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)

Softmax Classifer 1D

Table of Contents

In this lab, you will use Softmax to classify three linearly separable classes, the features are in one dimension

Estimated Time Needed: **25 min**

Preparation

We'll need the following libraries:

In [1]:

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

import torch.nn as nn
import torch
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import Dataset, DataLoader
```

Use the helper function to plot labeled data points:

In [2]:

```
# Create class for plotting

def plot_data(data_set, model = None, n = 1, color = False):
    X = data_set[:][0]
    Y = data_set[:][1]
    plt.plot(X[Y == 0, 0].numpy(), Y[Y == 0].numpy(), 'bo', label = 'y = 0')
    plt.plot(X[Y == 1, 0].numpy(), 0 * Y[Y == 1].numpy(), 'ro', label = 'y = 1')
    plt.plot(X[Y == 2, 0].numpy(), 0 * Y[Y == 2].numpy(), 'go', label = 'y = 2')
    plt.ylim((-0.1, 3))
    plt.legend()
    if model != None:
        w = list(model.parameters())[0][0].detach()
        b = list(model.parameters())[1][0].detach()
        y_label = ['yhat=0', 'yhat=1', 'yhat=2']
        y_color = ['b', 'r', 'g']
        Y = []
        for w, b, y_l, y_c in zip(model.state_dict()['0.weight'], model.state_dict
()['0.bias'], y_label, y_color):
            Y.append((w * X + b).numpy())
            plt.plot(X.numpy(), (w * X + b).numpy(), y_c, label = y_l)
        if color == True:
            x = X.numpy()
            x = x.reshape(-1)
            top = np.ones(x.shape)
            y0 = Y[0].reshape(-1)
            y1 = Y[1].reshape(-1)
            y2 = Y[2].reshape(-1)
            plt.fill_between(x, y0, where = y1 > y1, interpolate = True, color = 'b
lue')
            plt.fill_between(x, y0, where = y1 > y2, interpolate = True, color = 'b
lue')
            plt.fill_between(x, y1, where = y1 > y0, interpolate = True, color = 'r
ed')
            plt.fill_between(x, y1, where = ((y1 > y2) * (y1 > y0)),interpolate = T
rue, color = 'red')
            plt.fill_between(x, y2, where = (y2 > y0) * (y0 > 0),interpolate = True
, color = 'green')
            plt.fill_between(x, y2, where = (y2 > y1), interpolate = True, color =
'green')
    plt.legend()
    plt.show()
```

Set the random seed:

In [3]:

```
#Set the random seed

torch.manual_seed(0)
```

Out[3]:

```
<torch._C.Generator at 0x7f09f404d090>
```

Make Some Data

Create some linearly separable data with three classes:

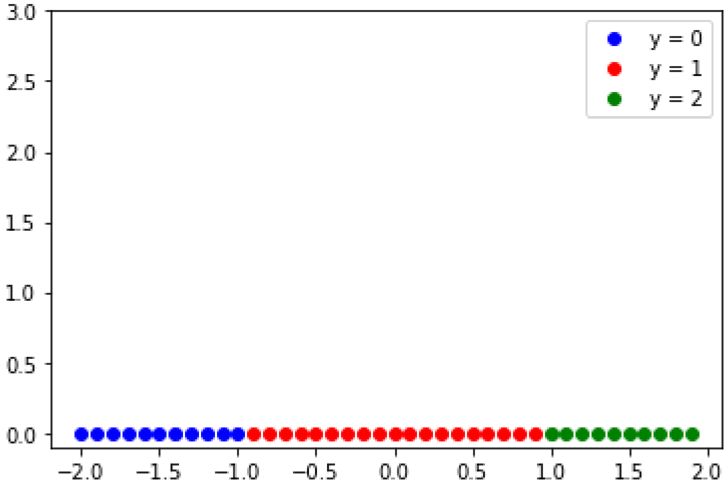
In [4]:

```
# Create the data class

class Data(Dataset):

    # Constructor
    def __init__(self):
        self.x = torch.arange(-2, 2, 0.1).view(-1, 1)
        self.y = torch.zeros(self.x.shape[0])
        self.y[(self.x > -1.0)[:, 0] * (self.x < 1.0)[:, 0]] = 1
        self.y[(self.x >= 1.0)[:, 0]] = 2
        self.y = self.y.type(torch.LongTensor)
        self.len = self.x.shape[0]

    # Getter
    def __getitem__(self,index):
        return self.x[index], self.y[index]

    # Get Length
    def __len__(self):
        return self.len
```

Create the dataset object:

In [5]:

```
# Create the dataset object and plot the dataset object

data_set = Data()
data_set.x
plot_data(data_set)
```



Build a Softmax Classifier

Build a Softmax classifier by using the Sequential module:

In [6]:

```
# Build Softmax Classifier technically you only need nn.Linear

model = nn.Sequential(nn.Linear(1, 3))
model.state_dict()
```

Out[6]:

```
OrderedDict([('0.weight',
              tensor([[-0.0075],
                      [ 0.5364],
                      [-0.8230]])),
             ('0.bias', tensor([-0.7359, -0.3852,  0.2682]))])
```

Train the Model

Create the criterion function, the optimizer and the dataloader

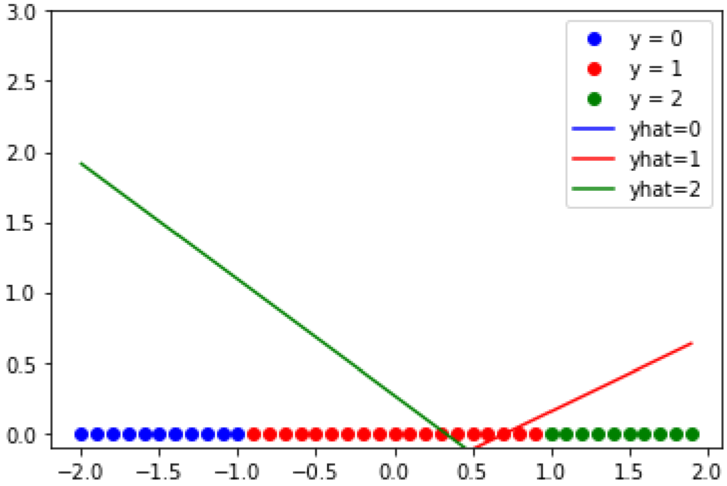
In [7]:

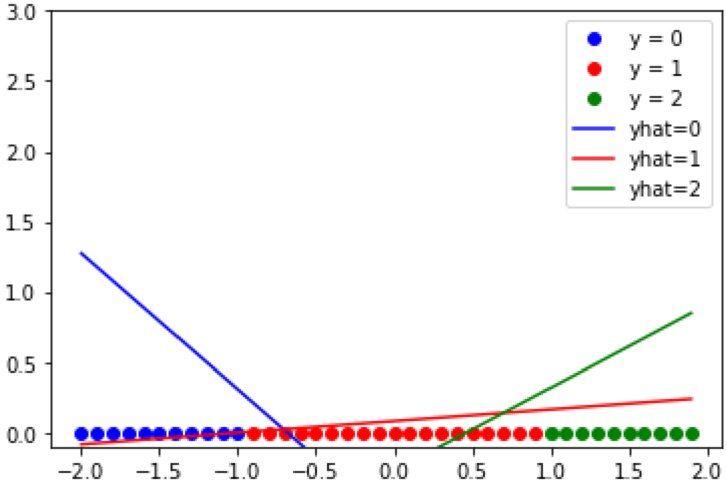
```
# Create criterion function, optimizer, and dataloader

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)
trainloader = DataLoader(dataset = data_set, batch_size = 5)
```

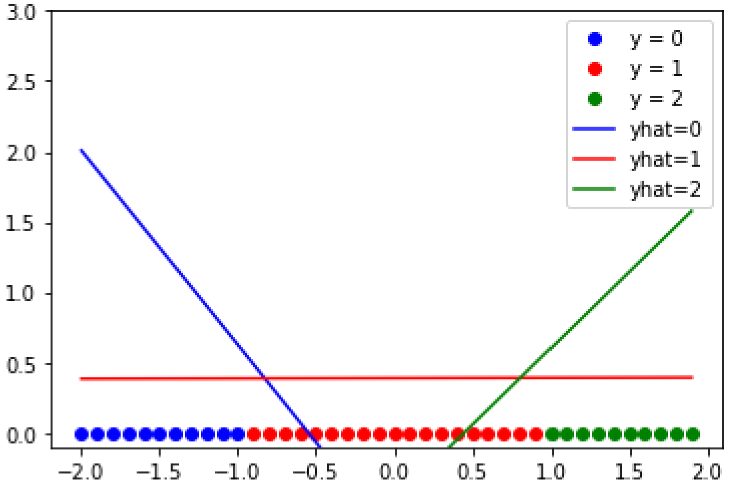
Train the model for every 50 epochs plot, the line generated for each class.

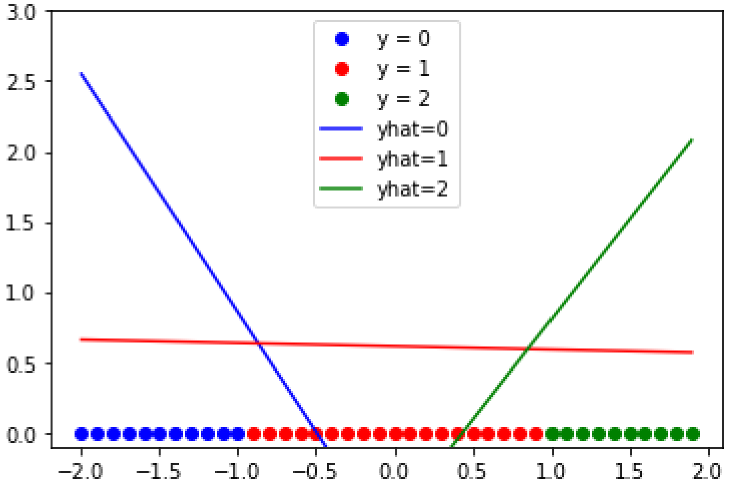
In [8]:

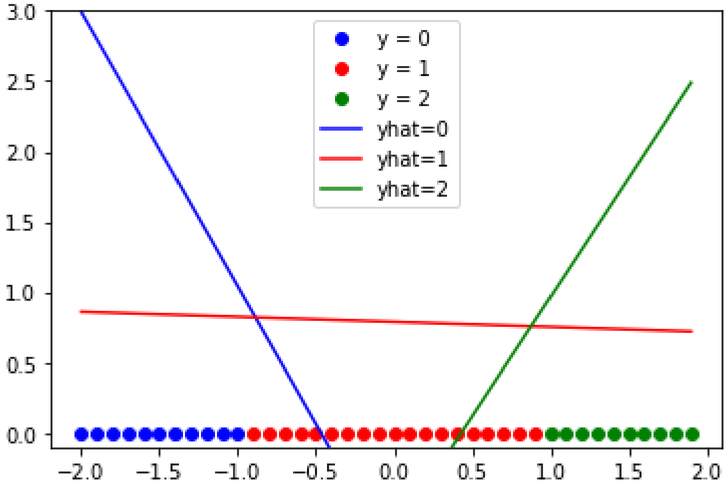
```
# Train the model

LOSS = []
def train_model(epochs):
    for epoch in range(epochs):
        if epoch % 50 == 0:
            pass
            plot_data(data_set, model)
        for x, y in trainloader:
            optimizer.zero_grad()
            yhat = model(x)
            loss = criterion(yhat, y)
            LOSS.append(loss)
            loss.backward()
            optimizer.step()
train_model(300)
```

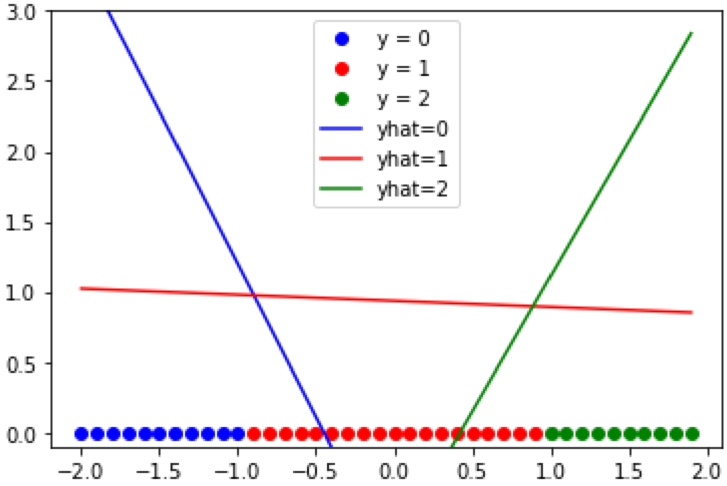

3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
● y = 0
● y = 1
● y = 2
yhat=0
yhat=1
yhat=2


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
● y = 0
● y = 1
● y = 2
yhat=0
yhat=1
yhat=2


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
y = 0
y = 1
y = 2
yhat=0
yhat=1
yhat=2


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
y = 0
y = 1
y = 2
yhat=0
yhat=1
yhat=2





Analyze Results

Find the predicted class on the test data:

In [9]:

```
# Make the prediction

z =  model(data_set.x)
_, yhat = z.max(1)
print("The prediction:", yhat)
```

```
The prediction: tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Calculate the accuracy on the test data:

In [10]:

```
# Print the accuracy

correct = (data_set.y == yhat).sum().item()
accuracy = correct / len(data_set)
print("The accuracy: ", accuracy)
```

```
The accuracy:  0.975
```

You can also use the softmax function to convert the output to a probability,first, we create a Softmax object:

In [11]:

```
Softmax_fn=nn.Softmax(dim=-1)
```

The result is a tensor `Probability` , where each row corresponds to a different sample, and each column corresponds to that sample belonging to a particular class

In [12]:

```
Probability =Softmax_fn(z)
```

we can obtain the probability of the first sample belonging to the first, second and third class respectively as follows:

In [13]:

```
for i in range(3):
    print("probability of class {} isg given by  {}".format(i, Probability[0,i]) )
```

```
probability of class 0 isg given by  0.9267547726631165
probability of class 1 isg given by  0.07310982048511505
probability of class 2 isg given by  0.00013548212882597
```

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)
