Softmax Classifer 1D

Objetivo

- Construir un clasificador Softmax.

Tabla de contenido

Usará el clasificador softmax para clasificar clases linealmente separables, las caracerísticas están en 1D

Preparación

In [14]:
```python
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

In [15]:
```python
# 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
```

Función ayudante para graficar puntos de datos:

In [16]:
```python
# 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()[
            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 = 'blue
            plt.fill_between(x, y0, where = y1 > y2, interpolate = True, color = 'blue
            plt.fill_between(x, y1, where = y1 > y0, interpolate = True, color = 'red
            plt.fill_between(x, y1, where = ((y1 > y2) * (y1 > y0)),interpolate = True
            plt.fill_between(x, y2, where = (y2 > y0) * (y0 > 0),interpolate = True, c
            plt.fill_between(x, y2, where = (y2 > y1), interpolate = True, color = 'gr
    plt.legend()
    plt.show()
```

Establecemos la semilla:

In [17]:
```python
#Set the random seed

torch.manual_seed(0)
```

Out[17]:
```
<torch._C.Generator at 0x1d3a6d6e270>
```

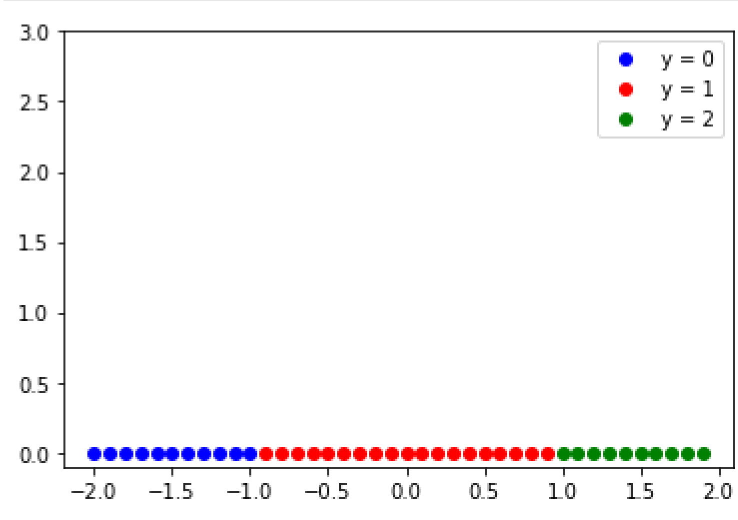
Crear algunos datos

Creamos 3 clases linealmente separables:

In [18]:
```python
# 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
```

Creamos el objeto dataset:

In [19]:
```python
# Create the dataset object and plot the dataset object

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



Construimos el clasificador softmax

Construimos el clasificador softmax usando el módulo Sequential:

In [20]:
```python
# Build Softmax Classifier technically you only need nn.Linear

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

Out[20]:
```
OrderedDict([('0.weight',
              tensor([[-0.0075],
                      [ 0.5364],
                      [-0.8230]])),
             ('0.bias', tensor([-0.7359, -0.3852,  0.2682]))])
```

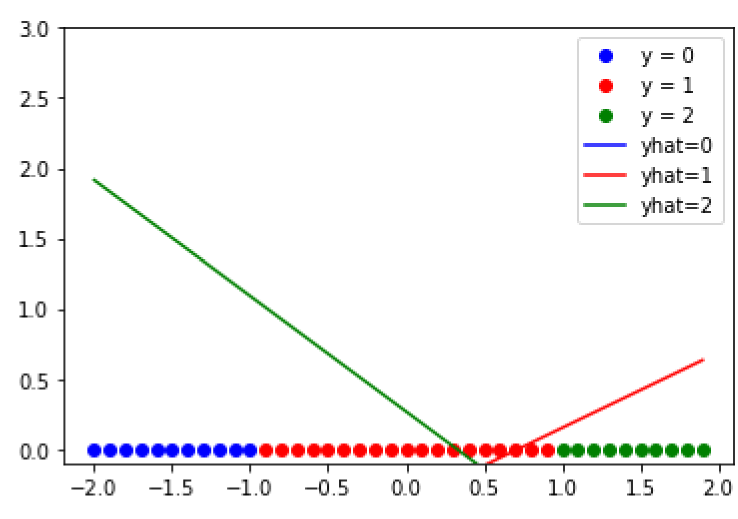
Entrenamos el modelo

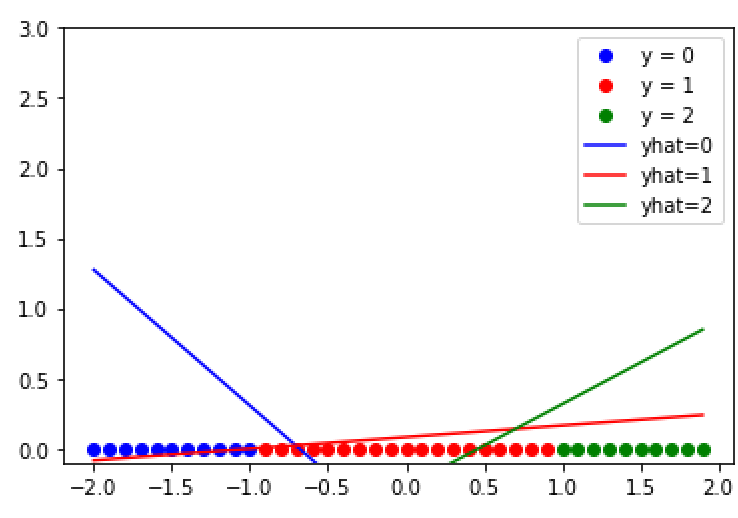
Creamos la función de criterio, el optimizador y el dataloader:

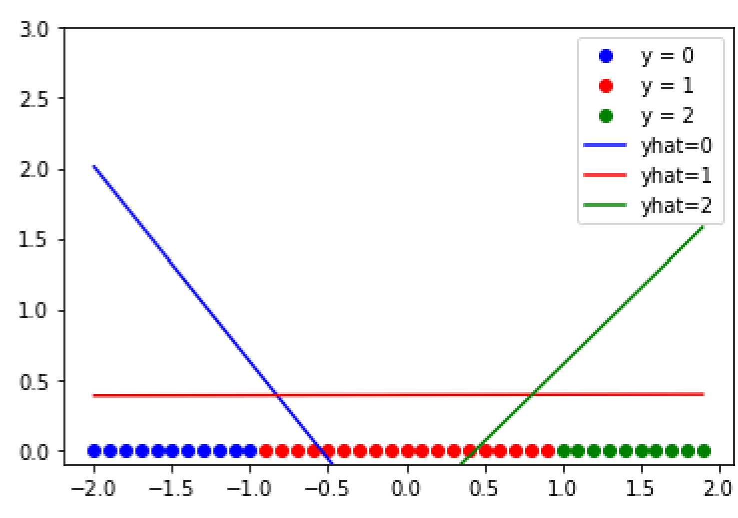
In [21]:
```python
# Create criterion function, optimizer, and dataloader

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

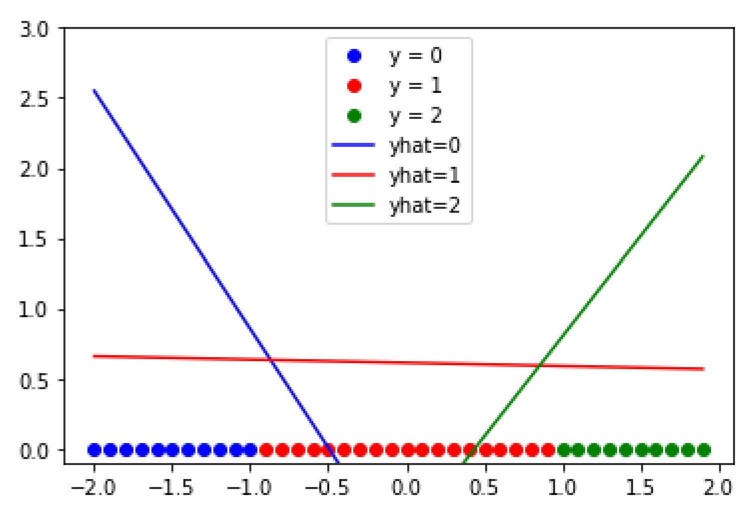
Entrenamos el modelo, por cada 50 epochs graficamos la recta generada por cada clase.

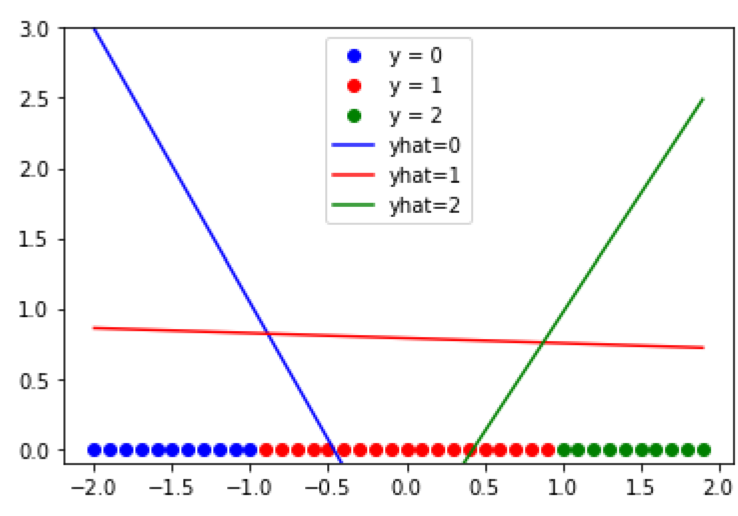
In [22]:
```python
# Train the model

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)
```

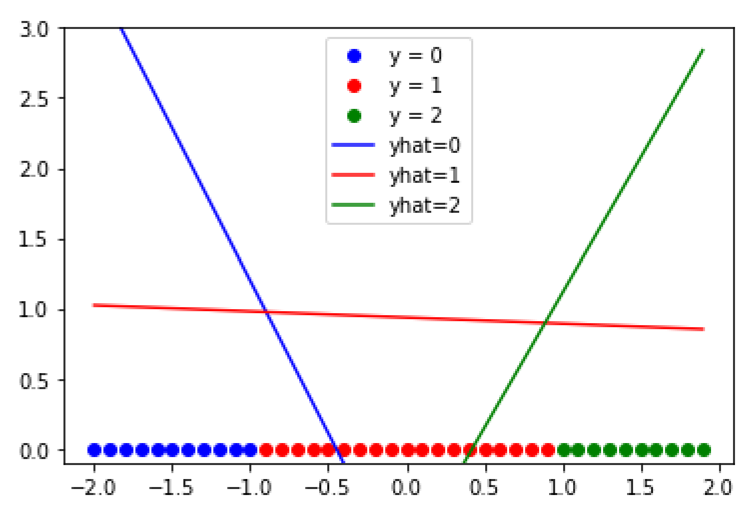












Analizamos los resultados

Encontramos la clase predicha sobre los datos de test:

In [23]:
```python
# 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])
```

Calculamos la precisión sobre los datos de test:

In [24]:
```python
# Print the accuracy

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

```
The accuracy:  0.975
```

Puede usar la función softmax para convertir la salida a una probabilidad; primero, creamos un objeto softmax:

In [25]:
```python
Softmax_fn=nn.Softmax(dim=-1)
```

El resultado es un tensor `Probability`, donde cada fila corresponde a una muestra diferente y cada columna a esa muestra perteneciendo a una clase particular.

In [26]:
```python
Probability =Softmax_fn(z)
```

podemos obtener la probabilidad de que la primer muestra pertenezca a la primer, segunda y tercera clase respectivamente como sigue:

In [27]:
```python
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
```