In [ ]:

```
%load_ext autoreload
%autoreload 2

import numpy as np
from mytorch import nn as mynn
from mytorch.optim import SGD
from torch import nn
import torch
from d2l import torch as d2l
```

# MSE Testing

In [ ]:

```
# set up synthetic data
N = 10
num_inputs = 7
num_outputs = 3

# numpy/our versions
W = np.random.rand(num_inputs, num_outputs)
b = np.random.rand(num_outputs, 1)
X = np.random.randn(N, num_inputs)
Y = X @ W + np.outer(np.ones(N), b) + 0.5 * np.random.randn(N, num_outputs)

# converted torch versions
Xt = torch.tensor(X).float()
Wt = torch.tensor(W).float()
bt = torch.tensor(b).float()
Yt = torch.tensor(Y).float()
```

In [ ]:

```
# initialize model and fix weights to true values
my_net = mynn.Linear(num_inputs, num_outputs)
my_net.W = W
my_net.b = b

# initialize torch model, loss, optimizer
net = nn.Linear(num_inputs, num_outputs)
net.weight = nn.Parameter(Wt.T)
net.bias = nn.Parameter(bt[:, 0])
torch_out = net(Xt)
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, momentum=0.0)
```

## Test `forward()`

In [ ]:

```
# torch loss function
torch_mse_fn = nn.MSELoss()
torch_mse = torch_mse_fn(torch_out, Yt)

# mytorch loss function
my_mse_fn = mynn.MSELoss()
my_mse = my_mse_fn.forward(torch_out.detach().numpy(), Y)
```

```
print('Torch MSE:', torch_mse.data)
print('My MSE:', my_mse, '\n')
```

```
Torch MSE: tensor(0.2180)
My MSE: 0.21803364618155888
```

## Test `backward()`

In [ ]:

```
# MSE
optimizer.zero_grad()
torch_out = net(Xt)
torch_mse = torch_mse_fn(torch_out, Yt)
torch_mse.backward(retain_graph=True)
torch_dLdW = net.weight.grad.data
torch_dLdb = net.bias.grad.data

dLdZ = my_mse_fn.backward()
my_net.forward(X)
my_net.backward(dLdZ)
my_dLdW = my_net.dLdW
my_dLdb = my_net.dLdb

print('MyTorch dLdW:\n', my_dLdW, '\n')
print('PyTorch dLdW:\n', torch_dLdW.T, '\n')
print('MyTorch dLdb:\n', my_dLdb, '\n')
print('PyTorch dLdb:\n', torch_dLdb, '\n')

print('Difference in dLdW:', np.linalg.norm(my_dLdW.T - torch_dLdW.data.numpy()))
print('Difference in dLdb:', np.linalg.norm(my_dLdb.flatten() - torch_dLdb.data.num
MyTorch dLdW:
 [[ 0.15629022 -0.01670823 -0.01395194]
 [ 0.02062998 -0.02817953 -0.14405865]
 [-0.06217844 -0.01217057  0.05367357]
 [ 0.09557618  0.01814454 -0.02136826]
 [ 0.03588844 -0.06888509 -0.01345359]
 [ 0.0222143  -0.0438693  -0.08048498]
 [ 0.12853376 -0.09517782  0.00418429]]

PyTorch dLdW:
 tensor([[ 0.1563, -0.0167, -0.0140],
        [ 0.0206, -0.0282, -0.1441],
        [-0.0622, -0.0122,  0.0537],
        [ 0.0956,  0.0181, -0.0214],
        [ 0.0359, -0.0689, -0.0135],
        [ 0.0222, -0.0439, -0.0805],
        [ 0.1285, -0.0952,  0.0042]])

MyTorch dLdb:
 [-0.03687998  0.13018967 -0.13604935]

PyTorch dLdb:
 tensor([-0.0369,  0.1302, -0.1360])

Difference in dLdW: 7.343191299264743e-08
Difference in dLdb: 1.4674534402403805e-08
```

# CE Testing

In [ ]:

```
# set up synthetic data
N = 10
num_inputs = 7
num_outputs = 3

# numpy/our versions
W = np.random.rand(num_inputs, num_outputs)
b = np.random.rand(num_outputs, 1)
# generate random one-hot matrix
x = np.eye(num_outputs)
x[np.random.choice(x.shape[0], size=N)]
Y = np.eye(num_outputs)[np.random.choice(num_outputs, N)]

# converted torch versions
Xt = torch.tensor(X).float()
Wt = torch.tensor(W).float()
bt = torch.tensor(b).float()
Yt = torch.tensor(Y).float()
```

In [ ]:

```
# initialize model and fix weights to true values
my_net = mynn.Linear(num_inputs, num_outputs)
my_net.W = W
my_net.b = b

# initialize torch model, loss, optimizer
```

```
net = nn.Linear(num_inputs, num_outputs)
net.weight = nn.Parameter(Wt.T)
net.bias = nn.Parameter(bt[:, 0])
torch_out = net(Xt)
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, momentum=0.0)
```

# Test `forward()`

In [ ]:

```
# torch loss functions
torch_ce_fn = nn.CrossEntropyLoss()
torch_ce = torch_ce_fn(torch_out, Yt)

# mytorch loss functions
my_ce_fn = mynn.CrossEntropyLoss()
my_ce = my_ce_fn.forward(torch_out.detach().numpy(), Y)

print('Torch CE:', torch_ce.data)
print('My CE:', my_ce, '\n')
```

```
Torch CE: tensor(1.6076)
My CE: 1.4981077154179694
```

# Test `backward()`

In [ ]:

```
optimizer.zero_grad()
torch_out = net(Xt)
torch_ce = torch_ce_fn(torch_out, Yt)
torch_ce.backward(retain_graph=True)
torch_dLdW = net.weight.grad.data
torch_dLdb = net.bias.grad.data

dLdZ = my_ce_fn.backward()
my_net.forward(X)
my_net.backward(dLdZ)
my_dLdW = my_net.dLdW
my_dLdb = my_net.dLdb

print('MyTorch dLdW:\n', my_dLdW, '\n')
print('PyTorch dLdW:\n', torch_dLdW.T, '\n')
print('MyTorch dLdb:\n', my_dLdb, '\n')
print('PyTorch dLdb:\n', torch_dLdb, '\n')

print('Difference in dLdW:', np.linalg.norm(my_dLdW.T - torch_dLdW.data.numpy()))
print('Difference in dLdb:', np.linalg.norm(my_dLdb.flatten() - torch_dLdb.data.num
MyTorch dLdW:
 [[-2.42025283 -0.74964799 -4.02664113]
 [ 1.271734   -2.08355641 -1.50988379]
 [-3.78258475  0.56260216 -0.22064712]
 [-0.12173315  0.01676124 -2.289436  ]
 [ 0.62239384 -0.81948011  0.28667337]
 [ 2.05611344  0.6249941   1.40096314]
 [ 0.54078871  0.71594616 -1.41667321]]

PyTorch dLdW:
 tensor([[ 0.1990, -0.1686, -0.0303],
        [-0.2816, -0.2838,  0.5654],
        [-0.0604,  0.1042, -0.0438],
        [ 0.0676,  0.1027, -0.1702],
        [ 0.0580, -0.1197,  0.0617],
        [-0.0905, -0.0410,  0.1315],
        [-0.0730, -0.3267,  0.3996]])

MyTorch dLdb:
 [-0.68258679 -1.88425163  1.19849675]

PyTorch dLdb:
 tensor([ 0.0650,  0.0972, -0.1622])

Difference in dLdW: 8.002632024295615
Difference in dLdb: 2.517237004940137
```