cs_4701_code_final

May 26, 2021

[1]:
```
#John Crossman jcc395, Steven Jiang ssj54
```

[2]:
```
#Imports
import qiskit
import numpy as np
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import IBMQ, Aer, transpile, assemble
from qiskit.extensions import Initialize
```

[3]:
```
#Class for quantum convolutional filter
class Quanvoluter:
    def __init__(self, kernel_size=2, stride=1):
        self.kernel_size = kernel_size
        self.stride = stride
        self.thetas = np.zeros((kernel_size, kernel_size))

    #This function converts grayscale values to r_y rotation values
    def quantum_data_encoder(self, img):
        img = np.array(img)
        init_thetas = img * np.pi / 2
        return init_thetas

    #img_sec is 2 by 2 section of image
    #This function creates out quantum circuit
    def circuit_builder(self, img_sec, thetas = None):
        if (thetas is None):
            thetas = self.thetas
        kernel_size = self.kernel_size
        thetas = thetas.flatten()

        img_sec = img_sec.flatten()
        qr = QuantumRegister(kernel_size * kernel_size, name="q")
        cr = ClassicalRegister(kernel_size * kernel_size, name="c")
        qc = QuantumCircuit(qr, cr)
        qubit_i = 0
        for img in img_sec:
            qc.ry(img, qubit_i)
            qubit_i += 1
```

```
        qubit_i = 0
        qc.barrier()
        for theta in thetas:
            qc.ry(theta, qubit_i)
            qubit_i += 1

        #CNOTS

        qubit_i = 0
        while(qubit_i < ((kernel_size * kernel_size) - 1)):
            qc.cx(qubit_i, qubit_i + 1)
            qubit_i += 2

        qubit_i = 1
        while(qubit_i < ((kernel_size * kernel_size) - 1)):
            qc.cx(qubit_i, qubit_i + 1)
            qubit_i += 2

        #MEASUREMENTS

        qc.barrier()

        qubit_i = 0
        while(qubit_i < ((kernel_size * kernel_size))):
            qc.measure(qubit_i, qubit_i)
            qubit_i += 1

        return qc

    #computes expectation of Z
    def corr_measure(self, circ):
        shots = 8192
        backend = qiskit.Aer.get_backend('qasm_simulator') #change this for␣
↪real quantum computer
        qobj = assemble(circ, shots=shots)
        result = backend.run(qobj).result()
        counts = result.get_counts(circ)
        sum = 0
        for key in counts:
            key_arr = list(key)
            key_arr = np.array(key_arr)
            key_arr_ints = np.array([int(x) for x in key_arr])
            key_sum = np.sum(key_arr_ints)
            weighted_val = ((-1)**key_sum) * counts[key]
            sum = sum + weighted_val
        return sum / shots
```

```python
    def feature_mapper(self, img): #img is set of init thetas for the image␣
↪data
        kernel_size = self.kernel_size
        stride = self.stride

        rows, cols = np.shape(img)
        feature_map_rows = int(((rows - kernel_size) / stride) + 1)
        feature_map_cols = int(((cols - kernel_size) / stride) + 1)
        feature_map = np.zeros((feature_map_rows, feature_map_cols))
        i = 0
        row_offset = 0
        col_offset = 0

        while(i < feature_map_rows * feature_map_cols):
            img_sec = img[row_offset:(row_offset+kernel_size), col_offset:
↪(col_offset+kernel_size)]
            circ = self.circuit_builder(img_sec)
            x = self.corr_measure(circ)
            feature_map[(i // feature_map_cols), (i % feature_map_cols)] = x
            i += 1
            col_offset += stride
            if (col_offset + kernel_size > cols):
                row_offset += stride
                col_offset = 0
        return feature_map

    #Helper function for quantum grad calcs
    def sub_grad_calc(self, img_sec):
        shift = np.pi / 2
        thetas = self.thetas
        thetas_copy = thetas.copy()
        grads = np.zeros(len(thetas), dtype=float)
        print(thetas)

        for i in range(len(thetas)):
            thetas_copy[i] += shift
            print(thetas_copy)
            pos_circ = self.circuit_builder(img_sec, thetas = thetas_copy)
            thetas_copy = thetas.copy()
            thetas_copy[i] -= shift
            print(thetas_copy)
            neg_circ = self.circuit_builder(img_sec, thetas = thetas_copy)
            thetas_copy = thetas.copy()
            grads[i] = 0.5 * (self.corr_measure(pos_circ) - self.
↪corr_measure(neg_circ))

        return grads
```

```python
    def grad_calc(self, img):
        shift = np.pi / 2
        thetas = self.thetas
        thetas_copy = thetas
        grads = np.zeros(len(thetas), dtype=float)

        #loop over all 2 by 2 windows, and add up all gradient calcs.

        kernel_size = self.kernel_size
        stride = self.stride

        rows, cols = np.shape(img)
        feature_map_rows = int(((rows - kernel_size) / stride) + 1)
        feature_map_cols = int(((cols - kernel_size) / stride) + 1)
        feature_map = np.zeros((feature_map_rows, feature_map_cols))
        i = 0
        row_offset = 0
        col_offset = 0

        while(i < feature_map_rows * feature_map_cols):
            img_sec = img[row_offset:(row_offset+kernel_size), col_offset:
↪(col_offset+kernel_size)]
            grads += self.sub_grad_calc(img_sec)
            i += 1
            col_offset += stride
            if (col_offset + kernel_size > cols):
                row_offset += stride
                col_offset = 0

        return grads

    def thetas_updates(self, img):
        grads = self.grad_calc(img)
        self.thetas = self.thetas - grads
```

[4]:
```python
#First, test circuit building
```

[5]:
```python
img_sec_test = np.array([[0.5, 1], [0, 1]]) #Greyscale values
```

[6]:
```python
img_sec_test
```

[6]:
```
array([[0.5, 1. ],
       [0. , 1. ]])
```

[7]:
```python
q_test = Quanvoluter()
```

[8]:
```python
theta_vals_test = q_test.quantum_data_encoder(img_sec_test)
```

[9]:
```python
theta_vals_test
```

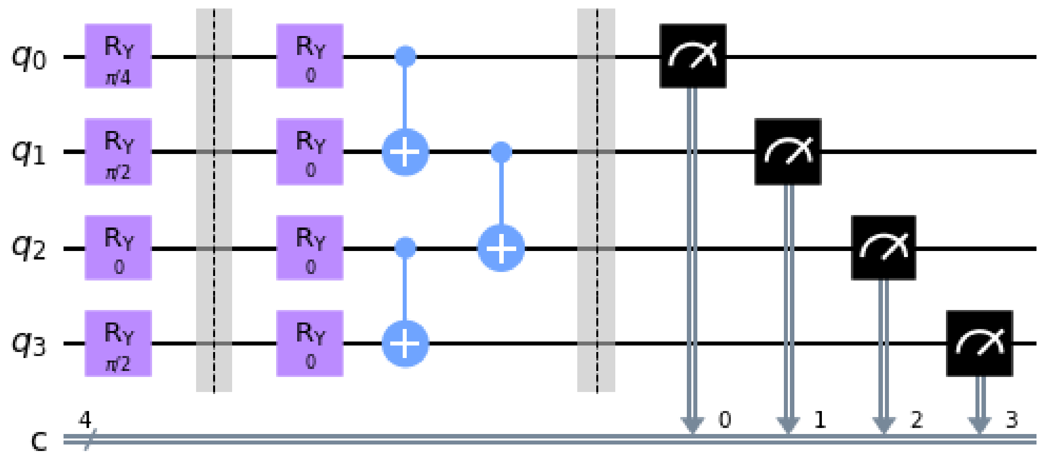
[9]:
```
array([[0.78539816, 1.57079633],
       [0.        , 1.57079633]])
```

[10]:
```
#quantum data encoder works!
```

[11]:
```
test_circ = q_test.circuit_builder(theta_vals_test)
```

[12]:
```
test_circ.draw(output = 'mpl')
```

[12]:



[13]:
```
#Looks like the circuit is building properly!
#Let's test generalization to arbitrary kernel size.
```

[14]:
```
q_test_big = Quanvoluter(kernel_size = 4)
```

[15]:
```
img_vals_test_big = [[1, 1, 0, 0.5], [1, 1, 0, 0.5], [0.5, 1, 0, 1],  [1, 1, 1,␣
 ↪1]]
```

[16]:
```
theta_vals_test_big = q_test_big.quantum_data_encoder(img_vals_test_big)
```

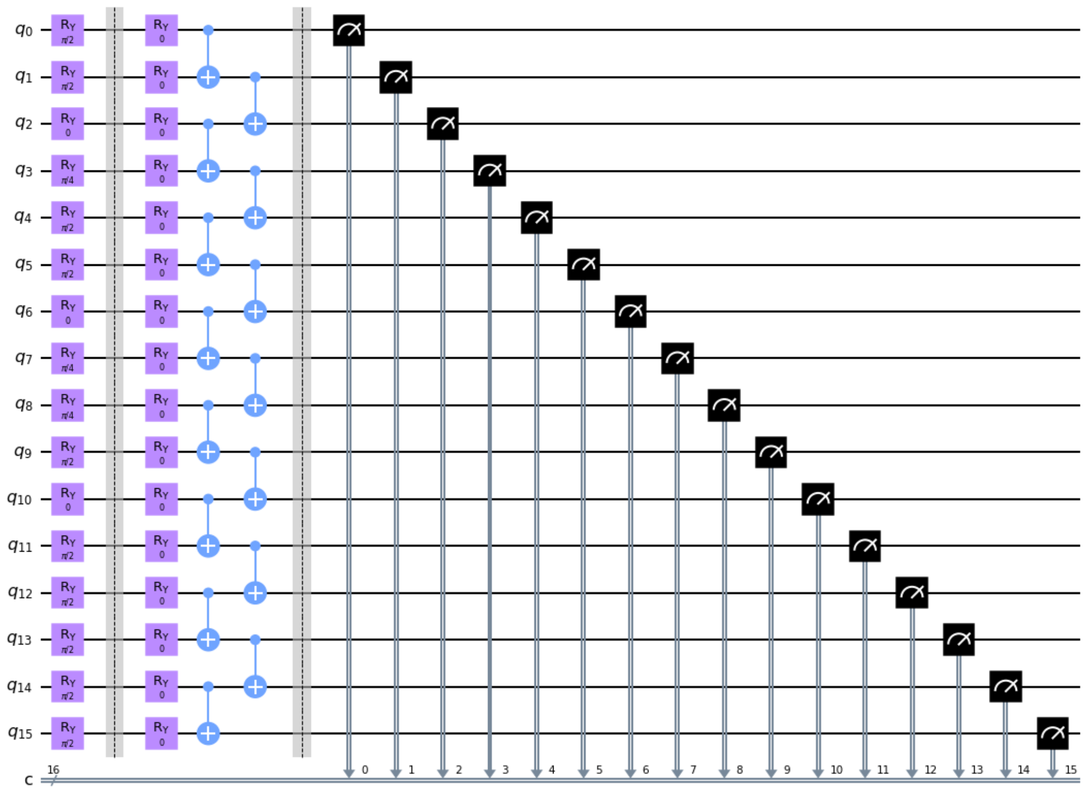
[17]:
```
theta_vals_test_big
```

[17]:
```
array([[1.57079633, 1.57079633, 0.        , 0.78539816],
       [1.57079633, 1.57079633, 0.        , 0.78539816],
       [0.78539816, 1.57079633, 0.        , 1.57079633],
       [1.57079633, 1.57079633, 1.57079633, 1.57079633]])
```

[18]:
```
test_circ_big = q_test_big.circuit_builder(theta_vals_test_big)
```

[19]:
```
test_circ_big.draw(output='mpl')
```

[19]:



[20]:
```
#It works!
```

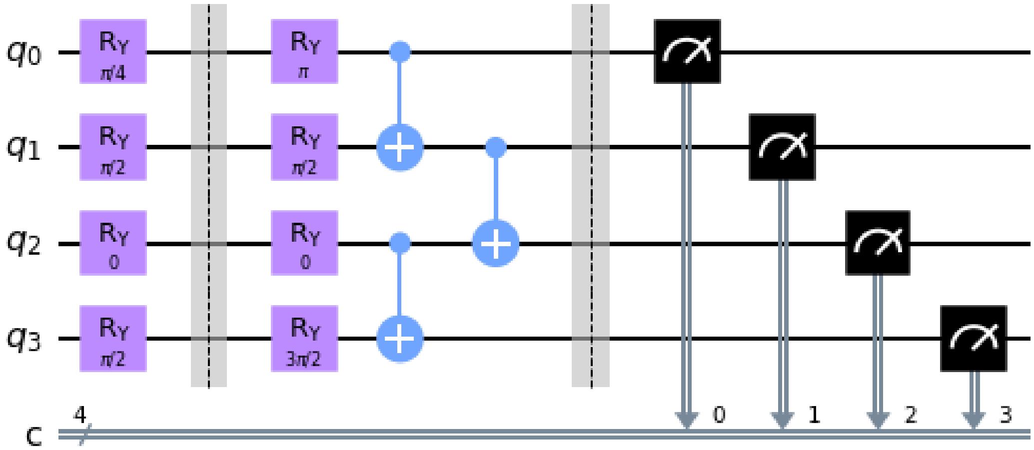
[21]:
```
#Test circuit builder for a "learned" theta vector for second set of R_y gates
```

[22]:
```
q_test.thetas = np.array([np.pi, np.pi / 2, 0, 3* np.pi/2])
```

[23]:
```
test_circ = q_test.circuit_builder(theta_vals_test)
```

[24]:
```
test_circ.draw(output='mpl')
```

[24]:



[25]:
```
#It works!
```

[26]:
```
#Test correlational measurement
```

[27]:
```
qr = QuantumRegister(4, name="q")
cr = ClassicalRegister(4, name="c")
special_circ = QuantumCircuit(qr, cr)
```

[28]:
```
special_circ.x(0)
```

[28]: <qiskit.circuit.instructionset.InstructionSet at 0x1ce18737b38>

[29]:
```
special_circ.barrier()
```

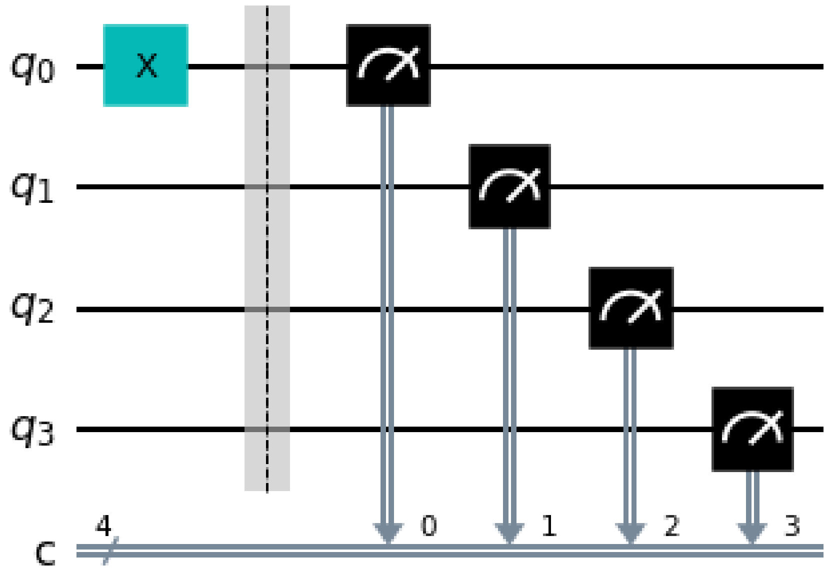
[29]: <qiskit.circuit.instructionset.InstructionSet at 0x1ce18737668>

[30]:
```
special_circ.measure(0, 0)
special_circ.measure(1, 1)
special_circ.measure(2, 2)
special_circ.measure(3, 3)
```

[30]: <qiskit.circuit.instructionset.InstructionSet at 0x1ce1865cb70>

[31]:
```
special_circ.draw(output='mpl')
```

[31]:



[32]:
```
q_test.corr_measure(special_circ)
```

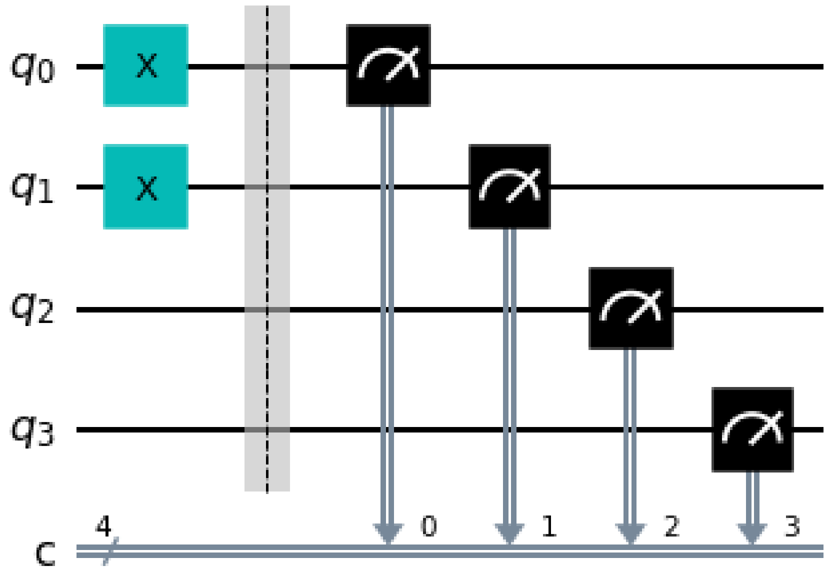
[32]: -1.0

[33]:
```
#negative one as expected. Next test should be positive one.
```

[34]:
```
qr = QuantumRegister(4, name="q")
cr = ClassicalRegister(4, name="c")
special_circ = QuantumCircuit(qr, cr)
```

[35]:
```
special_circ.x(0)
special_circ.x(1)
special_circ.barrier()
special_circ.measure(0, 0)
special_circ.measure(1, 1)
special_circ.measure(2, 2)
special_circ.measure(3, 3)
special_circ.draw(output='mpl')
```

[35]:



[36]:
```
q_test.corr_measure(special_circ)
```

[36]:
```
1.0
```

[37]:
```
#corr_measure works!
```

[38]:
```
#test subgradient
```

[39]:
```
q_test.thetas = np.array([0.0, 0.0, 0.0, 0.0])
```

[40]:
```
q_test.thetas
```

[40]:
```
array([0., 0., 0., 0.])
```

[41]:
```
q_test.sub_grad_calc(np.array([0.0,0.0,0.0,0.0]))
```

```
[0. 0. 0. 0.]
[1.57079633 0.         0.         0.        ]
[-1.57079633  0.          0.          0.        ]
[0.         1.57079633 0.         0.        ]
[ 0.         -1.57079633  0.          0.        ]
[0.         0.         1.57079633 0.        ]
[ 0.          0.         -1.57079633  0.        ]
[0.         0.         0.         1.57079633]
[ 0.          0.          0.         -1.57079633]
```

[41]:
```
array([0.0065918 , 0.        , 0.        , 0.00256348])
```

[42]:
```
#sub_grad works!
```

[43]:
```
test_img = np.array([[0.0,0.0],[0.0,0.0]])
```

[44]:
```
q_test.thetas_updates(test_img)
```

```
[0. 0. 0. 0.]
[1.57079633 0.         0.         0.        ]
[-1.57079633  0.          0.          0.        ]
[0.         1.57079633 0.         0.        ]
[ 0.          -1.57079633  0.          0.        ]
[0.         0.         1.57079633 0.        ]
[ 0.          0.          -1.57079633  0.        ]
[0.         0.         0.         1.57079633]
[ 0.          0.          0.          -1.57079633]
```

[45]:
```
q_test.thetas
```

[45]:
```
array([0.00402832, 0.        , 0.        , 0.00109863])
```

[46]:
```
#This works!
```

[47]:
```
#finally, test feature mapper
```

[48]:
```
test_img = np.array([[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1,␣
↪1, 1, 1], [1, 1, 1, 1, 1]])
```

[49]:
```
test_img
```

[49]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

[50]:
```
test_img = q_test.quantum_data_encoder(test_img)
```

[51]:
```
test_img
```

[51]:
```
array([[1.57079633, 1.57079633, 1.57079633, 1.57079633, 1.57079633],
       [1.57079633, 1.57079633, 1.57079633, 1.57079633, 1.57079633],
       [1.57079633, 1.57079633, 1.57079633, 1.57079633, 1.57079633],
       [1.57079633, 1.57079633, 1.57079633, 1.57079633, 1.57079633],
       [1.57079633, 1.57079633, 1.57079633, 1.57079633, 1.57079633]])
```

[52]:
```
q_test.thetas = np.array([-np.pi/2, -np.pi/2, -np.pi/2, -np.pi/2])
```

[53]:
```
#should map everything to one.
```

[54]:
```
q_test.feature_mapper(test_img)
```

[54]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[55]:
```
q_test.thetas = np.array([0.0, 0.0, 0.0, 0.0])
```

[56]:
```
q_test.feature_mapper(test_img)
```

[56]:
```
array([[ 0.00024414, -0.01855469, -0.01342773, -0.02514648],
       [ 0.00537109, -0.00195312,  0.00708008, -0.00805664],
       [ 0.00512695,  0.00024414, -0.00878906,  0.00488281],
       [-0.01733398, -0.02172852, -0.00512695, -0.00512695]])
```

[146]:
```
#The quantum convolutional filter appears to be working.
```

[57]:
```
import numpy as np
import matplotlib.pyplot as plt

import torch
from torch.autograd import Function
from torchvision import datasets, transforms
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F

import qiskit
from qiskit import transpile, assemble
from qiskit.visualization import *
```

[58]:
```
import torch
import torchvision
```

[59]:
```
import tensorflow as tf
```

[60]:
```
#Hybrid network class for a single convolutional layer.
class Hybrid(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = Quanvoluter()
        self.fc1 = nn.Linear(169, 25)
        self.fc2 = nn.Linear(25, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(torch.from_numpy(np.array([self.conv1.
 ↪feature_mapper(x)])), 2))
        x = x.view(-1, 169)
        x = F.relu(self.fc1(x.float()))
        x = self.fc2(x)
        return F.log_softmax(x)
```

[61]:
```
#Note: the rest of the blow code is for the classical network and is largely␣
 ↪taken from
#"https://nextjournal.com/gkoehler/pytorch-mnist."
#We were going to use the classical network solely to test it against the␣
 ↪hybrid CNN.
```

```
#However, we have modified the network module to include the right number of␣
 ↪convolutional layers and fully connected
#layers/neurons we planned on using. However, the rest of the code is largely␣
 ↪the same as that found at
#"https://nextjournal.com/gkoehler/pytorch-mnist."
#Credit: Gregor Koehler Feb 17, 2020
```

[62]:
```
#Modified class for CCNN
```

[63]:
```
class CNet(nn.Module):
    def __init__(self):
        super(CNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 1, kernel_size=2)
        self.conv2 = nn.Conv2d(1, 1, kernel_size=2)
        self.fc1 = nn.Linear(36, 25)
        self.fc2 = nn.Linear(25, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = x.view(-1, 36)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x)
```

[64]:
```
net = CNet() #net = Hybrid()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[65]:
```
n_epochs = 3
batch_size_train = 64
batch_size_test = 1000
log_interval = 10

random_seed = 1
torch.backends.cudnn.enabled = False
torch.manual_seed(random_seed)
```

[65]:
```
<torch._C.Generator at 0x1463d987ab0>
```

[66]:
```
train_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=True, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,)) #mean, std normalization
                             ])),
  batch_size=batch_size_train, shuffle=True)

test_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=False, download=True,
```

```
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_test, shuffle=True)
```

[67]:
```
train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]
```

[68]:
```
def train(epoch):
    net.train
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = net(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        #net.conv1.thetas_updates(data) #back-propagation for the QCNN
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.
 ↪format(epoch, batch_idx * len(data), len(train_loader.dataset), 100. *␣
 ↪batch_idx / len(train_loader), loss.item()))
            train_losses.append(loss.item())
            train_counter.append((batch_idx*64) + ((epoch-1)*len(train_loader.
 ↪dataset)))
```

[69]:
```
def test():
    net.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = net(data)
            test_loss += F.nll_loss(output, target, size_average=False).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).sum()
    test_loss /= len(test_loader.dataset)
    test_losses.append(test_loss)
    print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)))
```

[70]:
```
test()
for epoch in range(1, n_epochs + 1):
    train(epoch)
    test()
```

```
C:\Users\johnn\.julia\conda\3\lib\site-packages\ipykernel_launcher.py:15:
UserWarning: Implicit dimension choice for log_softmax has been deprecated.
Change the call to include dim=X as an argument.
  from ipykernel import kernelapp as app
C:\Users\johnn\.julia\conda\3\lib\site-packages\torch\nn\_reduction.py:42:
UserWarning: size_average and reduce args will be deprecated, please use
reduction='sum' instead.
  warnings.warn(warning.format(ret))


Test set: Avg. loss: 2.3059, Accuracy: 1044/10000 (10%)

Train Epoch: 1 [0/60000 (0%)]   Loss: 2.317006
Train Epoch: 1 [640/60000 (1%)] Loss: 2.276374
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.284849
Train Epoch: 1 [1920/60000 (3%)]        Loss: 2.299467
Train Epoch: 1 [2560/60000 (4%)]        Loss: 2.281609
Train Epoch: 1 [3200/60000 (5%)]        Loss: 2.276863
Train Epoch: 1 [3840/60000 (6%)]        Loss: 2.286090
Train Epoch: 1 [4480/60000 (7%)]        Loss: 2.300937
Train Epoch: 1 [5120/60000 (9%)]        Loss: 2.290903
Train Epoch: 1 [5760/60000 (10%)]       Loss: 2.308707
Train Epoch: 1 [6400/60000 (11%)]       Loss: 2.300022
Train Epoch: 1 [7040/60000 (12%)]       Loss: 2.301605
Train Epoch: 1 [7680/60000 (13%)]       Loss: 2.281618
Train Epoch: 1 [8320/60000 (14%)]       Loss: 2.284157
Train Epoch: 1 [8960/60000 (15%)]       Loss: 2.272540
Train Epoch: 1 [9600/60000 (16%)]       Loss: 2.283509
Train Epoch: 1 [10240/60000 (17%)]      Loss: 2.271096
Train Epoch: 1 [10880/60000 (18%)]      Loss: 2.267133
Train Epoch: 1 [11520/60000 (19%)]      Loss: 2.290675
Train Epoch: 1 [12160/60000 (20%)]      Loss: 2.248080
Train Epoch: 1 [12800/60000 (21%)]      Loss: 2.276143
Train Epoch: 1 [13440/60000 (22%)]      Loss: 2.248928
Train Epoch: 1 [14080/60000 (23%)]      Loss: 2.211706
Train Epoch: 1 [14720/60000 (25%)]      Loss: 2.241568
Train Epoch: 1 [15360/60000 (26%)]      Loss: 2.213793
Train Epoch: 1 [16000/60000 (27%)]      Loss: 2.215102
Train Epoch: 1 [16640/60000 (28%)]      Loss: 2.249144
Train Epoch: 1 [17280/60000 (29%)]      Loss: 2.212794
Train Epoch: 1 [17920/60000 (30%)]      Loss: 2.231268
Train Epoch: 1 [18560/60000 (31%)]      Loss: 2.209783
Train Epoch: 1 [19200/60000 (32%)]      Loss: 2.186702
Train Epoch: 1 [19840/60000 (33%)]      Loss: 2.136395
Train Epoch: 1 [20480/60000 (34%)]      Loss: 2.158204
Train Epoch: 1 [21120/60000 (35%)]      Loss: 2.139778
Train Epoch: 1 [21760/60000 (36%)]      Loss: 2.051205
Train Epoch: 1 [22400/60000 (37%)]      Loss: 2.079193
```

```
Train Epoch: 1 [23040/60000 (38%)]      Loss: 1.940106
Train Epoch: 1 [23680/60000 (39%)]      Loss: 1.853021
Train Epoch: 1 [24320/60000 (41%)]      Loss: 1.737386
Train Epoch: 1 [24960/60000 (42%)]      Loss: 1.739511
Train Epoch: 1 [25600/60000 (43%)]      Loss: 1.641935
Train Epoch: 1 [26240/60000 (44%)]      Loss: 1.530867
Train Epoch: 1 [26880/60000 (45%)]      Loss: 1.522743
Train Epoch: 1 [27520/60000 (46%)]      Loss: 1.381263
Train Epoch: 1 [28160/60000 (47%)]      Loss: 1.164993
Train Epoch: 1 [28800/60000 (48%)]      Loss: 1.002063
Train Epoch: 1 [29440/60000 (49%)]      Loss: 0.950321
Train Epoch: 1 [30080/60000 (50%)]      Loss: 0.932801
Train Epoch: 1 [30720/60000 (51%)]      Loss: 0.894762
Train Epoch: 1 [31360/60000 (52%)]      Loss: 0.844923
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.929125
Train Epoch: 1 [32640/60000 (54%)]      Loss: 0.644575
Train Epoch: 1 [33280/60000 (55%)]      Loss: 0.885385
Train Epoch: 1 [33920/60000 (57%)]      Loss: 0.826786
Train Epoch: 1 [34560/60000 (58%)]      Loss: 0.686843
Train Epoch: 1 [35200/60000 (59%)]      Loss: 0.590488
Train Epoch: 1 [35840/60000 (60%)]      Loss: 0.677508
Train Epoch: 1 [36480/60000 (61%)]      Loss: 0.903768
Train Epoch: 1 [37120/60000 (62%)]      Loss: 0.716876
Train Epoch: 1 [37760/60000 (63%)]      Loss: 0.659394
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.467410
Train Epoch: 1 [39040/60000 (65%)]      Loss: 0.543620
Train Epoch: 1 [39680/60000 (66%)]      Loss: 0.695765
Train Epoch: 1 [40320/60000 (67%)]      Loss: 0.708018
Train Epoch: 1 [40960/60000 (68%)]      Loss: 0.553371
Train Epoch: 1 [41600/60000 (69%)]      Loss: 0.746657
Train Epoch: 1 [42240/60000 (70%)]      Loss: 0.605082
Train Epoch: 1 [42880/60000 (71%)]      Loss: 0.402656
Train Epoch: 1 [43520/60000 (72%)]      Loss: 0.374308
Train Epoch: 1 [44160/60000 (74%)]      Loss: 0.566640
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.638361
Train Epoch: 1 [45440/60000 (76%)]      Loss: 0.290819
Train Epoch: 1 [46080/60000 (77%)]      Loss: 0.440146
Train Epoch: 1 [46720/60000 (78%)]      Loss: 0.498616
Train Epoch: 1 [47360/60000 (79%)]      Loss: 0.470777
Train Epoch: 1 [48000/60000 (80%)]      Loss: 0.528002
Train Epoch: 1 [48640/60000 (81%)]      Loss: 0.555432
Train Epoch: 1 [49280/60000 (82%)]      Loss: 0.794076
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.553197
Train Epoch: 1 [50560/60000 (84%)]      Loss: 0.815044
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.471096
Train Epoch: 1 [51840/60000 (86%)]      Loss: 0.419146
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.452945
Train Epoch: 1 [53120/60000 (88%)]      Loss: 0.462643
```

```
Train Epoch: 1 [53760/60000 (90%)]	Loss: 0.398157
Train Epoch: 1 [54400/60000 (91%)]	Loss: 0.340942
Train Epoch: 1 [55040/60000 (92%)]	Loss: 0.497506
Train Epoch: 1 [55680/60000 (93%)]	Loss: 0.274524
Train Epoch: 1 [56320/60000 (94%)]	Loss: 0.408104
Train Epoch: 1 [56960/60000 (95%)]	Loss: 0.720681
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.441772
Train Epoch: 1 [58240/60000 (97%)]	Loss: 0.447398
Train Epoch: 1 [58880/60000 (98%)]	Loss: 0.409275
Train Epoch: 1 [59520/60000 (99%)]	Loss: 0.356630

Test set: Avg. loss: 0.4804, Accuracy: 8512/10000 (85%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.311900
Train Epoch: 2 [640/60000 (1%)]	Loss: 0.716672
Train Epoch: 2 [1280/60000 (2%)]	Loss: 0.408842
Train Epoch: 2 [1920/60000 (3%)]	Loss: 0.507536
Train Epoch: 2 [2560/60000 (4%)]	Loss: 0.766106
Train Epoch: 2 [3200/60000 (5%)]	Loss: 0.408973
Train Epoch: 2 [3840/60000 (6%)]	Loss: 0.424057
Train Epoch: 2 [4480/60000 (7%)]	Loss: 0.440319
Train Epoch: 2 [5120/60000 (9%)]	Loss: 0.444634
Train Epoch: 2 [5760/60000 (10%)]	Loss: 0.866229
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.656391
Train Epoch: 2 [7040/60000 (12%)]	Loss: 0.480387
Train Epoch: 2 [7680/60000 (13%)]	Loss: 0.402641
Train Epoch: 2 [8320/60000 (14%)]	Loss: 0.379849
Train Epoch: 2 [8960/60000 (15%)]	Loss: 0.375752
Train Epoch: 2 [9600/60000 (16%)]	Loss: 0.429873
Train Epoch: 2 [10240/60000 (17%)]	Loss: 0.512670
Train Epoch: 2 [10880/60000 (18%)]	Loss: 0.532292
Train Epoch: 2 [11520/60000 (19%)]	Loss: 0.504641
Train Epoch: 2 [12160/60000 (20%)]	Loss: 0.396313
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.556257
Train Epoch: 2 [13440/60000 (22%)]	Loss: 0.528019
Train Epoch: 2 [14080/60000 (23%)]	Loss: 0.429725
Train Epoch: 2 [14720/60000 (25%)]	Loss: 0.375599
Train Epoch: 2 [15360/60000 (26%)]	Loss: 0.396417
Train Epoch: 2 [16000/60000 (27%)]	Loss: 0.205615
Train Epoch: 2 [16640/60000 (28%)]	Loss: 0.641002
Train Epoch: 2 [17280/60000 (29%)]	Loss: 0.335170
Train Epoch: 2 [17920/60000 (30%)]	Loss: 0.584923
Train Epoch: 2 [18560/60000 (31%)]	Loss: 0.380237
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.369711
Train Epoch: 2 [19840/60000 (33%)]	Loss: 0.642031
Train Epoch: 2 [20480/60000 (34%)]	Loss: 0.425254
Train Epoch: 2 [21120/60000 (35%)]	Loss: 0.415857
Train Epoch: 2 [21760/60000 (36%)]	Loss: 0.445800
```

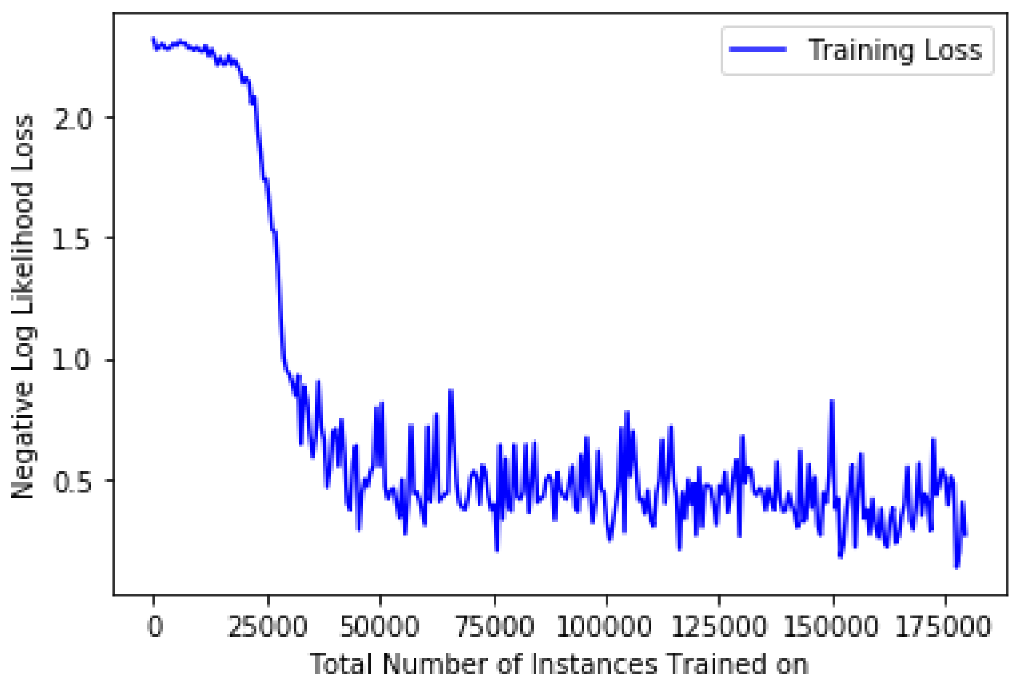
```
Train Epoch: 2 [22400/60000 (37%)]      Loss: 0.643625
Train Epoch: 2 [23040/60000 (38%)]      Loss: 0.362731
Train Epoch: 2 [23680/60000 (39%)]      Loss: 0.483383
Train Epoch: 2 [24320/60000 (41%)]      Loss: 0.651412
Train Epoch: 2 [24960/60000 (42%)]      Loss: 0.405484
Train Epoch: 2 [25600/60000 (43%)]      Loss: 0.422318
Train Epoch: 2 [26240/60000 (44%)]      Loss: 0.424879
Train Epoch: 2 [26880/60000 (45%)]      Loss: 0.493845
Train Epoch: 2 [27520/60000 (46%)]      Loss: 0.515969
Train Epoch: 2 [28160/60000 (47%)]      Loss: 0.492462
Train Epoch: 2 [28800/60000 (48%)]      Loss: 0.332650
Train Epoch: 2 [29440/60000 (49%)]      Loss: 0.528572
Train Epoch: 2 [30080/60000 (50%)]      Loss: 0.455274
Train Epoch: 2 [30720/60000 (51%)]      Loss: 0.430370
Train Epoch: 2 [31360/60000 (52%)]      Loss: 0.417630
Train Epoch: 2 [32000/60000 (53%)]      Loss: 0.483221
Train Epoch: 2 [32640/60000 (54%)]      Loss: 0.553562
Train Epoch: 2 [33280/60000 (55%)]      Loss: 0.379868
Train Epoch: 2 [33920/60000 (57%)]      Loss: 0.368067
Train Epoch: 2 [34560/60000 (58%)]      Loss: 0.599905
Train Epoch: 2 [35200/60000 (59%)]      Loss: 0.427137
Train Epoch: 2 [35840/60000 (60%)]      Loss: 0.671508
Train Epoch: 2 [36480/60000 (61%)]      Loss: 0.451627
Train Epoch: 2 [37120/60000 (62%)]      Loss: 0.320496
Train Epoch: 2 [37760/60000 (63%)]      Loss: 0.403829
Train Epoch: 2 [38400/60000 (64%)]      Loss: 0.617050
Train Epoch: 2 [39040/60000 (65%)]      Loss: 0.463037
Train Epoch: 2 [39680/60000 (66%)]      Loss: 0.443129
Train Epoch: 2 [40320/60000 (67%)]      Loss: 0.313833
Train Epoch: 2 [40960/60000 (68%)]      Loss: 0.248638
Train Epoch: 2 [41600/60000 (69%)]      Loss: 0.313635
Train Epoch: 2 [42240/60000 (70%)]      Loss: 0.391667
Train Epoch: 2 [42880/60000 (71%)]      Loss: 0.469486
Train Epoch: 2 [43520/60000 (72%)]      Loss: 0.711705
Train Epoch: 2 [44160/60000 (74%)]      Loss: 0.281724
Train Epoch: 2 [44800/60000 (75%)]      Loss: 0.777245
Train Epoch: 2 [45440/60000 (76%)]      Loss: 0.510440
Train Epoch: 2 [46080/60000 (77%)]      Loss: 0.696875
Train Epoch: 2 [46720/60000 (78%)]      Loss: 0.544573
Train Epoch: 2 [47360/60000 (79%)]      Loss: 0.412184
Train Epoch: 2 [48000/60000 (80%)]      Loss: 0.419394
Train Epoch: 2 [48640/60000 (81%)]      Loss: 0.357719
Train Epoch: 2 [49280/60000 (82%)]      Loss: 0.453466
Train Epoch: 2 [49920/60000 (83%)]      Loss: 0.330466
Train Epoch: 2 [50560/60000 (84%)]      Loss: 0.306245
Train Epoch: 2 [51200/60000 (85%)]      Loss: 0.442483
Train Epoch: 2 [51840/60000 (86%)]      Loss: 0.495216
Train Epoch: 2 [52480/60000 (87%)]      Loss: 0.661913
```

```
Train Epoch: 2 [53120/60000 (88%)]	Loss: 0.400485
Train Epoch: 2 [53760/60000 (90%)]	Loss: 0.510434
Train Epoch: 2 [54400/60000 (91%)]	Loss: 0.716218
Train Epoch: 2 [55040/60000 (92%)]	Loss: 0.495436
Train Epoch: 2 [55680/60000 (93%)]	Loss: 0.427473
Train Epoch: 2 [56320/60000 (94%)]	Loss: 0.211444
Train Epoch: 2 [56960/60000 (95%)]	Loss: 0.446863
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.337923
Train Epoch: 2 [58240/60000 (97%)]	Loss: 0.497523
Train Epoch: 2 [58880/60000 (98%)]	Loss: 0.397201
Train Epoch: 2 [59520/60000 (99%)]	Loss: 0.482900

Test set: Avg. loss: 0.4294, Accuracy: 8626/10000 (86%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.269598
Train Epoch: 3 [640/60000 (1%)]	Loss: 0.548275
Train Epoch: 3 [1280/60000 (2%)]	Loss: 0.303664
Train Epoch: 3 [1920/60000 (3%)]	Loss: 0.474222
Train Epoch: 3 [2560/60000 (4%)]	Loss: 0.475376
Train Epoch: 3 [3200/60000 (5%)]	Loss: 0.464051
Train Epoch: 3 [3840/60000 (6%)]	Loss: 0.396434
Train Epoch: 3 [4480/60000 (7%)]	Loss: 0.312276
Train Epoch: 3 [5120/60000 (9%)]	Loss: 0.473323
Train Epoch: 3 [5760/60000 (10%)]	Loss: 0.441715
Train Epoch: 3 [6400/60000 (11%)]	Loss: 0.528105
Train Epoch: 3 [7040/60000 (12%)]	Loss: 0.363129
Train Epoch: 3 [7680/60000 (13%)]	Loss: 0.434546
Train Epoch: 3 [8320/60000 (14%)]	Loss: 0.521688
Train Epoch: 3 [8960/60000 (15%)]	Loss: 0.580065
Train Epoch: 3 [9600/60000 (16%)]	Loss: 0.265042
Train Epoch: 3 [10240/60000 (17%)]	Loss: 0.677130
Train Epoch: 3 [10880/60000 (18%)]	Loss: 0.486141
Train Epoch: 3 [11520/60000 (19%)]	Loss: 0.548198
Train Epoch: 3 [12160/60000 (20%)]	Loss: 0.533649
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.446689
Train Epoch: 3 [13440/60000 (22%)]	Loss: 0.434510
Train Epoch: 3 [14080/60000 (23%)]	Loss: 0.462949
Train Epoch: 3 [14720/60000 (25%)]	Loss: 0.451103
Train Epoch: 3 [15360/60000 (26%)]	Loss: 0.373045
Train Epoch: 3 [16000/60000 (27%)]	Loss: 0.464096
Train Epoch: 3 [16640/60000 (28%)]	Loss: 0.409222
Train Epoch: 3 [17280/60000 (29%)]	Loss: 0.374731
Train Epoch: 3 [17920/60000 (30%)]	Loss: 0.571948
Train Epoch: 3 [18560/60000 (31%)]	Loss: 0.421245
Train Epoch: 3 [19200/60000 (32%)]	Loss: 0.367988
Train Epoch: 3 [19840/60000 (33%)]	Loss: 0.371897
Train Epoch: 3 [20480/60000 (34%)]	Loss: 0.443765
Train Epoch: 3 [21120/60000 (35%)]	Loss: 0.395096
```

```
Train Epoch: 3 [21760/60000 (36%)]	Loss: 0.360405
Train Epoch: 3 [22400/60000 (37%)]	Loss: 0.301758
Train Epoch: 3 [23040/60000 (38%)]	Loss: 0.617630
Train Epoch: 3 [23680/60000 (39%)]	Loss: 0.324174
Train Epoch: 3 [24320/60000 (41%)]	Loss: 0.338212
Train Epoch: 3 [24960/60000 (42%)]	Loss: 0.560300
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.380840
Train Epoch: 3 [26240/60000 (44%)]	Loss: 0.508559
Train Epoch: 3 [26880/60000 (45%)]	Loss: 0.309848
Train Epoch: 3 [27520/60000 (46%)]	Loss: 0.270184
Train Epoch: 3 [28160/60000 (47%)]	Loss: 0.443408
Train Epoch: 3 [28800/60000 (48%)]	Loss: 0.403121
Train Epoch: 3 [29440/60000 (49%)]	Loss: 0.517548
Train Epoch: 3 [30080/60000 (50%)]	Loss: 0.823069
Train Epoch: 3 [30720/60000 (51%)]	Loss: 0.380279
Train Epoch: 3 [31360/60000 (52%)]	Loss: 0.421570
Train Epoch: 3 [32000/60000 (53%)]	Loss: 0.178419
Train Epoch: 3 [32640/60000 (54%)]	Loss: 0.228875
Train Epoch: 3 [33280/60000 (55%)]	Loss: 0.372336
Train Epoch: 3 [33920/60000 (57%)]	Loss: 0.455830
Train Epoch: 3 [34560/60000 (58%)]	Loss: 0.558375
Train Epoch: 3 [35200/60000 (59%)]	Loss: 0.219630
Train Epoch: 3 [35840/60000 (60%)]	Loss: 0.461096
Train Epoch: 3 [36480/60000 (61%)]	Loss: 0.604162
Train Epoch: 3 [37120/60000 (62%)]	Loss: 0.340416
Train Epoch: 3 [37760/60000 (63%)]	Loss: 0.375458
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.273162
Train Epoch: 3 [39040/60000 (65%)]	Loss: 0.418682
Train Epoch: 3 [39680/60000 (66%)]	Loss: 0.307665
Train Epoch: 3 [40320/60000 (67%)]	Loss: 0.258279
Train Epoch: 3 [40960/60000 (68%)]	Loss: 0.378362
Train Epoch: 3 [41600/60000 (69%)]	Loss: 0.249881
Train Epoch: 3 [42240/60000 (70%)]	Loss: 0.220288
Train Epoch: 3 [42880/60000 (71%)]	Loss: 0.335674
Train Epoch: 3 [43520/60000 (72%)]	Loss: 0.381881
Train Epoch: 3 [44160/60000 (74%)]	Loss: 0.237417
Train Epoch: 3 [44800/60000 (75%)]	Loss: 0.265776
Train Epoch: 3 [45440/60000 (76%)]	Loss: 0.347844
Train Epoch: 3 [46080/60000 (77%)]	Loss: 0.398524
Train Epoch: 3 [46720/60000 (78%)]	Loss: 0.550844
Train Epoch: 3 [47360/60000 (79%)]	Loss: 0.331517
Train Epoch: 3 [48000/60000 (80%)]	Loss: 0.291431
Train Epoch: 3 [48640/60000 (81%)]	Loss: 0.393633
Train Epoch: 3 [49280/60000 (82%)]	Loss: 0.564937
Train Epoch: 3 [49920/60000 (83%)]	Loss: 0.349749
Train Epoch: 3 [50560/60000 (84%)]	Loss: 0.438983
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.408171
Train Epoch: 3 [51840/60000 (86%)]	Loss: 0.287638
```

```
Train Epoch: 3 [52480/60000 (87%)]      Loss: 0.663988
Train Epoch: 3 [53120/60000 (88%)]      Loss: 0.434672
Train Epoch: 3 [53760/60000 (90%)]      Loss: 0.472564
Train Epoch: 3 [54400/60000 (91%)]      Loss: 0.540497
Train Epoch: 3 [55040/60000 (92%)]      Loss: 0.508209
Train Epoch: 3 [55680/60000 (93%)]      Loss: 0.394271
Train Epoch: 3 [56320/60000 (94%)]      Loss: 0.511334
Train Epoch: 3 [56960/60000 (95%)]      Loss: 0.482670
Train Epoch: 3 [57600/60000 (96%)]      Loss: 0.136698
Train Epoch: 3 [58240/60000 (97%)]      Loss: 0.198434
Train Epoch: 3 [58880/60000 (98%)]      Loss: 0.404308
Train Epoch: 3 [59520/60000 (99%)]      Loss: 0.271020

Test set: Avg. loss: 0.3888, Accuracy: 8755/10000 (88%)
```

[71]:
```
fig = plt.figure()
plt.plot(train_counter, train_losses, color='blue')
#plt.scatter(test_counter, test_losses, color='red')
plt.legend(['Training Loss', 'Test Loss'], loc='upper right')
plt.xlabel('Total Number of Instances Trained on')
plt.ylabel('Negative Log Likelihood Loss')
plt.show()
```



[ ]:

[ ]:

[ ]:

[ ]: