TENSOR BASICS

```
import torch
import numpy as np
```

NumPy Arrays to PyTorch Tensors

```
arr = np.array([1,2,3,4,5])
print(arr)
print(arr.dtype)
print(type(arr))
```

```
[1 2 3 4 5]
int64
<class 'numpy.ndarray'>
```

```
x = torch.from_numpy(arr)
# Equivalent to x = torch.as_tensor(arr)
print(x)
# Print the type of data held by the tensor
print(x.dtype)
# Print the tensor object type
print(type(x))
print(x.type()) # this is more specific!
```

```
tensor([1, 2, 3, 4, 5])
torch.int64
<class 'torch.Tensor'>
torch.LongTensor
```

```
arr2 = np.arange(0.,12.).reshape(4,3)
print(arr2)
```

```
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]
 [ 9. 10. 11.]]
```

```
x2 = torch.from_numpy(arr2)
print(x2)
print(x2.type())
```

```
tensor([[ 0.,  1.,  2.],
        [ 3.,  4.,  5.],
        [ 6.,  7.,  8.],
        [ 9., 10., 11.]], dtype=torch.float64)
torch.DoubleTensor
```

Copying vs Sharing

torch.from_numpy(),torch.as_tensor() vs torch.tensor()

```
# Using torch.from_numpy()
arr = np.arange(0,5)
t = torch.from_numpy(arr)
print(t)
```

```
tensor([0, 1, 2, 3, 4])
```

```
arr[2]=77
print(t)
```

```
tensor([ 0,  1, 77,  3,  4])
```

```
# Using torch.tensor()
arr = np.arange(0,5)
t = torch.tensor(arr)
print(t)
```

```
tensor([0, 1, 2, 3, 4])
```

```
arr[2]=77
print(t)
```

```
tensor([0, 1, 2, 3, 4])
```

Class Constructors

torch.Tensor() torch.FloatTensor() torch.LongTensor()

```
data = np.array([1,2,3])
```

```
a = torch.Tensor(data)  # Equivalent to cc = torch.FloatTensor(data)
print(a, a.type())
```

```
tensor([1., 2., 3.]) torch.FloatTensor
```

```
b = torch.tensor(data)
print(b, b.type())
```

```
tensor([1, 2, 3]) torch.LongTensor
```

```
c = torch.tensor(data, dtype=torch.long)
print(c, c.type())
```

```
tensor([1, 2, 3]) torch.LongTensor
```

Creating tensors from scratch

Creating tensors from scratch

Uninitialized tensors with .empty() torch.empty()

```
x = torch.empty(4, 3)
print(x)
```

```
tensor([[3.5763e-36, 0.0000e+00, 2.8026e-45],
        [0.0000e+00, 4.2039e-45, 0.0000e+00],
        [6.3067e-10, 4.3915e-05, 1.3567e-19],
        [4.2482e-05, 1.4588e-19, 6.4892e-07]])
```

Initialized tensors with .zeros() and .ones() torch.zeros(size),torch.ones(size)

```
x = torch.zeros(4, 3, dtype=torch.int64)
print(x)
```

```
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

Tensors from ranges torch.arange(start,end,step), torch.linspace(start,end,steps)

```
x = torch.arange(0,18,2).reshape(3,3)
print(x)
```

```
tensor([[ 0,  2,  4],
        [ 6,  8, 10],
        [12, 14, 16]])
```

```
x = torch.linspace(0,18,12).reshape(3,4)
print(x)
```

```
tensor([[ 0.0000,  1.6364,  3.2727,  4.9091],
        [ 6.5455,  8.1818,  9.8182, 11.4545],
        [13.0909, 14.7273, 16.3636, 18.0000]])
```

Tensors from data torch.tensor()

```
x = torch.tensor([1, 2, 3, 4])
print(x)
print(x.dtype)
print(x.type())
```

```
tensor([1, 2, 3, 4])
torch.int64
```

```
x = torch.FloatTensor([5,6,7])
```

```
x = torch.FloatTensor([5,6,7])
print(x)
print(x.dtype)
print(x.type())
```

```
tensor([5., 6., 7.])
torch.float32
torch.FloatTensor
```

```
x = torch.tensor([8,9,-3], dtype=torch.int)
print(x)
print(x.dtype)
print(x.type())
```

```
tensor([ 8,  9, -3], dtype=torch.int32)
torch.int32
torch.IntTensor
```

Random number tensors torch.rand(size),torch.randn(size),torch.randint(low,high,size)

```
x = torch.rand(4, 3)
print(x)
```

```
tensor([[0.6625, 0.2297, 0.9545],
        [0.6099, 0.5643, 0.0594],
        [0.7099, 0.4250, 0.2709],
        [0.9295, 0.6115, 0.2234]])
```

```
x = torch.randn(4, 3)
print(x)
```

```
tensor([[ 0.7205, -0.1121, -0.0309],
        [-0.1503,  1.8928,  1.3067],
        [-0.0662, -0.4235, -2.3768],
        [ 0.0641, -0.3435,  1.2287]])
```

```
x = torch.randint(0, 5, (4, 3))
print(x)
```

```
tensor([[2, 3, 3],
        [1, 1, 4],
        [1, 4, 4],
        [3, 4, 2]])
```

Random number tensors that follow the input size torch.rand_like(input), torch.randn_like(input), torch.randint_like(input,low,high)

```
x = torch.zeros(2,5)
print(x)
```

```
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]])
```

```
x2 = torch.randn_like(x)
print(x2)
#The same syntax can be used with torch.zeros_like(input),torch.ones_like(input
```

```
tensor([[-1.4594, -0.5025, -1.0614, -0.3485, -0.8923],
        [ 0.9655, -0.5791, -0.8505,  0.0859,  0.1861]])
```

```
x3 = torch.ones_like(x2)
print(x3)
```

```
tensor([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]])
```

Setting the random seed torch.manual_seed(int)

```
torch.manual_seed(42)
x = torch.rand(2, 3)
print(x)
```

```
tensor([[0.8823, 0.9150, 0.3829],
        [0.9593, 0.3904, 0.6009]])
```

```
torch.manual_seed(42)
x = torch.rand(2, 3)
print(x)
```

```
tensor([[0.8823, 0.9150, 0.3829],
        [0.9593, 0.3904, 0.6009]])
```

Tensor attributes

```
x.shape
```

```
torch.Size([2, 3])
```

```
x.size()
```

```
torch.Size([2, 3])
```

```
x.device
```

```
device(type='cpu')
```

```
x.layout
```

```
torch.strided
```

TENSOR OPERATIONS

```
x = torch.arange(6).reshape(3,2)
print(x)
```

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

```
# Grabbing the right hand column values
x[:,1]
```

```
tensor([1, 3, 5])
```

```
# Grabbing the right hand column as a (3,1) slice
x[:,1:]
```

```
tensor([[1],
        [3],
        [5]])
```

Reshape tensors with .view()

```
x = torch.arange(10)
print(x)
```

```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
x.view(2,5)
```

```
tensor([[0, 1, 2, 3, 4],
        [5, 6, 7, 8, 9]])
```

```
x.view(5,2)
```

```
tensor([[0, 1],
        [2, 3],
        [4, 5],
        [6, 7],
        [8, 9]])
```

```
# x is unchanged
x
```

```
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Views reflect the most current data

```
z = x.view(2,5)
x[0]=234
print(z)
```

```
tensor([[234,   1,   2,   3,   4],
        [  5,   6,   7,   8,   9]])
```

Views can infer the correct size

```
x.view(2,-1)
```

```
tensor([[234,   1,   2,   3,   4],
        [  5,   6,   7,   8,   9]])
```

```
x.view(-1,5)
```

```
tensor([[234,   1,   2,   3,   4],
        [  5,   6,   7,   8,   9]])
```

Adopt another tensor's shape with .view_as()

```
x.view_as(z)
```

```
tensor([[234,   1,   2,   3,   4],
        [  5,   6,   7,   8,   9]])
```

Tensor Arithmetic

```
a = torch.tensor([1,2,3], dtype=torch.float)
b = torch.tensor([4,5,6], dtype=torch.float)
print(a + b)
```

```
tensor([5., 7., 9.])
```

```
print(torch.add(a, b))
```

```
tensor([5., 7., 9.])
```

```
result = torch.empty(3)
torch.add(a, b, out=result)  # equivalent to result=torch.add(a,b)
print(result)
```

```
tensor([5., 7., 9.])
```

```
a.add_(b)  # equivalent to a=torch.add(a,b)
print(a)
```

```
tensor([5., 7., 9.])
```

Basic Tensor Operations

Arithmetic OPERATION FUNCTION DESCRIPTION

- a + b a.add(b) element wise addition
- a - b a.sub(b) subtraction
- a * b a.mul(b) multiplication
- a / b a.div(b) division
- a % b a.fmod(b) modulo (remainder after division)
- ab a.pow(b) power

Monomial Operations OPERATION FUNCTION DESCRIPTION

- |a| torch.abs(a) absolute value
- 1/a torch.reciprocal(a) reciprocal
- $\sqrt{a}$ torch.sqrt(a) square root
- log(a) torch.log(a) natural log
- e^a torch.exp(a) exponential
- 12.34 ==> 12. torch.trunc(a) truncated integer
- 12.34 ==> 0.34 torch.frac(a) fractional component

Trigonometry OPERATION FUNCTION DESCRIPTION

- sin(a) torch.sin(a) sine
- cos(a) torch.sin(a) cosine
- tan(a) torch.sin(a) tangent
- arcsin(a) torch.asin(a) arc sine
- arccos(a) torch.acos(a) arc cosine
- arctan(a) torch.atan(a) arc tangent
- sinh(a) torch.sinh(a) hyperbolic sine
- cosh(a) torch.cosh(a) hyperbolic cosine
- tanh(a) torch.tanh(a) hyperbolic tangent

Summary Statistics OPERATION FUNCTION DESCRIPTION

- $\sum a$ torch.sum(a) sum
- $\bar{a}$ torch.mean(a) mean
- amax torch.max(a) maximum
- amin torch.min(a) minimum
- torch.max(a,b) returns a tensor of size a containing the element wise max between a and b

```
a = torch.tensor([1,2,3], dtype=torch.float)
b = torch.tensor([4,5,6], dtype=torch.float)
print(torch.add(a,b).sum())
```

```
tensor(21.)
```

Dot products

```
#torch.dot(a,b) or a.dot(b) or b.dot(a)
a = torch.tensor([1,2,3], dtype=torch.float)
b = torch.tensor([4,5,6], dtype=torch.float)
print(a.mul(b)) # for reference
print()
print(a.dot(b))
```

```
tensor([ 4., 10., 18.])

tensor(32.)
```

Matrix multiplication

```
#torch.mm(a,b) or a.mm(b) or a @ b
a = torch.tensor([[0,2,4],[1,3,5]], dtype=torch.float)
b = torch.tensor([[6,7],[8,9],[10,11]], dtype=torch.float)

print('a: ',a.size())
print('b: ',b.size())
print('a x b: ',torch.mm(a,b).size())
```

```
a:  torch.Size([2, 3])
b:  torch.Size([3, 2])
a x b:  torch.Size([2, 2])
```

```
print(torch.mm(a,b))
```

```
tensor([[56., 62.],
        [80., 89.]])
```

```
print(a.mm(b))
```

```
tensor([[56., 62.],
        [80., 89.]])
```

```
print(a @ b)
```

```
tensor([[56., 62.],
        [80., 89.]])
```

Matrix multiplication with broadcasting

```
#torch.matmul(a,b) or a.matmul(b) or a @ b
t1 = torch.randn(2, 3, 4)
t2 = torch.randn(4, 5)

print(torch.matmul(t1, t2).size())
```

```
torch.Size([2, 3, 5])
```

```
print(torch.mm(t1, t2).size())
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-143-edaac219da2b> in <module>()
----> 1 print(torch.mm(t1, t2).size())

RuntimeError: matrices expected, got 3D, 2D tensors at
/pytorch/aten/src/TH/generic/THTensorMath.cpp:36
```

SEARCH STACK OVERFLOW

L2 or Euclidian Norm torch.norm()

The Euclidian Norm gives the vector norm of x where x=(x1,x2,...,xn) . It is calculated as

||x|| base 2:=√x^2 base1+⋯+x^2 base$n$ When applied to a matrix, torch.norm() returns the Frobenius norm by default.

```
x = torch.tensor([2.,5.,8.,14.])
x.norm()
```

Number of elements torch.numel()

```
x = torch.ones(3,7)
x.numel()
```

```
##Mean Squared Error:
def mse(t1, t2):
    diff = t1 - t2
    return torch.sum(diff * diff) / diff.numel()
```