IBM Developer
SKILLS NETWORK

Logistic Regression

Objective

- How to create a logistic regression object with the nn.Sequential model.

Table of Contents

In this lab, we will cover logistic regression using PyTorch.

Estimated Time Needed: **15 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
```

Set the random seed:

In [2]:

```
# Set the random seed

torch.manual_seed(2)
```

Out[2]:

```
<torch._C.Generator at 0x146bd1542f0>
```

Logistic Function

Create a tensor ranging from -100 to 100:

In [3]:

```
z = torch.arange(-100, 100, 0.1).view(-1, 1)
print("The tensor: ", z)
```

```
The tensor:  tensor([[-100.0000],
        [ -99.9000],
        [ -99.8000],
        ...,
        [  99.7000],
        [  99.8000],
        [  99.9000]])
```

Create a sigmoid object:

In [4]:

```
# Create sigmoid object

sig = nn.Sigmoid()
```

Apply the element-wise function Sigmoid with the object:

In [5]:

```
# Use sigmoid object to calculate the

yhat = sig(z)
```

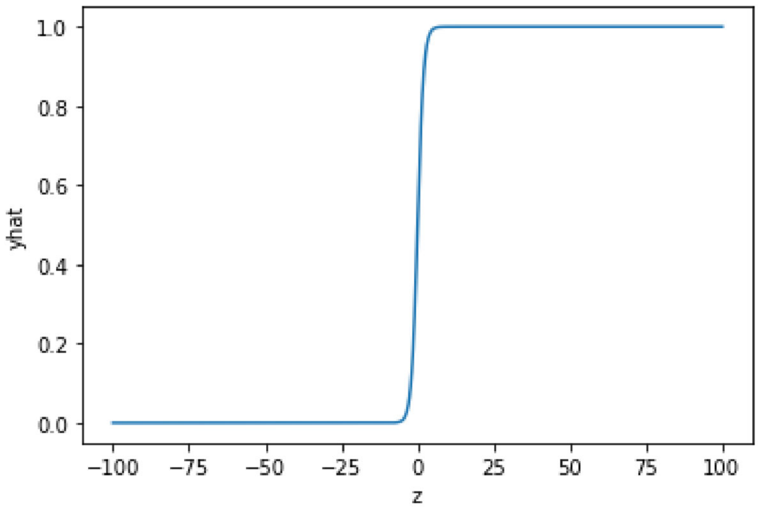
Plot the results:

In [6]:

```
plt.plot(z.numpy(), yhat.numpy())
plt.xlabel('z')
plt.ylabel('yhat')
```

Out[6]:

```
Text(0, 0.5, 'yhat')
```



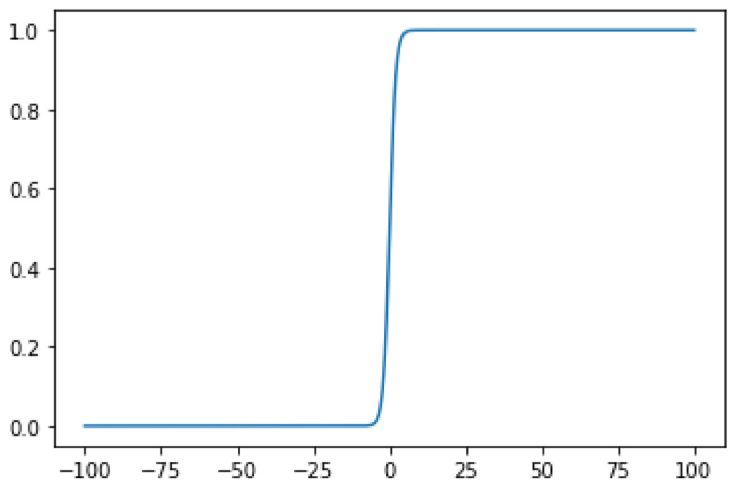
Apply the element-wise Sigmoid from the function module and plot the results:

In [7]:

```
yhat = torch.sigmoid(z)
plt.plot(z.numpy(), yhat.numpy())
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x146c2593d90>]
```



Build a Logistic Regression with `nn.Sequential`

Create a 1x1 tensor where x represents one data sample with one dimension, and 2x1 tensor X represents two data samples of one dimension:

In [8]:

```
# Create x and X tensor

x = torch.tensor([[1.0]])
X = torch.tensor([[1.0], [100]])
print('x = ', x)
print('X = ', X)
```

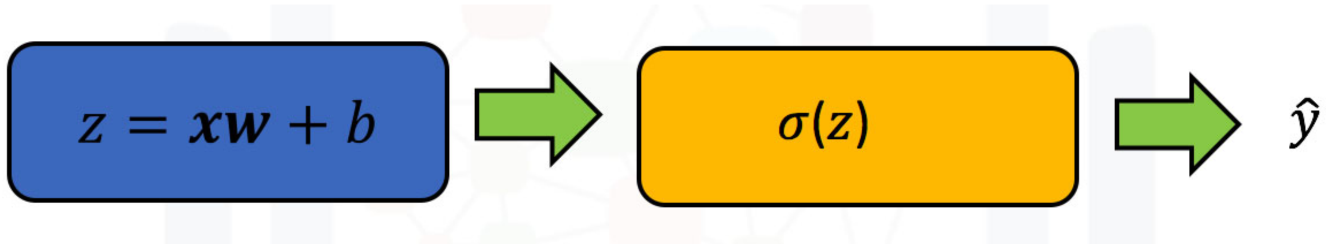
```
x =  tensor([[1.]])
X =  tensor([[  1.],
        [100.]])
```

Create a logistic regression object with the `nn.Sequential` model with a one-dimensional input:

In [9]:

```
# Use sequential function to create model

model = nn.Sequential(nn.Linear(1, 1), nn.Sigmoid())
```

The object is represented in the following diagram:



In this case, the parameters are randomly initialized. You can view them the following ways:

In [10]:

```
# Print the parameters

print("list(model.parameters()):\n ", list(model.parameters()))
print("\nmodel.state_dict():\n ", model.state_dict())
```

```
list(model.parameters()):
  [Parameter containing:
tensor([[0.2294]], requires_grad=True), Parameter containing:
tensor([-0.2380], requires_grad=True)]

model.state_dict():
  OrderedDict([('0.weight', tensor([[0.2294]])), ('0.bias', tensor([-0.2380]))])
```

Make a prediction with one sample:

In [11]:

```
# The prediction for x

yhat = model(x)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([[0.4979]], grad_fn=<SigmoidBackward>)
```

Calling the object with tensor `X` performed the following operation **(code values may not be the same as the diagrams value depending on the version of PyTorch)** :

$$\mathbf{x} = \begin{bmatrix} 1 \\ 100 \end{bmatrix}$$

$$\hat{y} = \sigma(-\mathbf{0.23} + 0.23 \begin{bmatrix} 1 \\ 100 \end{bmatrix})$$

$$\hat{y} = \sigma(\begin{bmatrix} 0 \\ 22 \end{bmatrix})$$

$$\hat{y} = \begin{bmatrix} \sigma(0) \\ \sigma(22) \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

Make a prediction with multiple samples:

In [12]:

```
# The prediction for X

yhat = model(X)
yhat
```

Out[12]:

```
tensor([[0.4979],
        [1.0000]], grad_fn=<SigmoidBackward>)
```

Calling the object performed the following operation:

Create a 1x2 tensor where x represents one data sample with one dimension, and 2x3 tensor X represents one data sample of two dimensions:

In [13]:

```
# Create and print samples

x = torch.tensor([[1.0, 1.0]])
X = torch.tensor([[1.0, 1.0], [1.0, 2.0], [1.0, 3.0]])
print('x = ', x)
print('X = ', X)
```

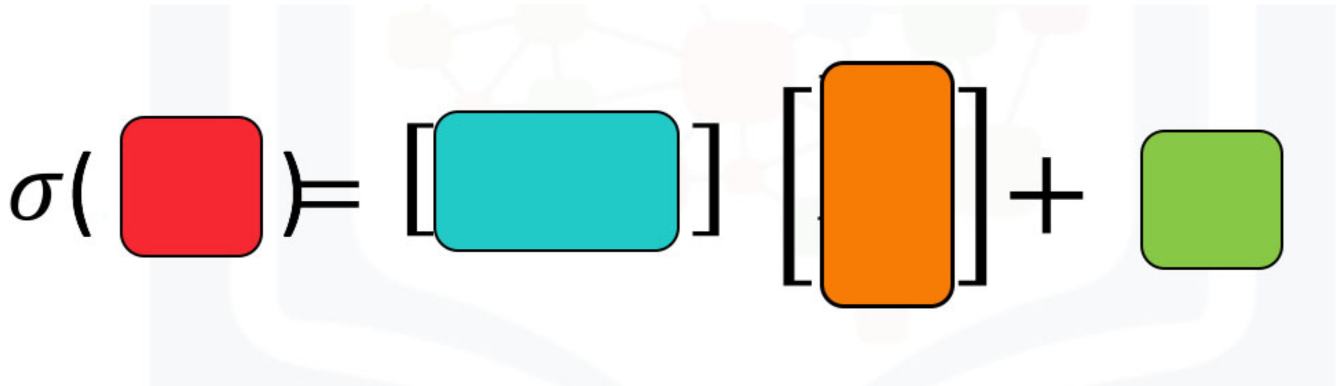
```
x =  tensor([[1., 1.]])
X =  tensor([[1., 1.],
        [1., 2.],
        [1., 3.]])
```

Create a logistic regression object with the `nn.Sequential` model with a two-dimensional input:

In [14]:

```
# Create new model using nn.sequential()

model = nn.Sequential(nn.Linear(2, 1), nn.Sigmoid())
```

The object will apply the Sigmoid function to the output of the linear function as shown in the following diagram:



In this case, the parameters are randomly initialized. You can view them the following ways:

In [15]:

```
# Print the parameters

print("list(model.parameters()):\n ", list(model.parameters()))
print("\nmodel.state_dict():\n ", model.state_dict())
```

```
list(model.parameters()):
  [Parameter containing:
tensor([[ 0.1939, -0.0361]], requires_grad=True), Parameter containing:
tensor([0.3021], requires_grad=True)]

model.state_dict():
  OrderedDict([('0.weight', tensor([[ 0.1939, -0.0361]])), ('0.bias', tensor([0.302
1]))])
```

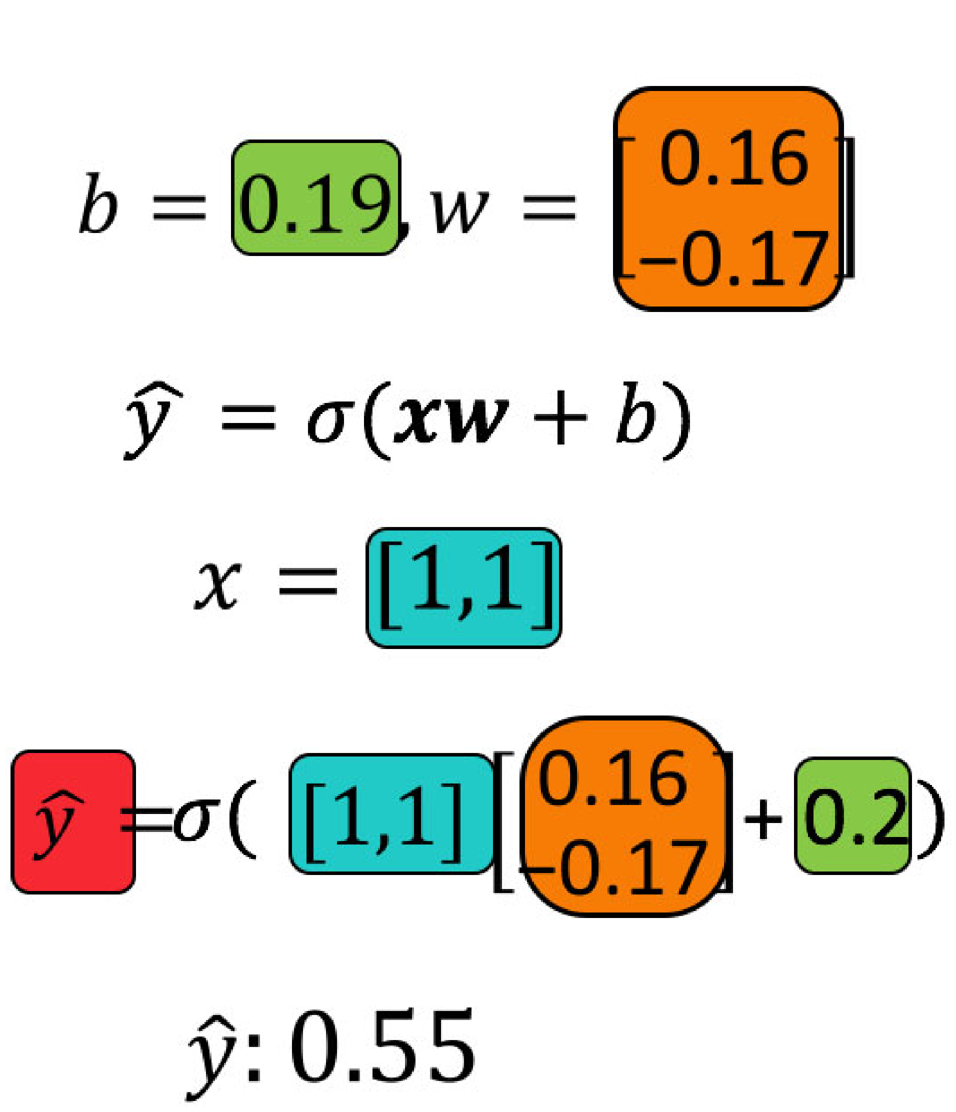
Make a prediction with one sample:

In [16]:

```
# Make the prediction of x

yhat = model(x)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([[0.6130]], grad_fn=<SigmoidBackward>)
```

The operation is represented in the following diagram:



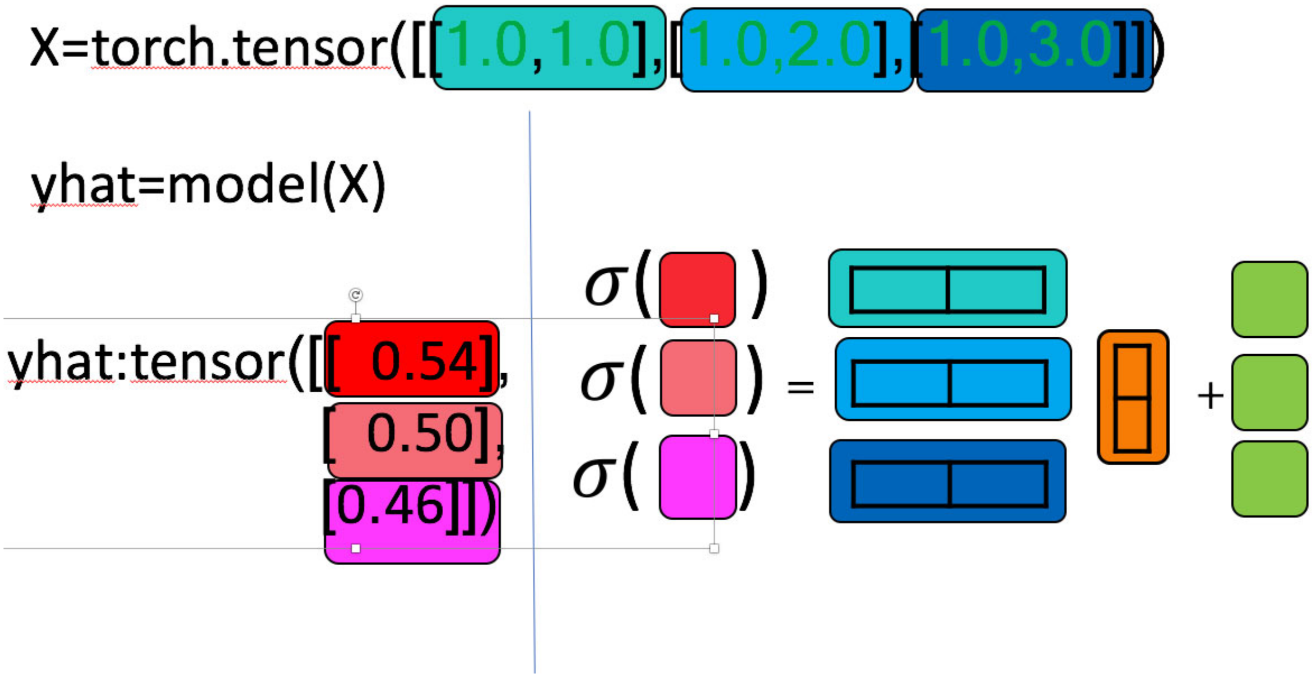
Make a prediction with multiple samples:

In [17]:

```
# The prediction of X

yhat = model(X)
print("The prediction: ", yhat)
```

```
The prediction:  tensor([[0.6130],
        [0.6044],
        [0.5957]], grad_fn=<SigmoidBackward>)
```

The operation is represented in the following diagram:



Build Custom Modules

In this section, you will build a custom Module or class. The model or object function is identical to using `nn.Sequential`.

Create a logistic regression custom module:

In [18]:

```
# Create logistic_regression custom class

class logistic_regression(nn.Module):

    # Constructor
    def __init__(self, n_inputs):
        super(logistic_regression, self).__init__()
        self.linear = nn.Linear(n_inputs, 1)

    # Prediction
    def forward(self, x):
        yhat = torch.sigmoid(self.linear(x))
        return yhat
```

Create a 1x1 tensor where x represents one data sample with one dimension, and 3x1 tensor where X represents one data sample of one dimension:

In [19]:

```
# Create x and X tensor

x = torch.tensor([[1.0]])
X = torch.tensor([[-100], [0], [100.0]])
print('x = ', x)
print('X = ', X)
```

```
x =  tensor([[1.]])
X =  tensor([[-100.],
        [   0.],
        [ 100.]])
```

Create a model to predict one dimension:

In [20]:

```
# Create logistic regression model

model = logistic_regression(1)
```

In this case, the parameters are randomly initialized. You can view them the following ways:

In [21]:

```
# Print parameters

print("list(model.parameters()):\n ", list(model.parameters()))
print("\nmodel.state_dict():\n ", model.state_dict())
```

```
list(model.parameters()):
  [Parameter containing:
tensor([[0.2381]], requires_grad=True), Parameter containing:
tensor([-0.1149], requires_grad=True)]

model.state_dict():
  OrderedDict([('linear.weight', tensor([[0.2381]])), ('linear.bias', tensor([-0.114
9]))])
```

Make a prediction with one sample:

In [22]:

```
# Make the prediction of x

yhat = model(x)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[0.5307]], grad_fn=<SigmoidBackward>)
```

Make a prediction with multiple samples:

In [23]:

```
# Make the prediction of X

yhat = model(X)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[4.0805e-11],
        [4.7130e-01],
        [1.0000e+00]], grad_fn=<SigmoidBackward>)
```

Create a logistic regression object with a function with two inputs:

In [24]:

```
# Create logistic regression model

model = logistic_regression(2)
```

Create a 1x2 tensor where x represents one data sample with one dimension, and 3x2 tensor X represents one data sample of one dimension:

In [25]:

```
# Create x and X tensor

x = torch.tensor([[1.0, 2.0]])
X = torch.tensor([[100, -100], [0.0, 0.0], [-100, 100]])
print('x = ', x)
print('X = ', X)
```

```
x =  tensor([[1., 2.]])
X =  tensor([[ 100., -100.],
        [   0.,    0.],
        [-100.,  100.]])
```

Make a prediction with one sample:

In [26]:

```
# Make the prediction of x

yhat = model(x)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[0.2943]], grad_fn=<SigmoidBackward>)
```

Make a prediction with multiple samples:

In [27]:

```
# Make the prediction of X

yhat = model(X)
print("The prediction result: \n", yhat)
```

```
The prediction result:
 tensor([[7.7529e-33],
        [3.4841e-01],
        [1.0000e+00]], grad_fn=<SigmoidBackward>)
```

Practice

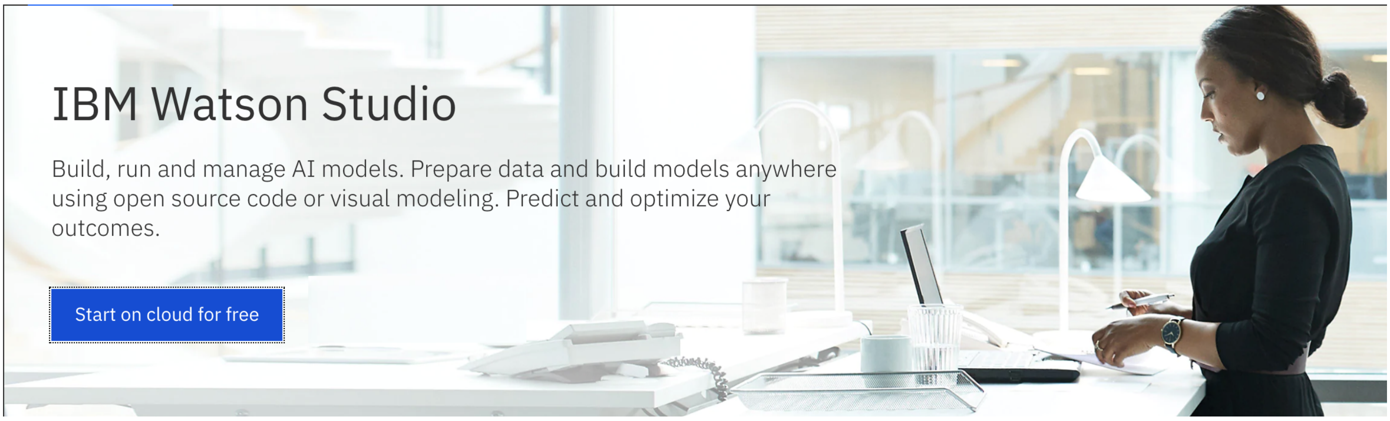
Make your own model `my_model` as applying linear regression first and then logistic regression using `nn.Sequential()` . Print out your prediction.

In [30]:

```
# Practice: Make your model and make the prediction

X = torch.tensor([-10.0])

class my_model(nn.Module):
    def __init__(self, input_size, output_size=1):
        super(my_model, self).__init__()
        self.model = nn.Sequential(nn.Linear(input_size, output_size), nn.Sigmoid())

    def forward(self, x):
        yhat = self.model(x)
        return yhat

mymodel = my_model(1,1)
Yhat = mymodel(X)
print(Yhat)
```

```
tensor([0.2231], grad_fn=<SigmoidBackward>)
```

Double-click **here** for the solution.



(https://dataplatform.cloud.ibm.com/registration/stepone?context=cpdaas&apps=data_science_experience,watson_machine_learning)

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)

Change Log

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

---