Testing

October 10, 2019

```
In [1]: import torch
        import torch.nn as nn
        from torch.autograd import Variable
        import torchvision
        import torchvision.transforms as T
        import random

        import numpy as np
        from scipy.ndimage.filters import gaussian_filter1d
        import matplotlib.pyplot as plt
        from cs7643.image_utils import SQUEEZENET_MEAN, SQUEEZENET_STD
        from PIL import Image
```

```
In [2]: class TestModel(nn.Module):
            def __init__(self):
                super(TestModel, self).__init__()
                self.model = nn.Sequential(
                    nn.Linear(20,10),
                    nn.Linear(10,5)
                )

            def forward(self, X):
                return self.model(X)
```

```
In [121]: model = TestModel()
```

```
In [133]: X = torch.ones((2,20), requires_grad=True)
          X_new = X + 2
```

```
In [134]: Y_out = model(X_new)
```

```
In [135]: criterion= torch.nn.MSELoss()
```

```
In [136]: Y = torch.ones((2,5))
          loss = criterion(Y_out, Y)
```

```
In [141]: print(Y.shape)
          print(loss.shape)
```

```
torch.Size([2, 5])
torch.Size([])
```

```
In [126]: # X_new_grad = torch.ones(X_new.shape)
          X_new_grad = []
          X_new.register_hook(lambda x : X_new_grad.append(x))
          loss.backward()
```

```
In [127]: X.grad
```

```
Out[127]: tensor([[ 0.0606,  0.0927,  0.0313,  0.1587, -0.0539, -0.0018,  0.0656,  0.0469,
                  -0.0516, -0.1260, -0.0999, -0.0849,  0.1635,  0.0202,  0.1155,  0.0172,
                   0.0189,  0.0575,  0.0208,  0.0257]])
```

```
In [128]: X_new_grad
```

```
Out[128]: [tensor([[ 0.0606,  0.0927,  0.0313,  0.1587, -0.0539, -0.0018,  0.0656,  0.0469,
                   -0.0516, -0.1260, -0.0999, -0.0849,  0.1635,  0.0202,  0.1155,  0.0172,
                    0.0189,  0.0575,  0.0208,  0.0257]])]
```

```
In [160]: a = torch.rand((3,5,5))
```

```
In [158]: torch.max(a, axis = 0).values
```

```
Out[158]: tensor([[0.3632, 0.7256, 0.7165, 0.7606, 0.5517],
                  [0.9315, 0.6703, 0.2354, 0.9261, 0.5223],
                  [0.9604, 0.6518, 0.9562, 0.8448, 0.9988],
                  [0.9207, 0.6964, 0.8018, 0.3931, 0.8649],
                  [0.8825, 0.7907, 0.3127, 0.8880, 0.5146]])
```

a

```
In [159]: a
```

```
Out[159]: tensor([[[0.1633, 0.3671, 0.3927, 0.4359, 0.2161],
                   [0.4180, 0.6703, 0.0236, 0.8541, 0.3635],
                   [0.4332, 0.3099, 0.9562, 0.6351, 0.4726],
                   [0.3012, 0.3428, 0.7109, 0.3931, 0.6972],
                   [0.3166, 0.7645, 0.0808, 0.8880, 0.4361]],

                  [[0.3632, 0.3626, 0.6399, 0.7606, 0.0436],
                   [0.6425, 0.2768, 0.0130, 0.6042, 0.5223],
                   [0.9604, 0.6518, 0.4344, 0.6643, 0.9988],
                   [0.4677, 0.3080, 0.8018, 0.0757, 0.0256],
                   [0.2254, 0.5474, 0.3127, 0.5026, 0.5146]],

                  [[0.2811, 0.7256, 0.7165, 0.3591, 0.5517],
                   [0.9315, 0.5468, 0.2354, 0.9261, 0.1073],
                   [0.6864, 0.0363, 0.4409, 0.8448, 0.2942],
                   [0.9207, 0.6964, 0.6875, 0.0379, 0.8649],
                   [0.8825, 0.7907, 0.0698, 0.0734, 0.3010]]])
```

```
In [173]: torch.stack((*[i for i in a])).shape


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-173-f6c74b061180> in <module>()
    ----> 1 torch.stack((*[i for i in a])).shape


        TypeError: stack() takes from 1 to 2 positional arguments but 3 were given


In [182]: torch.stack(( *[i for i in a]))


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-182-140eeca0d949> in <module>()
    ----> 1 torch.stack(( *[i for i in a]))


        TypeError: stack() takes from 1 to 2 positional arguments but 3 were given


In [3]: a = torch.Tensor([[1,2,3], [4,5,6]])

In [6]: torch.sqrt(a.sum()**2)

Out[6]: tensor(21.)

In [8]: (a**2).sum()

Out[8]: tensor(91.)

In [11]: if torch.argmax(a) == 5:
             print(1)

1


In [18]: # a = 10

         def update_a(a):
             print(id(a))
             a = np.array([3,2,1])
             print(id(a))
         a = np.array([1,2,3])
         print(id(a))
         update_a(a)
         print(a)
```

```
139864067365744
139864067365744
139864067392496
[1 2 3]
```

```
In [19]: a = 3
         b = a
         b = 5
         print(a)
```

```
3
```

```
In [20]: a = torch.Tensor([1,2,3])
         b = 3
         print(id(a))
         a += b
         print(id(a))
```

```
139864067429504
139864067429504
```

```
In [21]: a = a + b
         print(id(a))
```

```
139864067454584
```

```
In [22]: a = torch.Tensor([1])
         b = a + 1 - 1

In [23]: id(a)

Out[23]: 139864067373816

In [24]: id(b)

Out[24]: 139864067053824

In [27]: a == b

Out[27]: tensor([True])
```

```
In [30]: a = torch.Tensor([1,2,3])

In [31]: a.requires_grad = False

In [32]: a

Out[32]: tensor([1., 2., 3.])

In [33]: b = Variable(a, requires_grad=True)

In [34]: b

Out[34]: tensor([1., 2., 3.], requires_grad=True)

In [35]: b[0] = 3

In [36]: b

Out[36]: tensor([3., 2., 3.], grad_fn=<CopySlices>)

In [37]: a

Out[37]: tensor([3., 2., 3.])

In [39]: a = torch.Tensor([[[2,3,4], [3,4,5]], [[1,2,3], [4,5,6]]])

In [40]: a.shape

Out[40]: torch.Size([2, 2, 3])

In [42]: torch.transpose(a, 1, 2).shape

Out[42]: torch.Size([2, 3, 2])

In [43]: a

Out[43]: tensor([[[2., 3., 4.],
                  [3., 4., 5.]],

                 [[1., 2., 3.],
                  [4., 5., 6.]]])

In [46]: a[:,:,1:]

Out[46]: tensor([[[3., 4.],
                  [4., 5.]],

                 [[2., 3.],
                  [5., 6.]]])
```