

PPGEEC2318

Machine Learning

Rethinking the training loop: a simple classification problem

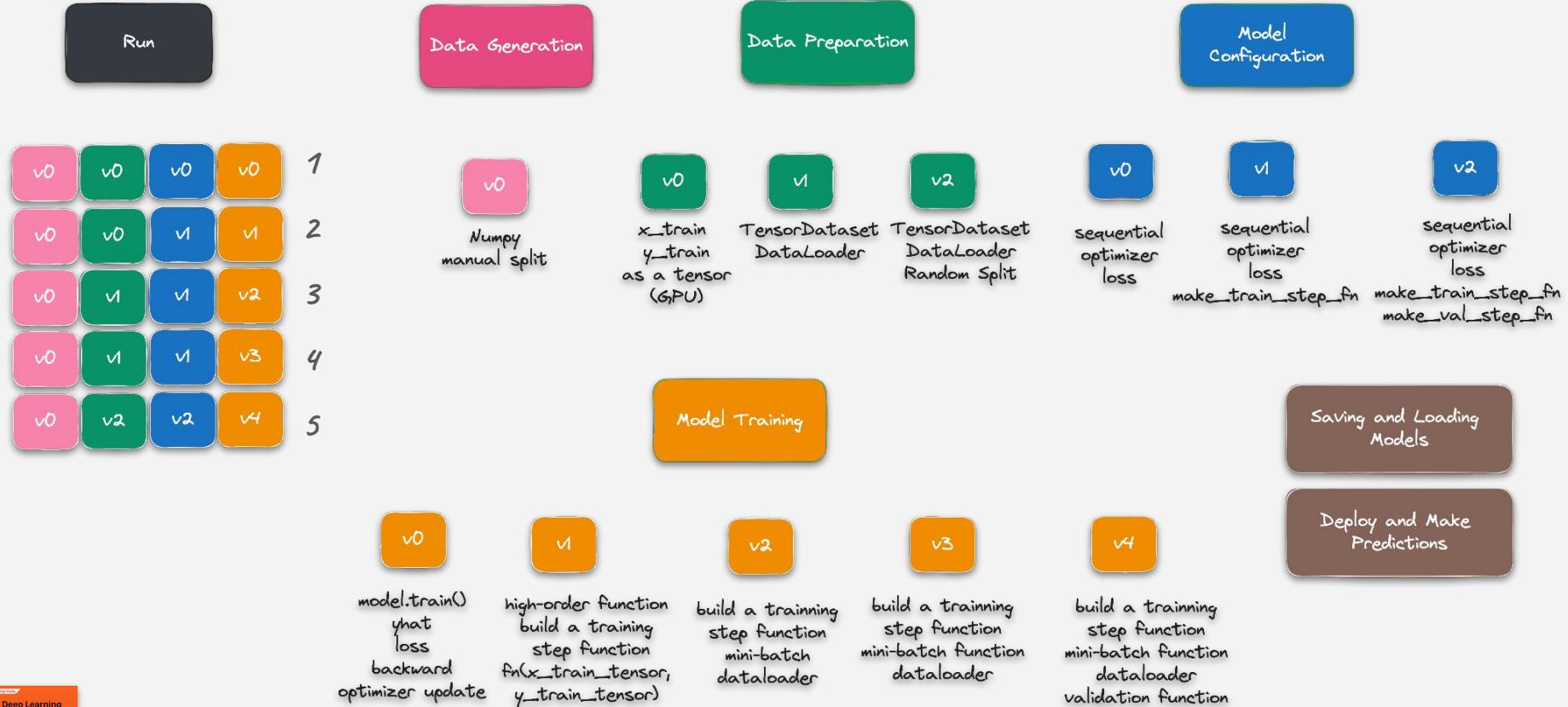
Ivanovitch Silva

ivanovitch.silva@ufrn.br



▼ Part I: Fundamentals

- Chapter 0: Visualizing Gradient Descent
 - Chapter 1: A Simple Regression Problem
 - Chapter 2: Rethinking the Training Loop
 - Chapter 2.1: Going Classy
 - Chapter 3: A Simple Classification Problem
-
- Part II: Computer Vision
 - Part III: Sequences
 - Part IV: Natural Language Processing



High-Order Functions

Higher-order functions in Python are functions that either **take other functions** as arguments or **return functions** as their results. This concept is key in functional programming and enhances flexibility and expressivity in your code.

Functions Accepting Functions as Arguments

```
def square(x):
    return x * x

numbers = [1, 2, 3, 4, 5]
squares = map(square, numbers)
print(list(squares)) # Output: [1, 4, 9, 16, 25]
```

High-Order Functions

Functions Returning Functions

```
def multiplier(n):
    def inner(x):
        return x * n
    return inner

double = multiplier(2)
triple = multiplier(3)

print(double(5)) # Output: 10
print(triple(5)) # Output: 15
```

Why Use High-Order Functions?

- **Code Reusability:** Using higher-order functions, you can write more generic and reusable code. For instance, you can write a general-purpose filter function that can be applied with different criterion functions.
- **Functional Programming:** Higher-order functions are a key component of functional programming, a paradigm emphasizing immutability and the use of functions to transform data.
- **Readability and Expressiveness:** They allow developers to express complex ideas more clearly and concisely.





Model lives
on GPU

Dataset

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):
    def __init__(self, x_tensor, y_tensor):
        self.x = x_tensor
        self.y = y_tensor

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

    def __len__(self):
        return len(self.x)

# Wait, is this a CPU tensor now? Why? Where is .to(device)?
x_train_tensor = torch.from_numpy(x_train).float()
y_train_tensor = torch.from_numpy(y_train).float()

train_data = CustomDataset(x_train_tensor, y_train_tensor)
print(train_data[0])
```

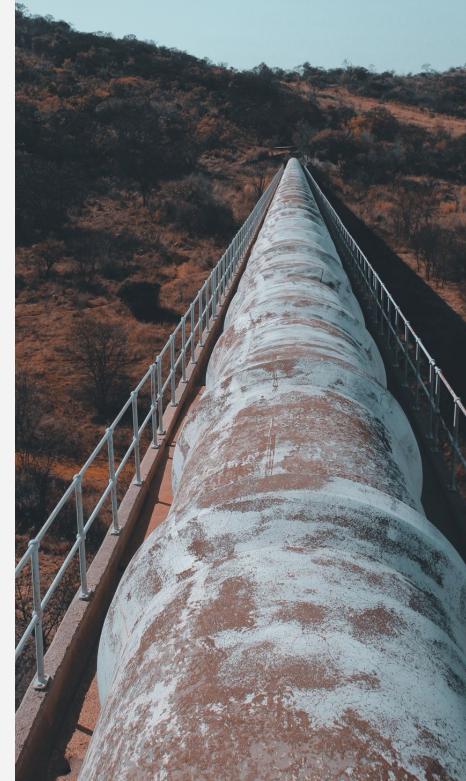


TensorDataset & DataLoader

```
# Our data was in Numpy arrays, but we need to transform them into
# PyTorch's Tensors
x_train_tensor = torch.from_numpy(x_train).float()
y_train_tensor = torch.from_numpy(y_train).float()

# Builds Dataset
train_data = TensorDataset(x_train_tensor, y_train_tensor)

# Builds DataLoader
train_loader = DataLoader(dataset=train_data,
                          batch_size=16,
                          shuffle=True)
```



TensorDataset & DataLoader

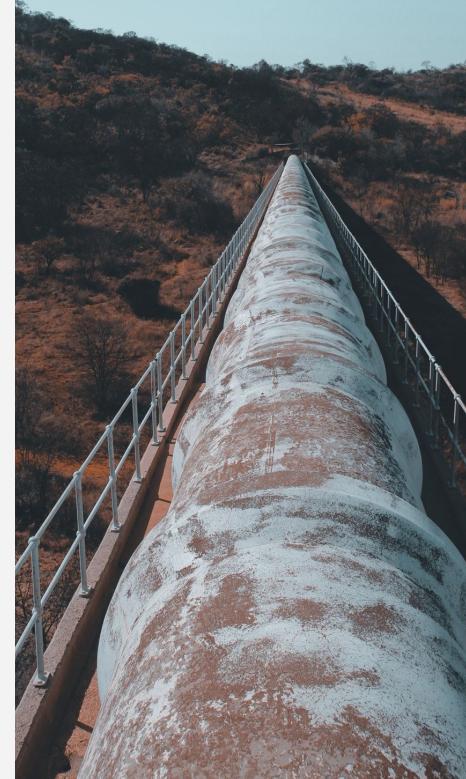
```
# Builds tensors from numpy arrays BEFORE split
x_tensor = torch.from_numpy(x).float()
y_tensor = torch.from_numpy(y).float()

# Builds dataset containing ALL data points
dataset = TensorDataset(x_tensor, y_tensor)

# Performs the split
ratio = .8
n_total = len(dataset)
n_train = int(n_total * ratio)
n_val = n_total - n_train

train_data, val_data = random_split(dataset, [n_train, n_val])

# Builds a loader of each set
train_loader = DataLoader(dataset=train_data,
                          batch_size=16, shuffle=True)
val_loader = DataLoader(dataset=val_data, batch_size=16)
```



Model Configuration

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Sets learning rate - this is "eta" ~ the "n" like Greek letter
lr = 0.1

torch.manual_seed(42)
# Now we can create a model and send it at once to the device
model = nn.Sequential(nn.Linear(1, 1)).to(device)

# Defines a SGD optimizer to update the parameters (now retrieved directly from
# the model)
optimizer = optim.SGD(model.parameters(), lr=lr)

# Defines a MSE loss function
loss_fn = nn.MSELoss(reduction='mean')

# Creates the train_step function for our model, loss function and optimizer
train_step_fn = make_train_step_fn(model, loss_fn, optimizer)

# Creates the val_step function for our model and loss function
val_step_fn = make_val_step_fn(model, loss_fn)
```

Model Configuration

```
def make_train_step_fn(model, loss_fn, optimizer):
    # Builds function that performs a step in the train loop
    def perform_train_step_fn(x, y):
        # Sets model to TRAIN mode
        model.train()

        # Step 1 - Computes our model's predicted output - forward pass
        yhat = model(x)
        # Step 2 - Computes the loss
        loss = loss_fn(yhat, y)
        # Step 3 - Computes gradients for both "a" and "b" parameters
        loss.backward()
        # Step 4 - Updates parameters using gradients and the learning rate
        optimizer.step()
        optimizer.zero_grad()

        # Returns the loss
        return loss.item()

    # Returns the function that will be called inside the train loop
    return perform_train_step_fn
```

Model Configuration

```
def make_val_step_fn(model, loss_fn):
    # Builds function that performs a step in the validation loop
    def perform_val_step_fn(x, y):
        # Sets model to EVAL mode
        model.eval()

        # Step 1 - Computes our model's predicted output - forward pass
        yhat = model(x)
        # Step 2 - Computes the loss
        loss = loss_fn(yhat, y)
        # There is no need to compute Steps 3 and 4, since we don't update
parameters during evaluation
        return loss.item()

    return perform_val_step_fn
```

```
# Defines number of epochs
n_epochs = 200

losses = []
val_losses = []

for epoch in range(n_epochs):
    # inner loop
    loss = mini_batch(device, train_loader, train_step_fn)
    losses.append(loss)

    # VALIDATION
    # no gradients in validation!
    with torch.no_grad():
        val_loss = mini_batch(device, val_loader, val_step_fn)
        val_losses.append(val_loss)
```

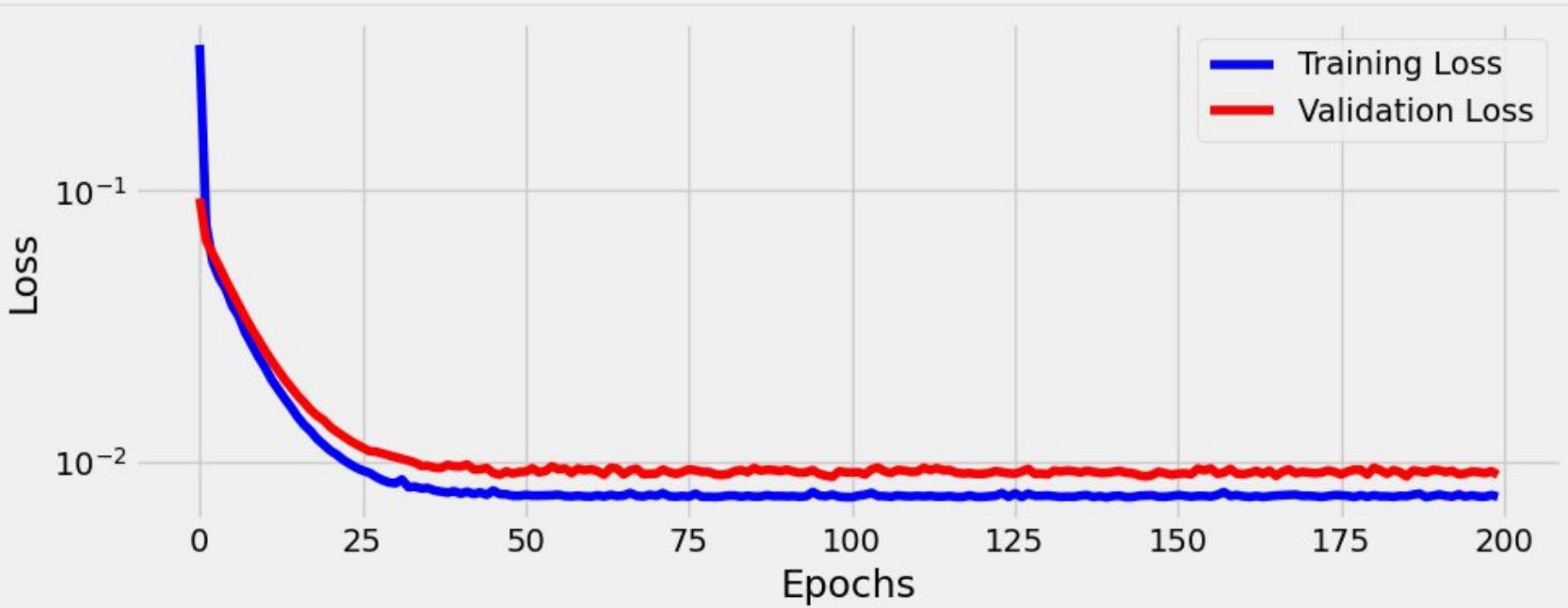


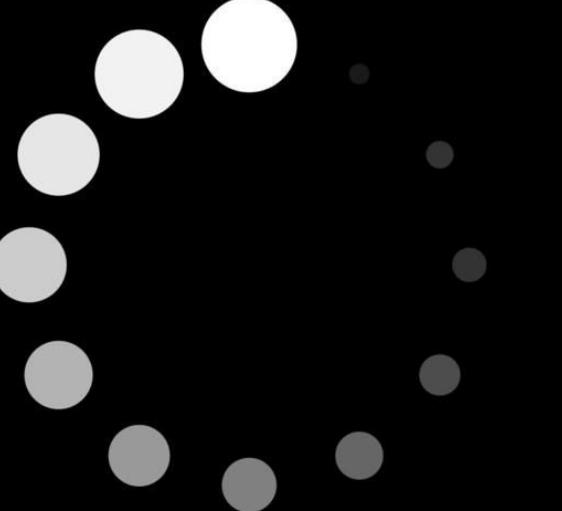
```
def mini_batch(device, data_loader, step_fn):
    mini_batch_losses = []
    for x_batch, y_batch in data_loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        mini_batch_loss = step_fn(x_batch, y_batch)
        mini_batch_losses.append(mini_batch_loss)

    loss = np.mean(mini_batch_losses)
    return loss
```







Saving and Loading Models

LOADING...

```
checkpoint = {'epoch': n_epochs,
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'loss': losses,
              'val_loss': val_losses}
```

```
torch.save(checkpoint, 'model_checkpoint.pth')
```

```
checkpoint = torch.load('model_checkpoint.pth')

model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])

saved_epoch = checkpoint['epoch']
saved_losses = checkpoint['loss']
saved_val_losses = checkpoint['val_loss']

model.train() # always use TRAIN for resuming training
```



Make Predictions

```
checkpoint = torch.load('model_checkpoint.pth')

model.load_state_dict(checkpoint['model_state_dict'])

new_inputs = torch.tensor([[.20], [.34], [.57]])

model.eval() # always use EVAL for fully trained models!
model(new_inputs.to(device))

tensor([[1.4174],
       [1.6896],
       [2.1366]], device='cuda:0')
```

Going Classy

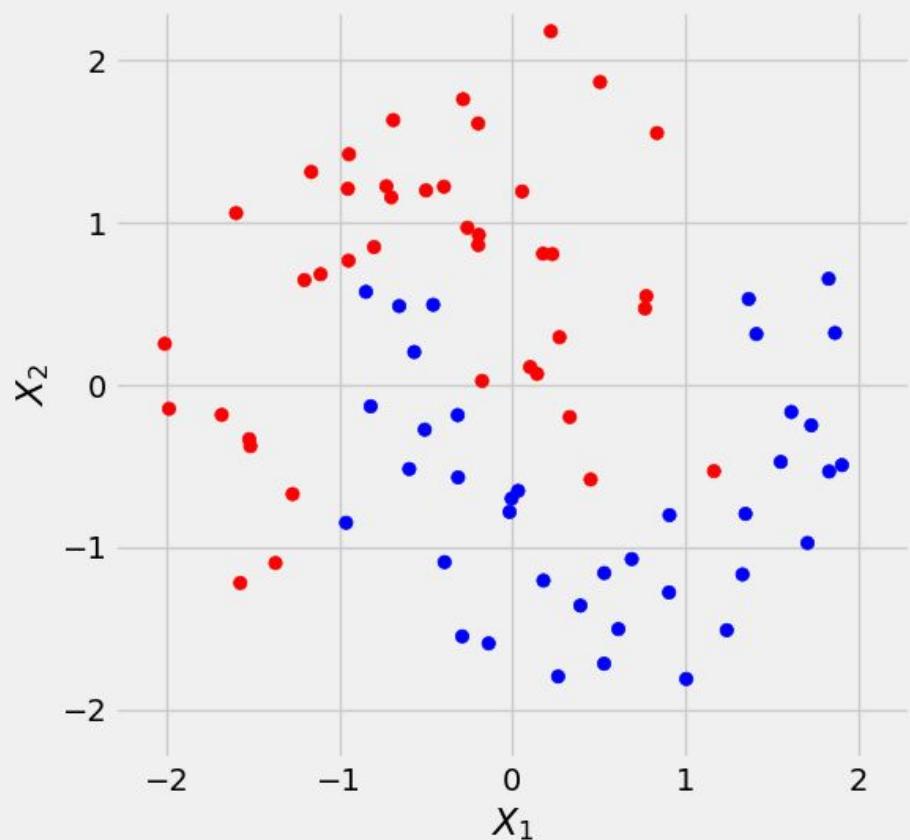
Object-Oriented
Programming - OOP

```
139     title="/"
140     targets="blank"
141     rel="noopener"
142     href={trackUrl}
143   >
144   Instagram
145   </a>
146   </li>
147 </ul>
148 </div>
149 }
150 }
151
152 renderWhatNewLinks() {
153   return (
154     <div className={styles.footer}>
155       <h4 className={style}>
156         <ul className={cls}>
157           {this.renderWhatNewLinks}
158         </ul>
159       </h4>
160       {this.renderWhatNewLinks}
161       {this.renderWhatNewLinks}
162       {this.renderWhatNewLinks}
163       {this.renderWhatNewLinks}
164       {this.renderWhatNewLinks}
165     </div>
166   );
167 }
168
169 renderWhatNewItem(title, url) {
170   return (
171     <li className={styles.footer}>
172       <a href={trackUrl(url)}
173         target="_blank"
174         rel="noopener noreferrer">
175         {title}
176       </a>
177     </li>
178   );
179 }
180
181 renderFooterSub() {
182   return (
183     <div className={styles.footerSub}>
184       <Link to="/" title="Home - Unsplash">
185         <Icon
186           type="logo"
187           className={styles.footerSubLogo}>
188         </Icon>
189       </Link>
190       <span className={styles.footerSlogan}>
191       </span>
192     </div>
193   );
194 }
195
196 render() {
197   return (
198     <footer className={styles.footerGlobal}>
199       <div className={style}>
200         <div className={container}>
201           {this.renderFooterMain()}
202           {this.renderFooterSub()}
203         </div>
204       </div>
205     </footer>
206   );
207 }
208 }
```

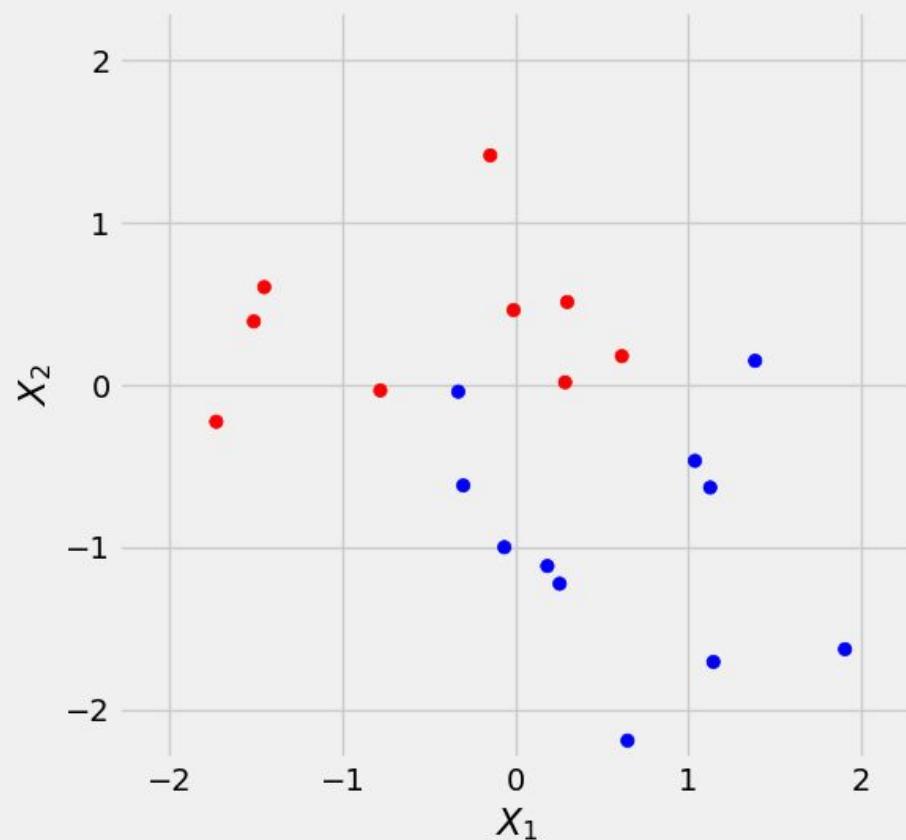
A simple Classification Problem



Generated Data - Train



Generated Data - Validation

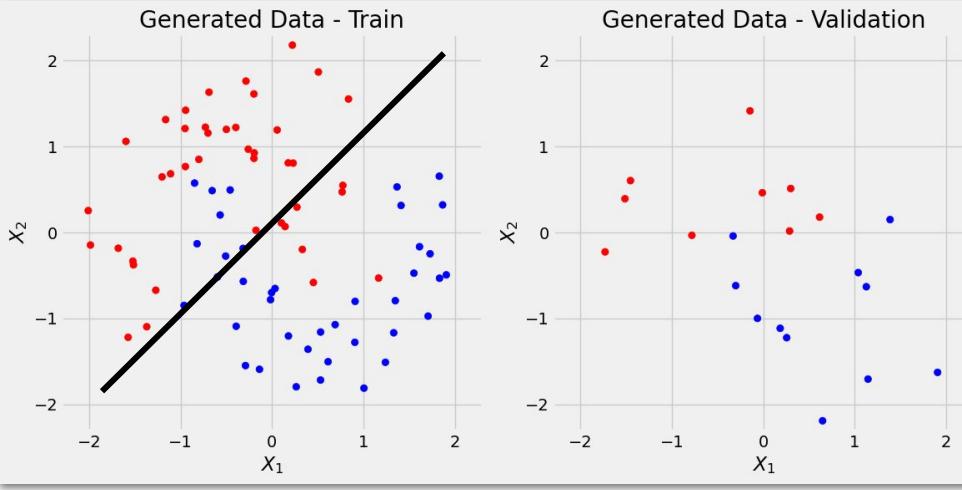




Linear Regression vs Logistic Regression

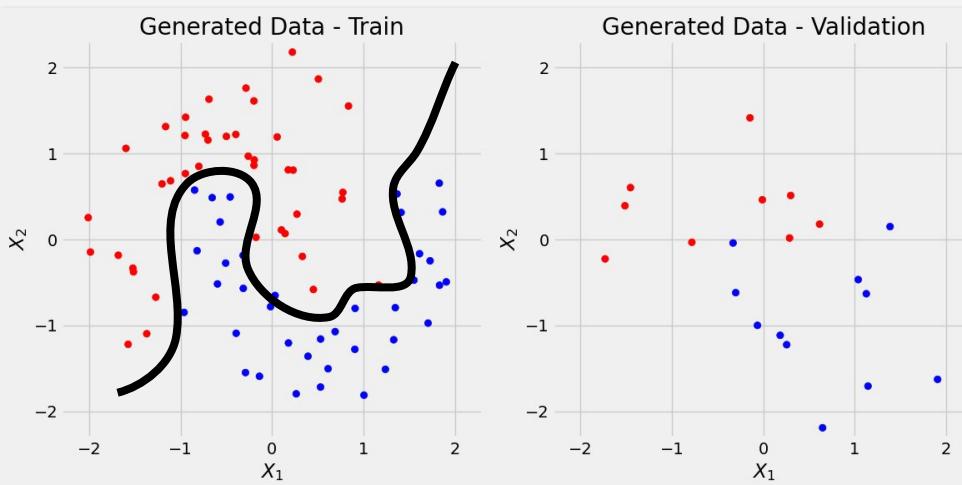
The **dependent variable** is a metric variable: eg. height, speed, salary, grade, etc

The **dependent variable** is a dichotomous variable: eg. red or blue, happy or sad, 0 or 1, gender, etc

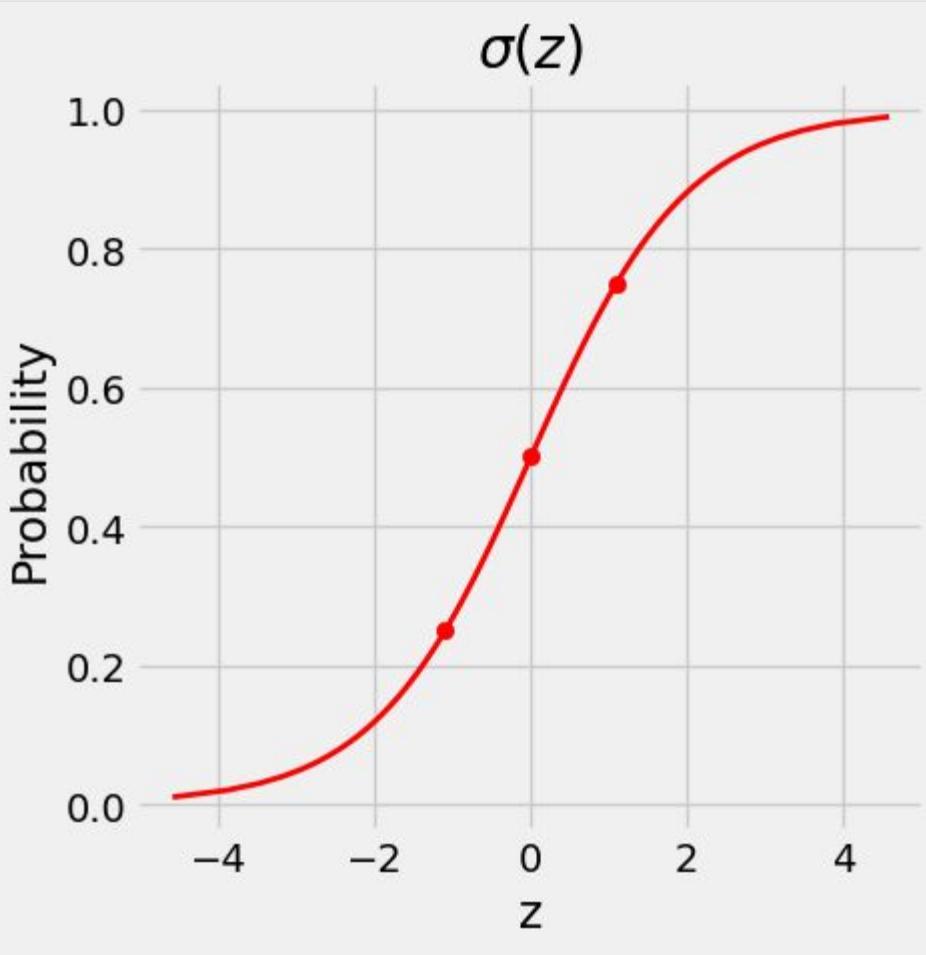


$$y = b + w_1x_1 + w_2x_2 + \epsilon$$

$$y = \begin{cases} 1, & \text{if } b + w_1x_1 + w_2x_2 \geq 0 \\ 0, & \text{if } b + w_1x_1 + w_2x_2 < 0 \end{cases}$$



Mapping a linear regression model to **discrete labels**.



Only apply linear regression is not enough. The goal of the logistic regression is to estimate the **probability of occurrence**.

Logits

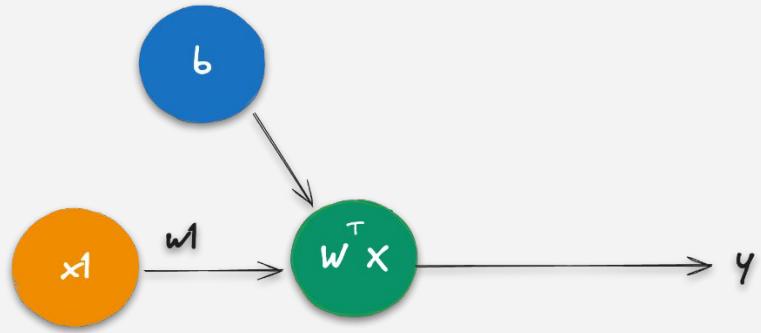
$$z = b + w_1x_1 + w_2x_2$$

$$P(y = 1) \approx 1.0, \text{ if } z \gg 0$$

$$P(y = 1) = 0.5, \text{ if } z = 0$$

$$P(y = 1) \approx 0.0, \text{ if } z \ll 0$$

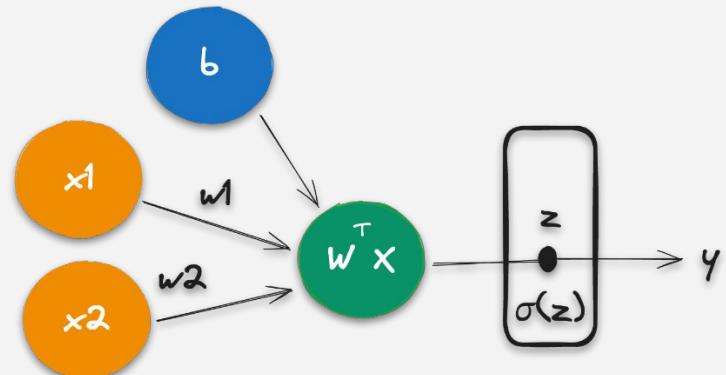
$$p = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Input Layer

Output Layer

Linear



Input Layer

Output Layer

Sigmoid

Linear Regression

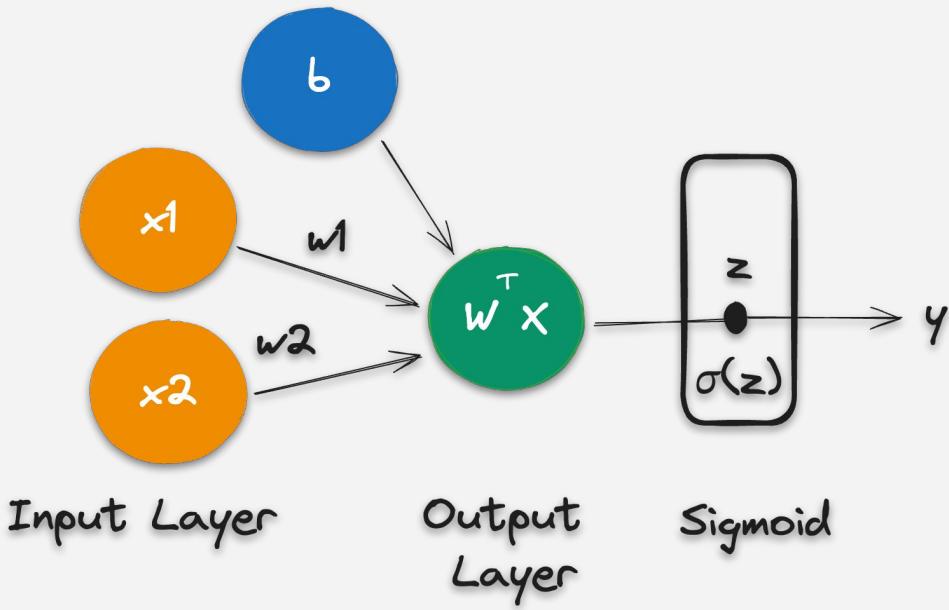
vs

Logistic Regression

Logistic Regression

A Note on Notation

$$W = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}_{(3 \times 1)}; X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}_{(3 \times 1)}$$



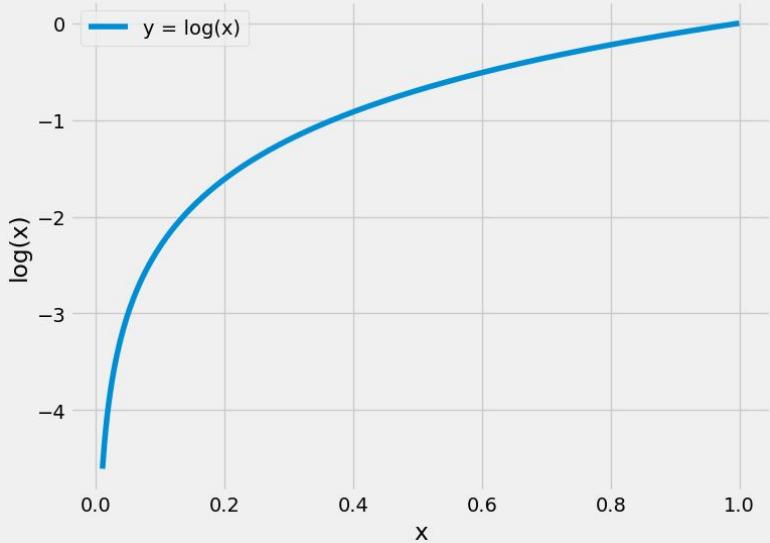
$$z = W^T X = \begin{bmatrix} - & w^T & - \end{bmatrix}_{(1 \times 3)} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}_{(3 \times 1)} = \begin{bmatrix} b & w_1 & w_2 \end{bmatrix}_{(1 \times 3)} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}_{(3 \times 1)}$$

$$= b + w_1 x_1 + w_2 x_2$$

```
torch.manual_seed(42)
model = nn.Sequential()
model.add_module('linear', nn.Linear(2, 1))
model.add_module('sigmoid', nn.Sigmoid())
print(model.state_dict())

OrderedDict([('linear.weight', tensor([[ 0.5406,
  0.5869]])), ('linear.bias', tensor([-0.1657]))])
```

Graph of the Logarithm Function



We already have a model, and now we need to define an appropriate **loss** for the logistic regression.

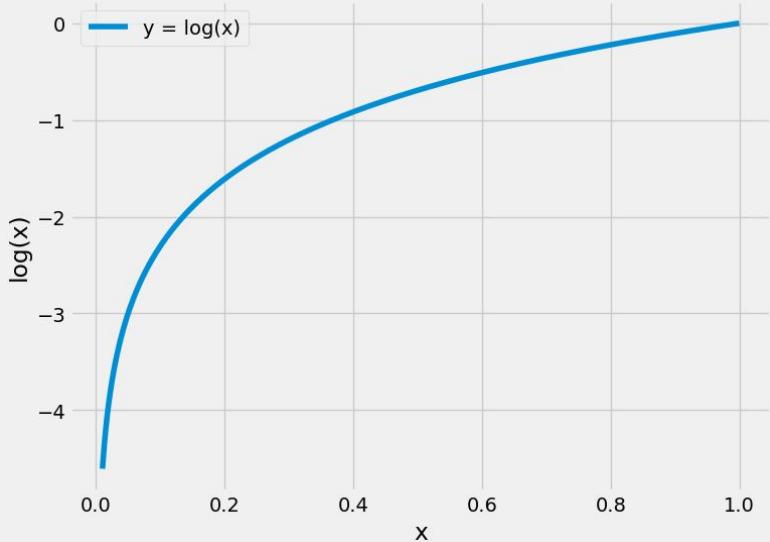
#intuitive-way

$$y_i = 1 \Rightarrow \text{error}_i = \log(P(y_i = 1))$$

$$P(y_i = 0) = 1 - P(y_i = 1)$$

$$y_i = 0 \Rightarrow \text{error}_i = \log(1 - P(y_i = 1))$$

Graph of the Logarithm Function



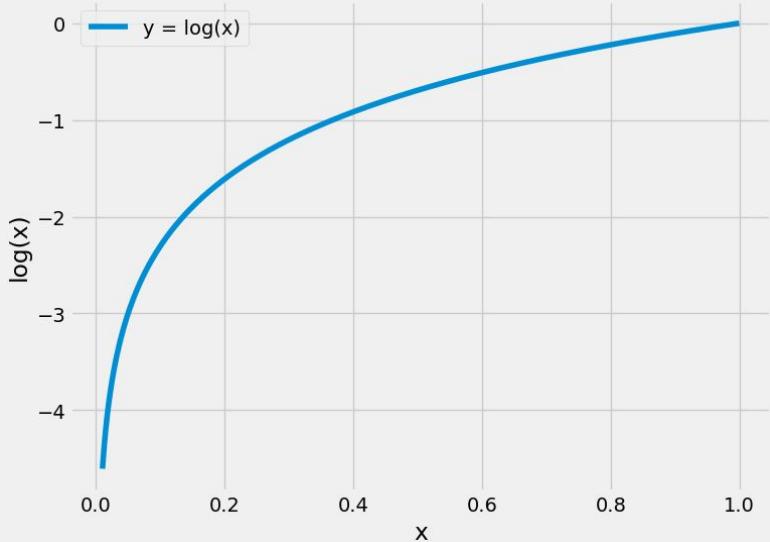
We already have a model, and now we need to define an appropriate **loss** for the logistic regression.

#intuitive-way

Binary Cross-Entropy (BCE)

$$BCE(y) = -\frac{1}{(N_{\text{pos}} + N_{\text{neg}})} \left[\sum_{i=1}^{N_{\text{pos}}} \log(P(y_i = 1)) + \sum_{i=1}^{N_{\text{neg}}} \log(1 - P(y_i = 1)) \right]$$

Graph of the Logarithm Function



We already have a model, and now we need to define an appropriate **loss** for the logistic regression.

#intuitive-way

Binary Cross-Entropy (BCE)

$$BCE(y) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(P(y_i = 1)) + (1 - y_i) \log(1 - P(y_i = 1))]$$

Binary Cross-Entropy (BCE)

$$BCE(y) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(P(y_i = 1)) + (1 - y_i) \log(1 - P(y_i = 1))]$$

```
loss_fn = nn.BCELoss(reduction='mean')

dummy_labels = torch.tensor([1.0, 0.0])
dummy_predictions = torch.tensor([.9, .2])

# RIGHT
right_loss = loss_fn(dummy_predictions, dummy_labels)

# WRONG
wrong_loss = loss_fn(dummy_labels, dummy_predictions)

print(right_loss, wrong_loss)
tensor(0.1643) tensor(15.0000)
```

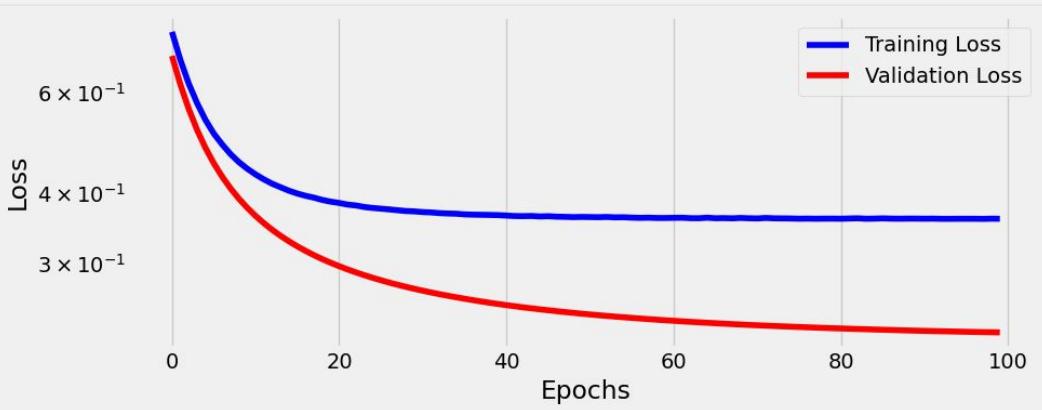
Model Configuration

```
# Sets learning rate - this is "eta" ~ the "n" like Greek letter
lr = 0.1

torch.manual_seed(42)
model = nn.Sequential()
model.add_module('linear', nn.Linear(2, 1))

# Defines a SGD optimizer to update the parameters
optimizer = optim.SGD(model.parameters(), lr=lr)

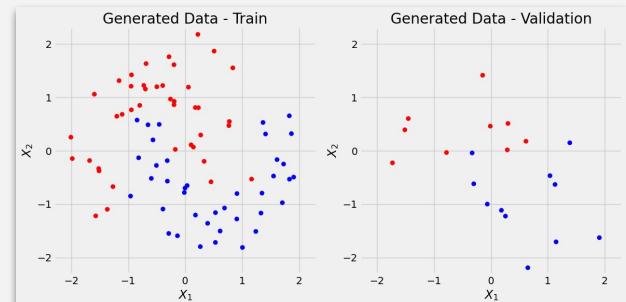
# Defines a BCE loss function
loss_fn = nn.BCEWithLogitsLoss()
```



Training

```
n_epochs = 100

arch = Architecture(model, loss_fn, optimizer)
arch.set_loaders(train_loader, val_loader)
arch.set_seed(42)
arch.train(n_epochs)
```



Making Predictions

$$y = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

$$y = \begin{cases} 1, & \text{if } P(y = 1) \geq 0.5 \\ 0, & \text{if } P(y = 1) < 0.5 \end{cases}$$

```
# prediction logits (z)
logits_val = arch.predict(X_val[:4])
logits_val

array([[ -0.37522304],
       [  0.7390274 ],
       [ -2.5800889 ],
       [ -0.93623203]], dtype=float32)
```

```
# prediction probabilities
prob_val = torch.sigmoid(torch.as_tensor(logits_val[:4]).float())
prob_val

tensor([[ 0.4073],
       [ 0.6768],
       [ 0.0704],
       [ 0.2817]])
```

Decision Boundary

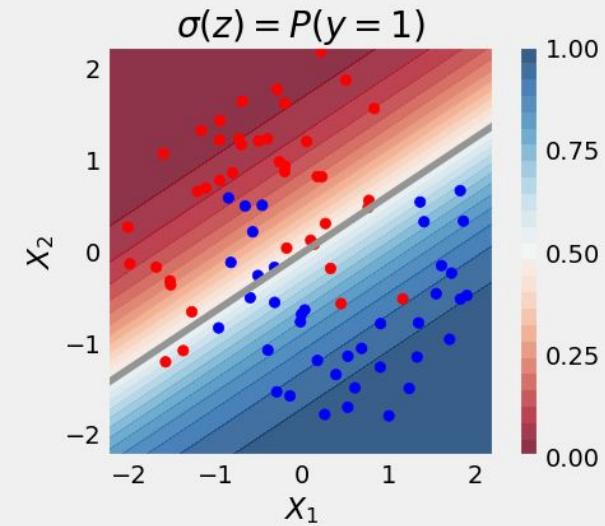
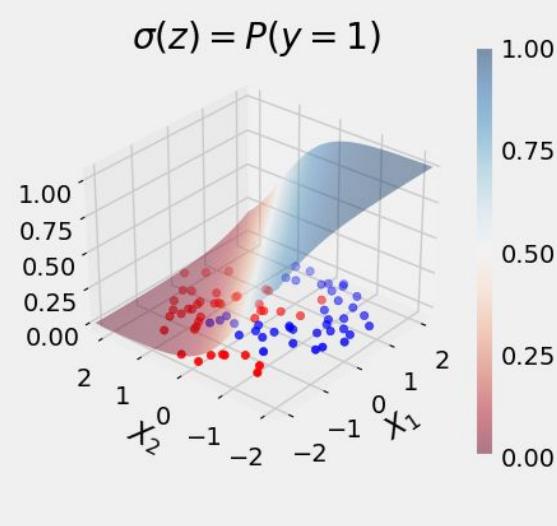
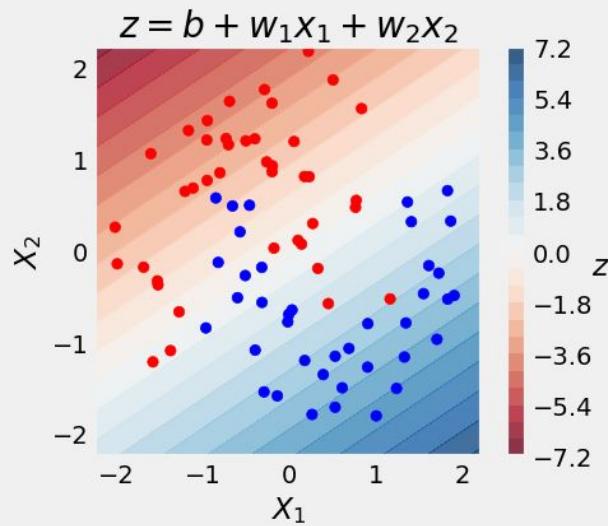
$$\begin{aligned} z &= 0 = b + w_1x_1 + w_2x_2 \\ -w_2x_2 &= b + w_1x_1 \\ x_2 &= -\frac{b}{w_2} - \frac{w_1}{w_2}x_1 \end{aligned}$$

$$\begin{aligned} x_2 &= -\frac{0.0591}{1.8693} + \frac{1.1806}{1.8693}x_1 \\ x_2 &= -0.0316 + 0.6315x_1 \end{aligned}$$

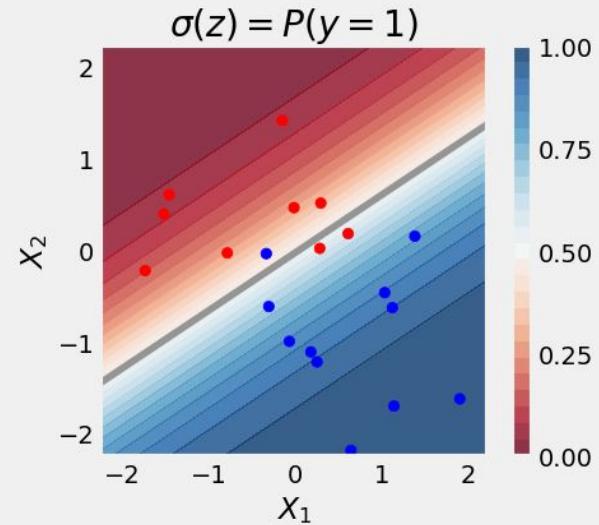
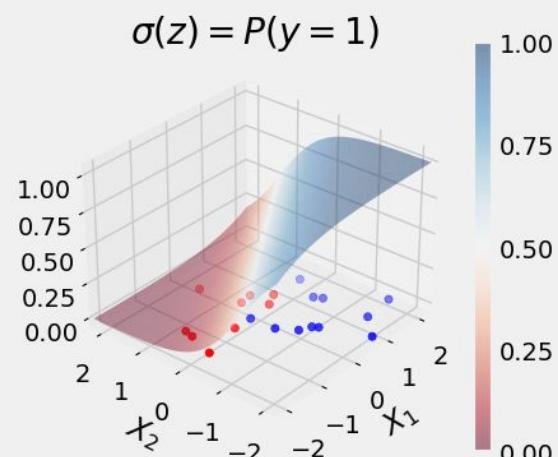
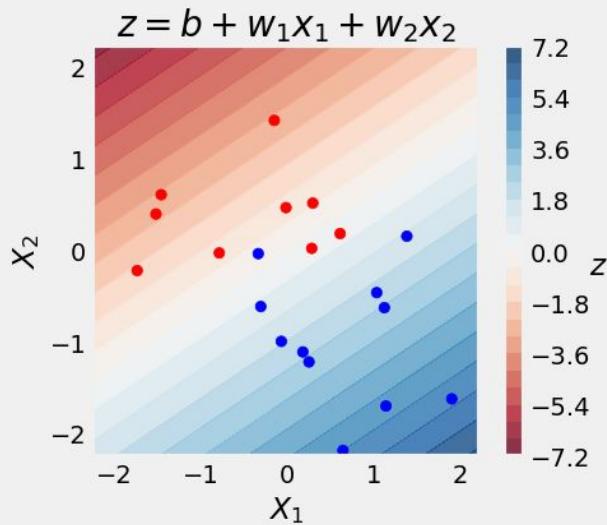
```
print(model.state_dict())
OrderedDict([('linear.weight', tensor([[ 1.1806, -1.8693]])), ('linear.bias', tensor([-0.0591]))])
```

$$\begin{aligned} z &= b + w_1x_1 + w_2x_2 \\ z &= -0.0591 + 1.1806x_1 - 1.8693x_2 \end{aligned}$$

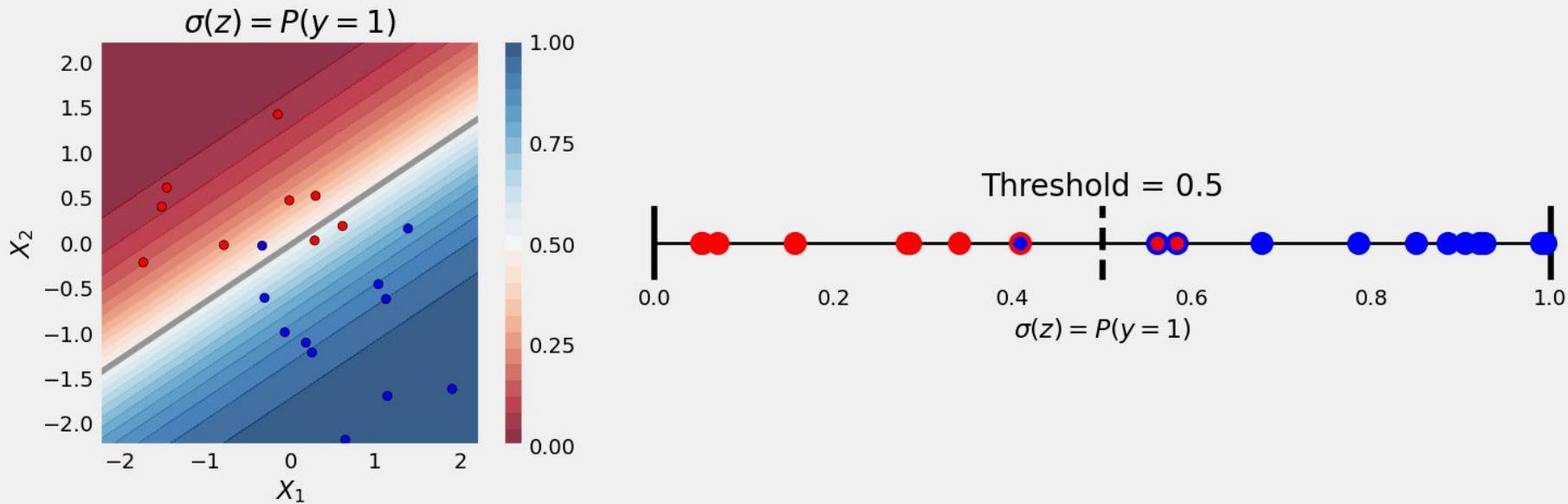
Decision Boundary (training data)



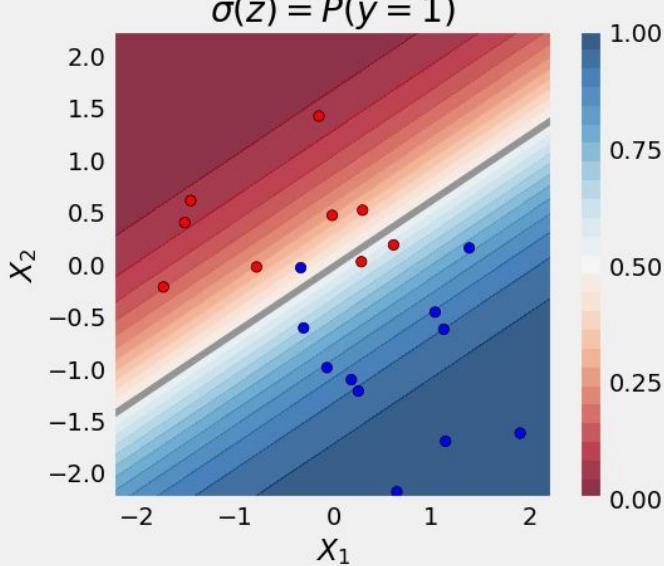
Decision Boundary (validation data)



Classification Threshold

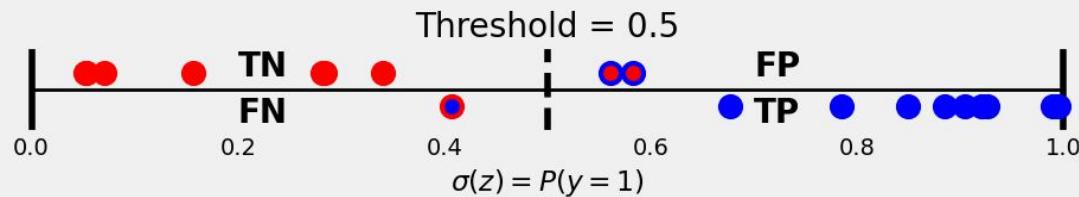


Confusion Matrix



$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$



$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$