```
import torch

P = 1000
N = 9
M = 3
s = 0.01

m = torch.randn( N+1, M)    # rand normal

x = torch.rand( P, N+1)     # rand unifom
x[:,-1] = 1
z = torch.mm( x, m)     # matrix multiply

w = torch.randn( N+1, M, requires_grad=True)
                        # w va a tener difen_
                        # ciacion automatica

xn = x + s*torch.randn( P, N+1) # agrego ruido

lr = 1e-4
E, t = 1., 0
while E>1e-3 and t<999:
    y = torch.mm( xn, w)
    error = (y-z).pow(2).sum()
    error.backward()
    with torch.no_grad():
        w -= lr*w.grad      # grad es miembro
        w.grad.zero_()      # _ al final = inplace
    E = error.item()/P      # item del tensor
    t += 1
    if t%100==0:
        print(t,E)

print( (torch.mm(x,w)-z).pow(2).mean().item() )

print(m)
print(w)
```

Ver también:

```
size()         (simil numpy.shape)
dim()
nelement()     (simil numpy.size)

from_numpy() / .numpy()
```

```
import torch

P = 100
N = 8
H = N+1
M = 1

x = torch.randn( P, N).sign()
z = torch.prod( x, dim=1).view(P,1)

w1 = torch.randn( N+1, H, requires_grad=True)
w2 = torch.randn( H+1, M, requires_grad=True)

bias = torch.ones( P, 1)

lr = 1e-2
t, e = 0, 1.
while e>0.01 and t<9999:
    h = torch.cat( (x,bias), dim=1).mm(w1).tanh()
    y = torch.cat( (h,bias), dim=1).mm(w2).tanh()
    error = (y-z).pow(2).sum()
    error.backward()
    with torch.no_grad():
        w1 -= lr*w1.grad
        w2 -= lr*w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
    e = error.item()/P
    t += 1
    if t%100==0:
        print(t,e)
```

```
import torch

P = 100
N = 8
H = N+1
M = 1

x = torch.randn( P, N).sign()
z = torch.prod( x, dim=1).view(P,1)

w1 = torch.randn( N+1, H, requires_grad=True)
w2 = torch.randn( H+1, M, requires_grad=True)

bias = torch.ones( P, 1)

### MINI-BATCH ###
bs = 10
lr = 1e-2
t, e = 0, 1.
while e>0.01 and t<9999:
    e = 0.0
    rp = torch.randperm(P)
    for mb in range( 0, P, bs):
        i = rp[mb:mb+bs]
        h = torch.cat( (x[i],bias[i]),
                       dim=1).mm(w1).tanh()
        y = torch.cat( (h, bias[i]),
                       dim=1).mm(w2).tanh()
        error = (y-z[i]).pow(2).sum()
        error.backward()
        with torch.no_grad():
            w1 -= lr*w1.grad
            w2 -= lr*w2.grad
            w1.grad.zero_()
            w2.grad.zero_()
        e += error.item()
    e /= P
    t += 1
    if t%100==0:
        print(t,e)
```