Hidden Layer Deep Network: Sigmoid, Tanh and Relu Activations Functions MNIST Dataset

Table of Contents

In this lab, you will test Sigmoid, Tanh and Relu activation functions on the MNIST dataset with two hidden Layers.

Estimated Time Needed: **25 min**

We'll need the following libraries

In [1]:

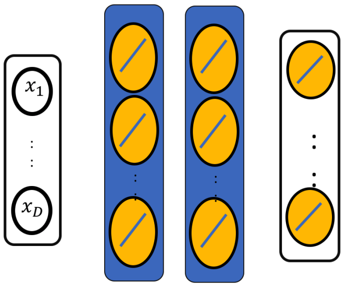
```
# Import the libraries we need for this lab

# 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 torch.nn.functional as F
import matplotlib.pylab as plt
import numpy as np
torch.manual_seed(2)
```

Out[1]:

```
<torch._C.Generator at 0x7f7f98081130>
```

Neural Network Module and Training Function

Define the neural network module or class, with two hidden Layers



2 Hidden Layers

In [2]:

```
# Create the model class using sigmoid as the activation function

class Net(nn.Module):

    # Constructor
    def __init__(self, D_in, H1, H2, D_out):
        super(Net, self).__init__()
        self.linear1 = nn.Linear(D_in, H1)
        self.linear2 = nn.Linear(H1, H2)
        self.linear3 = nn.Linear(H2, D_out)

    # Prediction
    def forward(self,x):
        x = torch.sigmoid(self.linear1(x))
        x = torch.sigmoid(self.linear2(x))
        x = self.linear3(x)
        return x
```

Define the class with the Tanh activation function

In [3]:

```
# Create the model class using Tanh as a activation function

class NetTanh(nn.Module):

    # Constructor
    def __init__(self, D_in, H1, H2, D_out):
        super(NetTanh, self).__init__()
        self.linear1 = nn.Linear(D_in, H1)
        self.linear2 = nn.Linear(H1, H2)
        self.linear3 = nn.Linear(H2, D_out)

    # Prediction
    def forward(self, x):
        x = torch.tanh(self.linear1(x))
        x = torch.tanh(self.linear2(x))
        x = self.linear3(x)
        return x
```

Define the class for the Relu activation function

In [4]:

```
# Create the model class using Relu as a activation function

class NetRelu(nn.Module):

    # Constructor
    def __init__(self, D_in, H1, H2, D_out):
        super(NetRelu, self).__init__()
        self.linear1 = nn.Linear(D_in, H1)
        self.linear2 = nn.Linear(H1, H2)
        self.linear3 = nn.Linear(H2, D_out)

    # Prediction
    def forward(self, x):
        x = torch.relu(self.linear1(x))
        x = torch.relu(self.linear2(x))
        x = self.linear3(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

In [5]:

```
# Train the model

def train(model, criterion, train_loader, validation_loader, optimizer, epochs=100
):
    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.data.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 int the argument `transform`

In [6]:

```
# Create the training 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`

In [7]:

```
# Create the validating dataset

validation_dataset = dsets.MNIST(root='./data', train=False, download=True, transform=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_size=5000, shuffle=False)
```

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

Create the model with 100 hidden layers

In [10]:

```
# Set the parameters for create the model

input_dim = 28 * 28
hidden_dim1 = 50
hidden_dim2 = 50
output_dim = 10
```

The epoch number in the video is 35. You can try 10 for now. If you try 35, it may take a long time.

In [11]:

```
# Set the number of iterations

cust_epochs = 10
```

Test Sigmoid ,Tanh and Relu

Train the network using the Sigmoid activation function

In [ ]:

```
# Train the model with sigmoid function

learning_rate = 0.01
model = Net(input_dim, hidden_dim1, hidden_dim2, output_dim)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
training_results = train(model, criterion, train_loader, validation_loader, optimiz
er, epochs=cust_epochs)
```

Train the network using the Tanh activation function

In [ ]:

```
# Train the model with tanh function

learning_rate = 0.01
model_Tanh = NetTanh(input_dim, hidden_dim1, hidden_dim2, 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=cust_epochs)
```

Train the network using the Relu activation function

In [ ]:

```
# Train the model with relu function

learning_rate = 0.01
modelRelu = NetRelu(input_dim, hidden_dim1, hidden_dim2, output_dim)
optimizer = torch.optim.SGD(modelRelu.parameters(), lr=learning_rate)
training_results_relu = train(modelRelu, criterion, train_loader, validation_loader
, optimizer, epochs=cust_epochs)
```

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()
```

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('Iteration')
plt.legend()
```

(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/), Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)