Tensors in Pytorch

Eunhui Kim/KISTI

https://fleuret.org/ee559/

Tensors in Pytorch

목차

1) Pytorch?

- PyTorch is Torch's THNN computational backend 를 기반으로 한 Python library 임.

- PyTorch 주요 특징:
  - CPU/GPU위에서 효율적인 tensor 연산들 지원,
  - 자동 미분 지원 (autograd),
  - optimizers,
  - data 입출력.

- 효율적인 tensor 연산들은 표준 linear algebra 연산들을 포함함.
  deep-learning 특화된 연산들 (convolution, pooling, etc.)을 또한 포함.

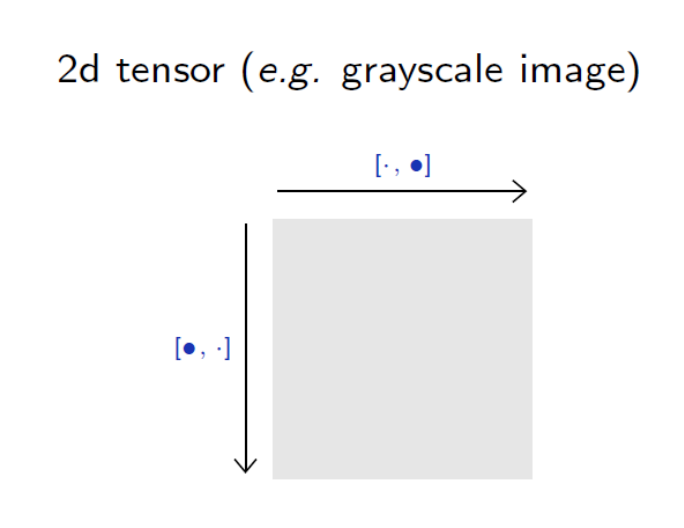
- PyTorch의 핵심이 되는 주요한 특징은 어떤 연산에 대해서도 미분을 지원하는 autograd 임!

- 이는 다음 수업시간에 다뤄짐

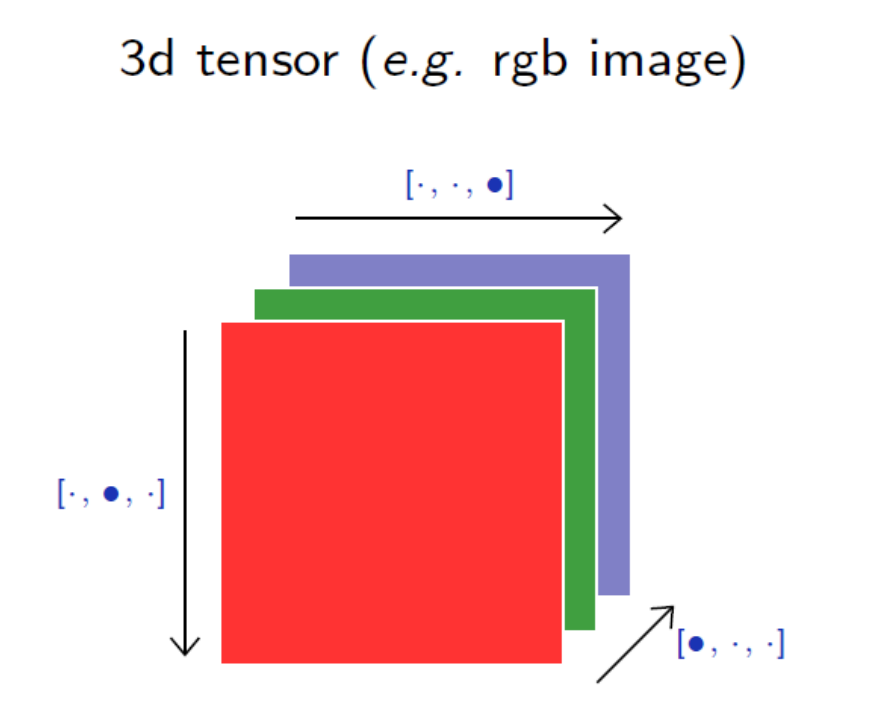
2) Tensor?

- tensor는 범용 matrix로 몇몇 이산 차원을 따라 색인된 산술 값을 갖는 table임
  - 0d tensor는 scalar(상수),
  - 1d tensor는 vector (e.g. a sound sample),
  - 2d tensor는 matrix (e.g. a grayscale image),
  - 3d tensor는 동일 크기의 matrix로 구성된 vector (e.g. a multi-channel image),
  - 4d tensor는 동일 크기의 matrix로 구성된 matrix, 혹은
    일련의 3d tensors sequence(e.g. a sequence of multi-channel images)
- Tensors는 처리할 신호를 encoding하는데 사용되지만, 모델의 내부 상태 및 매개 변수도 encoding합니다.
- **이 제한된 구조를 통해 데이터를 조작하면 CPU와 GPU를 최고 성능으로 사용할 수 있습니다.**
- 복합 데이터 구조는 더 다양한 데이터 유형을 나타낼 수 있습니다.

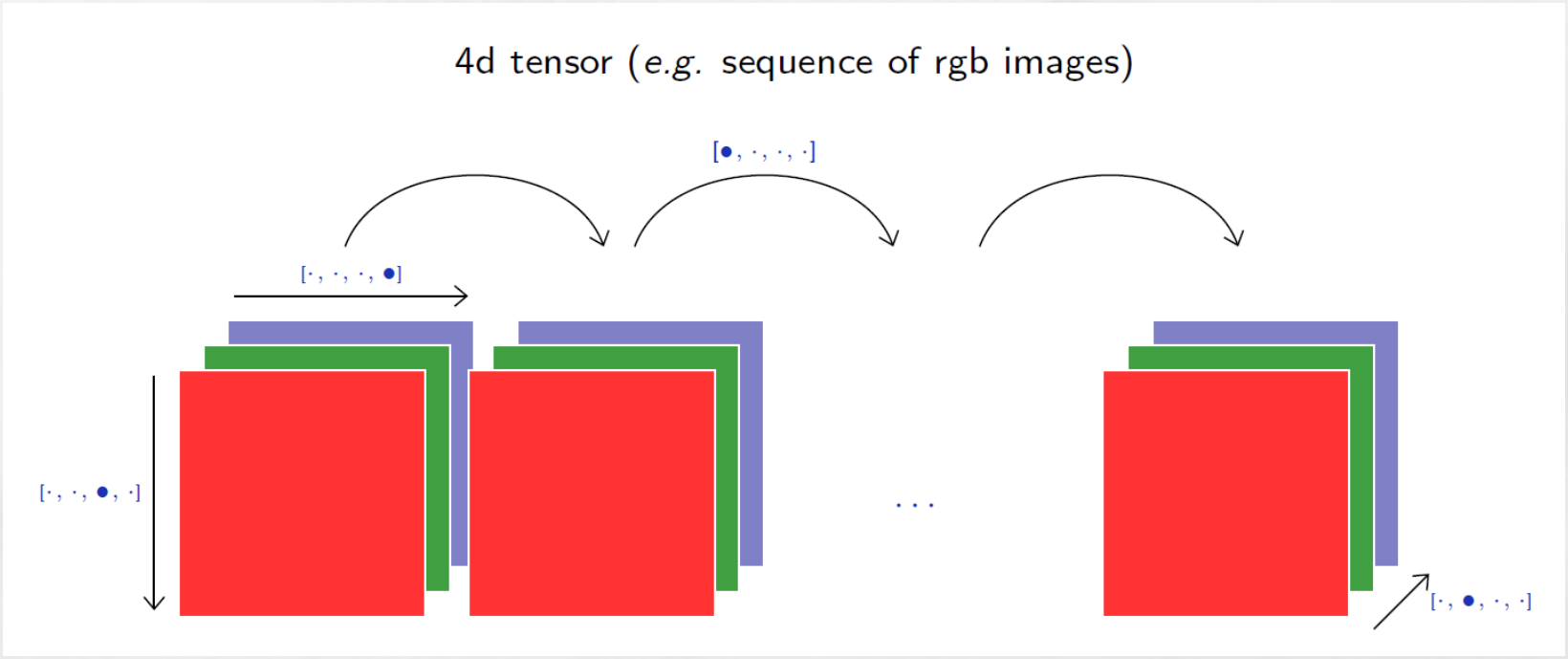
2) Tensor?



2) Tensor?



2) Tensor?



3) Tensor Types & Devices

- Tensor types

  - torch.float16, torch.float32, torch.float64,
  - torch.uint8,
  - torch.int8, torch.int16, torch.int32, torch.int64

- CPU의 메모리 혹은 GPU의 memory에 위치함

- 특정 장치의 메모리에 저장된 tensor 연산은 해당 장치에서 수행됨.

- 이는 Optimization 단원에서 다시 설명이 이뤄질 예정임.

3) Tensor types & device

In [2]:
```
1 import torch
2 x = torch.zeros(1, 3)
3 x.dtype, x.device
```

Out[2]:
```
(torch.float32, device(type='cpu'))
```

In [4]:
```
1 x = x.long()
2 x.dtype, x.device
```

Out[4]:
```
(torch.int64, device(type='cpu'))
```

In [5]:
```
1 x = x.to('cuda')
2 x.dtype, x.device
```

Out[5]:
```
(torch.int64, device(type='cuda', index=0))
```

4) Tensor Scalar

```
>>> x = torch.empty(2, 5)
>>> x.size()
torch.Size([2, 5])
>>> x.fill_(1.125)
tensor([[ 1.1250,  1.1250,  1.1250,  1.1250,  1.1250],
        [ 1.1250,  1.1250,  1.1250,  1.1250,  1.1250]])
>>> x.mean()
tensor(1.1250)
>>> x.std()
tensor(0.)
>>> x.sum()
tensor(11.2500)
>>> x.sum().item()
11.25
```

- In-place 연산들은 밑줄이 붙고, 0d tensor는 item()을 사용하여 Python scalar로 전환될 수 있음.
- 계수(coefficient)를 읽으면 0d tensor를 생성함.

```
>>> x = torch.tensor([[11., 12., 13.], [21., 22., 23.]])
>>> x[1, 2]
tensor(23.)
```

5) Tensor Vector/Matrix Operations

- PyTorch는 구성 요소 및 vector/matrix 연산을 위한 연산자를 제공함.

```
>>> x = torch.tensor([ 10., 20., 30.])
>>> y = torch.tensor([ 11., 21., 31.])
>>> x + y
tensor([ 21.,  41.,  61.])
>>> x * y
tensor([ 110.,  420.,  930.])
>>> x**2
tensor([ 100.,  400.,  900.])
>>> m = torch.tensor([[ 0., 0., 3. ],
...                   [ 0., 2., 0. ],
...                   [ 1., 0., 0. ]])
>>> m.mv(x)
tensor([ 90.,  40.,  10.])
>>> m @ x
tensor([ 90.,  40.,  10.])
```

6) Tensor Indexing, Slicing

- 그리고 numpy에서와 같이 : symbol은 index값의 범위를 정의하고, tensor 를 슬라이스 함.

```
>>> import torch
>>> x = torch.empty(2, 4).random_(10)
>>> x
tensor([[8., 1., 1., 3.],
        [7., 0., 7., 5.]])
>>> x[0]
tensor([8., 1., 1., 3.])
>>> x[0, :]
tensor([8., 1., 1., 3.])
>>> x[:, 0]
tensor([8., 7.])
>>> x[:, 1:3] = -1
>>> x
tensor([[ 8., -1., -1.,  3.],
        [ 7., -1., -1.,  5.]])
```

7) Tensor Standard Operations

다음은 방대한 Tensor 연산 library의 몇 가지 예:

- Creation
  - torch.empty(*size, ...)
  - torch.zeros(*size, ...)
  - torch.full(size, value, ...)
  - torch.tensor(sequence, ...)
  - torch.eye(n, ...)
  - torch.from numpy(ndarray)

- Indexing, Slicing, Joining, Mutating torch.empty(*size, ...)
  - torch.Tensor.view(*size)
  - torch.cat(inputs, dimension=0)
  - torch.chunk(tensor, chunks, dim=0)[source]
  - torch.split(tensor, split size, dim=0)[source]
  - torch.index select(input, dim, index, out=None)
  - torch.t(input, out=None)
  - torch.transpose(input, dim0, dim1, out=None)

7) Tensor Standard Operations

다음은 방대한 Tensor 연산 library의 몇 가지 예:

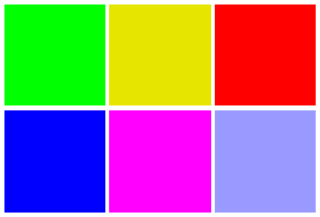
- Filling
  - Tensor.fill (value)
  - torch.bernoulli (proba)
  - torch.normal ([mu, [std]])
- Pointwise math
  - torch.abs(input, out=None)
  - torch.add()
  - torch.cos(input, out=None)
  - torch.sigmoid(input, out=None)
  - (+ many operators)
- Math reduction
  - torch.dist(input, other, p=2, out=None)
  - torch.mean()
  - torch.norm()
  - torch.std()
  - torch.sum()

7) Tensor Standard Operations

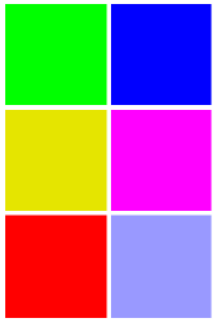
다음은 방대한 Tensor 연산 library의 몇 가지 예:

- BLAS and LAPACK Operations
  - torch.eig(a, eigenvectors=False, out=None)
  - ~~torch.gels(B, A, out=None)~~
  - torch.lstsq(B, A, out=None)
  - torch.inverse(input, out=None)
  - torch.mm(mat1, mat2, out=None)
  - torch.mv(mat, vec, out=None)

7) Tensor Standard Operations

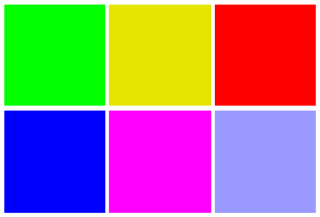


```
x = torch.tensor([ [ 1, 3, 0 ],
                   [ 2, 4, 6 ] ])
```



x.t()

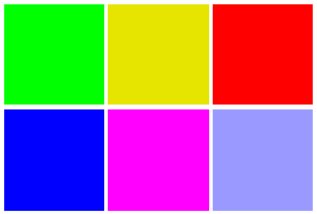
7) Tensor Standard Operations



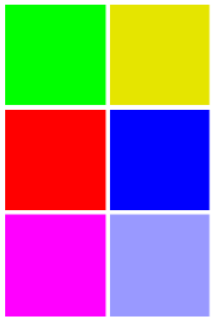
x.view(-1)

```
x = torch.tensor([ [ 1, 3, 0 ],
                   [ 2, 4, 6 ] ])
```

7) Tensor Standard Operations

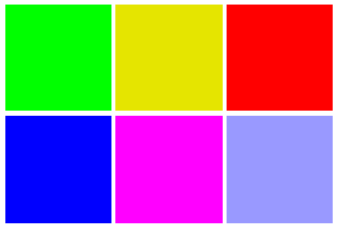


```
x = torch.tensor([ [ 1, 3, 0 ],
                   [ 2, 4, 6 ] ])
```

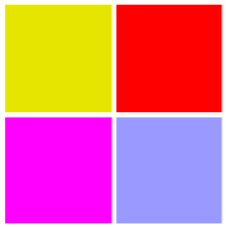


```
x.view(3, -1)
```

7) Tensor Standard Operations



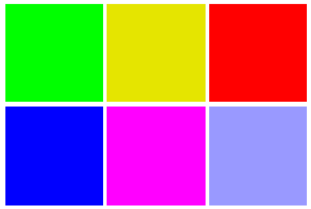
```
x = torch.tensor([ [ 1, 3, 0 ],
                   [ 2, 4, 6 ] ])
```



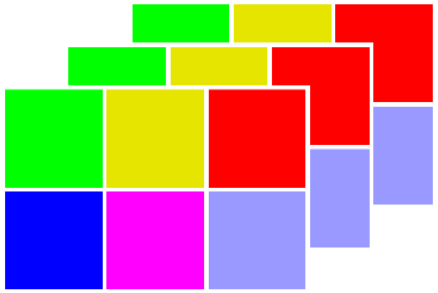
```
x.narrow(1, 1, 2)
```

`torch.narrow(`*input*, *dim*, *start*, *length*`)` → Tensor

7) Tensor Standard Operations



```
x = torch.tensor([ [ 1, 3, 0 ],
                   [ 2, 4, 6 ] ])
```

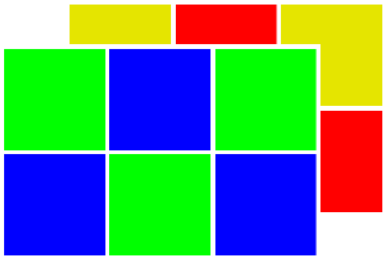


```
x.view(1, 2, 3).expand(3, 2, 3)
```

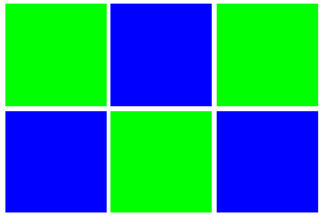
expand(**sizes*) → Tensor

***sizes** (*torch.Size or int...*) – the desired expanded size

7) Tensor Standard Operations

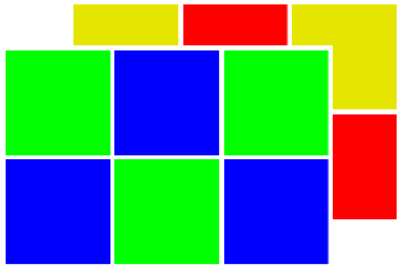


```
x = torch.tensor([ [ [ 1, 2, 1 ],
                     [ 2, 1, 2 ] ],
                   [ [ 3, 0, 3 ],
                     [ 0, 3, 0 ] ] ])
```

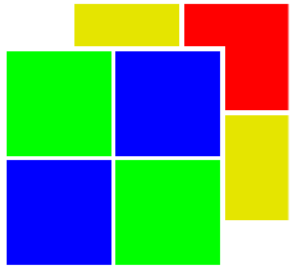


```
x.narrow(0, 0, 1)
```

7) Tensor Standard Operations

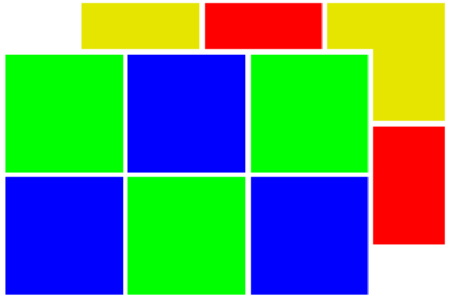


```
x = torch.tensor([ [ [ 1, 2, 1 ],
                     [ 2, 1, 2 ] ],
                   [ [ 3, 0, 3 ],
                     [ 0, 3, 0 ] ] ])
```

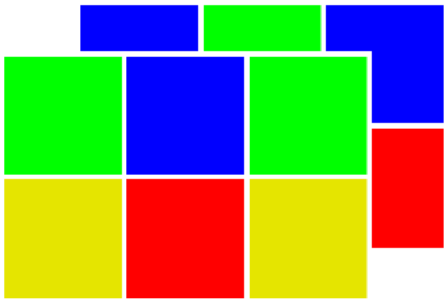


```
x.narrow(2, 0, 2)
```

7) Tensor Standard Operations



```
x = torch.tensor([ [ [ 1, 2, 1 ],
                     [ 2, 1, 2 ] ],
                   [ [ 3, 0, 3 ],
                     [ 0, 3, 0 ] ] ])
```



x.transpose(0, 1)

`torch.transpose(`*input*, *dim0*, *dim1*`)` → Tensor

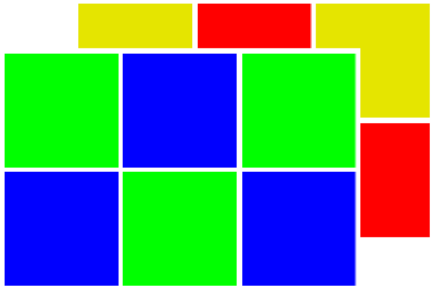
Returns a tensor that is a transposed version of `input`.

The given dimensions `dim0` and `dim1` are swapped.

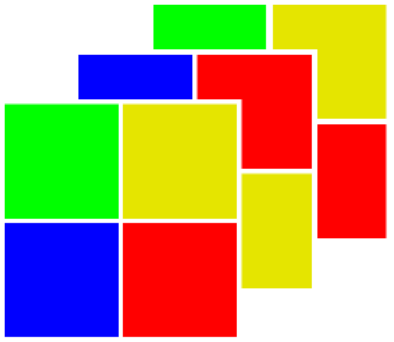
```
x = torch.tensor([[[1, 2, 1],
                   [2, 1, 2]],
                  [[3, 0, 3],
                   [0, 3, 0]]])
y = x.transpose(0, 1)
y
```

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

        [[2, 1, 2],
         [0, 3, 0]]])
```

7) Tensor Standard Operations



```
x = torch.tensor([ [ [ 1, 2, 1 ],
                     [ 2, 1, 2 ] ],
                   [ [ 3, 0, 3 ],
                     [ 0, 3, 0 ] ] ])
```

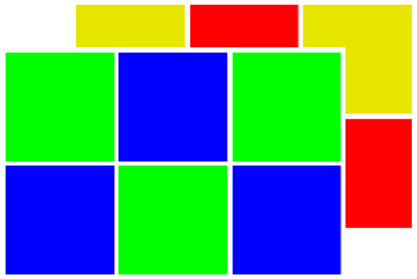


x.transpose(0, 2)

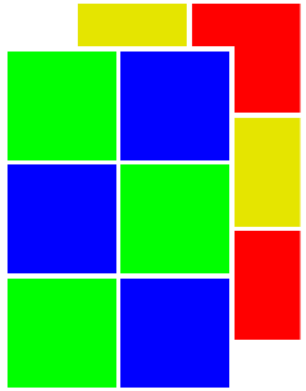
```
z = x.transpose(0, 2)
z
```

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

        [[2, 0],
         [1, 3]],

        [[1, 3],
         [2, 0]]])
```

7) Tensor Standard Operations



```
x = torch.tensor([ [ [ 1, 2, 1 ],
                     [ 2, 1, 2 ] ],
                   [ [ 3, 0, 3 ],
                     [ 0, 3, 0 ] ] ])
```



```
x.transpose(1, 2)
```

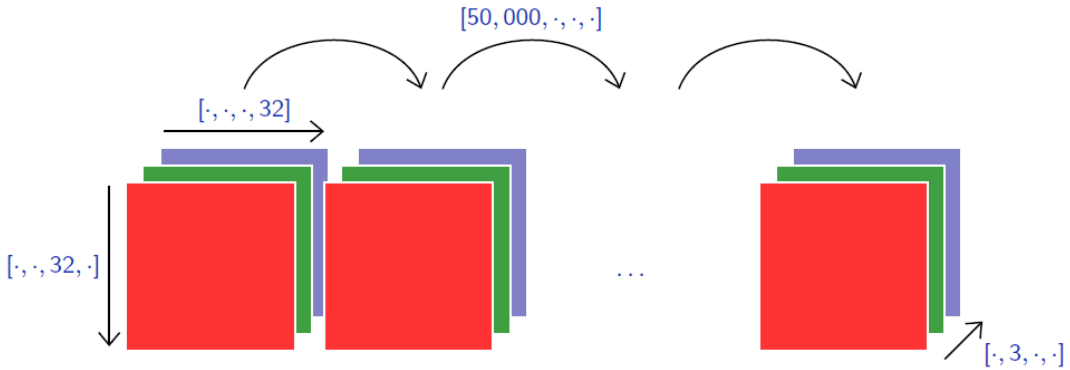
8) Pytorch interface to image database

- PyTorch 표준 image database에 대한 interface를 지원함.

```
import torch, torchvision
cifar = torchvision.datasets.CIFAR10('./cifar10/', train = True, download = True)
x = torch.from_numpy(cifar.train_data).transpose(1, 3).transpose(2, 3).float()
x = x / 255
print(x.type(), x.size(), x.min().item(), x.max().item())
```

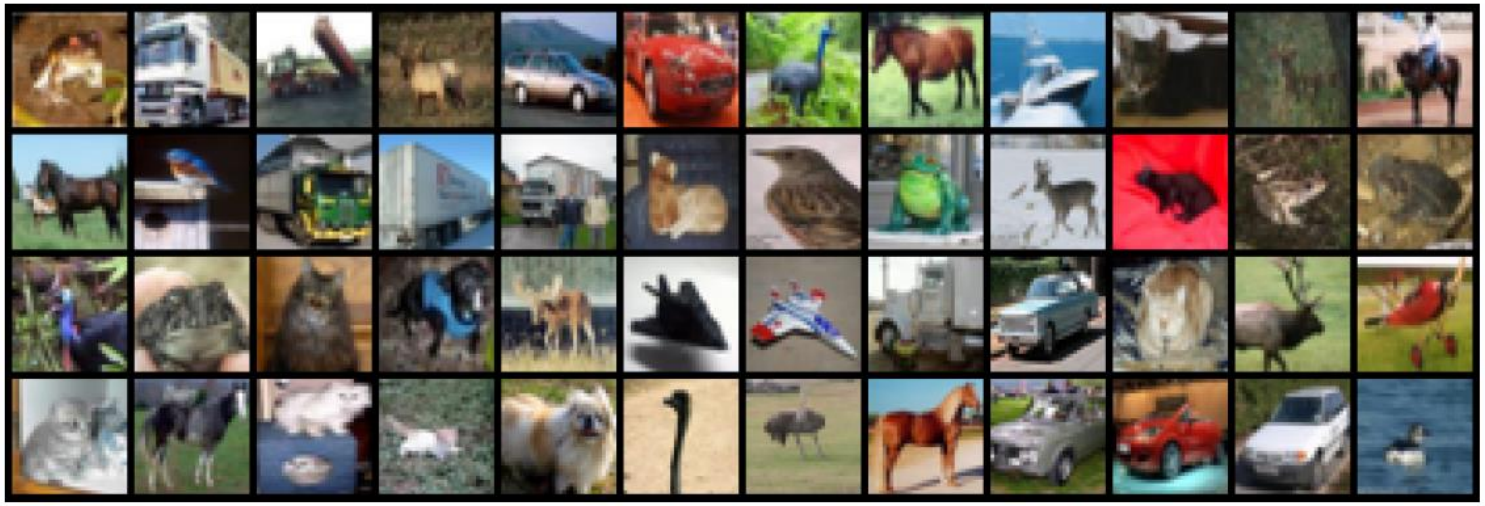
prints

```
Files already downloaded and verified
torch.FloatTensor torch.Size([50000, 3, 32, 32]) 0.0 1.0
```



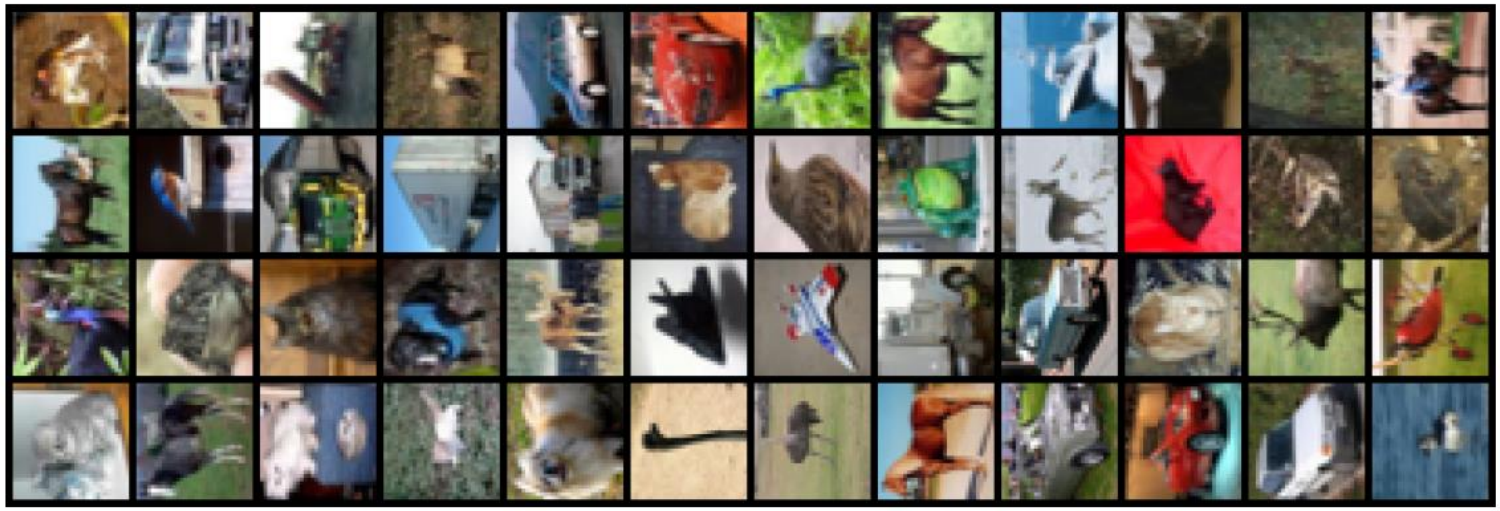
8) Pytorch interface to image database

```
# Narrows to the first images, converts to float
x = x.narrow(0, 0, 48).float()

# Saves these samples as a single image
torchvision.utils.save_image(x, 'cifar-4x12.png', nrow = 12)
```



8) Pytorch interface to image database

```
# Switches the row and column indexes
x.transpose_(2, 3)
torchvision.utils.save_image(x, 'cifar-4x12-rotated.png', nrow = 12)
```



8) Pytorch interface to image database

```
# Kills the green and blue channels
x.narrow(1, 1, 2).fill_(0)
torchvision.utils.save_image(x, 'cifar-4x12-rotated-and-red.png', nrow = 12)
```

9) BroadCasting

- Broadcasting은 "직관적으로 합리적인" 작업을 수행해야 할 때, 계수를 복제하여 자동으로 자원을 확장하는 연산.

For instance:

```
>>> x = torch.empty(100, 4).normal_(2)
>>> x.mean(0)
tensor([2.0476, 2.0133, 1.9109, 1.8588])
>>> x -= x.mean(0) # This should not work!
>>> x.mean(0)
tensor([-4.0531e-08, -4.4703e-07, -1.3471e-07,  3.5763e-09])
```

9) BroadCasting

- 정확하게, broadcasting은 다음과 같이 진행됨:

1. Tensor중 하나의 차원이 다른 tensor보다 적으면, 전면에 필요한 만큼 크기 1의 치수를 추가하여 모양을 변경함;

그 이후,

2. 모든 차원 불일치에 대해, 두 Tensor중 하나의 크기가 1이면 계수를 복제하여 축을 따라 확장함.

차원 중 한 차원의 tensor 크기가 일치하지 않고, 두 차원 중 1인 차원이 없는 경우는 Broadcasting 작업이 실패함.

9) BroadCasting

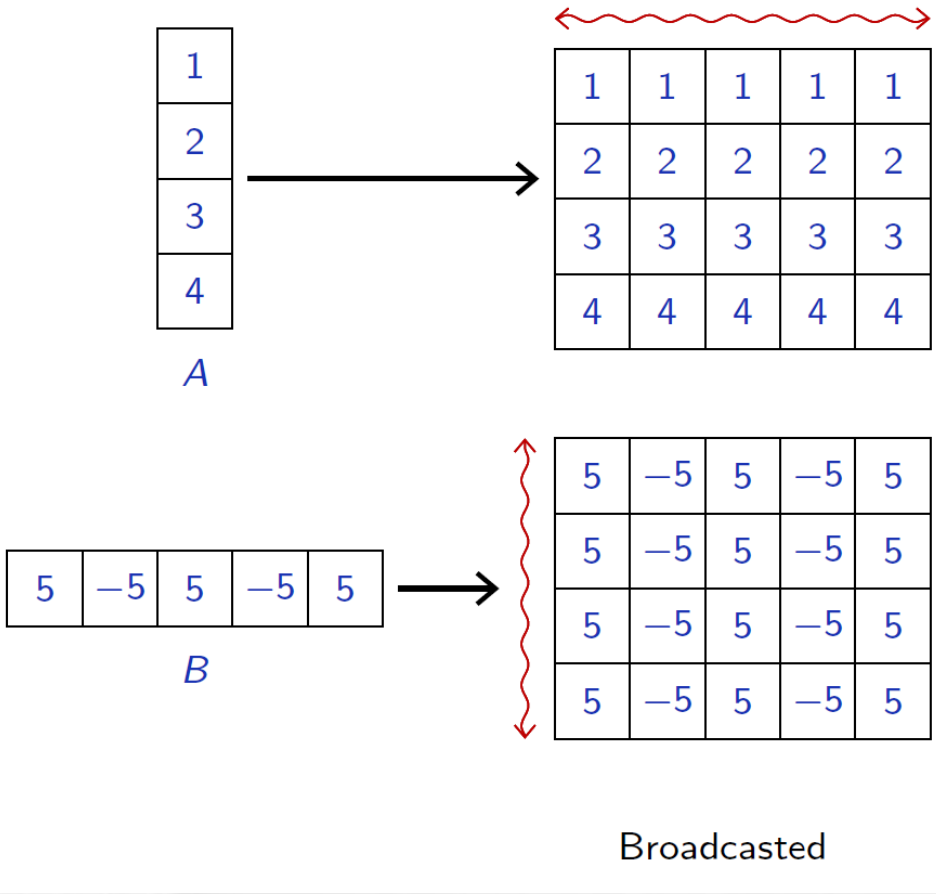
```
A = torch.tensor([[1.], [2.], [3.], [4.]])
B = torch.tensor([[5., -5., 5., -5., 5.]])
C = A + B
```

A

B

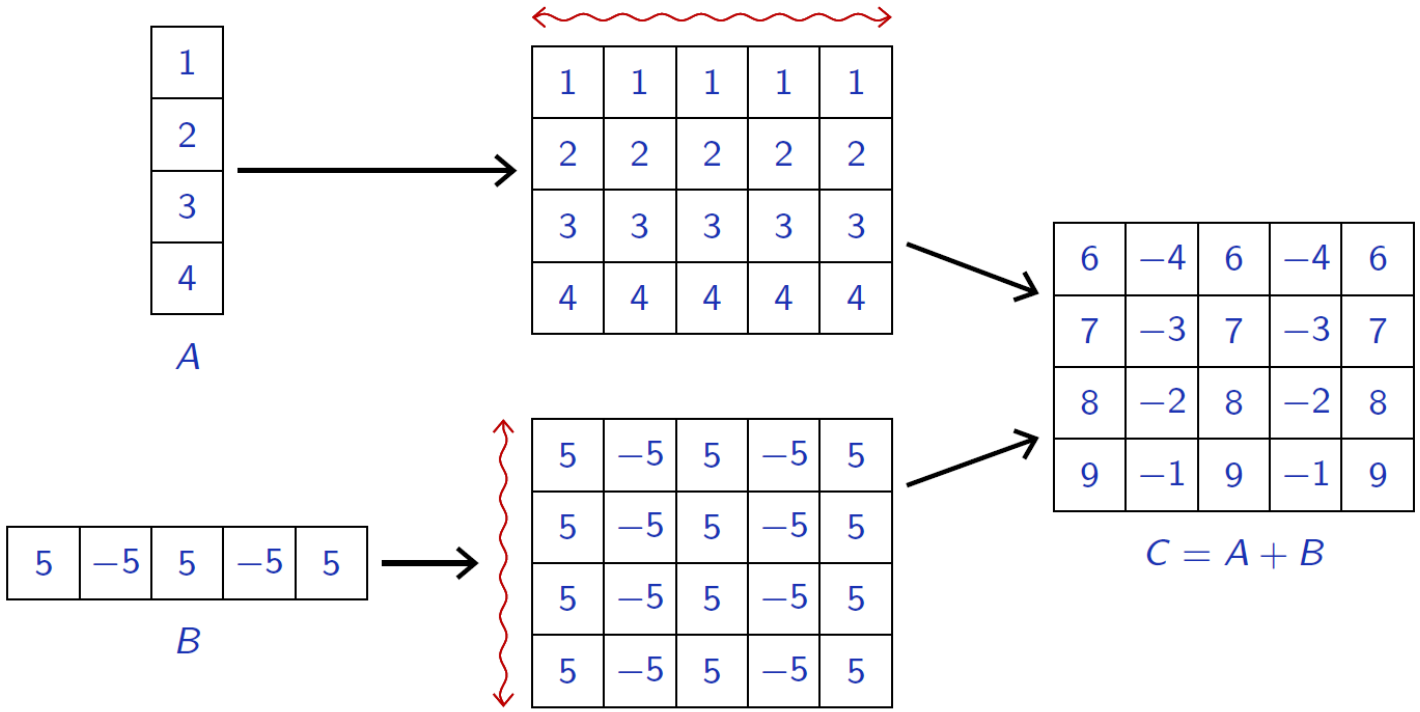
9) BroadCasting

```
A = torch.tensor([[1.], [2.], [3.], [4.]])
B = torch.tensor([[5., -5., 5., -5., 5.]])
C = A + B
```



9) BroadCasting

```
A = torch.tensor([[1.], [2.], [3.], [4.]])
B = torch.tensor([[5., -5., 5., -5., 5.]])
C = A + B
```



10) Tensor Internals

- Tensor는 저 수준 1-d vector로 일련의 저장소로 간주할 수 있음.

```
>>> x = torch.zeros(2, 4)
>>> x.storage()
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
[torch.FloatStorage of size 8]
>>> q = x.storage()
>>> q[4] = 1.0
>>> x
tensor([[ 0.,  0.,  0.,  0.],
        [ 1.,  0.,  0.,  0.]])
```

10) Tensor Internals

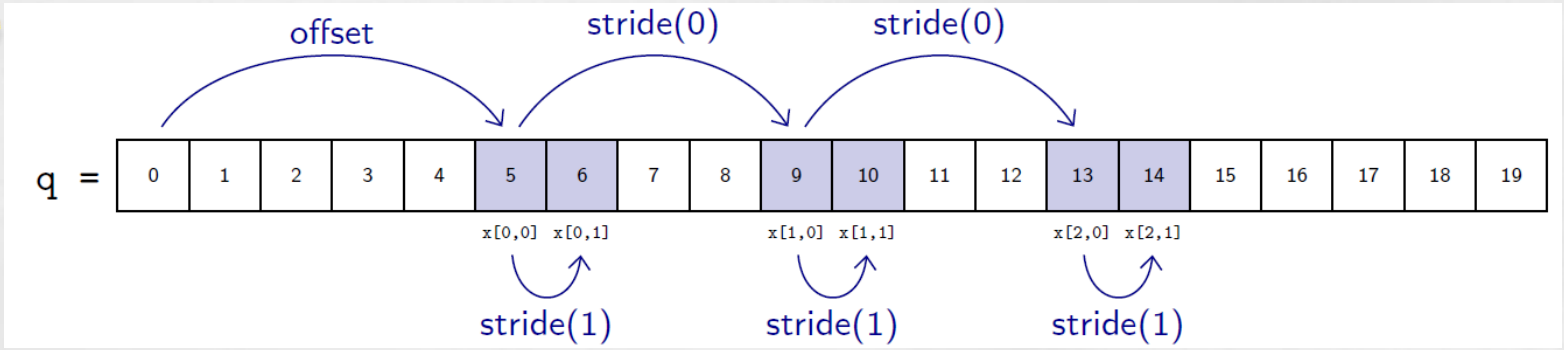
- Multiple tensors는 같은 저장소를 공유 가능함.
  view(), expand() or transpose()와 같은 연산을 사용할 경우 공유 가능.

```
>>> y = x.view(2, 2, 2)
>>> y
tensor([[[ 0.,  0.],
         [ 0.,  0.]],

        [[ 1.,  0.],
         [ 0.,  0.]]])
>>> y[1, 1, 0] = 7.0
>>> x
tensor([[ 0.,  0.,  0.,  0.],
        [ 1.,  0.,  7.,  0.]])
>>> y.narrow(0, 1, 1).fill_(3.0)
tensor([[[ 3.,  3.],
         [ 3.,  3.]]])
>>> x
tensor([[ 0.,  0.,  0.,  0.],
        [ 3.,  3.,  3.,  3.]])
```

10) Tensor Internals

- Tensor의 첫 번째 계수는 storage()의 storage_offset()에 있는 계수임.
- index k 를 1씩 증가시키려면, storage에서 stride(k) 요소로 이동해야 함.

```
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
tensor([[  5.,   6.],
        [  9.,  10.],
        [ 13.,  14.]])
```



10) Tensor Internals

- 동일한 storage에 대해 명시적으로 서로 다른 “view”를 생성 가능함

```
>>> n = torch.linspace(1, 4, 4)
>>> n
tensor([ 1.,  2.,  3.,  4.])
>>> torch.tensor(0.).set_(n.storage(), 1, (3, 3), (0, 1))
tensor([[ 2.,  3.,  4.],
        [ 2.,  3.,  4.],
        [ 2.,  3.,  4.]])
>>> torch.tensor(0.).set_(n.storage(), 1, (2, 4), (1, 0))
tensor([[ 2.,  2.,  2.,  2.],
        [ 3.,  3.,  3.,  3.]])
```

- 이는 특히 transpositions 과broadcasting이 구현되는 방법임.

10) Tensor Internals

- 아래 예시는 (아마도 놀라운) error를 설명함

```
>>> x = torch.empty(100, 100)
>>> x.stride()
(100, 1)
>>> y = x.t()
>>> y.stride()
(1, 100)
>>> y.view(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: invalid argument 2: view size is not compatible with
input tensor's size and stride (at least one dimension spans across
two contiguous subspaces).
```

- x.t() 는 x's storage를 공유하고 1d로 flattened될 수 없음.
- 이것은 Tensor의 연속 버전을 반환하는 contiguous()로 고칠 수 있으며, 필요한 경우 복사본을 만들거나 view()와 contiguous()를 결합하는 reshape()으로 해결 가능함.

10) Pytorch 실습 on Google CoLab

- Linear Regression

지점들로 구성된 dataset에 대해, 이 dataset을 표현하는 "best line"을 찾을 수 있음.

$$(x_n, y_n) \in \mathbb{R} \times \mathbb{R}, \quad n = 1, \ldots, N,$$

예로) 지점들을 통과하는 선을 통해 mean squared error를 최소화 할 수 있음.

$$f(x; a, b) = ax + b$$

이 모델은 새로운 입력 x에 대해 관련된 f(x;a,b)를 계산하여 y예측이 가능함.

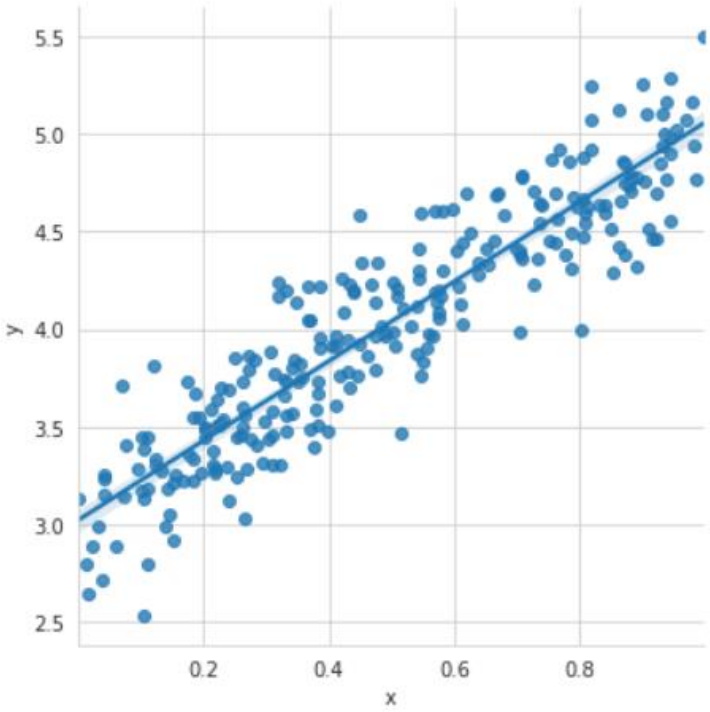
$$\underset{a,b}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} (\underbrace{ax_n + b}_{f(x_n;a,b)} - y_n)^2.$$

10) Pytorch 실습 on Google CoLab

In [19]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import seaborn as sns
import pandas as pd
%matplotlib inline

sns.set_style(style = 'whitegrid')
plt.rcParams["patch.force_edgecolor"] = True

m = 2 # slope
c = 3 # interceptm = 2 # slope
c = 3 # intercept

x = np.random.rand(256)

noise = np.random.randn(256) / 4

y = x * m + c + noise

df = pd.DataFrame()
df['x'] = x
df['y'] = y

sns.lmplot(x ='x', y ='y', data = df)
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x7f9cfd52b910>



10) Pytorch 실습 on Google CoLab

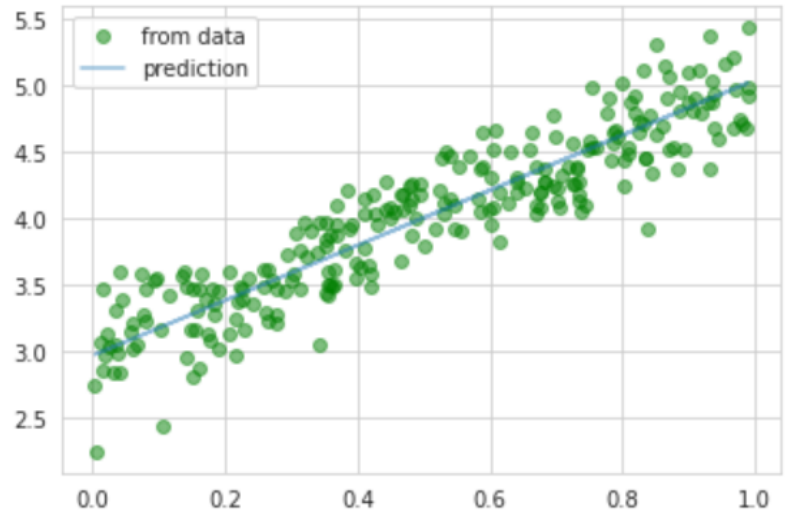
```python
import torch
import torch.nn as nn
from torch.autograd import Variable

x_train = x.reshape(-1, 1).astype('float32')
y_train = y.reshape(-1, 1).astype('float32')

class LinearRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        out = self.linear(x)
        return out

input_dim = x_train.shape[1]
output_dim = y_train.shape[1]
print('input_d, output_d  = {}, {}'.format(input_dim, output_dim))

#model setting
model = LinearRegressionModel(input_dim, output_dim)
criterion = nn.MSELoss()

#optimizer setting
l_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr = l_rate)
epochs = 2000
```

```python
#training the model
for epoch in range(epochs):
    epoch += 1

    inputs = Variable(torch.from_numpy(x_train))
    labels = Variable(torch.from_numpy(y_train))

    #clear grads
    optimizer.zero_grad()

    #forward to get predicted values
    outputs = model.forward(inputs)
    loss = criterion(outputs, labels)
    loss.backward() #back props
    optimizer.step() #update the parameters
    #print('epoch {}, loss {}'.format(epoch, loss.data))

predicted = model.forward(Variable(torch.from_numpy(x_train))).data.numpy()

plt.plot(x_train, y_train, 'go', label='from data', alpha = .5)
plt.plot(x_train, predicted, label='prediction', alpha = .5)
plt.legend()
plt.show()
print(model.state_dict())
```

10) Pytorch 실습 on Google CoLab

```
46
47 predicted = model.forward(Variable(torch.from_numpy(x_train))).data.numpy()
48
49 plt.plot(x_train, y_train, 'go', label='from data', alpha = .5)
50 plt.plot(x_train, predicted, label='prediction', alpha = .5)
51 plt.legend()
52 plt.show()
53 print(model.state_dict())
54
```

```
input_d, output_d  = 1, 1
```




```
OrderedDict([('linear.weight', tensor([[2.0803]])), ('linear.bias', tensor([2.9609]))])
```

감사합니다.