# LAB 3-1 : The choice of weight initialization in training

## construct the LeNet-5

class Net(nn.Module):

def \_\_init\_\_(self, mode):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 10, 5)

self.conv2 = nn.Conv2d(10, 20, 5)

self.fc1 = nn.Linear(320, 50)

self.fc2 = nn.Linear(50, 10)

self.pool = nn.MaxPool2d(2, 2)

self.dropout = nn.Dropout(0.5)

self.relu = nn.ReLU()

def forward(self, x):

# input is 28x28x1

# conv1(kernel=5, filters=10) 28x28x10 -> 24x24x10

# max\_pool(kernel=2) 24x24x10 -> 12x12x10

x = self.relu(self.pool(self.conv1(x)))

# conv2(kernel=5, filters=20) 12x12x20 -> 8x8x20

# max\_pool(kernel=2) 8x8x20 -> 4x4x20

x = self.relu(self.pool(self.dropout(self.conv2(x))))

# flatten 4x4x20 = 320

x = x.view(-1, 320)

# 320 -> 50

x = self.relu(self.fc1(x))

x = self.dropout(x)

# 50 -> 10

x = self.relu(self.fc2(x))

return x

## use MNIST as dataset

#The transform function for train data

transform\_train = transforms.transforms.Compose([

transforms.ToTensor(), # first, convert image to PyTorch tensor

transforms.Normalize((0.1307,), (0.3081,)) # normalize inputs

])

#The transform function for test data

transform\_test = transforms.Compose([

transforms.ToTensor(), # first, convert image to PyTorch tensor

transforms.Normalize((0.1307,), (0.3081,)) # normalize inputs

])

path = '../data/mnist\_data'

trainset = torchvision.datasets.MNIST(root=path, train=True,

download=True, transform=transform\_train)

testset = torchvision.datasets.MNIST(root=path, train=False,

download=True, transform=transform\_test)

## **Solution**

5 kinds of weight initialization methods:

1. All weights are zero
2. Uniform distribution (-1, 1)
3. Normal distribution, mean = 0 & standard deviation = 0.1
4. Xavier initialization
5. Kaiming initialization

### Initialize the value layer by layer

if(mode == 'constant'):

print('constant mode')

with torch.no\_grad():

nn.init.constant\_(self.conv1.weight, 0)

nn.init.constant\_(self.conv2.weight, 0)

nn.init.constant\_(self.fc1.weight, 0)

nn.init.constant\_(self.fc2.weight, 0)

elif(mode == 'uniform'):

print('uniform mode')

with torch.no\_grad():

nn.init.uniform\_(self.conv1.weight, -0.1, 0.1)

nn.init.uniform\_(self.conv2.weight, -0.1, 0.1)

nn.init.uniform\_(self.fc1.weight, -0.1, 0.1)

nn.init.uniform\_(self.fc2.weight, -0.1, 0.1)

elif(mode == 'normal'):

print('normal mode')

with torch.no\_grad():

nn.init.normal\_(self.conv1.weight, 0, 0.2)

nn.init.normal\_(self.conv2.weight, 0, 0.1)

nn.init.normal\_(self.fc1.weight, 0, 0.1)

nn.init.normal\_(self.fc2.weight, 0, 0.1)

elif(mode == 'xavier'):

print('xavier mode')

with torch.no\_grad():

nn.init.xavier\_normal\_(self.conv1.weight, gain=nn.init.calculate\_gain('relu'))

nn.init.xavier\_normal\_(self.conv2.weight, gain=nn.init.calculate\_gain('relu'))

nn.init.xavier\_normal\_(self.fc1.weight, gain=nn.init.calculate\_gain('relu'))

nn.init.xavier\_normal\_(self.fc2.weight, gain=nn.init.calculate\_gain('relu'))

elif(mode == 'kaiming'):

print('kaiming mode')

self.relu = nn.PReLU()

with torch.no\_grad():

nn.init.kaiming\_normal\_(self.conv1.weight, a=0.6,

mode='fan\_in', nonlinearity='relu')

nn.init.kaiming\_normal\_(self.conv2.weight, a=0.2,

mode='fan\_in', nonlinearity='relu')

nn.init.kaiming\_normal\_(self.fc1.weight, a=0.06,

mode='fan\_in', nonlinearity='relu')

nn.init.kaiming\_normal\_(self.fc2.weight, a=0.06,

mode='fan\_in', nonlinearity='relu')

else:

print('original mode')

### **Using net.apply**

def init\_weights\_constant(m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

nn.init.constant\_(m.weight, 0)

def init\_weights\_uniform(m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

nn.init.uniform\_(m.weight, -0.1, 0.1)

def init\_weights\_normal(m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

nn.init.normal\_(m.weight, 0, 0.01)

def init\_weights\_xavier(m):

if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):

nn.init.xavier\_normal\_(m.weight, gain=nn.init.calculate\_gain('relu'))

def init\_weights\_kaiming(m):

if isinstance(m, nn.Conv2d):

nn.init.kaiming\_normal\_(m.weight, a=0.5, mode='fan\_out', nonlinearity='relu')

if isinstance(m, nn.Linear):

nn.init.kaiming\_normal\_(m.weight, a=0.1, mode='fan\_out', nonlinearity='relu')

def init\_weights\_original(m):

Return

def construct\_net(mode):

#declare a new model

net = Net()

if(mode == 'constant'):

print('constant mode')

init\_func = init\_weights\_constant

elif(mode == 'uniform'):

print('uniform mode')

init\_func = init\_weights\_uniform

elif(mode == 'normal'):

print('normal mode')

init\_func = init\_weights\_normal

elif(mode == 'xavier'):

print('xavier mode')

init\_func = init\_weights\_xavier

elif(mode == 'kaiming'):

print('kaiming mode')

init\_func = init\_weights\_kaiming

else:

print('original mode')

init\_func = init\_weights\_original

net.apply(init\_func)

if device == 'cuda':

net = net.cuda(0)

else:

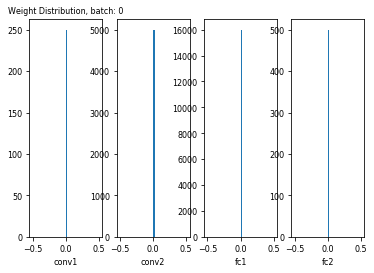
net = net.cpu()

return net

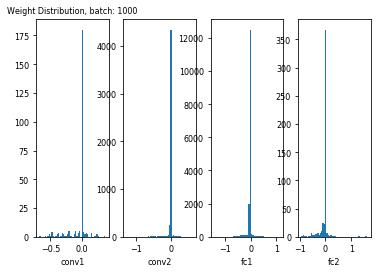
## **The** result

### constant **mod**e

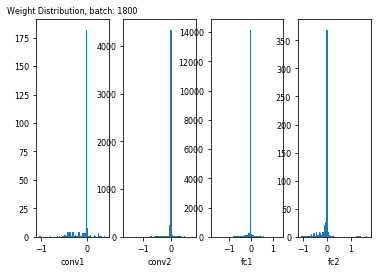
Weight distribution: Initial



Weight distribution: batch 1000

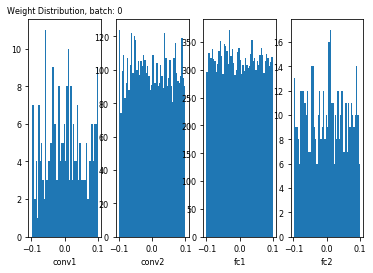


Weight distribution: batch 1800

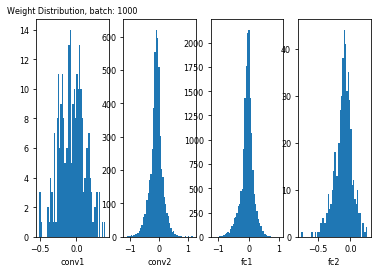


### **uniform** mode

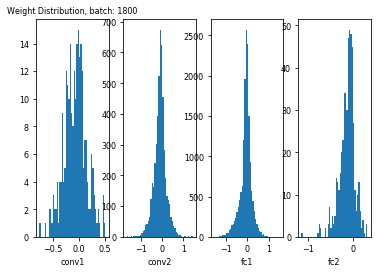
Weight distribution: Initial



Weight distribution: batch 1000

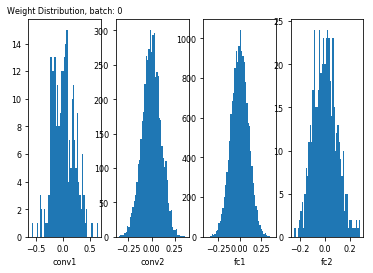


Weight distribution: batch 1800

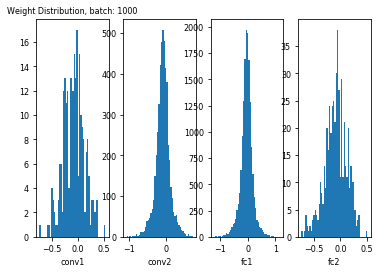


### normal mode

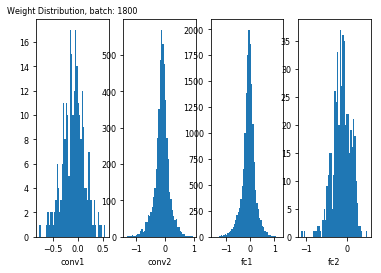
Weight distribution: Initial



Weight distribution: batch 1000

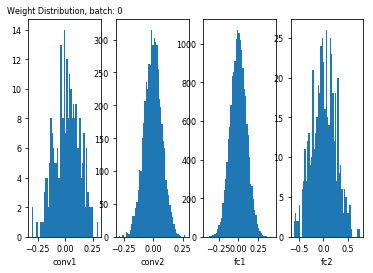


Weight distribution: batch 1800

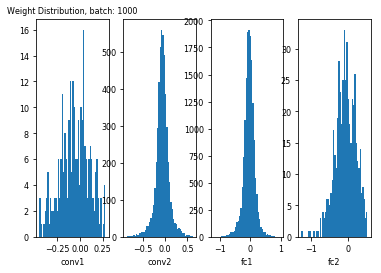


### **xavier** mode

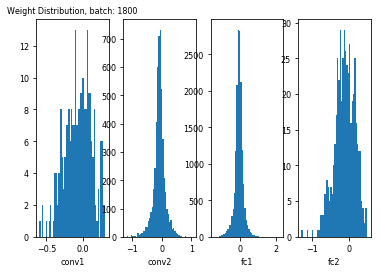
Weight distribution: Initial



Weight distribution: batch 1000

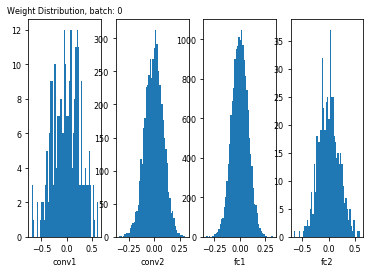


Weight distribution: batch 1800

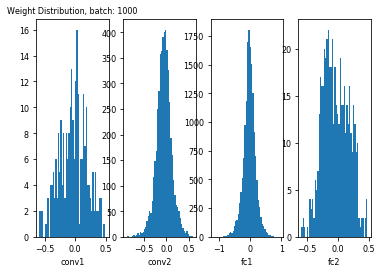


### **kaiming** mode

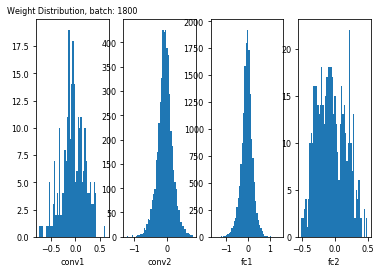
Weight distribution: Initial



Weight distribution: batch 1000

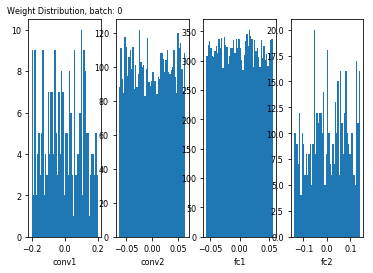


Weight distribution: batch 1800

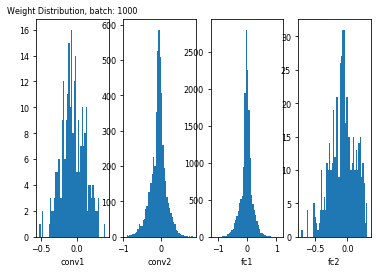


### **original** mode

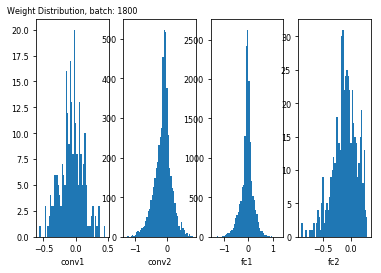
Weight distribution: Initial



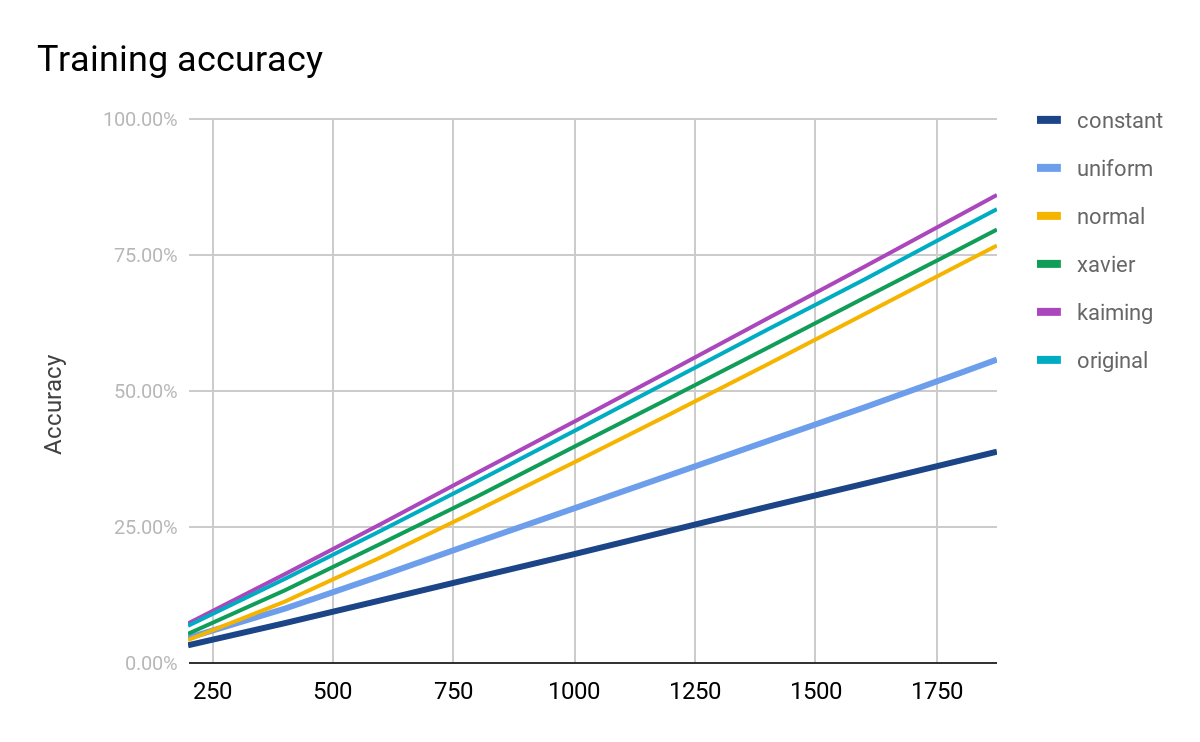
Weight distribution: batch 1000



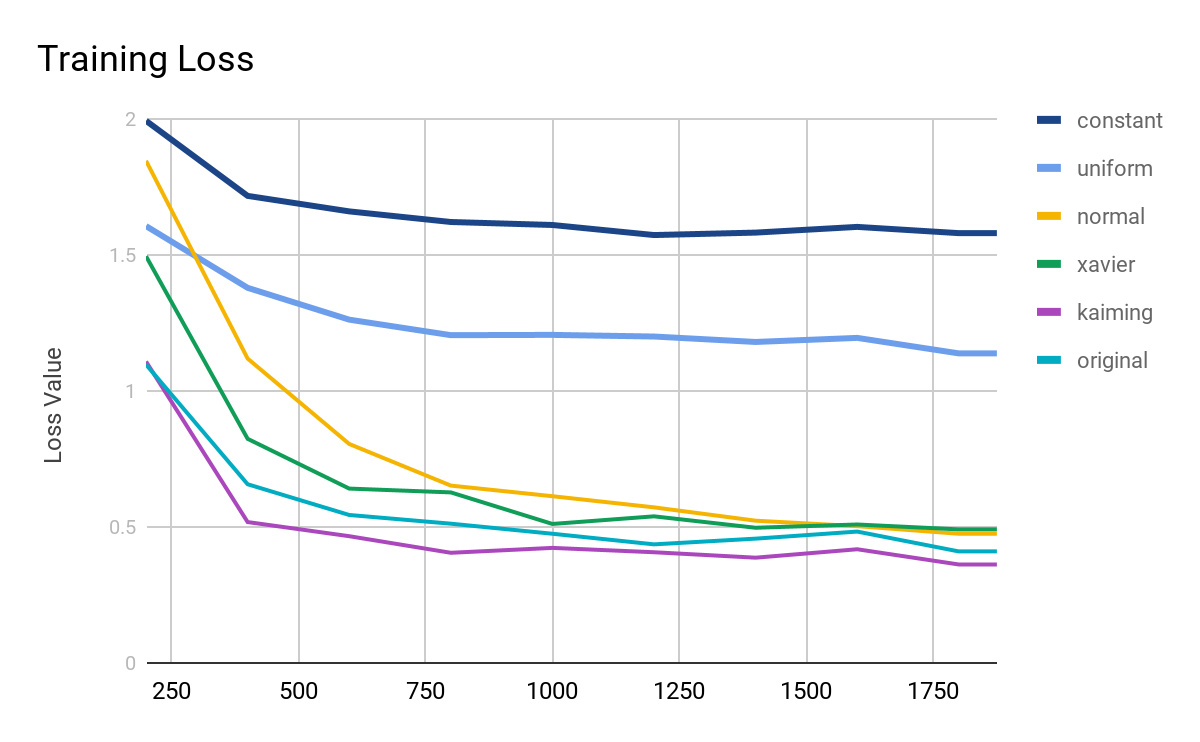
Weight distribution: batch 1800



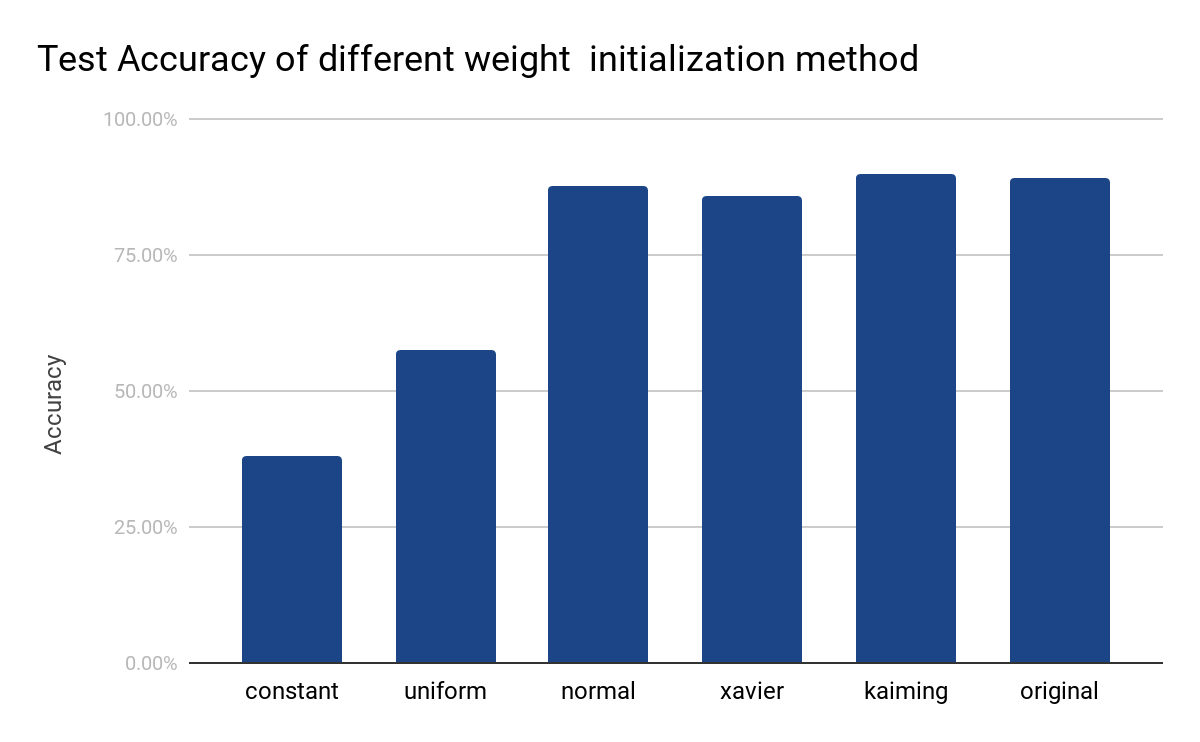
Training accuracy



Training Loss



### Test **Accuracy** of different weight initialization method



* Problems & solutions :
* Experiment setup :
* Results :
* Analysis :