Test Uniform, Default and Xavier Uniform Initialization on MNIST dataset with tanh activation

Objective for this Notebook

1. Define Several Neural Network, Criterion function, Optimizer
2. Test Uniform, Default and Xavier Initialization

Table of Contents

In this lab, you will test PyTroch Default Initialization, Xavier Initialization and Uniform Initialization on the MNIST dataset.

Estimated Time Needed: **25 min**

Preparation

We'll need the following libraries:

[ ]:
```
# 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

torch.manual_seed(0)
```

Neural Network Module and Training Function

Define the neural network module or class with Xavier Initialization

[ ]:
```
# Define the neural network with Xavier initialization

class Net_Xavier(nn.Module):

------
    # Constructor
    def __init__(self, Layers):
        super(Net_Xavier, self).__init__()
        self.hidden = nn.ModuleList()

        for input_size, output_size in zip(Layers, Layers[1:]):
            linear = nn.Linear(input_size, output_size)
            torch.nn.init.xavier_uniform_(linear.weight)
            self.hidden.append(linear)

------
    # Prediction
    def forward(self, x):
        L = len(self.hidden)
        for (l, linear_transform) in zip(range(L), self.hidden):
            if l < L - 1:
                x = torch.tanh(linear_transform(x))
            else:
                x = linear_transform(x)
        return x
```

Define the neural network module with Uniform Initialization:

[ ]:
```
# Define the neural network with Uniform initialization

class Net_Uniform(nn.Module):

------
    # Constructor
    def __init__(self, Layers):
        super(Net_Uniform, self).__init__()
        self.hidden = nn.ModuleList()

        for input_size, output_size in zip(Layers, Layers[1:]):
            linear = nn.Linear(input_size, output_size)
```

```
            linear.weight.data.uniform_(0, 1)
            self.hidden.append(linear)

    # Prediction
    def forward(self, x):
        L = len(self.hidden)
        for (l, linear_transform) in zip(range(L), self.hidden):
            if l < L - 1:
                x = torch.tanh(linear_transform(x))
            else:
                x = linear_transform(x)
        return x
```

Define the neural network module with PyTroch Default Initialization

```
[ ]: # Define the neural network with Default initialization

class Net(nn.Module):

    # Constructor
    def __init__(self, Layers):
        super(Net, self).__init__()
        self.hidden = nn.ModuleList()

        for input_size, output_size in zip(Layers, Layers[1:]):
            linear = nn.Linear(input_size, output_size)
            self.hidden.append(linear)

    # Prediction
    def forward(self, x):
        L = len(self.hidden)
        for (l, linear_transform) in zip(range(L), self.hidden):
            if l < L - 1:
                x = torch.tanh(linear_transform(x))
            else:
                x = linear_transform(x)
        return x
```

Define a function to train the model, in this case the function returns a Python dictionary to store the training loss and accuracy on the validation data

```
[ ]: # function to Train the model

def train(model, criterion, train_loader, validation_loader, optimizer, epochs = 100):
    i = 0
    loss_accuracy = {'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()
            loss_accuracy['training_loss'].append(loss.data.item())

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

    return loss_accuracy
```

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 int the argument `transform`

```
[ ]: # Create the train dataset

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

Load the testing dataset by setting the parameters `train` to `False` and convert it to a tensor by placing a transform object int the argument `transform`

```
[ ]: # Create the validation dataset

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

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

```
[ ]: # Create Dataloader for both train dataset and validation dataset

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

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

Create the criterion function

```
[ ]: # Define criterion function

criterion = nn.CrossEntropyLoss()
```

Create the model with 100 hidden layers

```
[ ]: # Set the parameters

input_dim = 28 * 28
output_dim = 10
layers = [input_dim, 100, 10, 100, 10, 100, output_dim]
epochs = 15
```

Test PyTorch Default Initialization, Xavier Initialization, Uniform Initialization

Train the network using PyTorch Default Initialization

[ ]:
```
# Train the model with default initialization

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

Train the network using Xavier Initialization function

[ ]:
```
# Train the model with Xavier initialization

model_Xavier = Net_Xavier(layers)
optimizer = torch.optim.SGD(model_Xavier.parameters(), lr=learning_rate)
training_results_Xavier = train(model_Xavier, criterion, train_loader, validation_loader, optimizer, epochs=epochs)
```

Train the network using Uniform Initialization

[ ]:
```
# Train the model with Uniform initialization

model_Uniform = Net_Uniform(layers)
optimizer = torch.optim.SGD(model_Uniform.parameters(), lr=learning_rate)
training_results_Uniform = train(model_Uniform, criterion, train_loader, validation_loader, optimizer, epochs=epochs)
```

Analyse Results

Compare the training loss for each initialization

[ ]:
```
# Plot the loss

plt.plot(training_results_Xavier['training_loss'], label='Xavier')
plt.plot(training_results['training_loss'], label='Default')
plt.plot(training_results_Uniform['training_loss'], label='Uniform')
plt.ylabel('loss')
plt.xlabel('iteration ')
plt.title('training loss iterations')
plt.legend()
```

compare the validation loss for each model

[ ]:
```
# Plot the accuracy

plt.plot(training_results_Xavier['validation_accuracy'], label='Xavier')
plt.plot(training_results['validation_accuracy'], label='Default')
plt.plot(training_results_Uniform['validation_accuracy'], label='Uniform')
plt.ylabel('validation accuracy')
plt.xlabel('epochs')
plt.legend()
```

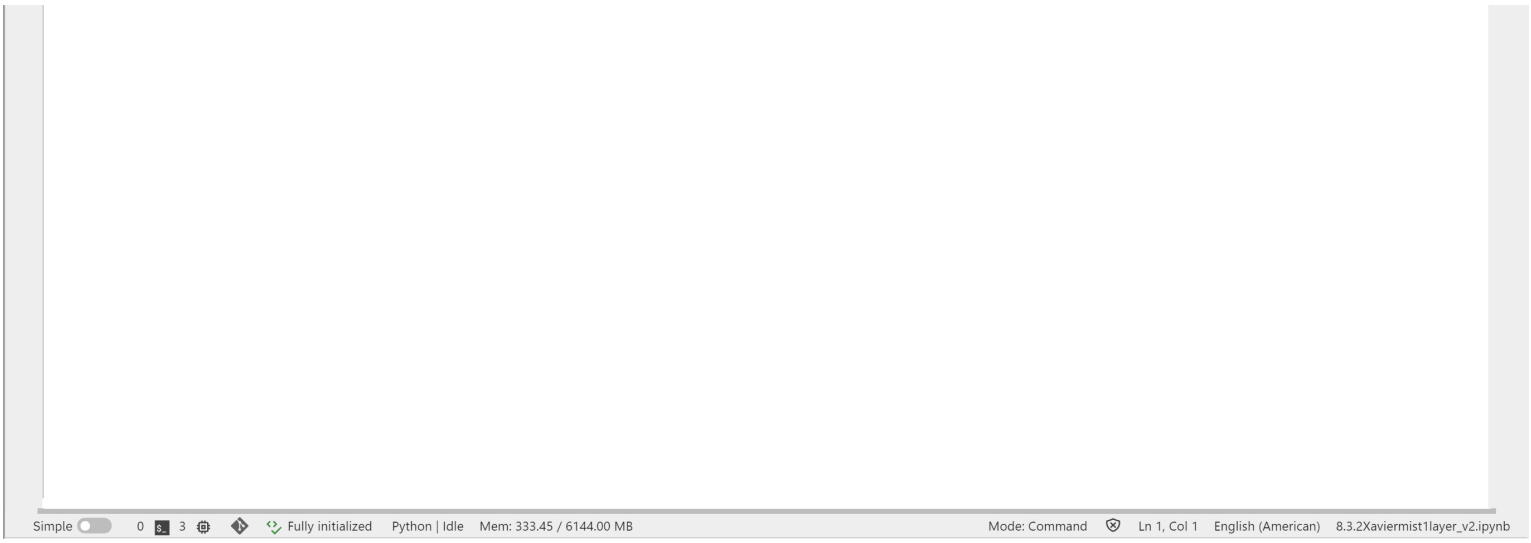

About the Authors:

Joseph Santarcangelo 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, Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-23	2.0	Srishti	Migrated Lab to Markdown and added to course repo in GitLab


Simple
0
3
Fully initialized
Python | Idle
Mem: 333.45 / 6144.00 MB
Mode: Command
Ln 1, Col 1
English (American)
8.3.2Xaviermist1layer_v2.ipynb