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)

Test Sigmoid, Tanh, and Relu Activations Functions on the MNIST Dataset

Table of Contents

In this lab, you will test sigmoid, tanh, and relu activation functions on the MNIST dataset.

- Neural Network Module and Training Function
- Make Some Data
- Define Several Neural Network, Criterion Function, and Optimizer
- Test Sigmoid, Tanh, and Relu
- Analyze Results

Estimated Time Needed: **25 min** </div>

Preparation

We'll need the following libraries

In [1]:

```
# Import the libraries we need for 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
```

Neural Network Module and Training Function

Define the neural network module or class using the sigmoid activation function:

In [2]:

```
# Build the model with sigmoid function

class Net(nn.Module):

    # Constructor
    def __init__(self, D_in, H, D_out):
        super(Net, self).__init__()
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)

    # Prediction
    def forward(self, x):
        x = torch.sigmoid(self.linear1(x))
        x = self.linear2(x)
        return x
```

Define the neural network module or class using the Tanh activation function:

In [3]:

```
# Build the model with Tanh function

class NetTanh(nn.Module):

    # Constructor
    def __init__(self, D_in, H, D_out):
        super(NetTanh, self).__init__()
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)

    # Prediction
    def forward(self, x):
        x = torch.tanh(self.linear1(x))
        x = self.linear2(x)
        return x
```

Define the neural network module or class using the Relu activation function:

In [4]:

```
# Build the model with Relu function

class NetRelu(nn.Module):

    # Constructor
    def __init__(self, D_in, H, D_out):
        super(NetRelu, self).__init__()
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)

    # Prediction
    def forward(self, x):
        x = torch.relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

Define a function to train the model. In this case, the function returns a Python dictionary to store the training loss for each iteration and accuracy on the validation data.

In [5]:

```
# Define the function for training the model

def train(model, criterion, train_loader, validation_loader, optimizer, epochs = 10
0):
    i = 0
    useful_stuff = {'training_loss':[], 'validation_accuracy':[]}

    for epoch in range(epochs):
        for i, (x, y) in enumerate(train_loader):
            optimizer.zero_grad()
            z = model(x.view(-1, 28 * 28))
            loss = criterion(z, y)
            loss.backward()
            optimizer.step()
            useful_stuff['training_loss'].append(loss.item())

        correct = 0
        for x, y in validation_loader:
            z = model(x.view(-1, 28 * 28))
            _, label=torch.max(z, 1)
            correct += (label == y).sum().item()
        accuracy = 100 * (correct / len(validation_dataset))
        useful_stuff['validation_accuracy'].append(accuracy)

    return useful_stuff
```

Make Some Data

Load the training dataset by setting the parameters `train` to `True` and convert it to a tensor by placing a transform object in the argument `transform`.

In [6]:

```
# Create the training dataset

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

Load the testing dataset by setting the parameter `train` to `False` and convert it to a tensor by placing a transform object in the argument `transform`.

In [7]:

```
# Create the validation  dataset

validation_dataset = dsets.MNIST(root='./data', train=False, download=True, transfo
rm=transforms.ToTensor())
```

Create the criterion function:

In [8]:

```
# Create the criterion function

criterion = nn.CrossEntropyLoss()
```

Create the training-data loader and the validation-data loader object:

In [9]:

```
# Create the training data loader and validation data loader object

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=2000,
shuffle=True)
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_s
ize=5000, shuffle=False)
```

Define the Neural Network, Criterion Function, Optimizer, and Train the Model

Create the criterion function:

In [10]:

```
# Create the criterion function

criterion = nn.CrossEntropyLoss()
```

Create the model with 100 hidden neurons:

In [11]:

```
# Create the model object

input_dim = 28 * 28
hidden_dim = 100
output_dim = 10

model = Net(input_dim, hidden_dim, output_dim)
```

Test Sigmoid, Tanh, and Relu

Train the network by using the sigmoid activations function:

In [12]:

```
# Train a model with sigmoid function

learning_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
training_results = train(model, criterion, train_loader, validation_loader, optimiz
er, epochs=30)
```

Train the network by using the Tanh activations function:

In [13]:

```
# Train a model with Tanh function

model_Tanh = NetTanh(input_dim, hidden_dim, output_dim)
optimizer = torch.optim.SGD(model_Tanh.parameters(), lr=learning_rate)
training_results_tanch = train(model_Tanh, criterion, train_loader, validation_load
er, optimizer, epochs=30)
```

Train the network by using the Relu activations function:

In [ ]:

```
# Train a model with Relu function

modelRelu = NetRelu(input_dim, hidden_dim, output_dim)
optimizer = torch.optim.SGD(modelRelu.parameters(), lr=learning_rate)
training_results_relu = train(modelRelu,criterion, train_loader, validation_loader,
optimizer, epochs=30)
```

Analyze Results

Compare the training loss for each activation:

In [ ]:

```
# Compare the training loss

plt.plot(training_results_tanch['training_loss'], label='tanh')
plt.plot(training_results['training_loss'], label='sigmoid')
plt.plot(training_results_relu['training_loss'], label='relu')
plt.ylabel('loss')
plt.title('training loss iterations')
plt.legend()
plt.show()
```

Compare the validation loss for each model:

In [ ]:

```
# Compare the validation loss

plt.plot(training_results_tanch['validation_accuracy'], label='tanh')
plt.plot(training_results['validation_accuracy'], label='sigmoid')
plt.plot(training_results_relu['validation_accuracy'], label='relu')
plt.ylabel('validation accuracy')
plt.xlabel('epochs ')
plt.legend()
plt.show()
```

what activation function performed best ?

[(http://cocl.us/pytorch_link_bottom)](http://cocl.us/pytorch_link_bottom)
(http://cocl.us/pytorch_link_bottom)

(http://cocl.us/pytorch_link_bottom)
(http://cocl.us/pytorch_link_bottom)

(http://cocl.us/pytorch_link_bottom)

(http://cocl.us/pytorch_link_bottom)
(http://cocl.us/pytorch_link_bottom)

About the Authors:
(http://cocl.us/pytorch_link_bottom)

(http://cocl.us/pytorch_link_bottom)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/)](https://www.linkedin.com/in/michelleccarey/), [Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)](www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)
