Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

COGNITIVE CLASS

Convolutional Neural Network with Batch-Normalization

Table of Contents

This lab takes a long time to run so the results are given. You can run the notebook your self but it may take a long time.

In this lab, we will compare a Convolutional Neural Network using Batch Normalization with a regular Convolutional Neural Network to classify handwritten digits from the MNIST database. We will reshape the images to make them faster to process.

Estimated Time Needed: **25 min**

Read me Batch Norm for Convolution Operation

Like a fully connected network, we create a `BatchNorm2d` object, but we apply it to the 2D convolution object. First, we create objects `Conv2d` object; we require the number of output channels, specified by the variable `OUT`.

```
self.cnn1 = nn.Conv2d(in_channels=1, out_channels=OUT, kernel_size=5, padding=2)
```

We then create a Batch Norm object for 2D convolution as follows:

```
self.conv1_bn = nn.BatchNorm2d(OUT)
```

The parameter out is the number of channels in the output. We can then apply batch norm after the convolution operation :

```
x = self.cnn1(x)
```

```
x=self.conv1_bn(x)
```

Preparation

In [1]:

```
# Import the libraries we need to use in this lab

# Using the following line code to install the torchvision library
# !conda install -y torchvision

import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.datasets as dsets
import matplotlib.pylab as plt
import numpy as np
def show_data(data_sample):
    plt.imshow(data_sample[0].numpy().reshape(IMAGE_SIZE, IMAGE_SIZE), cmap='gray')
    plt.title('y = '+ str(data_sample[1].item()))
```

Get the Data

we create a transform to resize the image and convert it to a tensor :

In [2]:

```
IMAGE_SIZE = 16

composed = transforms.Compose([transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)), transforms.ToTensor()])
```

Load the training dataset by setting the parameters `train` to `True`. We use the transform defined above.

In [3]:

```
train_dataset = dsets.MNIST(root='./data', train=True, download=True, transform=composed)
```

Load the testing dataset by setting the parameters train `False`.

In [4]:

```
# Make the validating

validation_dataset = dsets.MNIST(root='./data', train=False, download=True, transform=composed)
```

We can see the data type is long.

In [5]:

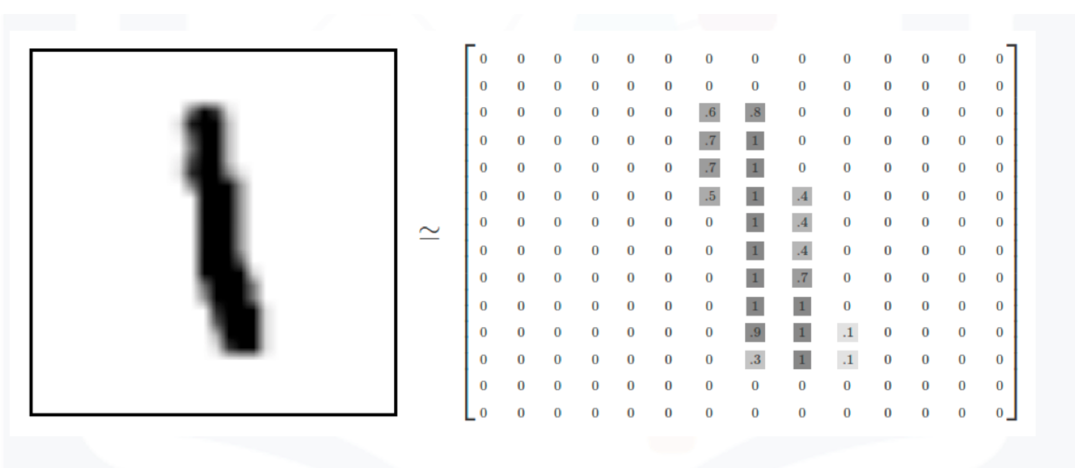
```
# Show the data type for each element in dataset

train_dataset[0][1].type()
```

Out[5]:

```
'torch.LongTensor'
```

Each element in the rectangular tensor corresponds to a number representing a pixel intensity as demonstrated by the following image.



Print out the fourth label

In [6]:

```
# The label for the fourth data element

train_dataset[3][1]
```

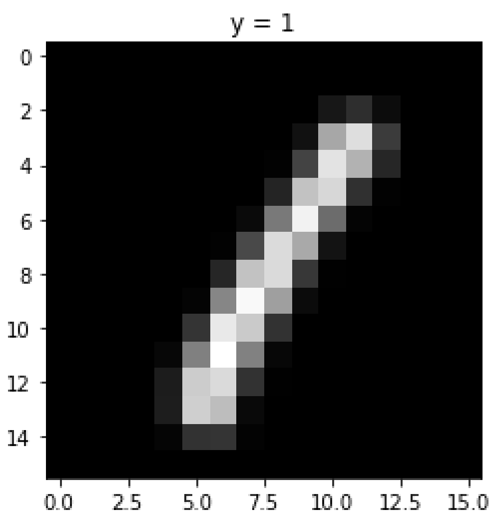
Out[6]:

```
tensor(1)
```

Plot the fourth sample

In [7]:

```
# The image for the fourth data element
show_data(train_dataset[3])
```



The fourth sample is a "1".

Build a Two Convolutional Neural Network Class

Build a Convolutional Network class with two Convolutional layers and one fully connected layer. Pre-determine the size of the final output matrix. The parameters in the constructor are the number of output channels for the first and second layer.

In [8]:

```
class CNN(nn.Module):

    # Contructor
    def __init__(self, out_1=16, out_2=32):
        super(CNN, self).__init__()
        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=out_1, kernel_size=5, pad
ding=2)
        self.maxpool1=nn.MaxPool2d(kernel_size=2)

        self.cnn2 = nn.Conv2d(in_channels=out_1, out_channels=out_2, kernel_size=5,
stride=1, padding=2)
        self.maxpool2=nn.MaxPool2d(kernel_size=2)
        self.fc1 = nn.Linear(out_2 * 4 * 4, 10)

    # Prediction
    def forward(self, x):
        x = self.cnn1(x)
        x = torch.relu(x)
        x = self.maxpool1(x)
        x = self.cnn2(x)
        x = torch.relu(x)
        x = self.maxpool2(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x
```

Build a Convolutional Network class with two Convolutional layers and one fully connected layer. But we add Batch Norm for the convolutional layers.

In [9]:

```
class CNN_batch(nn.Module):

    # Contructor
    def __init__(self, out_1=16, out_2=32,number_of_classes=10):
        super(CNN_batch, self).__init__()
        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=out_1, kernel_size=5, pad
ding=2)
        self.conv1_bn = nn.BatchNorm2d(out_1)

        self.maxpool1=nn.MaxPool2d(kernel_size=2)

        self.cnn2 = nn.Conv2d(in_channels=out_1, out_channels=out_2, kernel_size=5,
stride=1, padding=2)
        self.conv2_bn = nn.BatchNorm2d(out_2)

        self.maxpool2=nn.MaxPool2d(kernel_size=2)
        self.fc1 = nn.Linear(out_2 * 4 * 4, number_of_classes)
        self.bn_fc1 = nn.BatchNorm1d(10)

    # Prediction
    def forward(self, x):
        x = self.cnn1(x)
        x=self.conv1_bn(x)
        x = torch.relu(x)
        x = self.maxpool1(x)
        x = self.cnn2(x)
        x=self.conv2_bn(x)
        x = torch.relu(x)
        x = self.maxpool2(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x=self.bn_fc1(x)
        return x
```

Function to train the model

In [10]:

```
def train_model(model,train_loader,validation_loader,optimizer,n_epochs=4):

    #global variable
    N_test=len(validation_dataset)
    accuracy_list=[]
    loss_list=[]
    for epoch in range(n_epochs):
        for x, y in train_loader:
            model.train()
            optimizer.zero_grad()
            z = model(x)
            loss = criterion(z, y)
            loss.backward()
            optimizer.step()
            loss_list.append(loss.data)

        correct=0
        #perform a prediction on the validation  data
        for x_test, y_test in validation_loader:
            model.eval()
            z = model(x_test)
            _, yhat = torch.max(z.data, 1)
            correct += (yhat == y_test).sum().item()
        accuracy = correct / N_test
        accuracy_list.append(accuracy)

    return accuracy_list, loss_list
```

Define the Convolutional Neural Network Classifier, Criterion function, Optimizer and Train the Model

There are 16 output channels for the first layer, and 32 output channels for the second layer

In [11]:

```
# Create the model object using CNN class
model = CNN(out_1=16, out_2=32)
```

Define the loss function, the optimizer and the dataset loader

In [12]:

```
criterion = nn.CrossEntropyLoss()
learning_rate = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100)
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_s
ize=5000)
```

Train the model and determine validation accuracy technically test accuracy **(This may take a long time)**

In [13]:

```
# Train the model
accuracy_list_normal, loss_list_normal=train_model(model=model,n_epochs=10,train_lo
ader=train_loader,validation_loader=validation_loader,optimizer=optimizer)
```

Repeat the Process for the model with batch norm

In [14]:

```
model_batch=CNN_batch(out_1=16, out_2=32)
criterion = nn.CrossEntropyLoss()
learning_rate = 0.1
optimizer = torch.optim.SGD(model_batch.parameters(), lr = learning_rate)
accuracy_list_batch, loss_list_batch=train_model(model=model_batch,n_epochs=10,trai
n_loader=train_loader,validation_loader=validation_loader,optimizer=optimizer)
```

Analyze Results

Plot the loss with both networks.

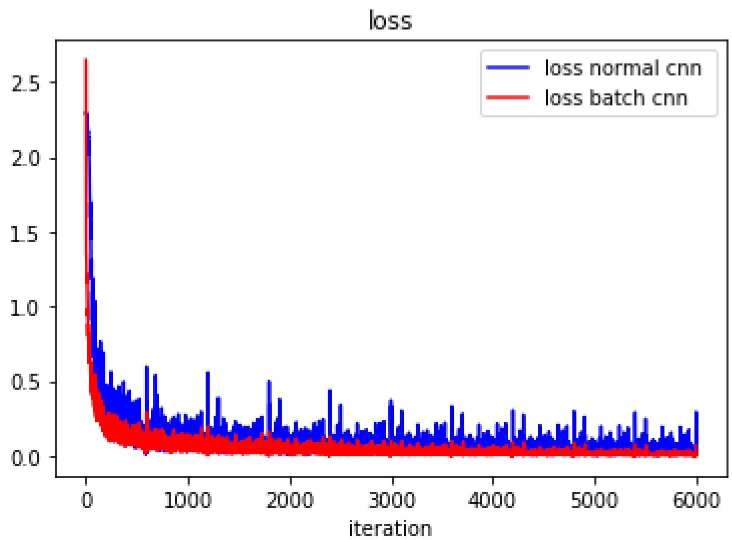
In [15]:

```
# Plot the loss and accuracy

plt.plot(loss_list_normal, 'b',label='loss normal cnn ')
plt.plot(loss_list_batch,'r',label='loss batch cnn')
plt.xlabel('iteration')
plt.title("loss")
plt.legend()
```

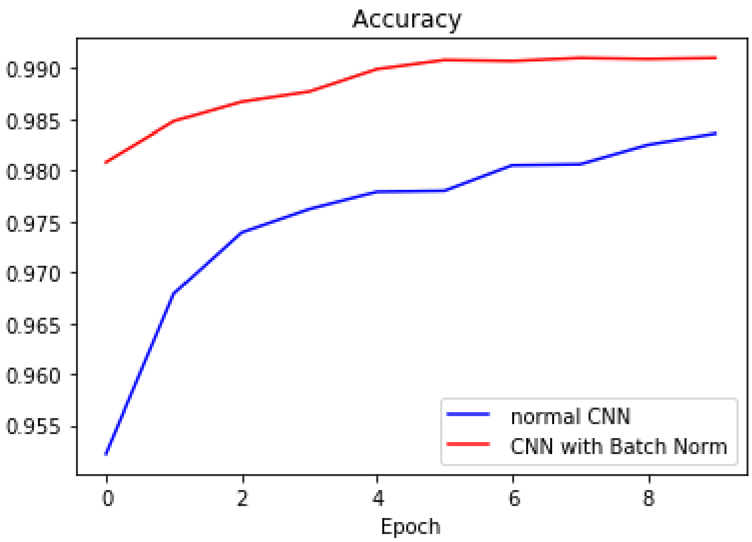
Out[15]:

```
<matplotlib.legend.Legend at 0x12a05da20>
```



In [16]:

```
plt.plot(accuracy_list_normal, 'b',label=' normal CNN')
plt.plot(accuracy_list_batch,'r',label=' CNN with Batch Norm')
plt.xlabel('Epoch')
plt.title("Accuracy ")
plt.legend()
plt.show()
```



We see the CNN with batch norm performers better, with faster convergence.

About the Authors:

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Michelle Carey (https://www.linkedin.com/in/michelleccarey/), Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)

Thanks to Magnus Erik Hvass Pedersen (http://www.hvass-labs.org/) whose tutorials helped me understand convolutional Neural Network

---