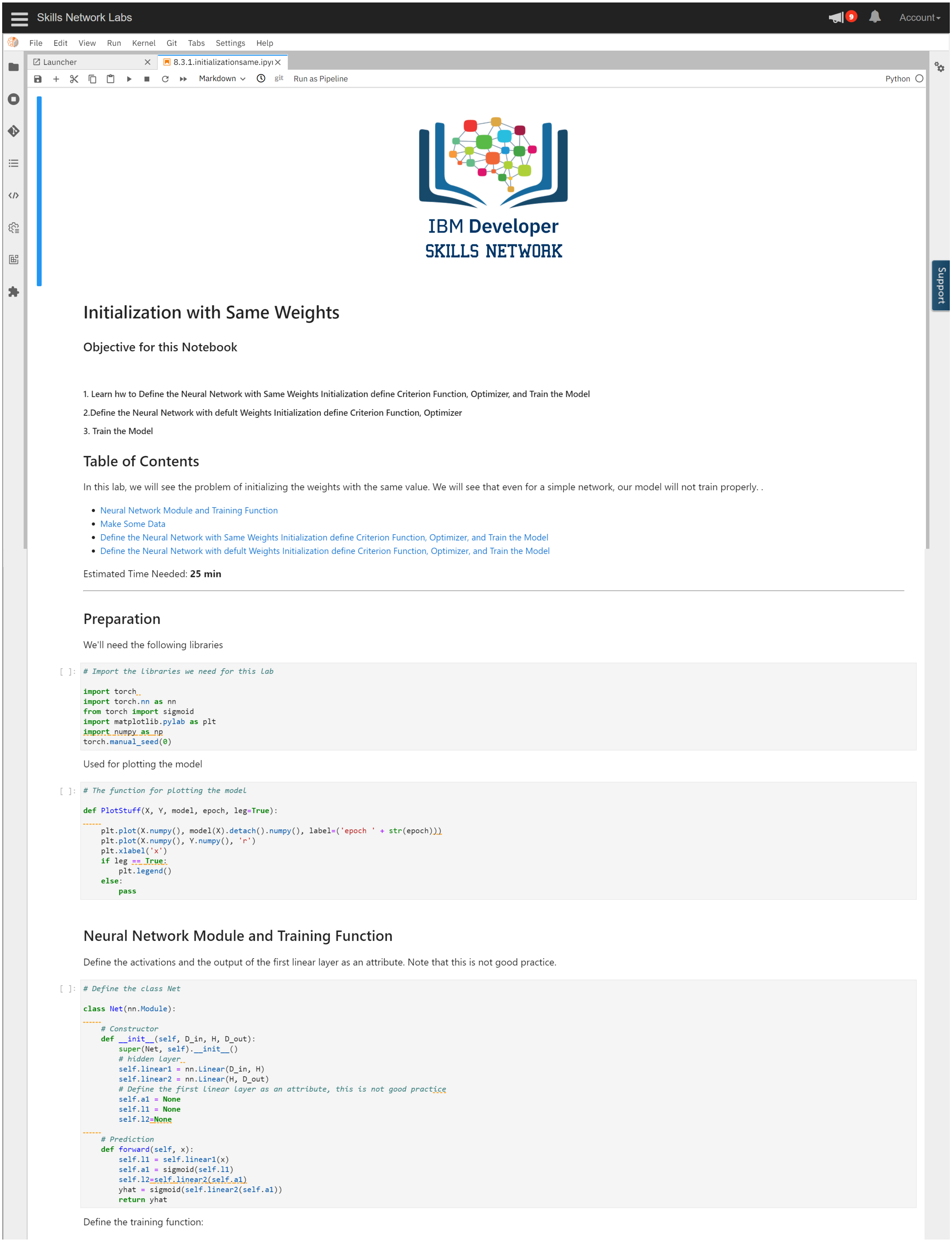

Skills Network Labs
9
Account
File Edit View Run Kernel Git Tabs Settings Help
Launcher
8.3.1.initializationsame.ipyı
Markdown
git
Run as Pipeline
Python
Support

IBM Developer
SKILLS NETWORK

Initialization with Same Weights

Objective for this Notebook

1. Learn hw to Define the Neural Network with Same Weights Initialization define Criterion Function, Optimizer, and Train the Model

2.Define the Neural Network with defult Weights Initialization define Criterion Function, Optimizer

3. Train the Model

Table of Contents

In this lab, we will see the problem of initializing the weights with the same value. We will see that even for a simple network, our model will not train properly. .

Estimated Time Needed: **25 min**

Preparation

We'll need the following libraries

```
[ ]: # Import the libraries we need for this lab

import torch 
import torch.nn as nn
from torch import sigmoid
import matplotlib.pylab as plt
import numpy as np
torch.manual_seed(0)
```

Used for plotting the model

```
[ ]: # The function for plotting the model

def PlotStuff(X, Y, model, epoch, leg=True):
    
    plt.plot(X.numpy(), model(X).detach().numpy(), label=('epoch ' + str(epoch)))
    plt.plot(X.numpy(), Y.numpy(), 'r')
    plt.xlabel('x')
    if leg == True:
        plt.legend()
    else:
        pass
```

Neural Network Module and Training Function

Define the activations and the output of the first linear layer as an attribute. Note that this is not good practice.

```
[ ]: # Define the class Net

class Net(nn.Module):
    
    # Constructor
    def __init__(self, D_in, H, D_out):
        super(Net, self).__init__()
        # hidden layer 
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)
        # Define the first linear layer as an attribute, this is not good practice
        self.a1 = None
        self.l1 = None
        self.l2=None
    
    # Prediction
    def forward(self, x):
        self.l1 = self.linear1(x)
        self.a1 = sigmoid(self.l1)
        self.l2=self.linear2(self.a1)
        yhat = sigmoid(self.linear2(self.a1))
        return yhat
```

Define the training function:

[ ]:

```
# Define the training function

def train(Y, X, model, optimizer, criterion, epochs=1000):
    cost = []
    total=0
    for epoch in range(epochs):
        total=0
        for y, x in zip(Y, X):
            yhat = model(x)
            loss = criterion(yhat, y)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            #cumulative loss 
            total+=loss.item() 
        cost.append(total)
        if epoch % 300 == 0:    
            PlotStuff(X, Y, model, epoch, leg=True)
            plt.show()
            model(X)
            plt.scatter(model.a1.detach().numpy()[:, 0], model.a1.detach().numpy()[:, 1], c=Y.numpy().reshape(-1))
            plt.title('activations')
            plt.show()
    return cost
```

Make Some Data

[ ]:

```
# Make some data

X = torch.arange(-20, 20, 1).view(-1, 1).type(torch.FloatTensor)
Y = torch.zeros(X.shape[0])
Y[(X[:, 0] > -4) & (X[:, 0] < 4)] = 1.0
```

Define the Neural Network with Same Weights Initialization define, Criterion Function, Optimizer and Train the Model

Create the Cross-Entropy loss function:

[ ]:

```
# The loss function

def criterion_cross(outputs, labels):
    out = -1 * torch.mean(labels * torch.log(outputs) + (1 - labels) * torch.log(1 - outputs))
    return out
```

Define the Neural Network

[ ]:

```
# Train the model
# size of input 
D_in = 1
# size of hidden layer 
H = 2
# number of outputs 
D_out = 1
# learning rate 
learning_rate = 0.1
# create the model 
model = Net(D_in, H, D_out)
```

This is the PyTorch default installation

[ ]:

```
model.state_dict()
```

Same Weights Initialization with all ones for weights and zeros for the bias.

[ ]:

```
model.state_dict()['linear1.weight'][0]=1.0
model.state_dict()['linear1.weight'][1]=1.0
model.state_dict()['linear1.bias'][0]=0.0
model.state_dict()['linear1.bias'][1]=0.0
model.state_dict()['linear2.weight'][0]=1.0
model.state_dict()['linear2.bias'][0]=0.0
model.state_dict()
```

Optimizer, and Train the Model:

[ ]:

```
#optimizer 
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
#train the model usein
cost_cross = train(Y, X, model, optimizer, criterion_cross, epochs=1000)
#plot the loss
plt.plot(cost_cross)
plt.xlabel('epoch')
plt.title('cross entropy loss')
```

By examining the output of the paramters all thought they have changed they are identical.

[ ]:

```
model.state_dict()
```

[ ]:

```
yhat=model(torch.tensor([[-2.0],[0.0],[2.0]]))
yhat
```

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

[ ]:

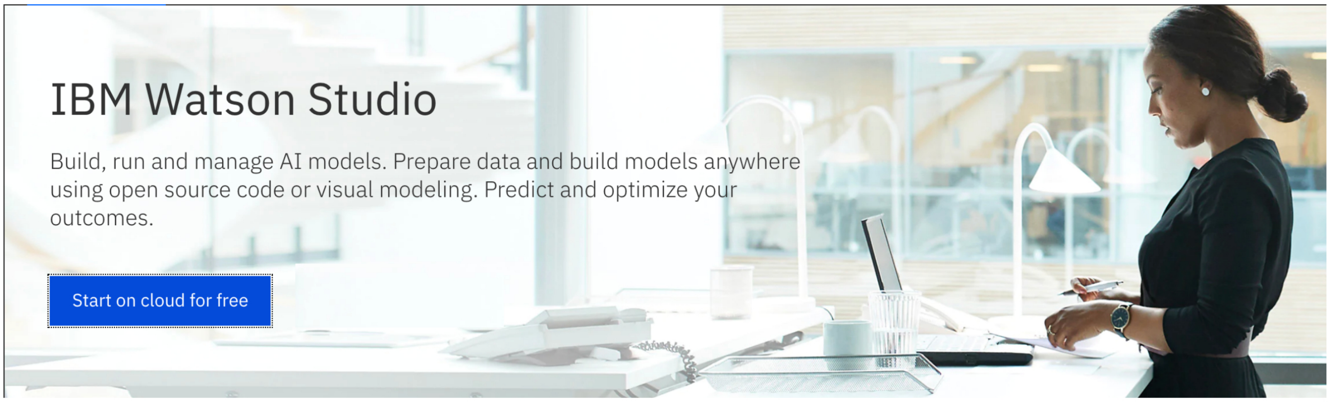
```
# Train the model
# size of input 
D_in = 1
# size of hidden layer 
H = 2
# number of outputs 
D_out = 1
# learning rate 
learning_rate = 0.1
# create the model 
model = Net(D_in, H, D_out)
```

Repeat the previous steps above by using the MSE cost or total loss:

[ ]:

```
#optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
#train the model usein
cost_cross = train(Y, X, model, optimizer, criterion_cross, epochs=1000)
#plot the loss
plt.plot(cost_cross)
plt.xlabel('epoch')
plt.title('cross entropy loss')
```

Double-click here for the solution.


IBM Watson Studio
Build, run and manage AI models. Prepare data and build models anywhere using open source code or visual modeling. Predict and optimize your outcomes.
Start on cloud for free

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 2 Fully initialized Python | Idle Mem: 283.79 / 6144.00 MB Mode: Command Ln 1, Col 1 English (American) 8.3.1.initializationsame.ipynb