```python
import torch
import torchvision as tv

trn_data = tv.datasets.MNIST(                       # Entrenamiento.
       root='./data', train=True, download=True,
       transform=tv.transforms.ToTensor() )
tst_data = tv.datasets.MNIST(                       # Validacion.
       root='./data', train=False, download=True,
       transform=tv.transforms.ToTensor() )

B = 100
trn_load = torch.utils.data.DataLoader(
       dataset=trn_data, batch_size=B, shuffle=True)
tst_load = torch.utils.data.DataLoader(
       dataset=tst_data, batch_size=B, shuffle=False)

P = len(trn_data)                    # Cant de instancias.
N = trn_data[0][0].nelement()        # Cant de entradas.
C = 10                               # Cant de clases de salida.

model = mlp( N, 128, C)              # El modelo que hicimos antes.
costf = torch.nn.MSELoss()
optim = torch.optim.Adam( model.parameters(), lr=1e-3)

t, E = 0, 1.
model.train()
while E>=0.001 and t<99:             # Mas datos, menos epocas.
    e = []
    for images, labels in trn_load:         # Itera en mini-batches.
        optim.zero_grad()
        x = images.reshape(-1,N)                    # -1 es auto.
        z = torch.zeros( size=(len(labels),C))      # labels es un nro.
        z[torch.arange(len(labels)),labels] = 1     # z sera un tensor.
        y = model( x)
        error = costf( y, z)
        error.backward()
        optim.step()
        e.append( error.item())
    E = sum(e)/len(e)                       # Promedio entre lotes.
    t += 1
    print( "Epoch: {},  Loss: {:.4f}".format( t, E))
```

```python
class MLP( torch.nn.Module):
    def __init__( _, sizes):
        assert( isinstance( sizes, list) or isinstance( sizes, tuple))
        assert( len(sizes)>1)
        super().__init__()
        _.layers = torch.nn.ModuleList()    # List con params incluidos.
        for i in range(len(sizes)-1):
            _.layers.append( torch.nn.Linear( sizes[i], sizes[i+1]))

    def forward( _, x):
        h = x
        for hidden in _.layers[:-1]:
            h = torch.sigmoid( hidden( h))  # Sigmoid en lugar de tanh.
        output = _.layers[-1]
        #y = torch.softmax( output( h), dim=1)     # Con MSE.
        y = output( h)                             # Con CrossEntropy.
        return y

model = MLP( [N, 256, 128, C])
costf = torch.nn.CrossEntropyLoss()         # Mejor para clasificacion.

#-#-#-#-#
       error = costf( y, labels)            # No hace falta armar z.
#-#-#-#-#

model.eval()
right = 0
total = 0
with torch.no_grad():
    for images, labels in tst_load:         # Porcentaje de aciertos.
        x = images.reshape( -1, N)
        y = model( x)
        right += (y.argmax( dim=1) == labels).sum().item()
        total += len(labels)
print( "Accuracy", right/total)

#-#-#-#-#

for Wi in model.layers[0].weight            # Alternativa al matshow ;-)
    rf = Wi.view(28,28)
    print('\n'.join([ ''.join(['_' if v<rf.mean() else '#' for v in row])
                                                    for row in rf ]))
    print('-'*28)
```