tf.contrib.framework.load_and_remap_matrix_initializer

```
load_and_remap_matrix_initializer(
    ckpt_path,
    old_tensor_name,
    new_row_vocab_size,
    new_col_vocab_size,
    old_row_vocab_file=None,
    new_row_vocab_file=None,
    old_col_vocab_file=None,
    new_col_vocab_file=None,
    num_row_oov_buckets=0,
    num_col_oov_buckets=0,
    initializer=None,
    max_rows_in_memory=-1
)
```

Defined in `tensorflow/python/training/checkpoint_ops.py`.

Returns a var initializer for loading and remapping a 2-D (matrix) tensor.

The returned initializer loads a 2-D (matrix) `Tensor` with name `old_tensor_name` from the checkpoint at `ckpt_path`. It will reorder the rows/columns according to the specified vocab files and append additional out-of-vocabulary rows/columns according to the number of OOV buckets.

The format of the file at the `{old,new}_{row,col}_vocab_file` path should be a text file, with each line containing a single entity within the vocabulary. Let the function `line_of(f, "x")` return the 0-indexed line number of the entity "x" in file f, and the function `entity_at(f, i)` return the entity at line i of file f. Then, row i of the new output matrix will be taken from row `line_of(old_row_vocab_file, entity_at(new_row_vocab_file, i))` of the old matrix. If any entity in `new_row_vocab_file` is not found in `old_row_vocab_file`, that row is considered a "missing" row, and its values will be initialized using the `initializer` arg. The same logic also applies for the columns.

For example, assuming that:

- `old_row_vocab_file` contains "mercury\nvenus\nmars"
- `new_row_vocab_file` contains "venus\njupiter\nmercury"
- `old_col_vocab_file` contains "good\nbetter\nbest"
- `new_col_vocab_file` contains "good\nbest\nfantastic"
- `initializer` returns the natural numbers `[1, 2, 3, 4, ...]`
- `w(i, j)` represents the value from row i, column j of the old matrix

Then the new output matrix will look like:

`[[w(1, 0), w(1, 2), 1], [2, 3, 4], [w(0, 0), w(0, 2), 5]]`

If we further specify that:

- `num_row_oov_buckets` == 2
- `num_col_oov_buckets` == 1

Then the new output matrix will look like:

`[[w(1, 0), w(1, 2), 1, 12], [2, 3, 4, 13], [w(0, 0), w(0, 2), 5, 14], [6, 7, 8, 15], [9, 10, 11, 16]]`

If `{old,new}_row_vocab_file` are None, we assume that the old and new row vocab files are the same, and no row remapping is done. If `{old,new}_col_vocab_file` are None, we assume that the old and new column vocab files are the same, and no column remapping is done.

The returned initializer only supports div-partitioning along the row axis. It does not support partitioning along the column axis or mod-partitioning.

NOTE: When this is used to warm-start variables, client code should use `tf.lookup.index_table_from_tensor()` like contrib/layers/python/layers/feature_column.py does, as opposed to `tf.feature_to_id()` - in order to ensure the underlying lookup tables are the same.

Args:

- `ckpt_path`: Path to the TensorFlow checkpoint (version 2, `TensorBundle`) from which the old matrix `Tensor` will be loaded.
- `old_tensor_name`: Name of the 2-D `Tensor` to load from checkpoint.
- `new_row_vocab_size`: `int` specifying the number of entries in `new_row_vocab_file`. If no row remapping is needed (no row vocab provided), this should be equal to the number of rows to load from the old matrix (which can theoretically be smaller than the number of rows in the old matrix).
- `new_col_vocab_size`: `int` specifying the number of entries in `new_col_vocab_file`. If no column remapping is needed (no column vocab provided), this should be equal to the number of columns in the old matrix.
- `old_row_vocab_file`: A scalar `Tensor` of type `string` containing the path to the old row vocabulary file. Can be None, which represents no remapping on the row axis.
- `new_row_vocab_file`: A scalar `Tensor` of type `string` containing the path to the new row vocabulary file. Can be None, which represents no remapping on the row axis.
- `old_col_vocab_file`: A scalar `Tensor` of type `string` containing the path to the old column vocabulary file. Can be None, which represents no remapping on the column axis.
- `new_col_vocab_file`: A scalar `Tensor` of type `string` containing the path to the new column vocabulary file. Can be None, which represents no remapping on the column axis.
- `num_row_oov_buckets`: `int` specifying the number of out-of-vocabulary rows to append. Must be >= 0.
- `num_col_oov_buckets`: `int` specifying the number of out-of-vocabulary columns to append. Must be >= 0.
- `initializer`: Initializer function to initialize missing values. Accepts a 1-D tensor as the arg to specify the shape of the returned tensor. If `None`, defaults to using `zeros_initializer()`.
- `max_rows_in_memory`: `int` specifying the maximum number of rows to load from the checkpoint at once. If less than or equal to 0, the entire matrix will be loaded into memory. Setting this arg trades increased disk reads for lower memory usage.

Returns:

A variable initializer function that should be used to initialize a (potentially partitioned) `Variable` whose complete shape is `[new_row_vocab_size + num_row_oov_buckets, new_col_vocab_size + num_col_oov_buckets]`.

Raises:

- `TypeError`: If `initializer` is specified but not callable.

Last updated November 2, 2017.