype`

`output_size`

Methods

`__init__`

```
__init__(
    cell,
    embedding,
    start_tokens,
    end_token,
    initial_state,
    beam_width,
    output_layer=None,
    length_penalty_weight=0.0
)
```

Initialize the BeamSearchDecoder.

Args:

- `cell`: An `RNNCell` instance.
- `embedding`: A callable that takes a vector tensor of `ids` (argmax ids), or the `params` argument for `embedding_lookup`.
- `start_tokens`: `int32` vector shaped `[batch_size]`, the start tokens.
- `end_token`: `int32` scalar, the token that marks end of decoding.
- `initial_state`: A (possibly nested tuple of...) tensors and TensorArrays.
- `beam_width`: Python integer, the number of beams.
- `output_layer`: (Optional) An instance of `tf.layers.Layer`, i.e., `tf.layers.Dense`. Optional layer to apply to the RNN output prior to storing the result or sampling.
- `length_penalty_weight`: Float weight to penalize length. Disabled with 0.0.

Raises:

- `TypeError`: if `cell` is not an instance of `RNNCell`, or `output_layer` is not an instance of `tf.layers.Layer`.
- `ValueError`: If `start_tokens` is not a vector or `end_token` is not a scalar.

`finalize`

```
finalize(
    outputs,
    final_state,
    sequence_lengths
)
```

Finalize and return the predicted_ids.

Args:

- `outputs`: An instance of BeamSearchDecoderOutput.
- `final_state`: An instance of BeamSearchDecoderState. Passed through to the output.

- `sequence_lengths`: An `int64` tensor shaped `[batch_size, beam_width]`. The sequence lengths determined for each beam during decode.

Returns:

- `outputs`: An instance of FinalBeamSearchDecoderOutput where the predicted_ids are the result of calling _gather_tree.
- `final_state`: The same input instance of BeamSearchDecoderState.

`initialize`

```
initialize(name=None)
```

Initialize the decoder.

Args:

- `name`: Name scope for any created operations.

Returns:

`(finished, start_inputs, initial_state)`.

`step`

```
step(
    time,
    inputs,
    state,
    name=None
)
```

Perform a decoding step.

Args:

- `time`: scalar `int32` tensor.
- `inputs`: A (structure of) input tensors.
- `state`: A (structure of) state tensors and TensorArrays.
- `name`: Name scope for any created operations.

Returns:

`(outputs, next_state, next_inputs, finished)`.