tf.map_fn

```
map_fn(
    fn,
    elems,
    dtype=None,
    parallel_iterations=10,
    back_prop=True,
    swap_memory=False,
    infer_shape=True,
    name=None
)
```

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

See the guide: Higher Order Functions > Higher Order Operators

map on the list of tensors unpacked from `elems` on dimension 0.

The simplest version of `map_fn` repeatedly applies the callable `fn` to a sequence of elements from first to last. The elements are made of the tensors unpacked from `elems`. `dtype` is the data type of the return value of `fn`. Users must provide `dtype` if it is different from the data type of `elems`.

Suppose that `elems` is unpacked into `values`, a list of tensors. The shape of the result tensor is `[values.shape[0]] + fn(values[0]).shape`.

This method also allows multi-arity `elems` and output of `fn`. If `elems` is a (possibly nested) list or tuple of tensors, then each of these tensors must have a matching first (unpack) dimension. The signature of `fn` may match the structure of `elems`. That is, if `elems` is `(t1, [t2, t3, [t4, t5]])`, then an appropriate signature for `fn` is: `fn = lambda (t1, [t2, t3, [t4, t5]]):`.

Furthermore, `fn` may emit a different structure than its input. For example, `fn` may look like: `fn = lambda t1: return (t1 + 1, t1 - 1)`. In this case, the `dtype` parameter is not optional: `dtype` must be a type or (possibly nested) tuple of types matching the output of `fn`.

To apply a functional operation to the nonzero elements of a SparseTensor one of the following methods is recommended. First, if the function is expressible as TensorFlow ops, use

```
  result = SparseTensor(input.indices, fn(input.values), input.dense_shape)
```

If, however, the function is not expressible as a TensorFlow op, then use

```
result = SparseTensor(
  input.indices, map_fn(fn, input.values), input.dense_shape)
```

instead.

Args:

- `fn`: The callable to be performed. It accepts one argument, which will have the same (possibly nested) structure as `elems`. Its output must have the same structure as `dtype` if one is provided, otherwise it must have the same structure as `elems`.

- `elems`: A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be applied to `fn`.
- `dtype`: (optional) The output type(s) of `fn`. If `fn` returns a structure of Tensors differing from the structure of `elems`, then `dtype` is not optional and must have the same structure as the output of `fn`.
- `parallel_iterations`: (optional) The number of iterations allowed to run in parallel.
- `back_prop`: (optional) True enables support for back propagation.
- `swap_memory`: (optional) True enables GPU-CPU memory swapping.
- `infer_shape`: (optional) False disables tests for consistent output shapes.
- `name`: (optional) Name prefix for the returned tensors.

Returns:

A tensor or (possibly nested) sequence of tensors. Each tensor packs the results of applying `fn` to tensors unpacked from `elems` along the first dimension, from first to last.

Raises:

- `TypeError`: if `fn` is not callable or the structure of the output of `fn` and `dtype` do not match, or if elems is a SparseTensor.
- `ValueError`: if the lengths of the output of `fn` and `dtype` do not match.

Examples:

```
elems = np.array([1, 2, 3, 4, 5, 6])
squares = map_fn(lambda x: x * x, elems)
# squares == [1, 4, 9, 16, 25, 36]

elems = (np.array([1, 2, 3]), np.array([-1, 1, -1]))
alternate = map_fn(lambda x: x[0] * x[1], elems, dtype=tf.int64)
# alternate == [-1, 2, -3]

elems = np.array([1, 2, 3])
alternates = map_fn(lambda x: (x, -x), elems, dtype=(tf.int64, tf.int64))
# alternates[0] == [1, 2, 3]
# alternates[1] == [-1, -2, -3]
```

Last updated November 2, 2017.