Assignment 9

# Original Results from DL20A.py

## Output

Text

Description automatically generated

## Code

# DL20A.py CS5173/6073 cheng 2023  
# 2D LeNet5 on MNIST  
# following d2l 7.6.1  
# Usage: python DL20A.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
X2 = torch.reshape(X, (len(X), 1, 28, 28))  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyConv2d(6, kernel\_size=5, padding=2), nn.Sigmoid(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.LazyConv2d(16, kernel\_size=5), nn.Sigmoid(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.Flatten(),  
 nn.LazyLinear(120), nn.Sigmoid(),  
 nn.LazyLinear(84), nn.Sigmoid(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X2[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X2)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
X2test = torch.reshape(Xtest, (len(Xtest), 1, 28, 28))  
o = model(X2test)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# Original Results from DL20P.py

## Output

Text

Description automatically generated

## Code

# DL20P.py CS5173/6073 cheng 2023  
# MLP on MNIST as vectors  
# Usage: python DL20P.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyLinear(1024), nn.ReLU(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
o = model(Xtest)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# DL20A.py using nn.ReLU

## Output

Text

Description automatically generated

## Code

# DL20A.py CS5173/6073 cheng 2023  
# 2D LeNet5 on MNIST  
# following d2l 7.6.1  
# Usage: python DL20A.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
X2 = torch.reshape(X, (len(X), 1, 28, 28))  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyConv2d(6, kernel\_size=5, padding=2), nn.ReLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.LazyConv2d(16, kernel\_size=5), nn.ReLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.Flatten(),  
 nn.LazyLinear(120), nn.ReLU(),  
 nn.LazyLinear(84), nn.ReLU(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X2[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X2)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
X2test = torch.reshape(Xtest, (len(Xtest), 1, 28, 28))  
o = model(X2test)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# DL20A.py using nn.PReLU

## Output

Text

Description automatically generated

## Code

# DL20A.py CS5173/6073 cheng 2023  
# 2D LeNet5 on MNIST  
# following d2l 7.6.1  
# Usage: python DL20A.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
X2 = torch.reshape(X, (len(X), 1, 28, 28))  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyConv2d(6, kernel\_size=5, padding=2), nn.PReLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.LazyConv2d(16, kernel\_size=5), nn.PReLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.Flatten(),  
 nn.LazyLinear(120), nn.PReLU(),  
 nn.LazyLinear(84), nn.PReLU(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X2[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X2)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
X2test = torch.reshape(Xtest, (len(Xtest), 1, 28, 28))  
o = model(X2test)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# DL20A.py using nn.LeakyReLU

## Output

Text

Description automatically generated

## Code

# DL20A.py CS5173/6073 cheng 2023  
# 2D LeNet5 on MNIST  
# following d2l 7.6.1  
# Usage: python DL20A.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
X2 = torch.reshape(X, (len(X), 1, 28, 28))  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyConv2d(6, kernel\_size=5, padding=2), nn.LeakyReLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.LazyConv2d(16, kernel\_size=5), nn.LeakyReLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.Flatten(),  
 nn.LazyLinear(120), nn.LeakyeLU(),  
 nn.LazyLinear(84), nn.LeakyReLU(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X2[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X2)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
X2test = torch.reshape(Xtest, (len(Xtest), 1, 28, 28))  
o = model(X2test)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# DL20A.py using nn.SiLU

## Output

A screenshot of a computer

Description automatically generated with medium confidence

## Code

# DL20A.py CS5173/6073 cheng 2023  
# 2D LeNet5 on MNIST  
# following d2l 7.6.1  
# Usage: python DL20A.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
X2 = torch.reshape(X, (len(X), 1, 28, 28))  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyConv2d(6, kernel\_size=5, padding=2), nn.SiLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.LazyConv2d(16, kernel\_size=5), nn.SiLU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.Flatten(),  
 nn.LazyLinear(120), nn.SiLU(),  
 nn.LazyLinear(84), nn.SiLU(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X2[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X2)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
X2test = torch.reshape(Xtest, (len(Xtest), 1, 28, 28))  
o = model(X2test)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# DL20A.py using nn.GELU

## Output

Text

Description automatically generated

## Code

# DL20A.py CS5173/6073 cheng 2023  
# 2D LeNet5 on MNIST  
# following d2l 7.6.1  
# Usage: python DL20A.py  
  
import torch  
import torchvision  
import torch.nn as nn  
import numpy as np  
import random  
import time  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = len(mnist)  
x = []  
targets = []  
for i in range(num\_samples):  
 x.append(list(mnist[i][0].getdata()))  
 targets.append(mnist[i][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
X2 = torch.reshape(X, (len(X), 1, 28, 28))  
y = torch.tensor(targets, dtype=torch.long)  
  
model = nn.Sequential(  
 nn.LazyConv2d(6, kernel\_size=5, padding=2), nn.GELU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.LazyConv2d(16, kernel\_size=5), nn.GELU(),  
 nn.AvgPool2d(kernel\_size=2),  
 nn.Flatten(),  
 nn.LazyLinear(120), nn.GELU(),  
 nn.LazyLinear(84), nn.GELU(),  
 nn.LazyLinear(10))  
  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 512  
rounds = 1000  
indices = list(range(num\_samples))  
t1 = time.process\_time()  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X2[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 if i % 100 == 0:  
 print(i, loss.item())  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
t2 = time.process\_time()  
print('Training time', t2 - t1)  
  
print(loss.item())  
o = model(X2)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('Training misclassified =', misclassified.item(), 'out of', num\_samples)  
  
testset = torchvision.datasets.MNIST('/data/', train=False)  
num\_test = len(testset)  
testImg = []  
testTgt = []  
for i in range(num\_test):  
 testImg.append(list(testset[i][0].getdata()))  
 testTgt.append(testset[i][1])  
Xtest = torch.tensor(testImg, dtype=torch.float32)  
ytest = torch.tensor(testTgt, dtype=torch.long)  
X2test = torch.reshape(Xtest, (len(Xtest), 1, 28, 28))  
o = model(X2test)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != ytest))  
print('Test misclassified =', misclassified.item(), 'out of', num\_test)

# Comments

## How do ReLU and other modern substitutes to Sigmoid improve the performance of LeNet5?

As you can see above, ReLU and other modern substitutes to sigmoid significantly improves both the training and testing accuracy of LeNet5.

## Which one has the best training accuracy, and which one has the best test accuracy, based on the misclassification numbers?

Based on the misclassification of numbers, DL20P.py (simple MLP) has the best training accuracy (117 misclassified out of 60000) and LeakyReLU has the test accuracy (8p misclassified out of 10000).

## What are the important improvements of these versions of LetNet5 over the simple MLP?

Some of the important improvements of these versions of LeNet5 over the simple MLP is that it allows for the sharing of parameters, uses convolution layers, and non-linear activation functions. This means that the number of parameters to be trained are reduced, the spatial relationship between pixels ins captured unlike simple MLP, and introducing non-linearity into the model captures complex data better.