The Tensor and Variable classes had slightly different semantics. There are some breaking changes from combining the two classes.

- Indexing a one-dim Tensor returns a zero-dim Tensor instead of a Python number. The zero-dim Tensor shares the storage of the indexed Tensor. For example:

```
>>> x = torch.Tensor([1, 1, 1])
>>> v = x[0]
>>> x += 1
>>> print(v)  # was 1
 2
[torch.FloatTensor of size ()]
```

- The `type()` of a Tensor no longer reflects the data type. Use `isinstance()` or `x.type()` instead:

```
>>> x = torch.DoubleTensor([1, 1, 1])
>>> print(type(x)) # was torch.DoubleTensor
<class 'torch.Tensor'>
>>> print(x.type())  # OK: 'torch.DoubleTensor'
'torch.DoubleTensor'
>>> print(isinstance(x, torch.DoubleTensor))  # OK: True
True
```

- Tensor.copy_() always obeys broadcasting rules. Tensor assignment (`x[idx] = value`) also follows broadcasting rules more closely:

```
>>> x = torch.randn(3, 3)
>>> y = torch.randn(9)
>>> x.copy_(y, broadcast=False) # ERROR
>>> x.copy_(y.view_as(x))  # OK
```

Minor:

- `matmul` , `ones_like` , and `zeros_like` no longer have an `out` parameter.
- Functions that construct tensors based on sizes now interpret empty tuples/sizes differently; previous a empty tuple was treated to mean the size of an empty tensor, now an empty tuple is treated to mean the size of a scalar (0-dimensional tensor).

New Issues:

- `masked_scatter(...)` and `masked_fill(...)` follow in place broadcasting rules.

Possible Issues

- If you have code that accumulates losses across iterations (e.g. to compute an average at the end of an epoch), such as `total_loss += loss` where `loss` is your per-iteration loss, you may find increased memory usage in your program. This is because `loss` probably used to be a Python float (such as when it is the result of `.sum()`), while it is now a zero-dim Tensor. `total_loss` is thus accumulating `Tensor` s and their gradient history, which may keep around large autograd graphs for much longer than

necessary. To fix this, be sure to convert the per-iteration loss to a Python number as soon as possible, for example with `total_loss += float(loss)`.