AI VIET NAM
@aivietnam.edu.vn

Multilayer Perceptrons (MLPs) and Activation functions

Exercise

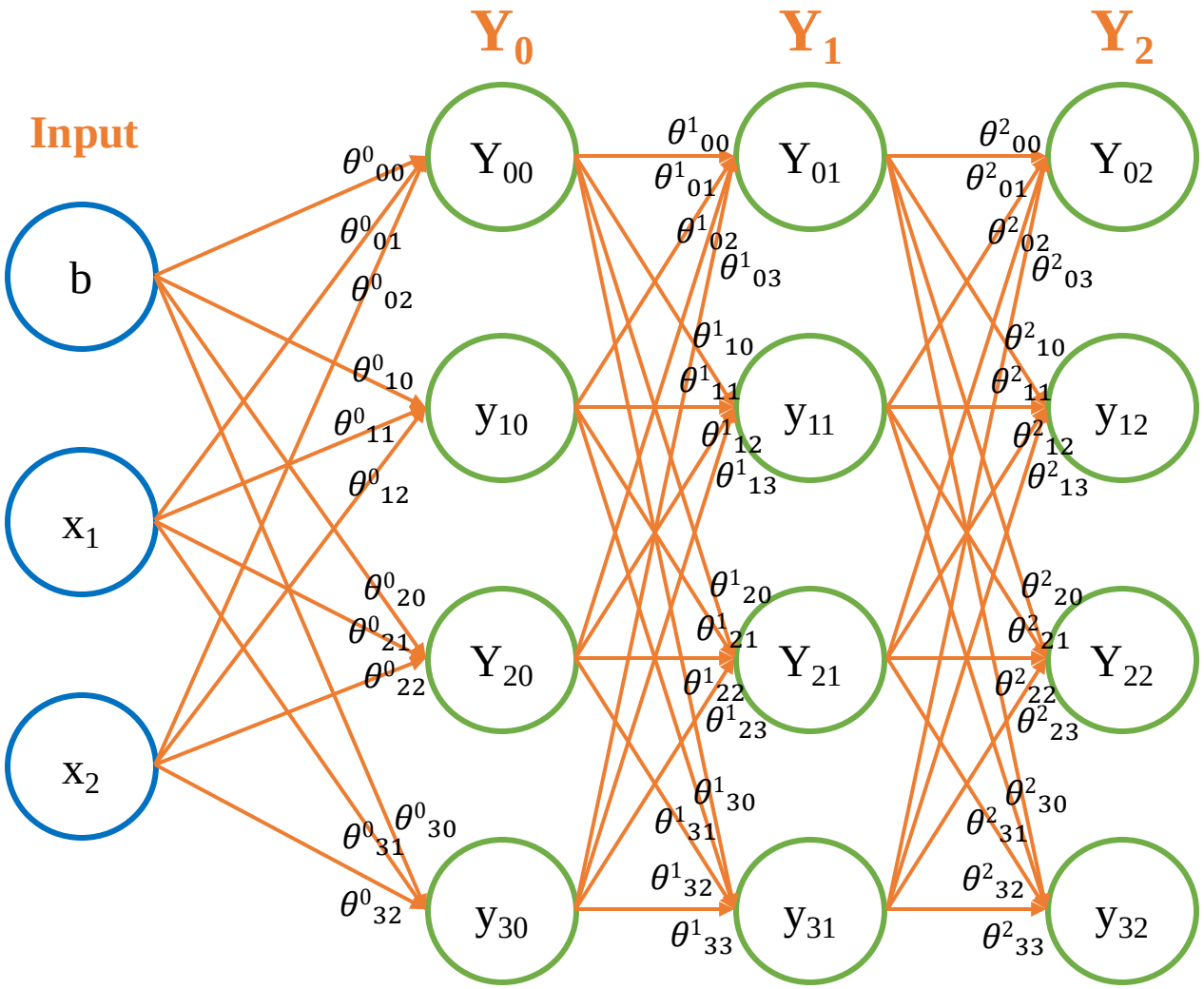
Dinh-Thang Duong – TA

Yen-Linh Vu – STA

AI VIET NAM
@aivietnam.edu.vn

Getting Started

❖ Objectives



Our objectives:

- Review the Multilayer Perceptrons and Activation functions concepts.
- Practice how to implement Multilayer Perceptrons with PyTorch on 3 problems: Car Performance Prediction, Non-linear Data Classification and Sentiment Image Analysis.
- Examine the performance Multilayer Perceptrons on multiple configurations.

AI VIET NAM
@aivietnam.edu.vn

Outline

- Review
- Car Performance Prediction
- Non-linear Data Classification
- Sentiment Image Analysis
- Question

AI VIET NAM
@aivietnam.edu.vn

Review

Review

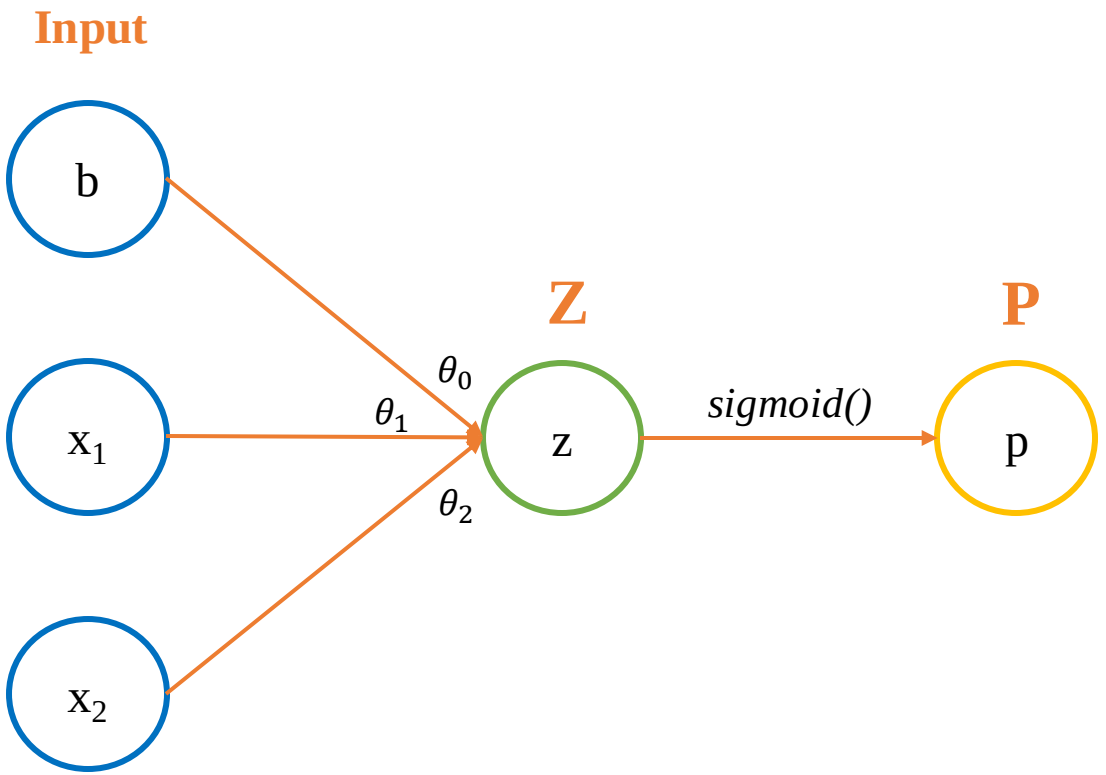
❖ Getting Started

x : **Input data x_i**

z : **Linear value (z)**

p : **Activation value (p)**

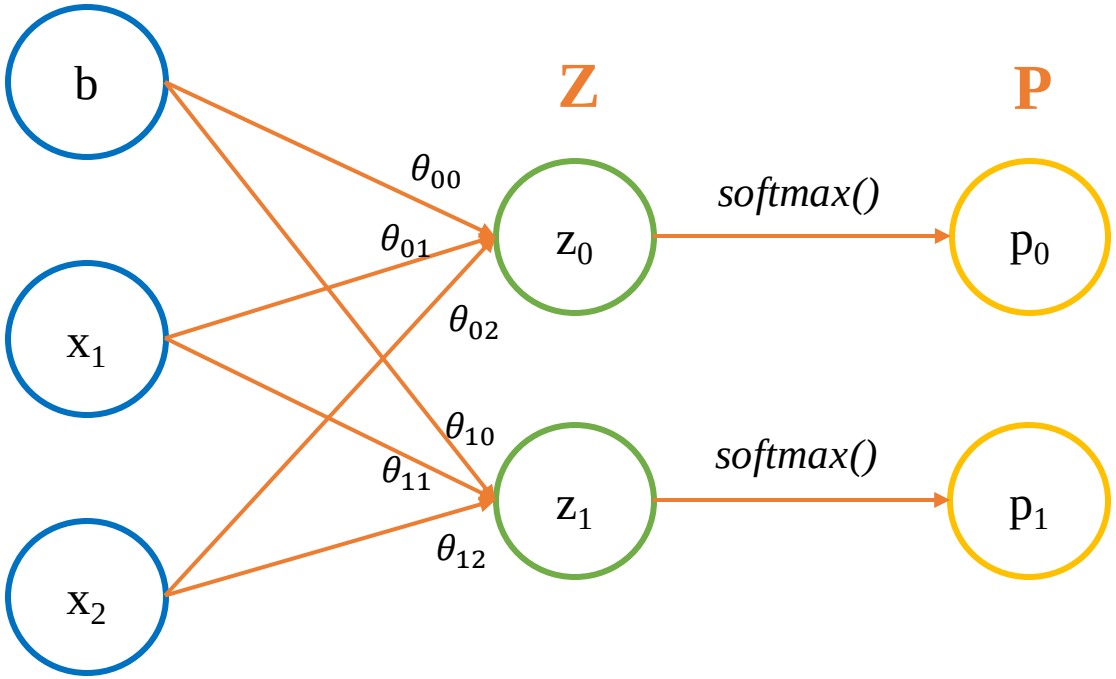
: Direction



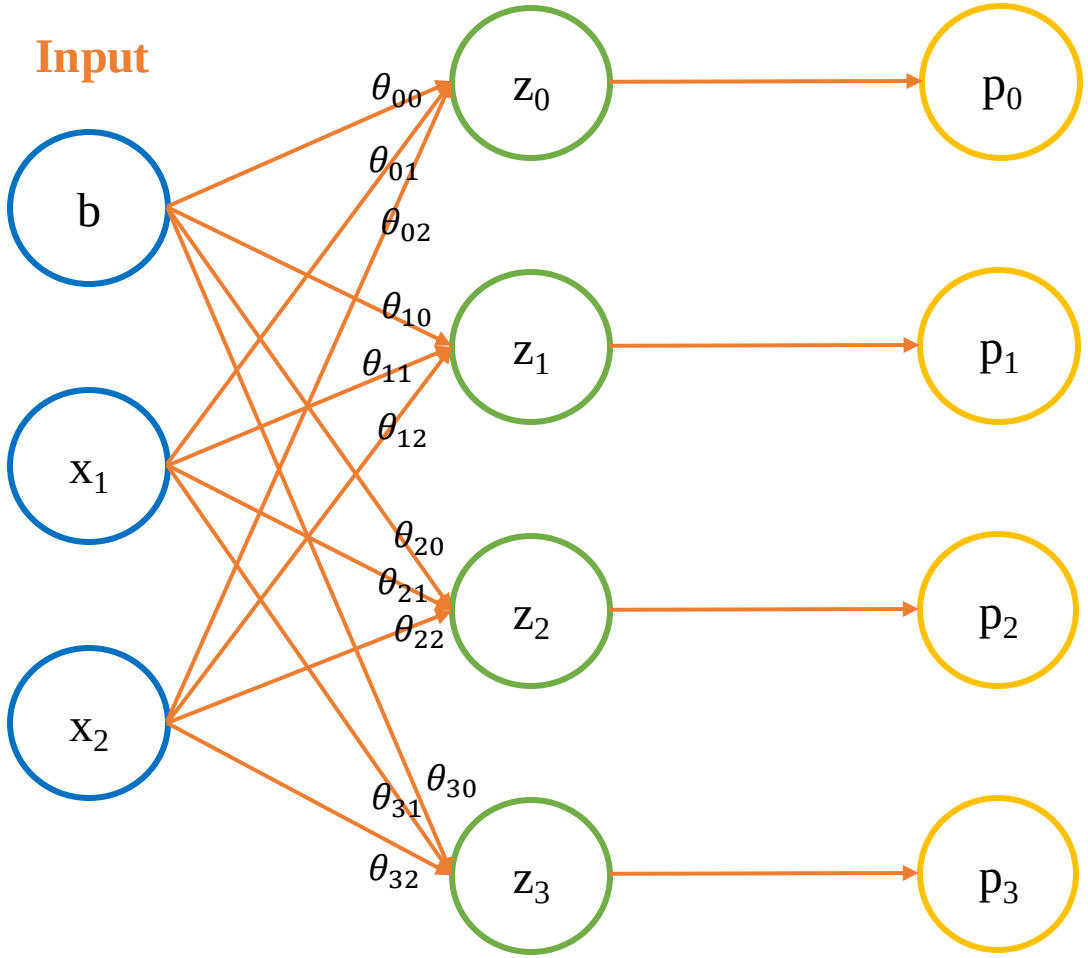
Logistic Regression Computation

Review

❖ Softmax Regression Illustration



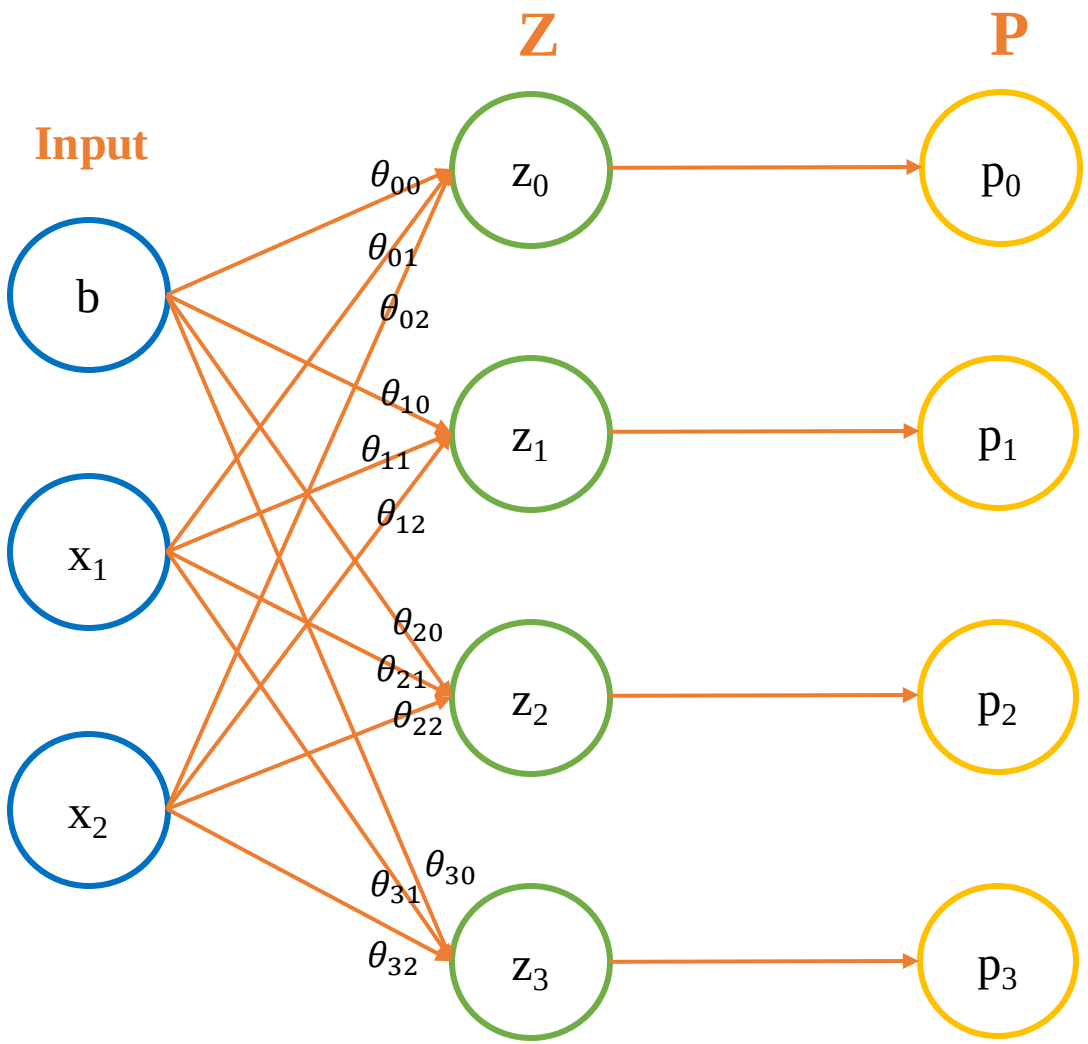
Softmax Regression (2 class)

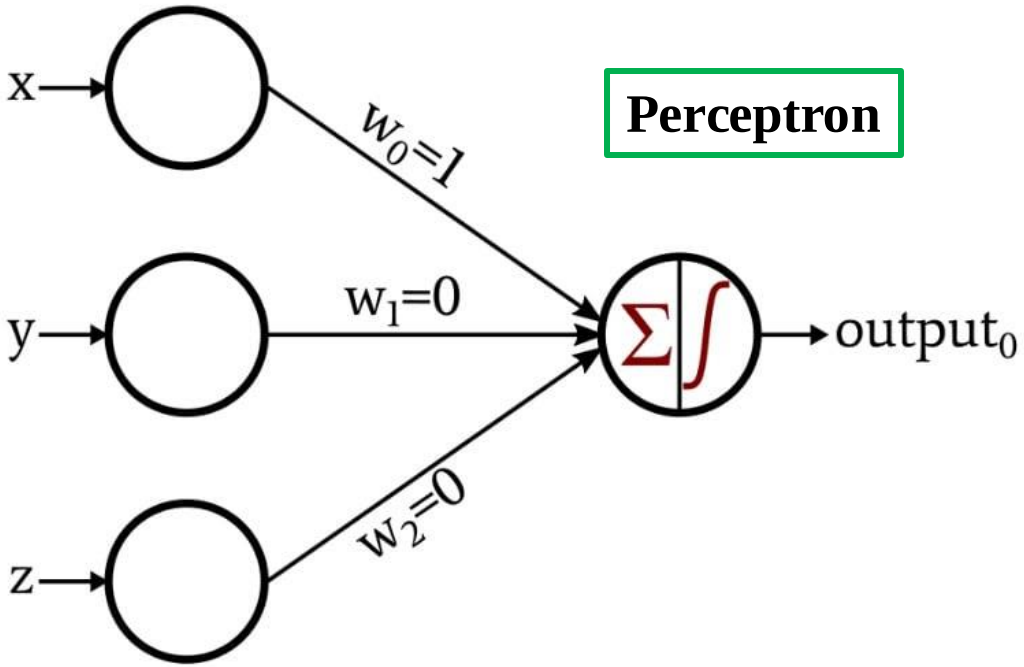


Softmax Regression (4 class)

Review

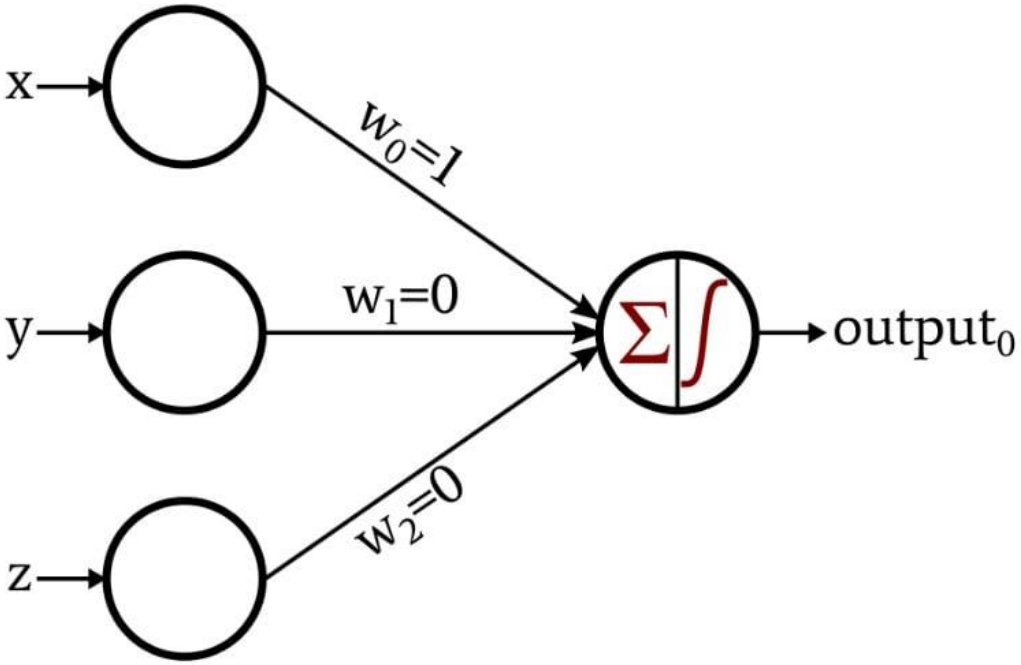
❖ Perceptron





Review

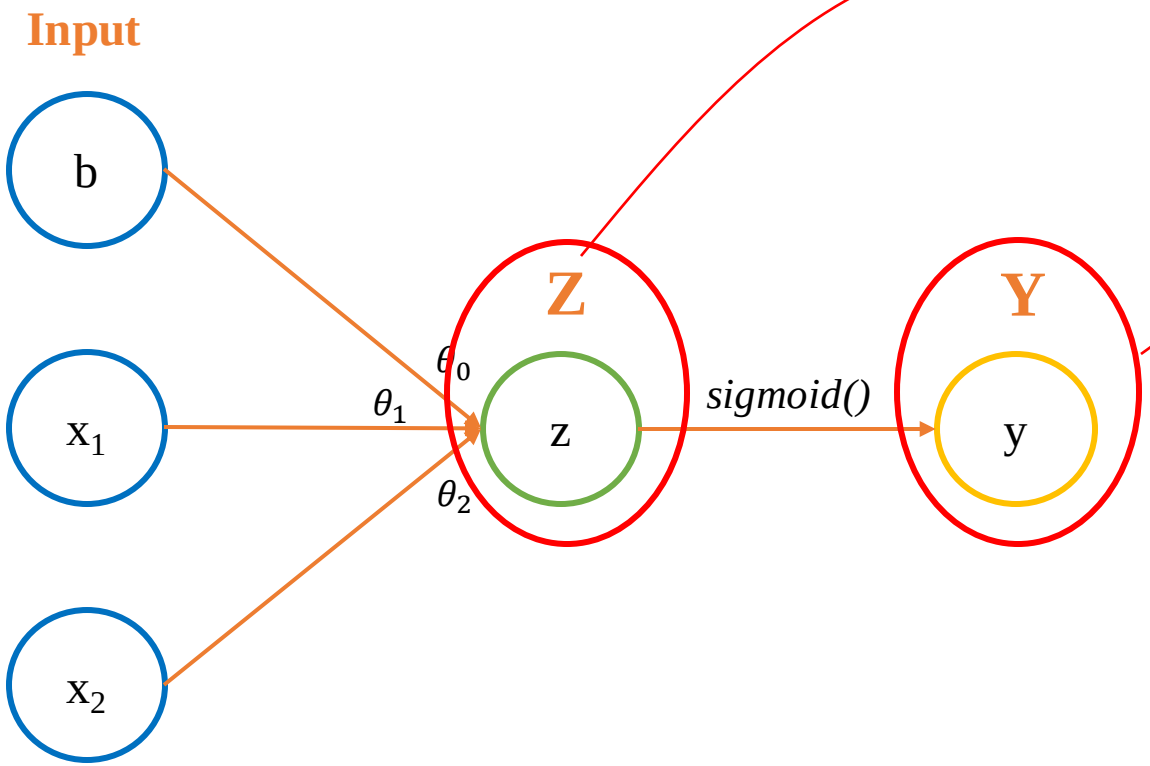
❖ Perceptron



Perceptron: A simple neural network with a single layer that processes inputs using weighted sums, a bias, and an activation function for binary classification of linearly separable data.

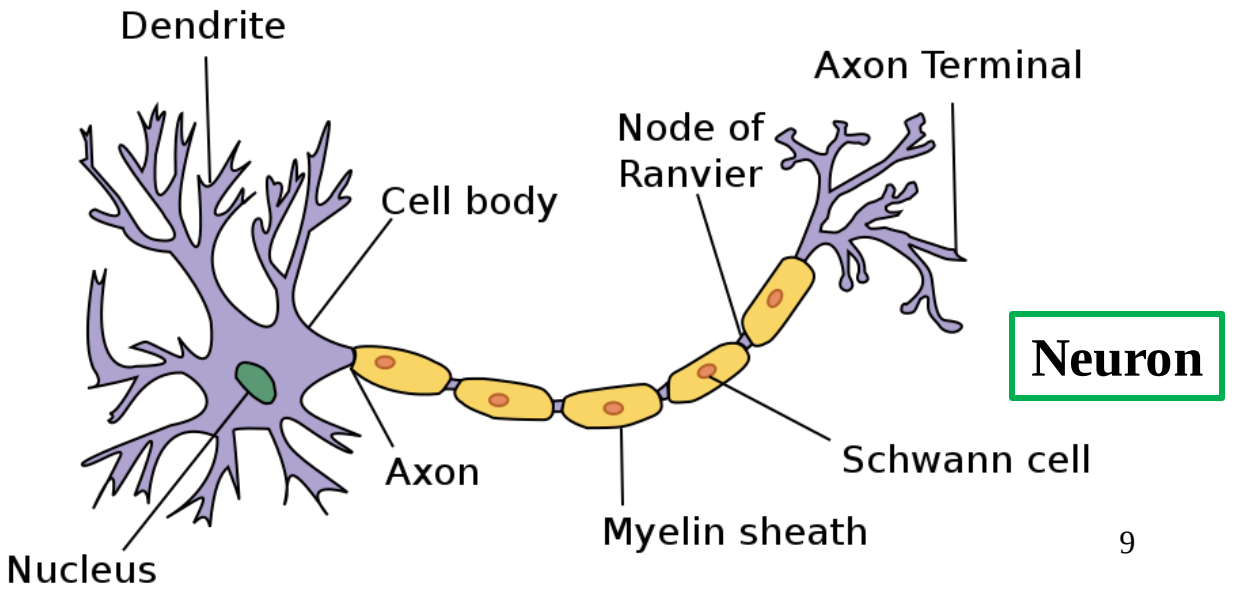
Review

❖ Perceptron



1. Dot product $Z = X \cdot \theta$

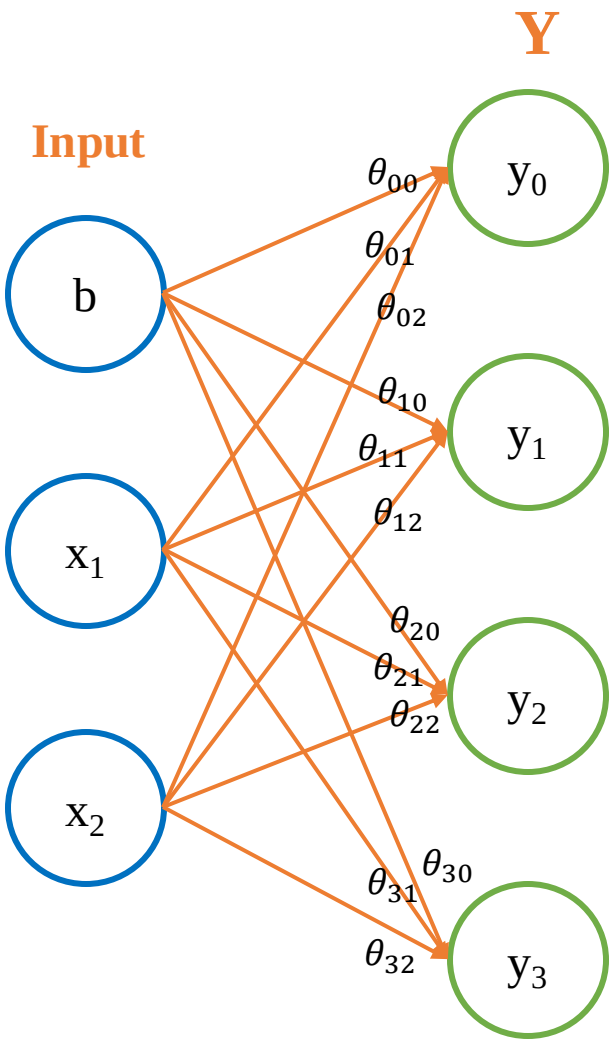
2. Output of some function (activation function):
- e.g: $y = sigmoid(Z)$

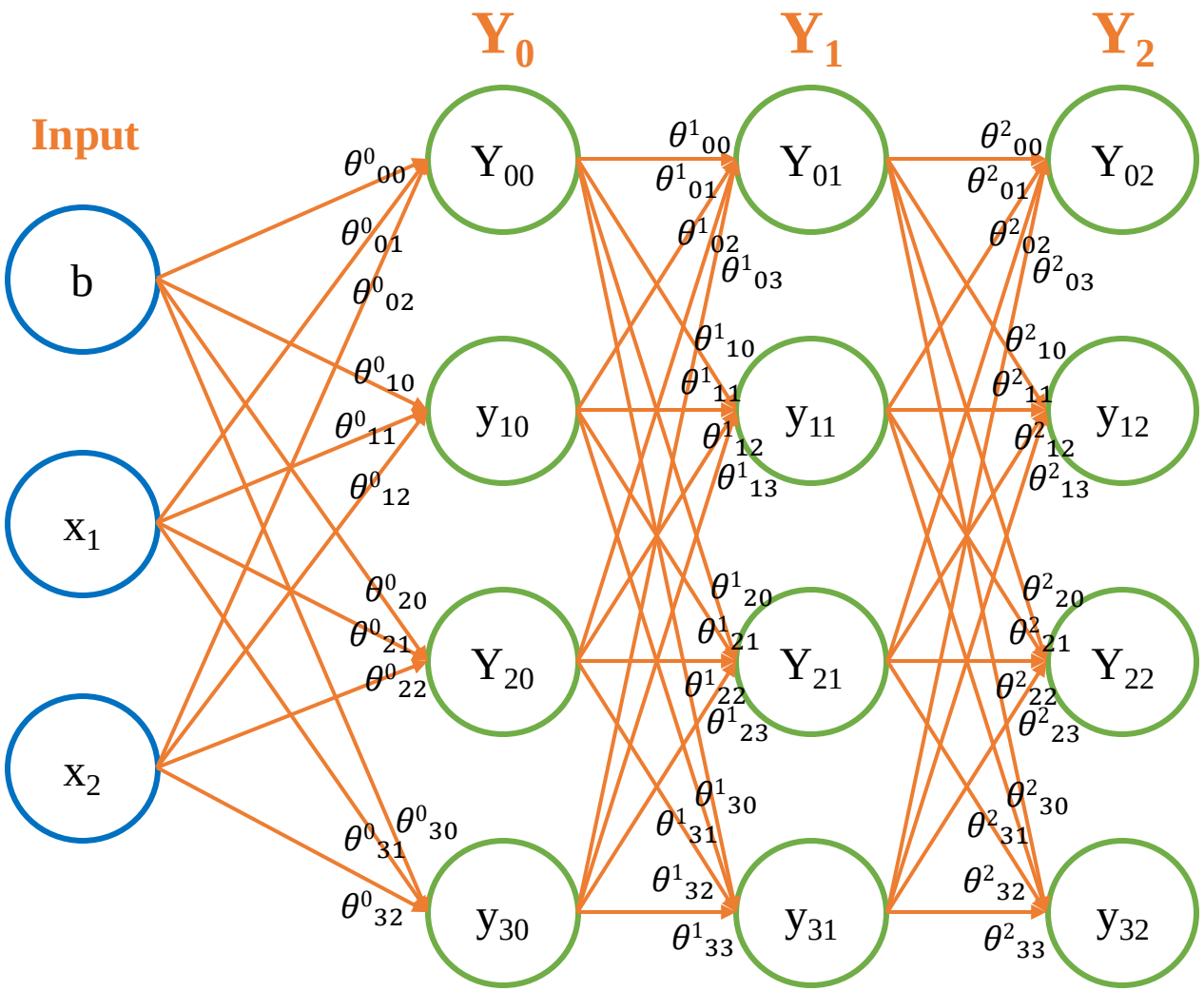


Neuron

Review

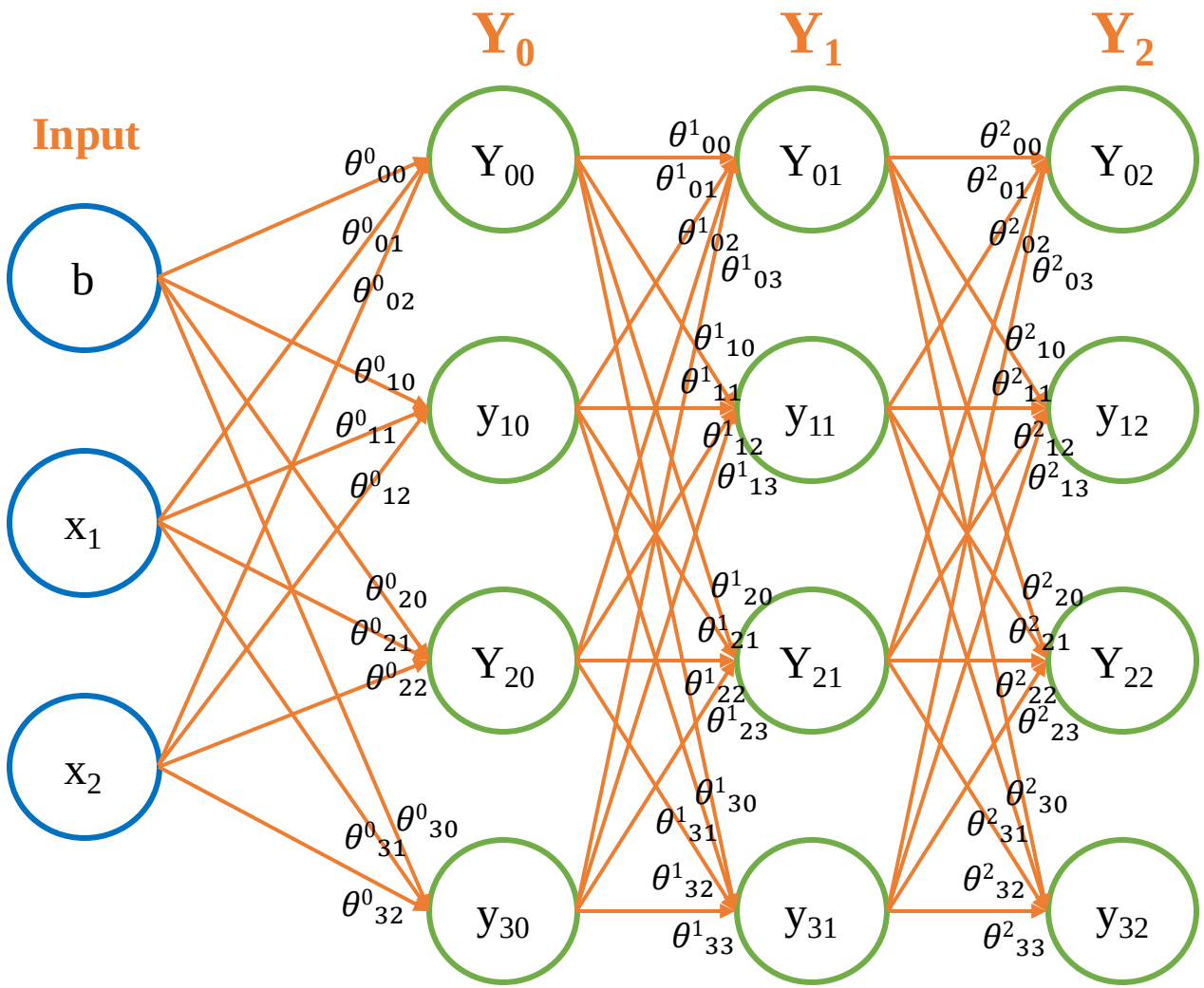
❖ Multilayer Perceptron





Review

❖ Multilayer Perceptron



Multilayer Perceptron (MLP): A type of neural network with an input layer, one or more hidden layers, and an output layer, using non-linear activation functions to handle both linear and non-linear problems.

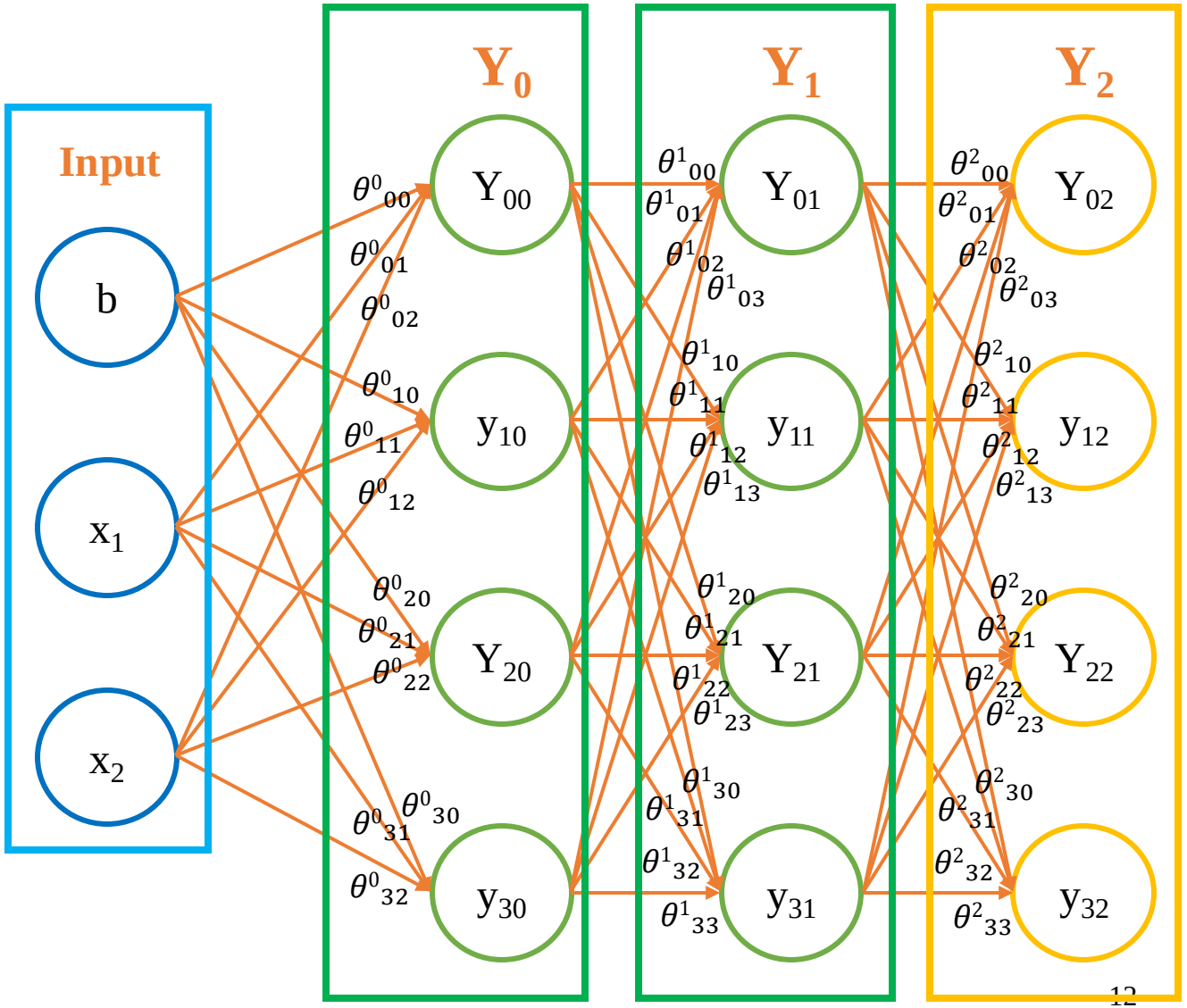
Review

❖ Multilayer Perceptron

Input layer

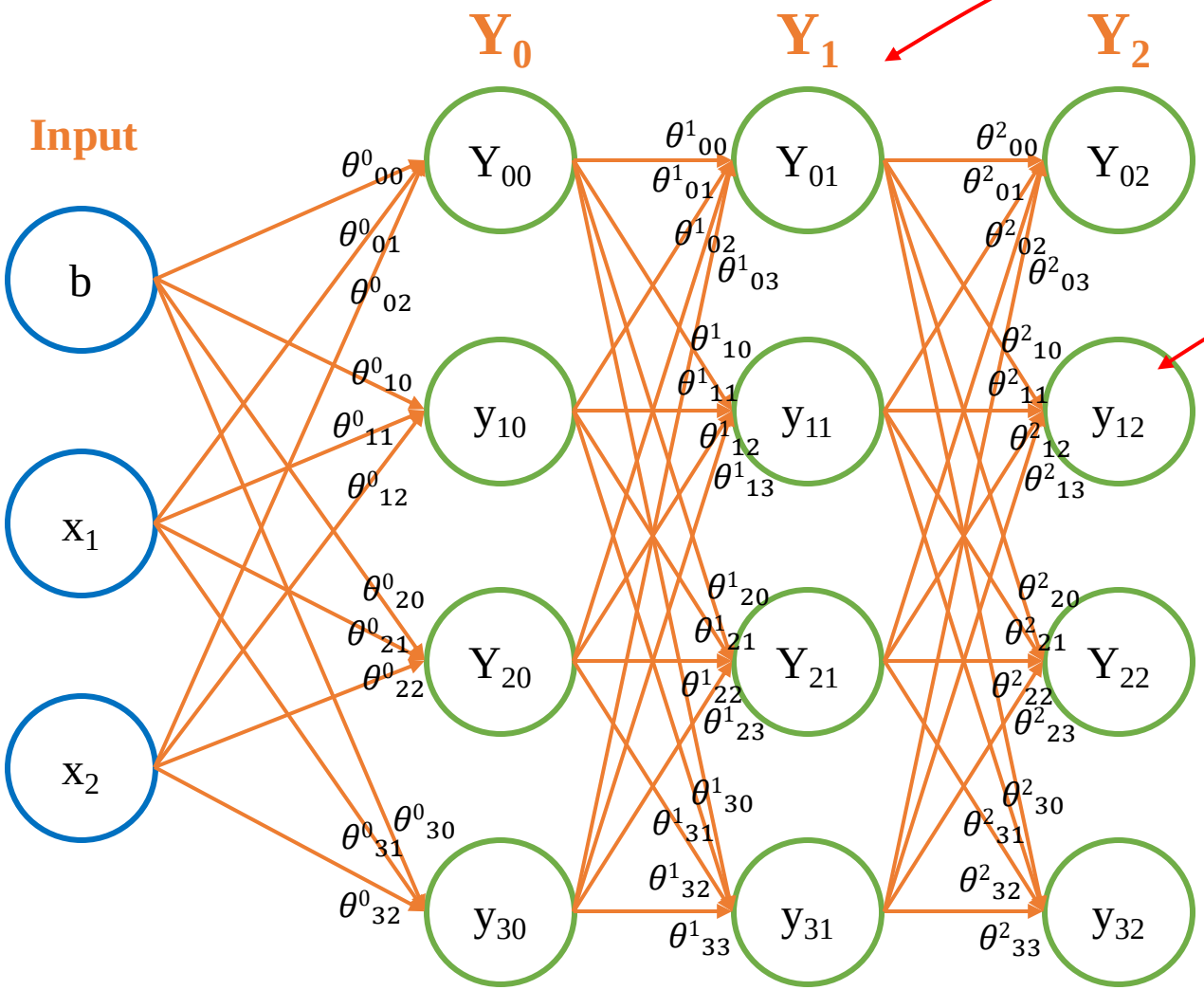
Hidden layer

Output layer



Review

❖ Multilayer Perceptron

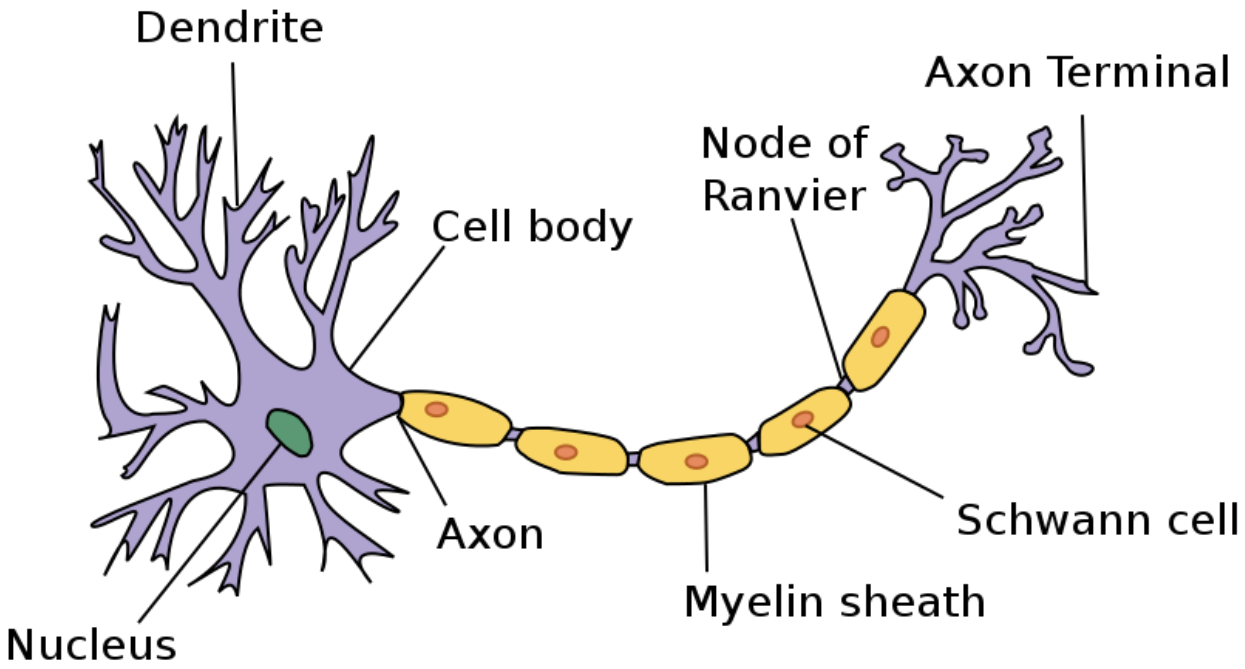


Could be any number nodes in layers (except input layer).

Could be any activation function in layers (including None).

Review

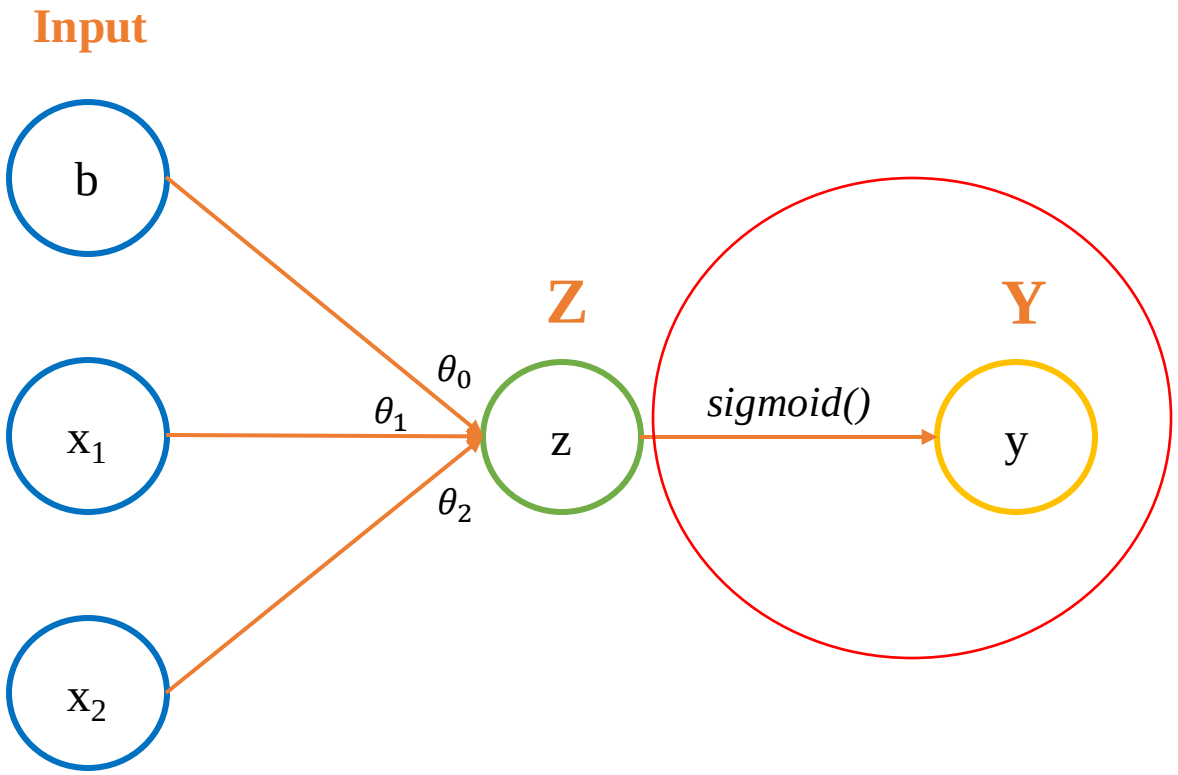
❖ Activation function

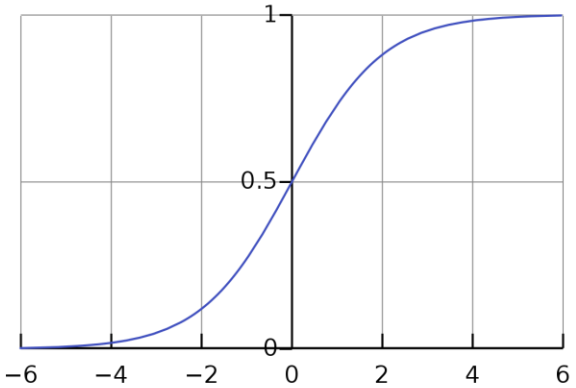


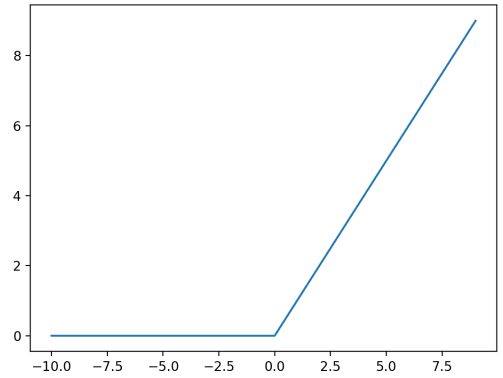
Activation Function: A function in neural networks that transforms the weighted sum of inputs into an output, introducing non-linearity and enabling the network to learn complex patterns in data.

Review

❖ Activation function







Activation functions:

- ❖ Final output of a node.
- ❖ Bring non-linearity to the network.

Review

❖ Activation function

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

2001

$$\text{softplus}(x) = \log(1+e^{x})$$

2010

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

2015

$$\text{ELU}(x) = \begin{cases} \alpha(e^{x}-1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

2015

$$\text{PReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

2017

$$SELU(x) = \begin{cases} \lambda x & \text{if } x \geq 0 \\ \lambda\alpha(e^{x}-1) & \text{if } x < 0 \end{cases}$$

$$\lambda \approx 1.0507$$

$$\alpha \approx 1.6733$$

2017

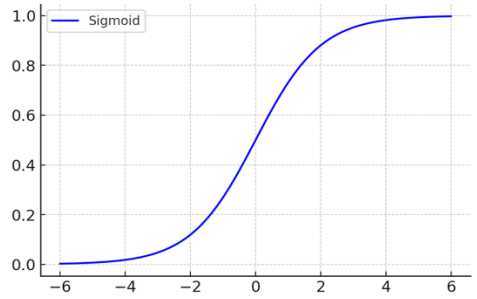
$$swish(x) = x * \frac{1}{1+e^{-x}}$$

Some famous activation functions.

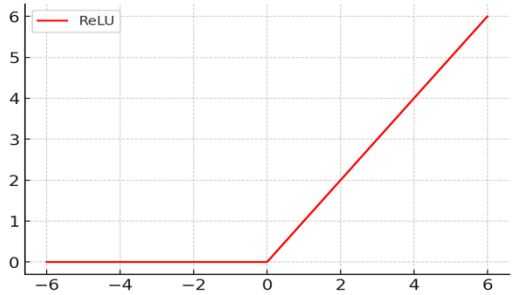
Review

❖ Activation function: Graphs

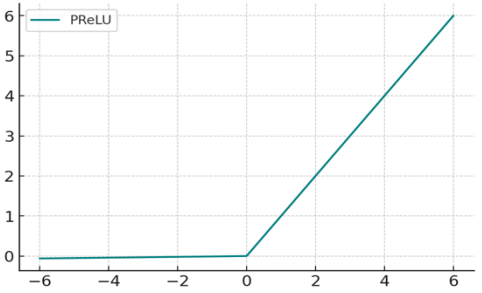
Sigmoid Function



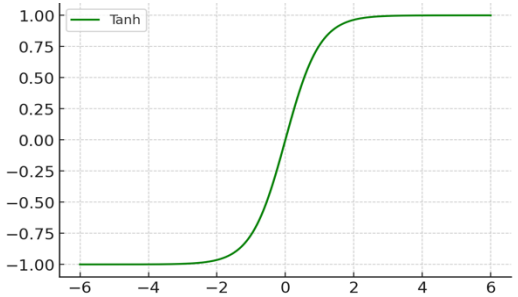
ReLU Function



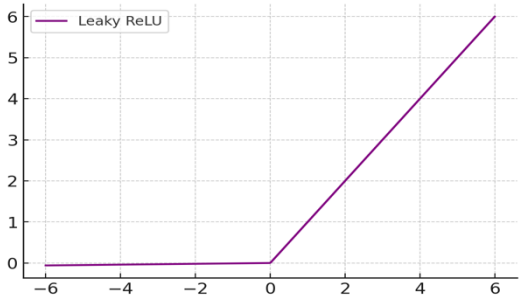
PReLU Function



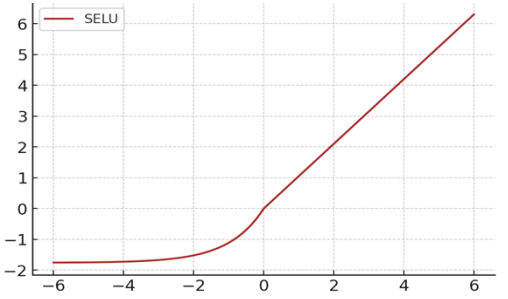
Tanh Function



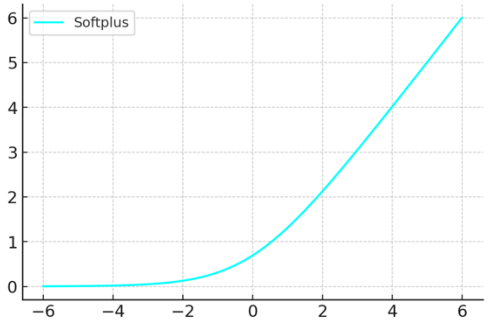
Leaky ReLU Function



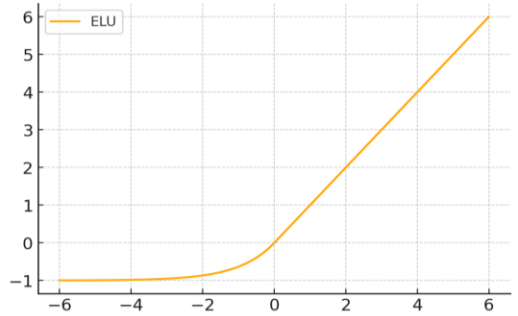
SELU Function



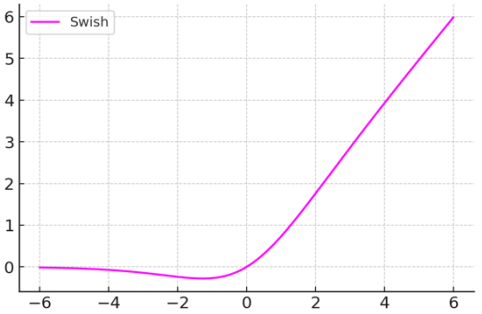
Softplus Function



ELU Function



Swish Function

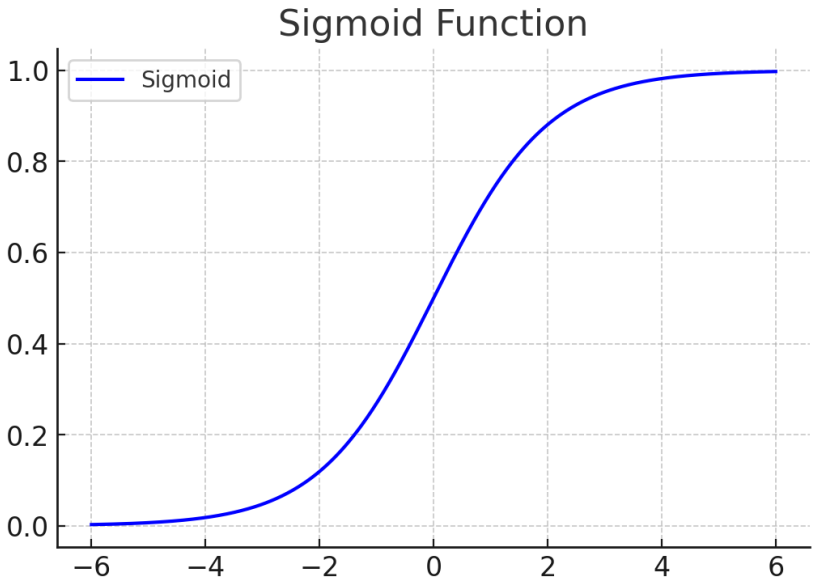


Review

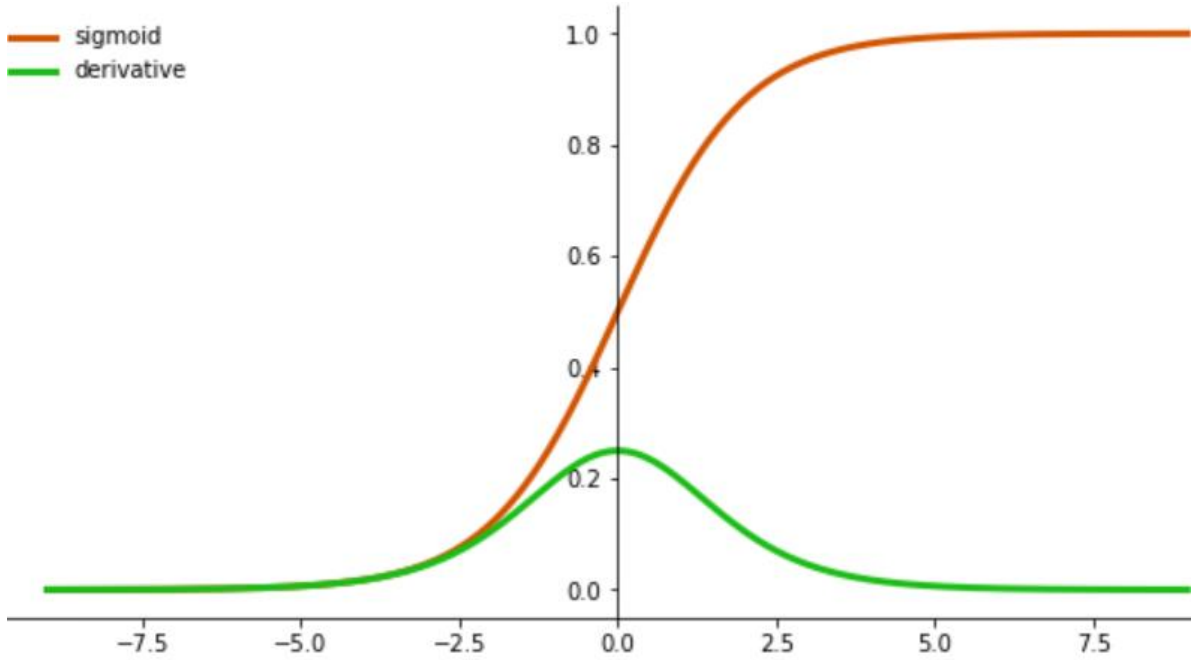
❖ Activation function: Sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\text{sigmoid}'(x) = \text{sigmoid}(x)\,(1 - \text{sigmoid}(x))$$



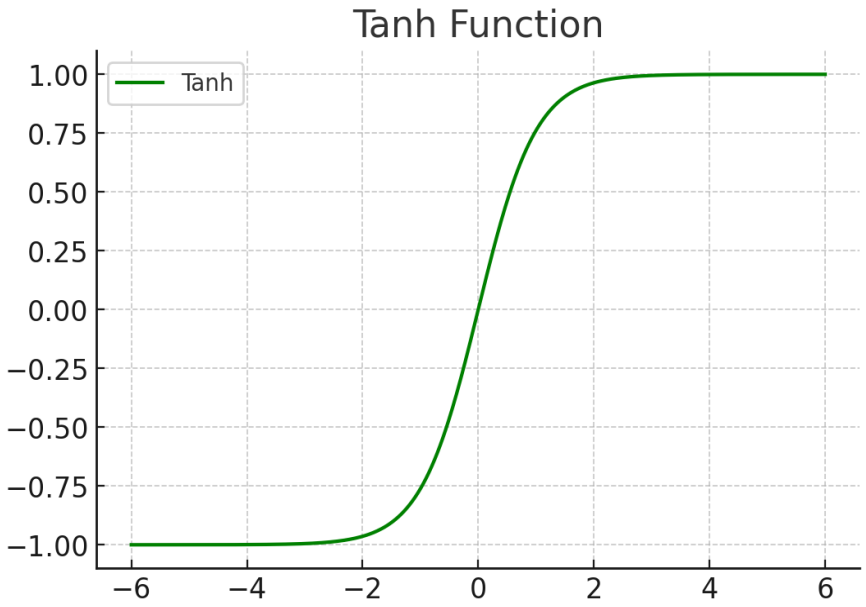
Domain: $(-\infty, +\infty)$,
Range: $(0,1)$



Review

❖ Activation function: Tanh

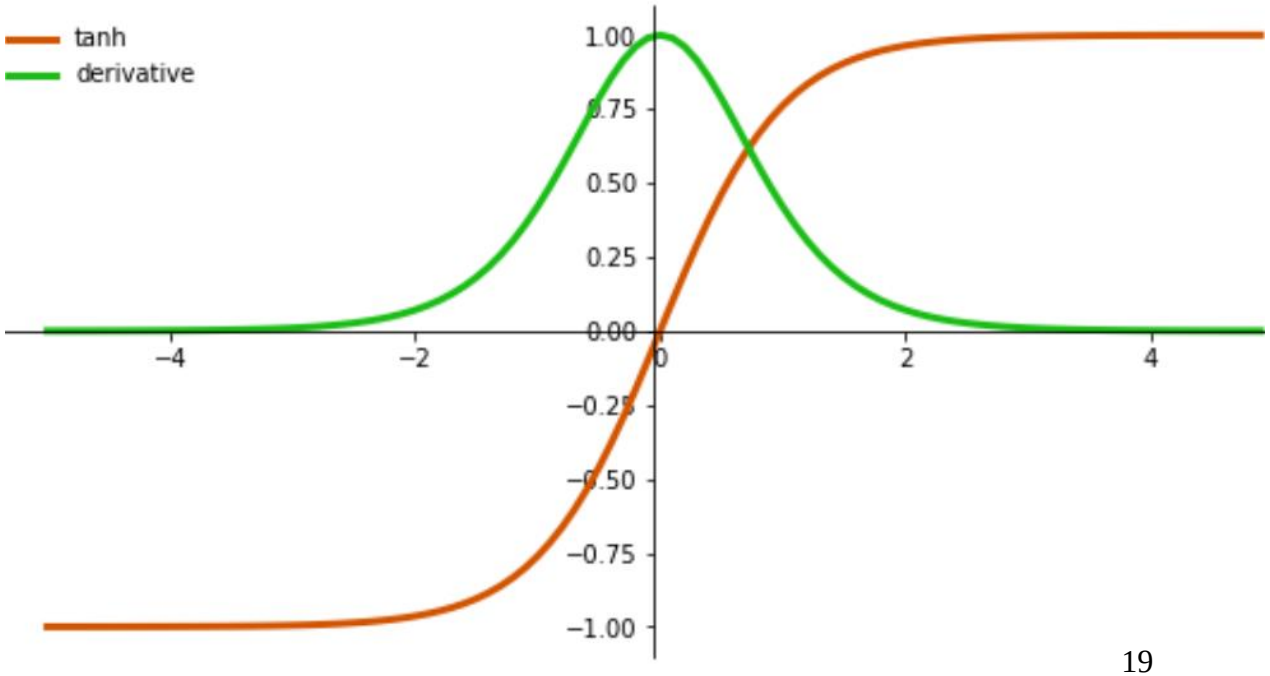
$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$



Domain: $(-\infty, +\infty)$,
Range: $(-1, 1)$

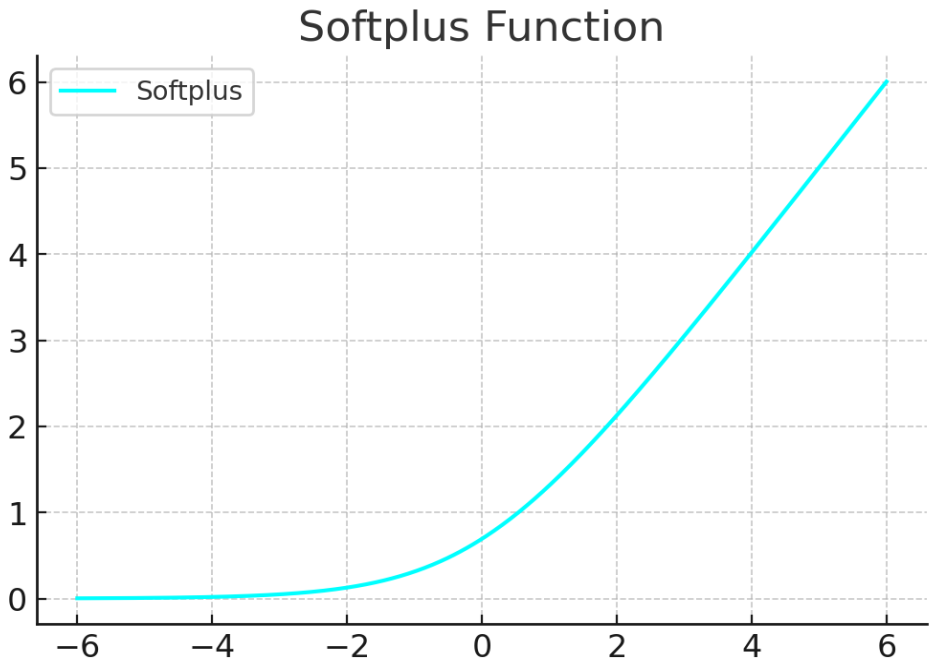
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1 = 1 - \frac{2}{e^{2x} + 1}$$

$$tanh'(x) = 1 - tanh^2(x)$$



Review

❖ Activation function: Softplus

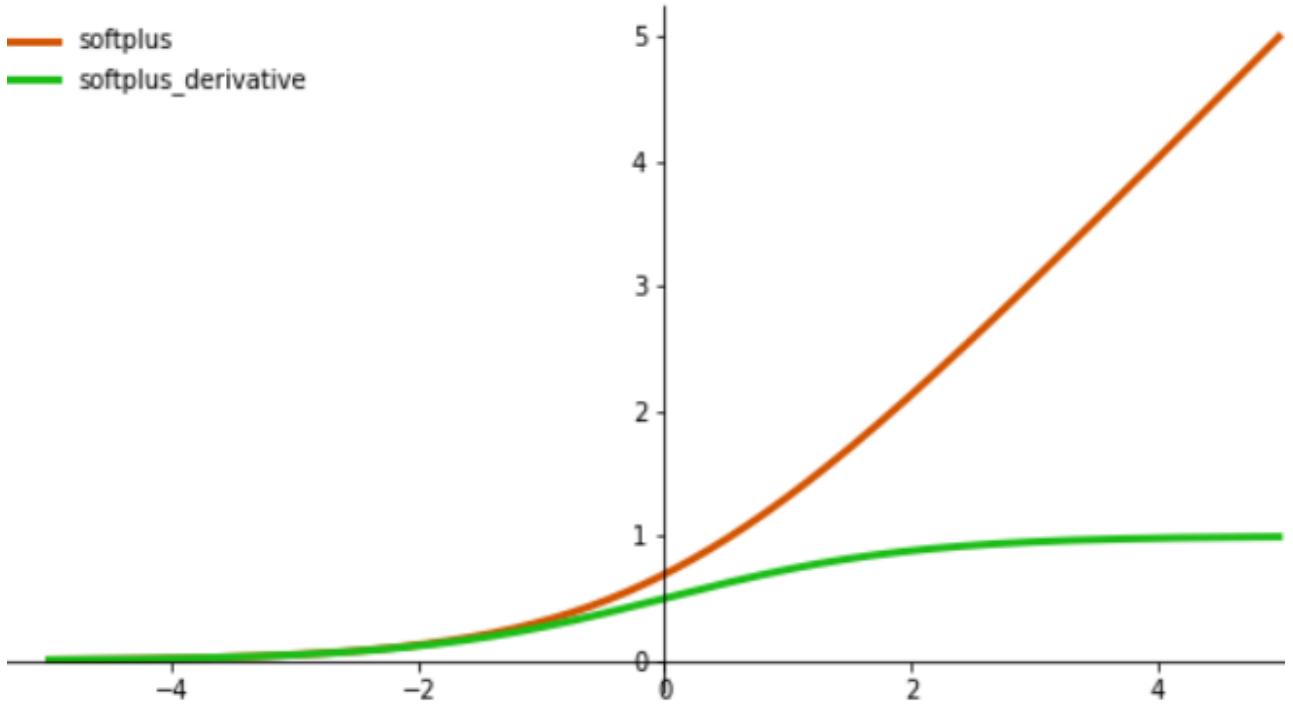


Domain: $(-\infty,+\infty)$,
Range: $(0,+\infty)$

2001

$$\text{softplus}(x) = \log(1 + e^x)$$

$$\text{softplus}'(x) = \frac{1}{1 + e^{-x}}$$



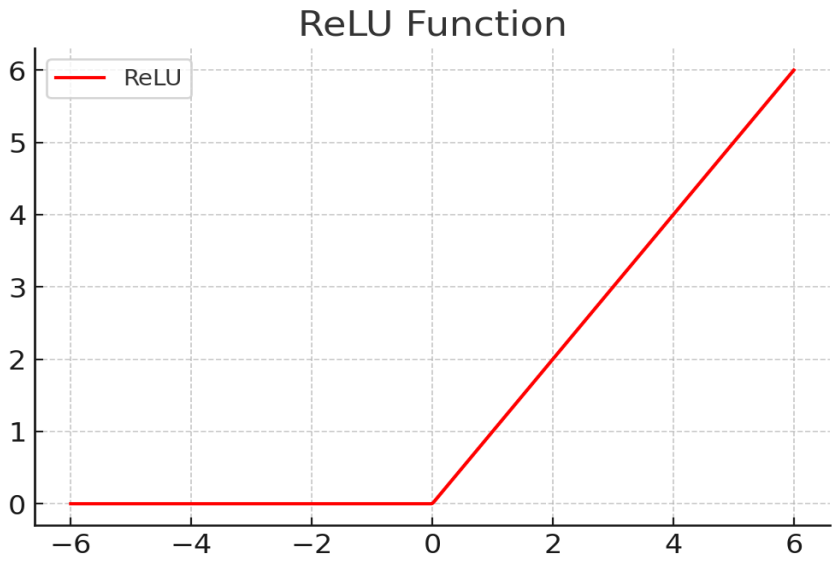
Review

❖ Activation function: ReLU

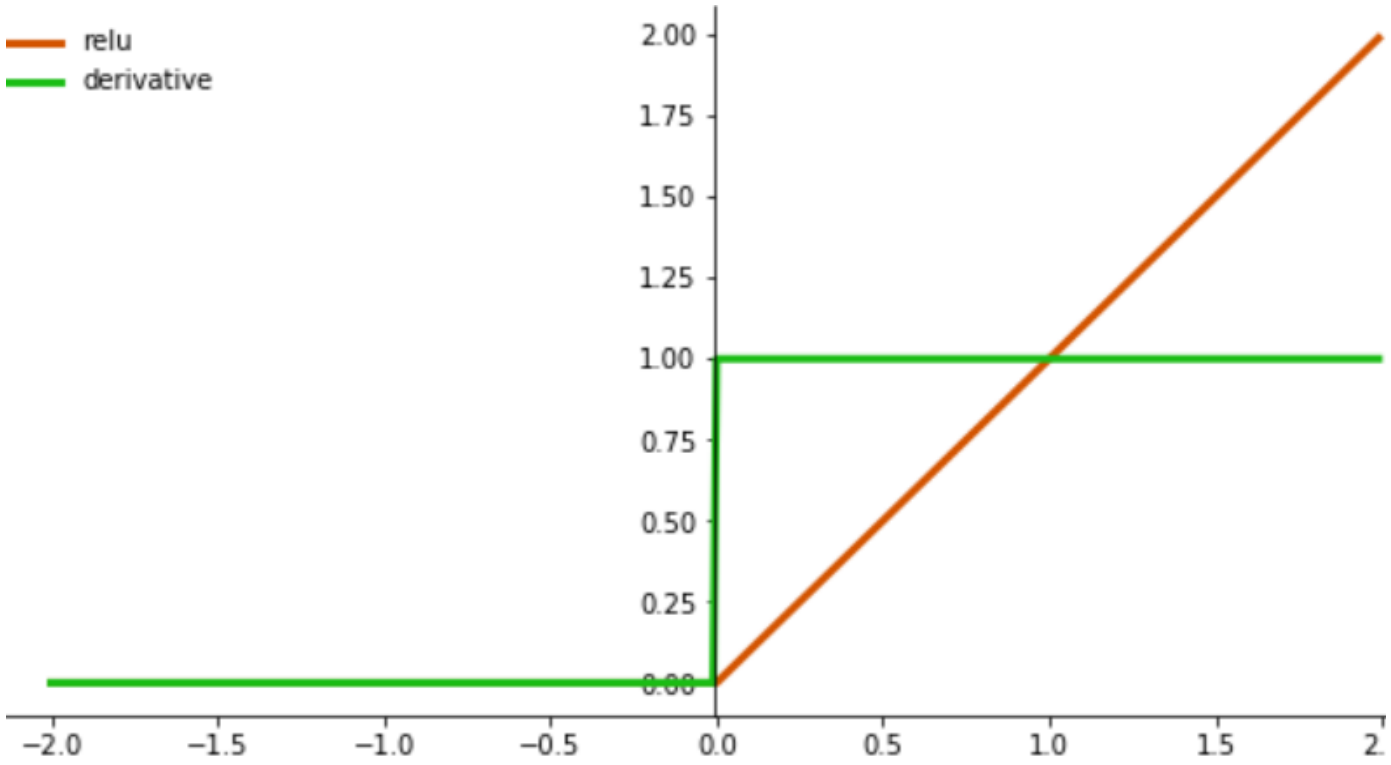
2010

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

$$\text{ReLU}'(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$



Domain: $(-\infty,+\infty)$,
Range: $[0,+\infty)$

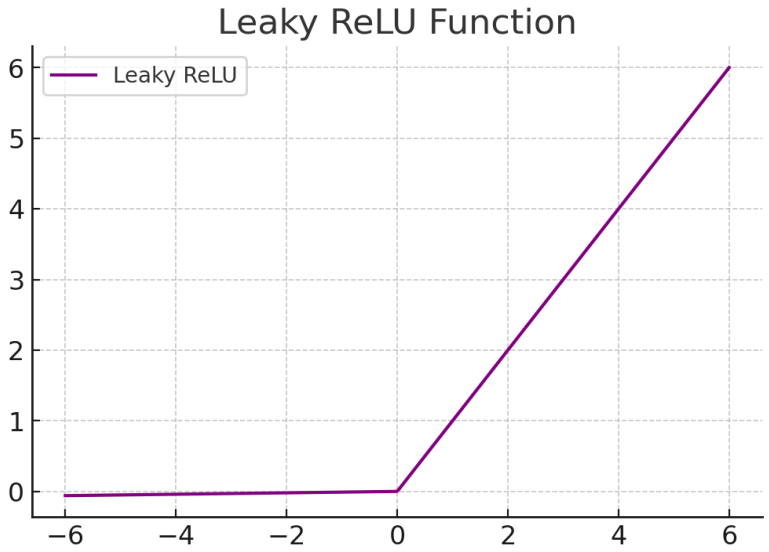


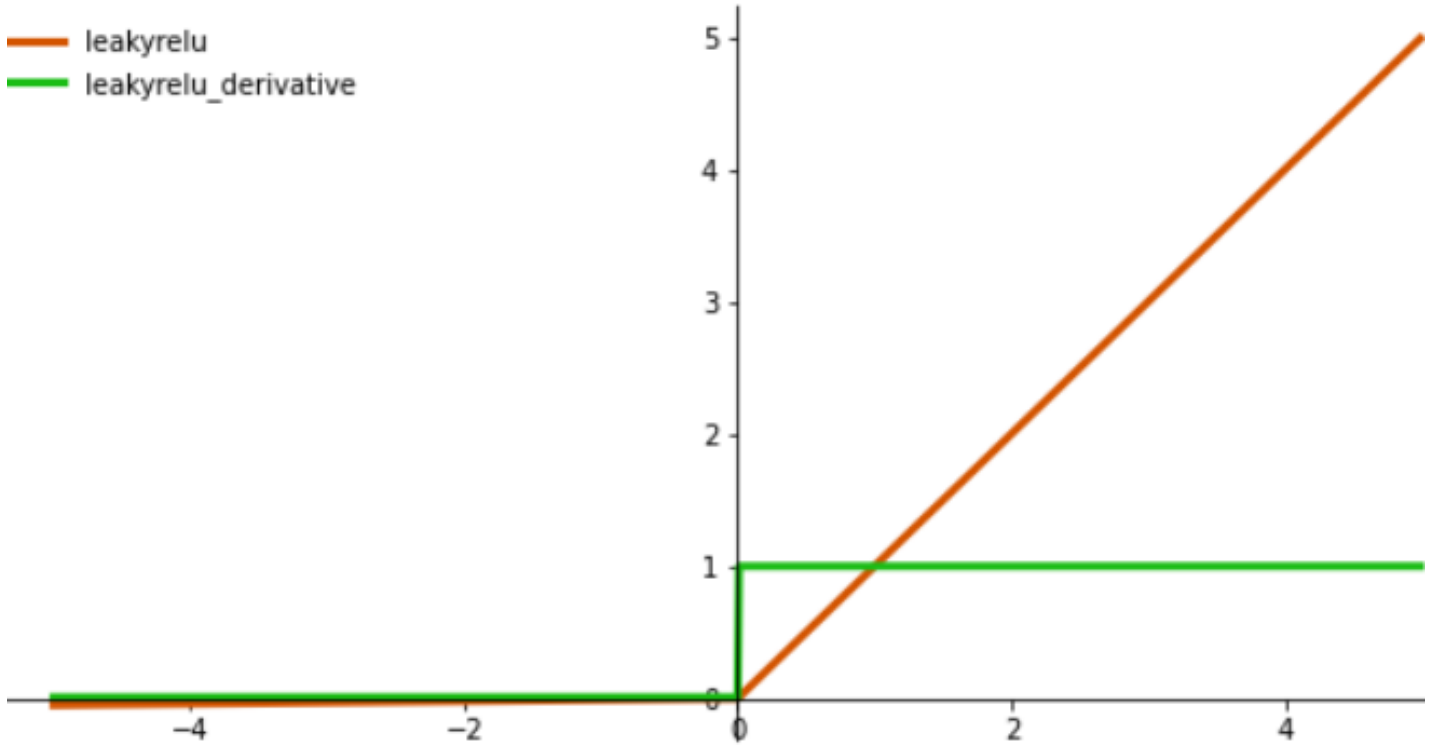
Review

❖ Activation function: LeakyReLU

$$\text{LeakyReLU}(x) = \begin{cases} 0.01x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

$$\text{LeakyReLU}'(x) = \begin{cases} 0.01 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

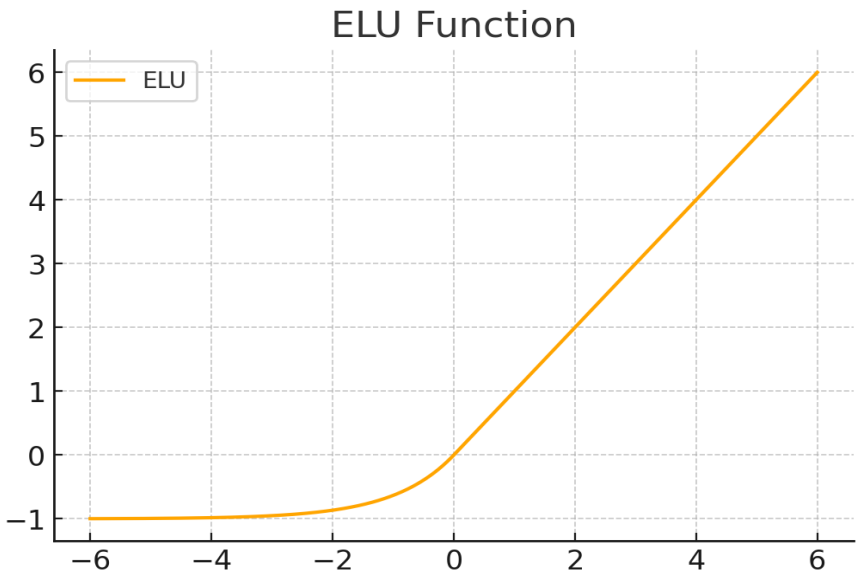




Domain: $(-\infty, +\infty)$,
Range: $(-\infty, +\infty)$

Review

❖ Activation function: ELU

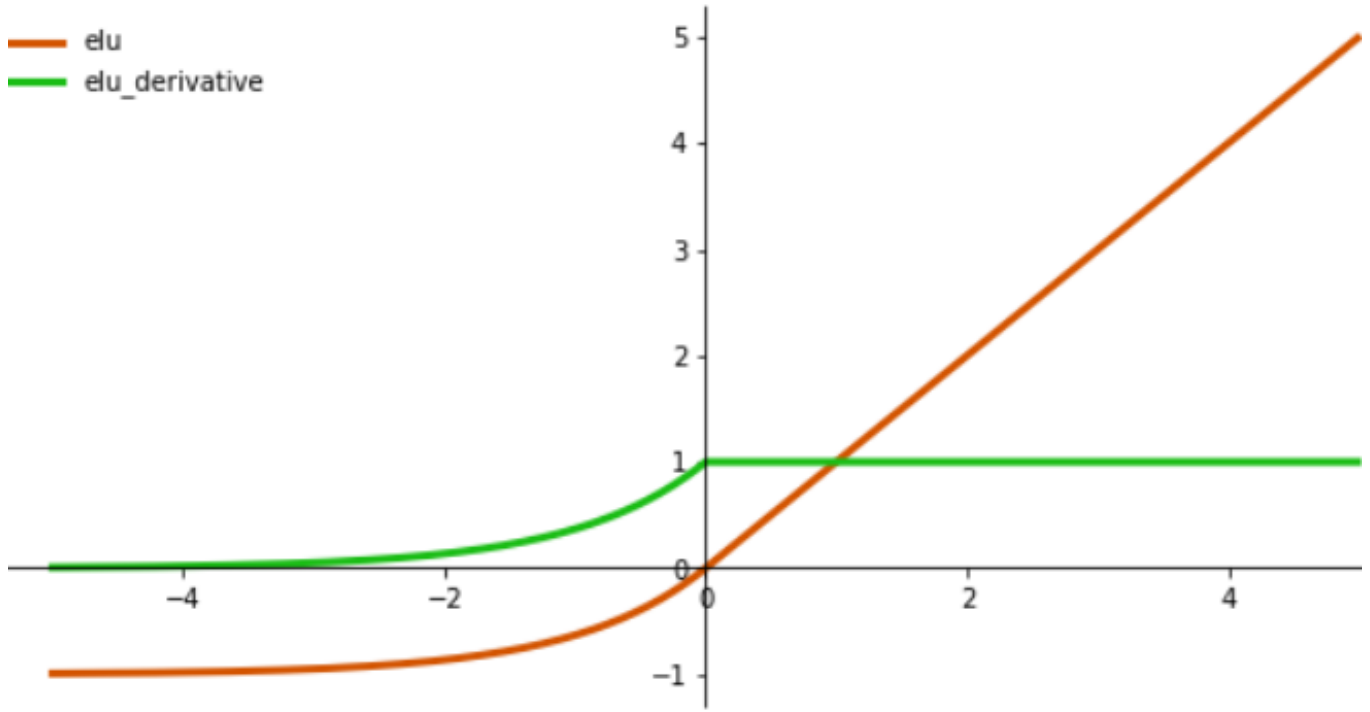


Domain: $(-\infty,+\infty)$,
Range: $(-\alpha,+\infty)$ with $\alpha>0$

2015

$$\mathrm{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

$$\mathrm{ELU}'(x) = \begin{cases} \alpha e^x & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$



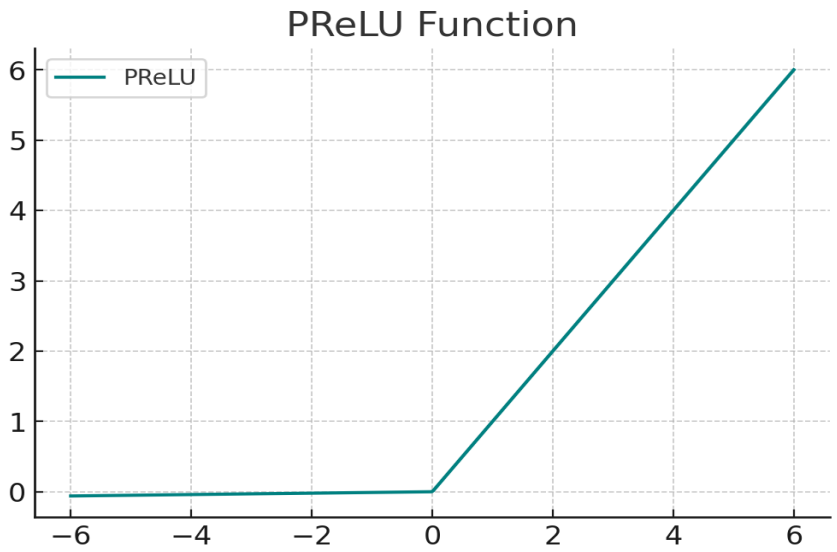
Review

❖ Activation function: PReLU

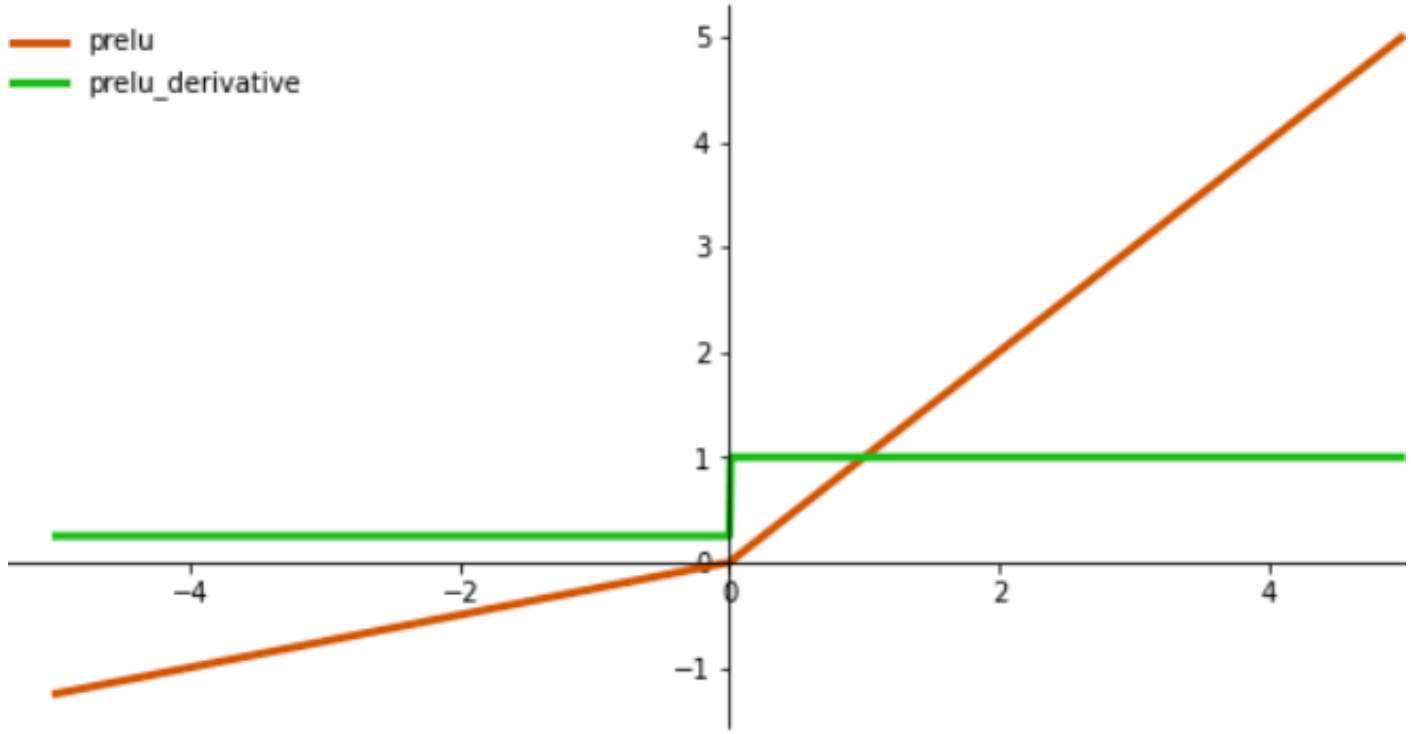
2015

$$\text{PReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

$$\text{PReLU}'(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

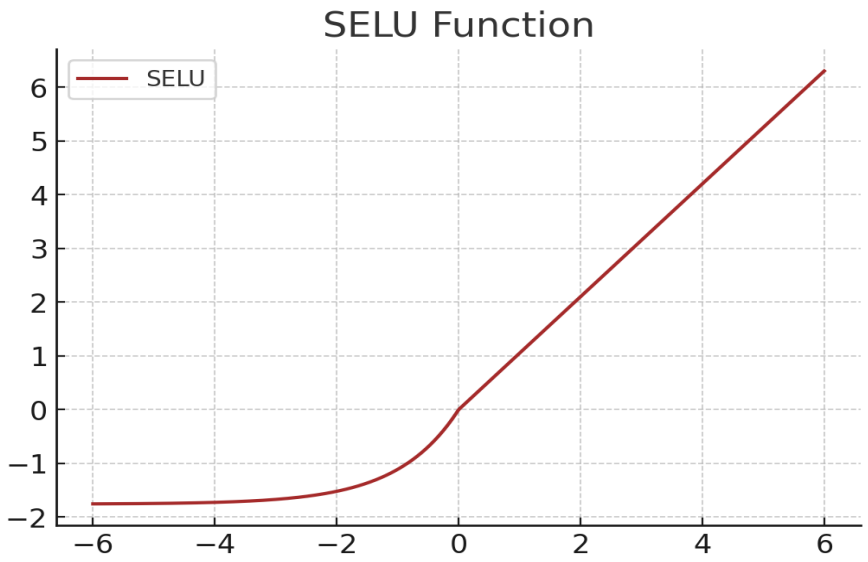


Domain: $(-\infty,+\infty)$,
Range: $(-\infty,+\infty)$



Review

❖ Activation function: SELU



2017

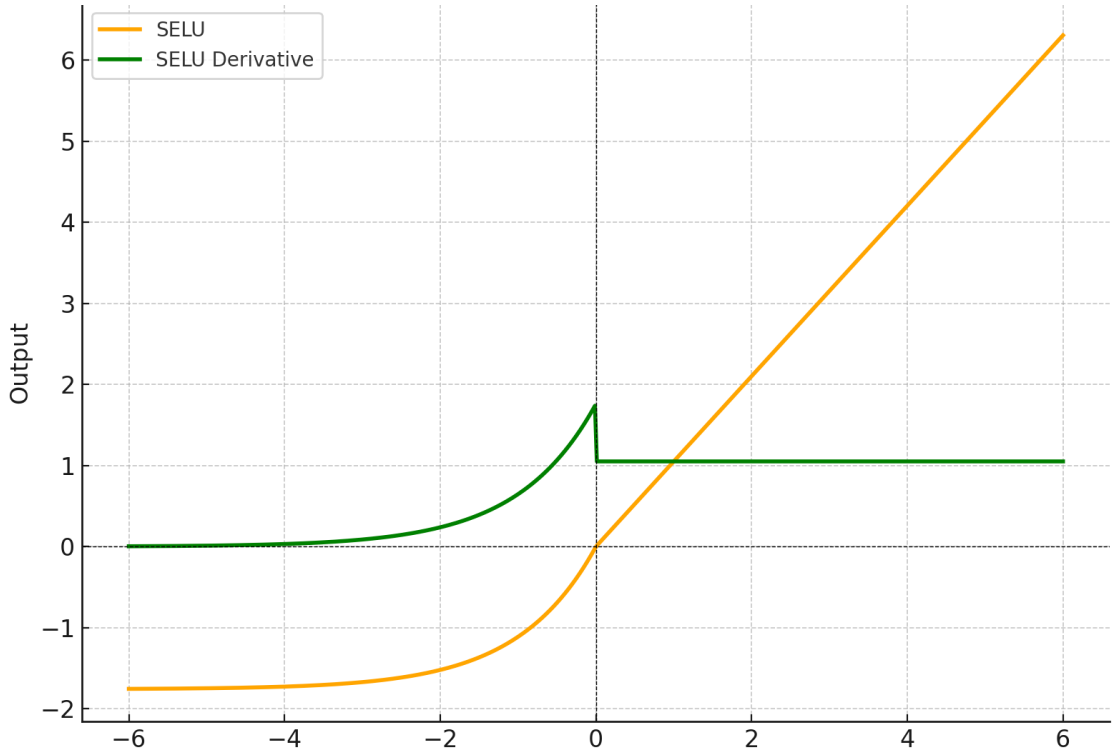
$$SELU(x) = \begin{cases} \lambda x & \text{if } x \geq 0 \\ \lambda\alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

$$\lambda \approx 1.0507$$

$$\alpha \approx 1.6733$$

$$\text{SELU}'(x) = \begin{cases} \lambda & \text{if } x \leq 0 \\ \lambda\alpha e^x & \text{if } x > 0 \end{cases}$$

Domain: $(-\infty,+\infty)$,
Range: $(-\lambda\alpha,+\infty)$
with $\lambda \approx 1.0507$, $\alpha \approx 1.6733$

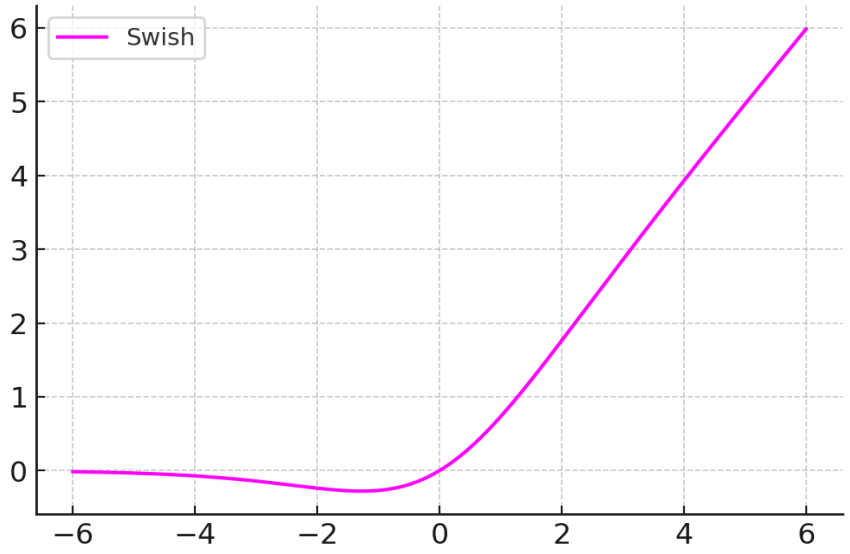


Review

❖ Activation function: Swish

2017

$$swish(x) = x * \frac{1}{1 + e^{-x}}$$

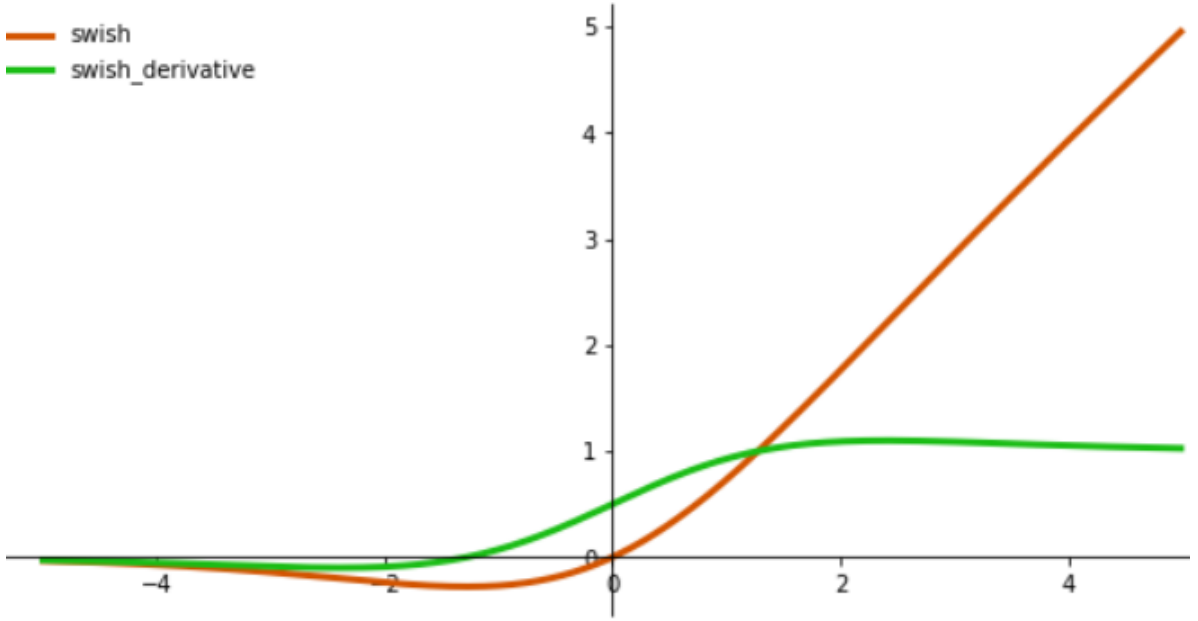


Domain: $(-\infty,+\infty)$,
Range: $(-\infty,+\infty)$

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

$$swish(x) = \frac{x}{1 + e^{-x}} = x\,\sigma(x)$$

$$swish'(x) = swish(x) + \sigma(x)\,(1 - swish(x))$$



AI VIET NAM
@aivietnam.edu.vn

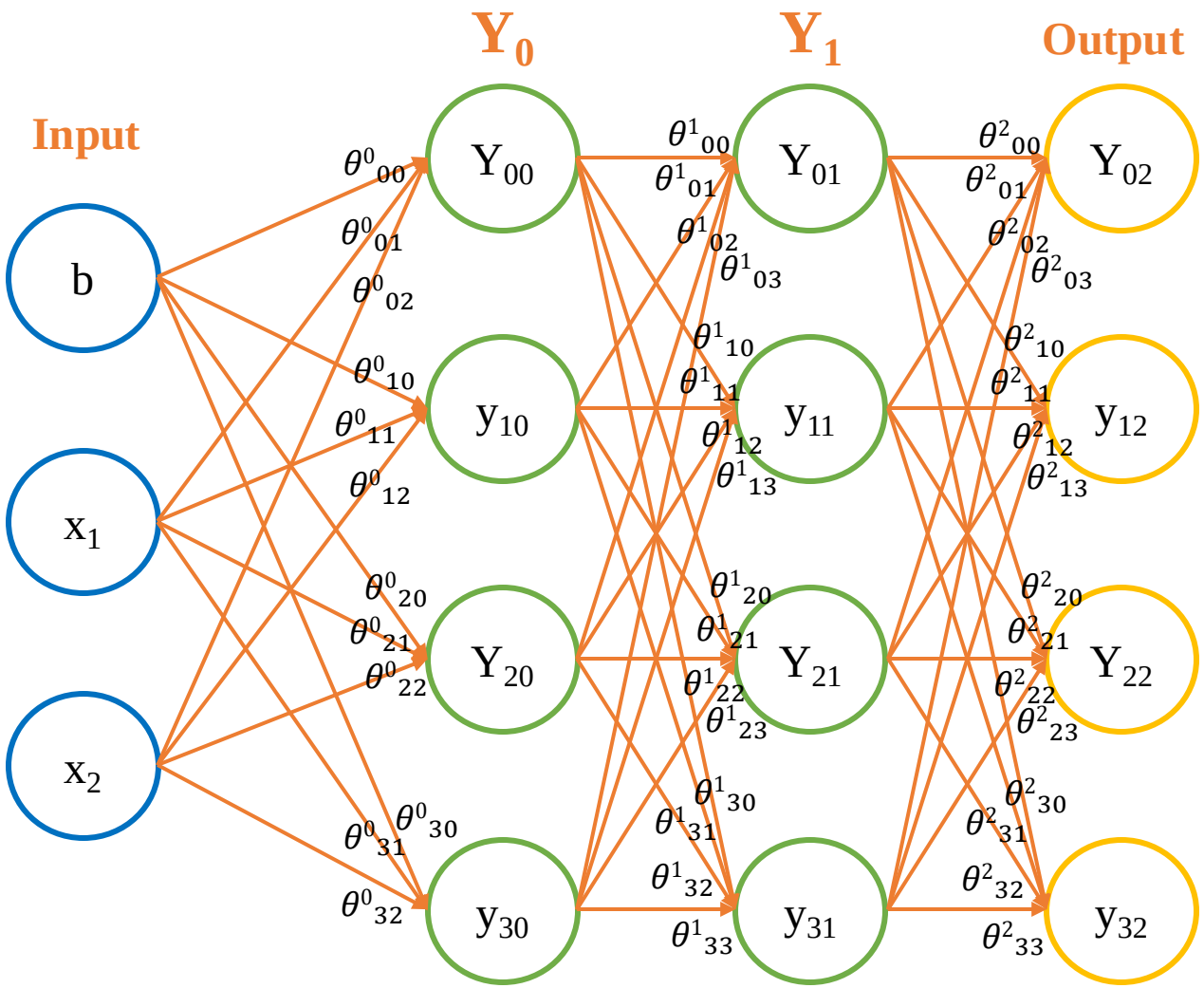
Review

❖ Activation function

Function	Pros	Cons	Applications
Sigmoid	Good for probabilities. Simple.	Vanishing gradient. Not centered.	Binary classification (e.g., logistic regression).
Tanh	Zero-centered. Good for RNNs.	Vanishing gradient.	RNN hidden layers (e.g., sentiment analysis, language modeling).
ReLU	Simple and efficient. No saturation.	Dying ReLU problem.	Default for CNNs, MLPs.
Softplus	Smooth ReLU approximation. Continuous and smooth.	Slower than ReLU.	Alternative to ReLU for smooth outputs in regression tasks
Leaky ReLU	Small slope for x<0. Solves Dying ReLU. Simple.	Small bias in outputs.	Networks with deeper layers (e.g., GANs like DCGAN for generating images).
PReLU	Learnable slope for x<0. Adaptive activation.	Higher computation cost.	Advanced deep networks (e.g., face recognition in VGGFace models).
ELU	Smooth. Zero-centered. Reduces vanishing gradient.	Slower than ReLU.	Robust networks (e.g., deep networks for speech recognition).
SELU	Self-normalizing. Stable training for deep layers.	Needs specific normalization.	Self-normalizing networks (SNNs) (e.g., tabular data in structured prediction).
Swish	Smooth. Non-monotonic. Improves deep model accuracy.	Computationally expensive.	State-of-the-art architectures (e.g., EfficientNet for image classification, Transformers for NLP tasks).

Review

❖ Activation function

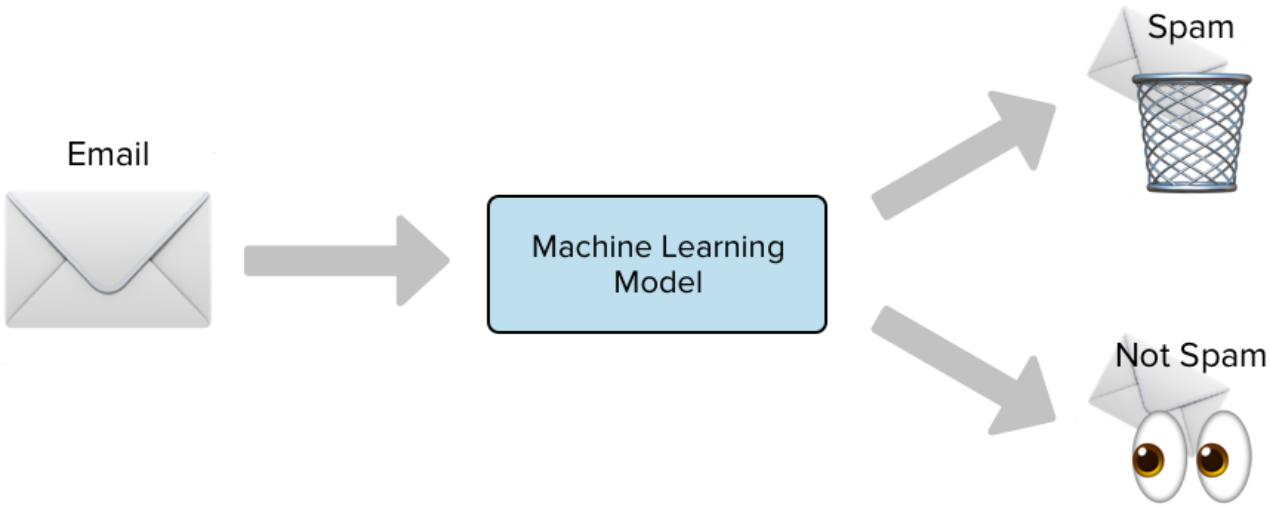


Question: Which activation function should we use?

Review

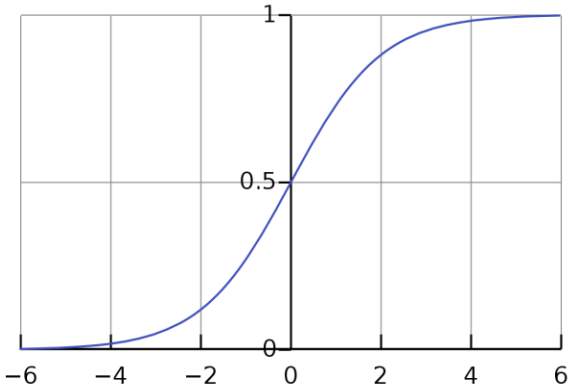
❖ Case study 1

Problem output:
categorial value



= 1

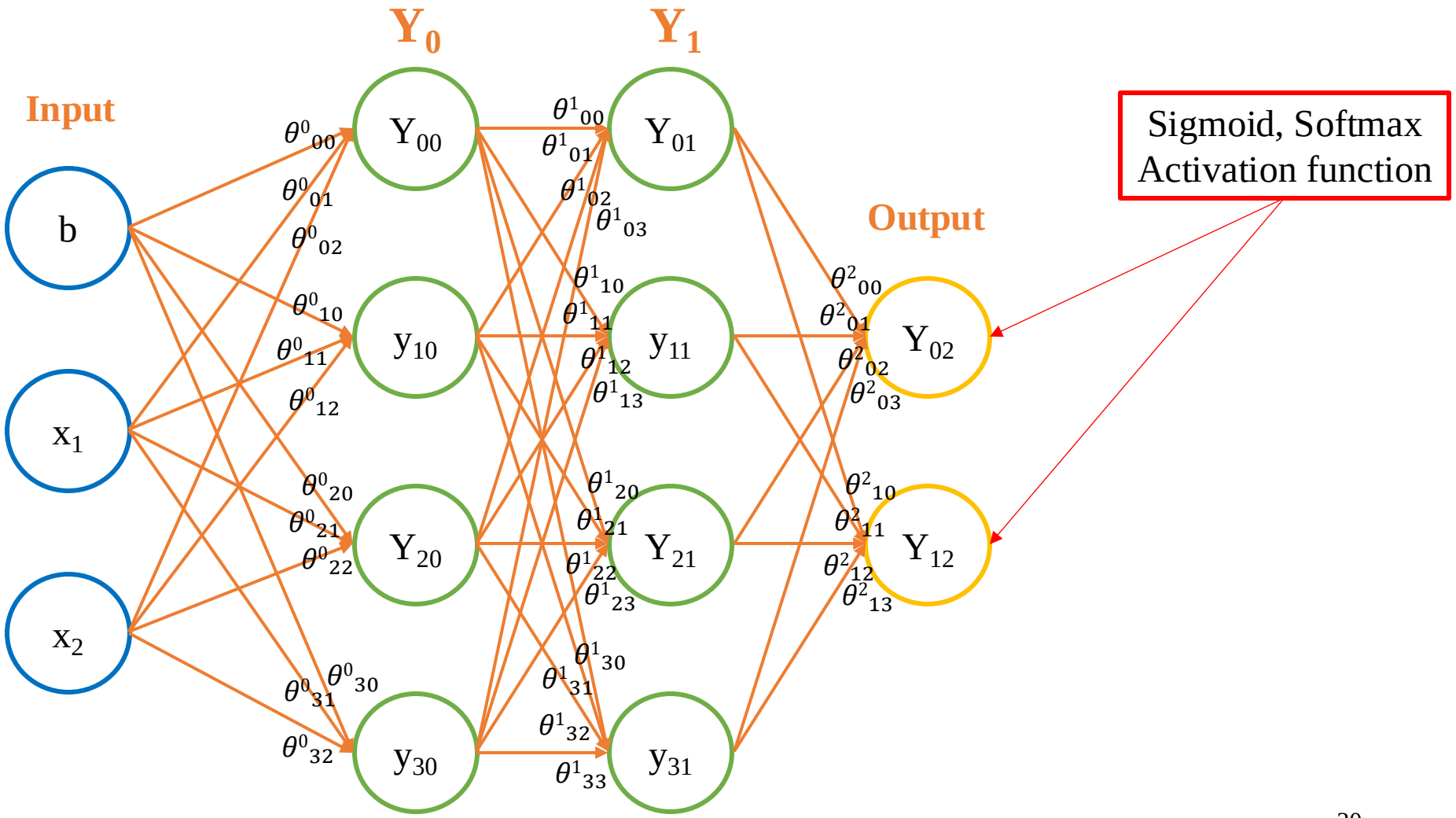
= 0



We could use Sigmoid, Tanh, Softmax… for Output Layer.

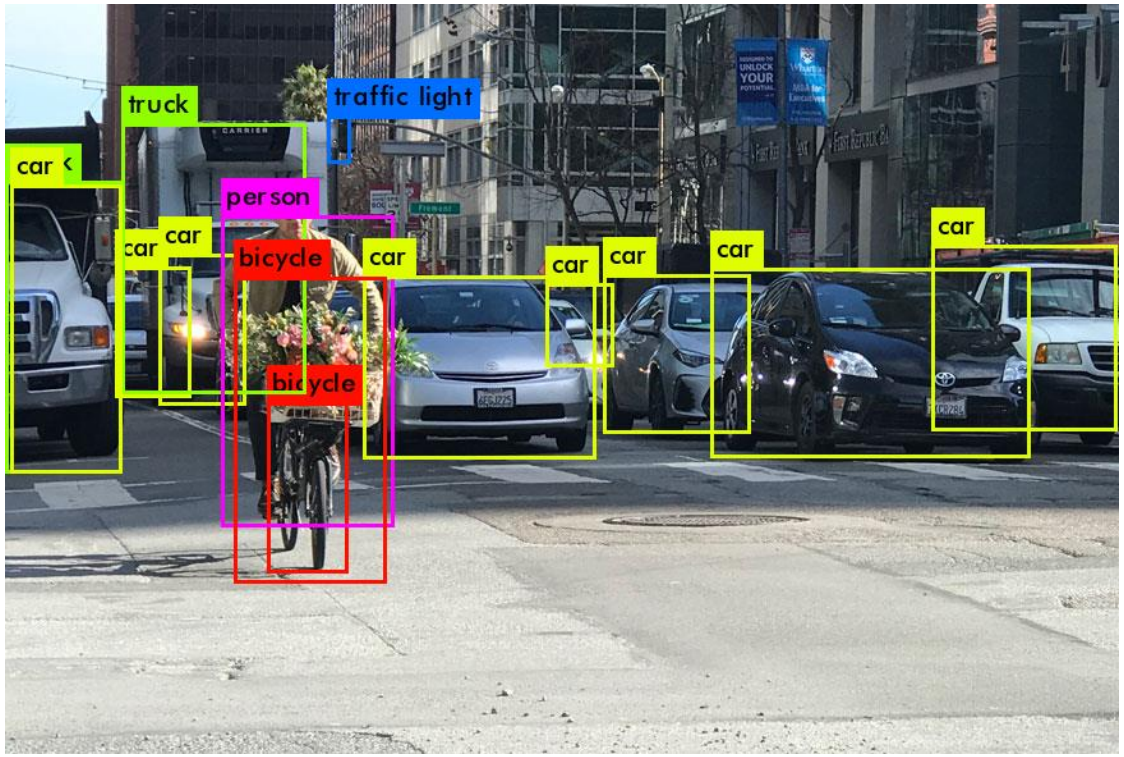
Review

❖ Case study 1



Review

❖ Case study 2



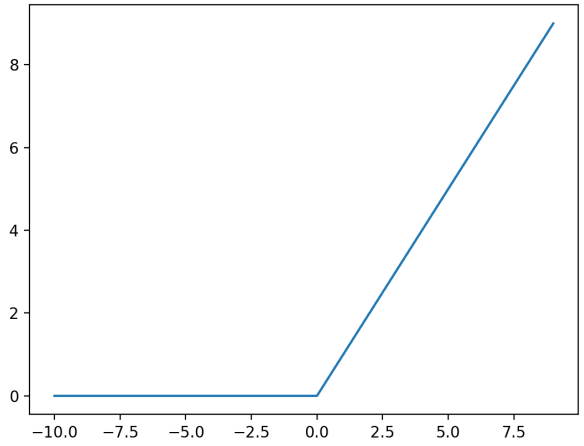
(x_left, y_left) width

height

(x_right, y_right)

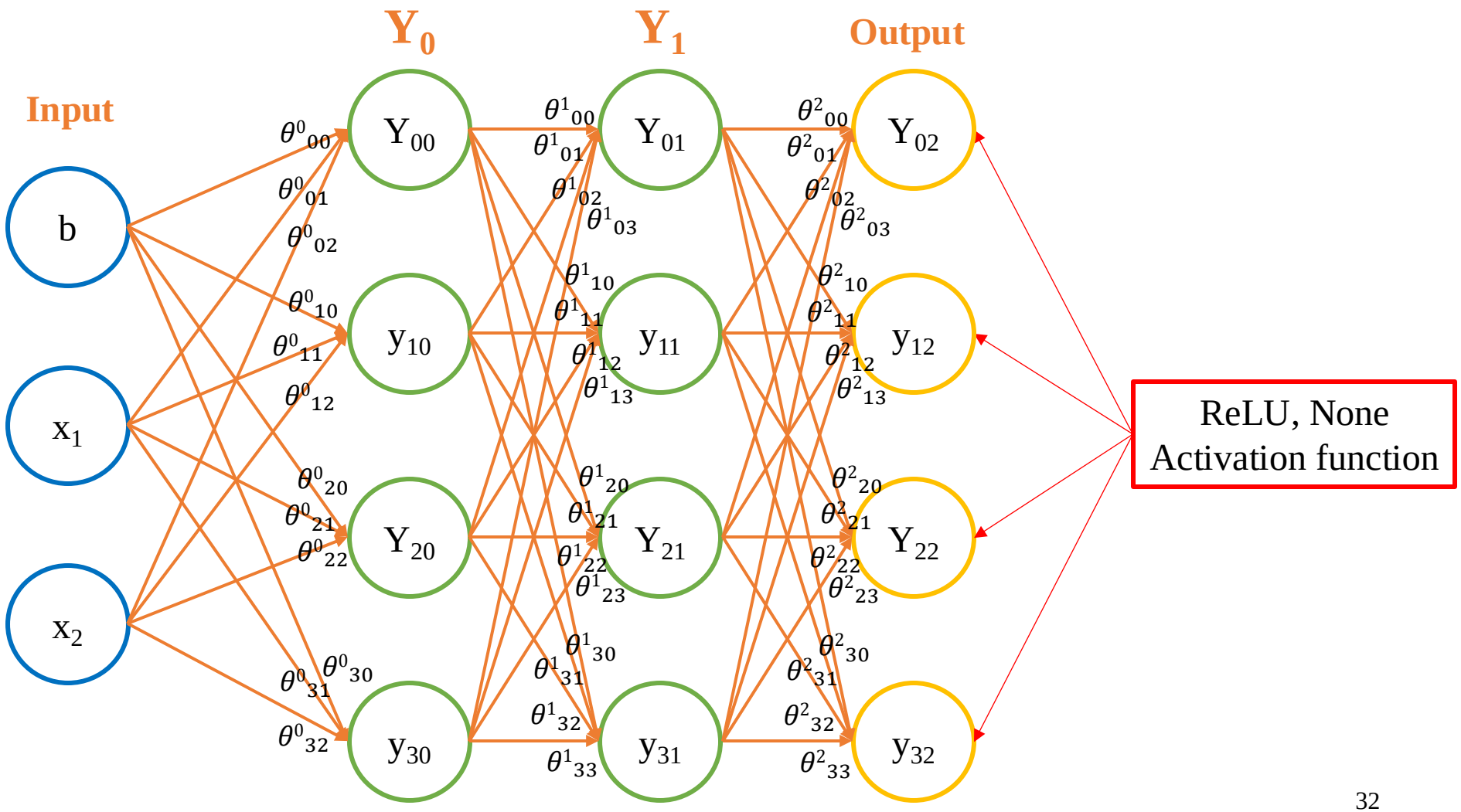
Object BB Coordinate: positive floating value.

We could use ReLU, None… for Output Layer.



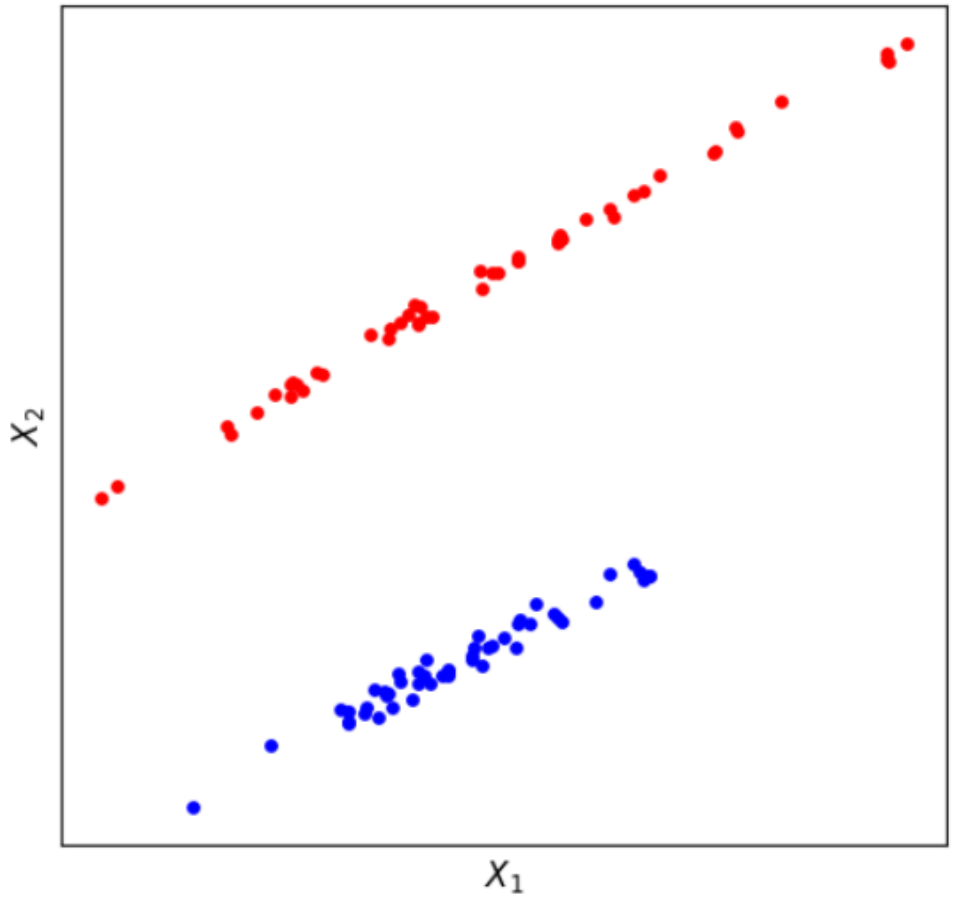
Review

❖ Case study 2



Review

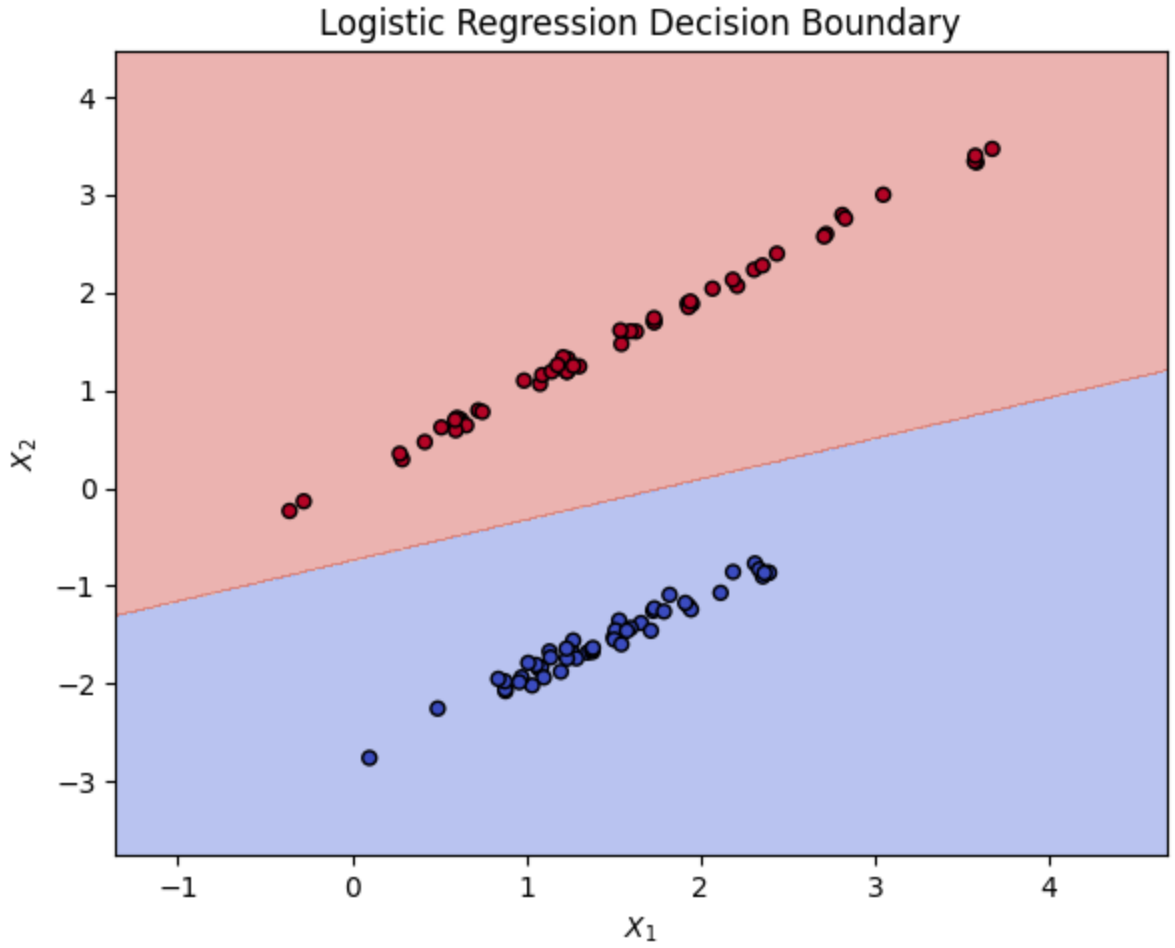
❖ Why we need MLP?

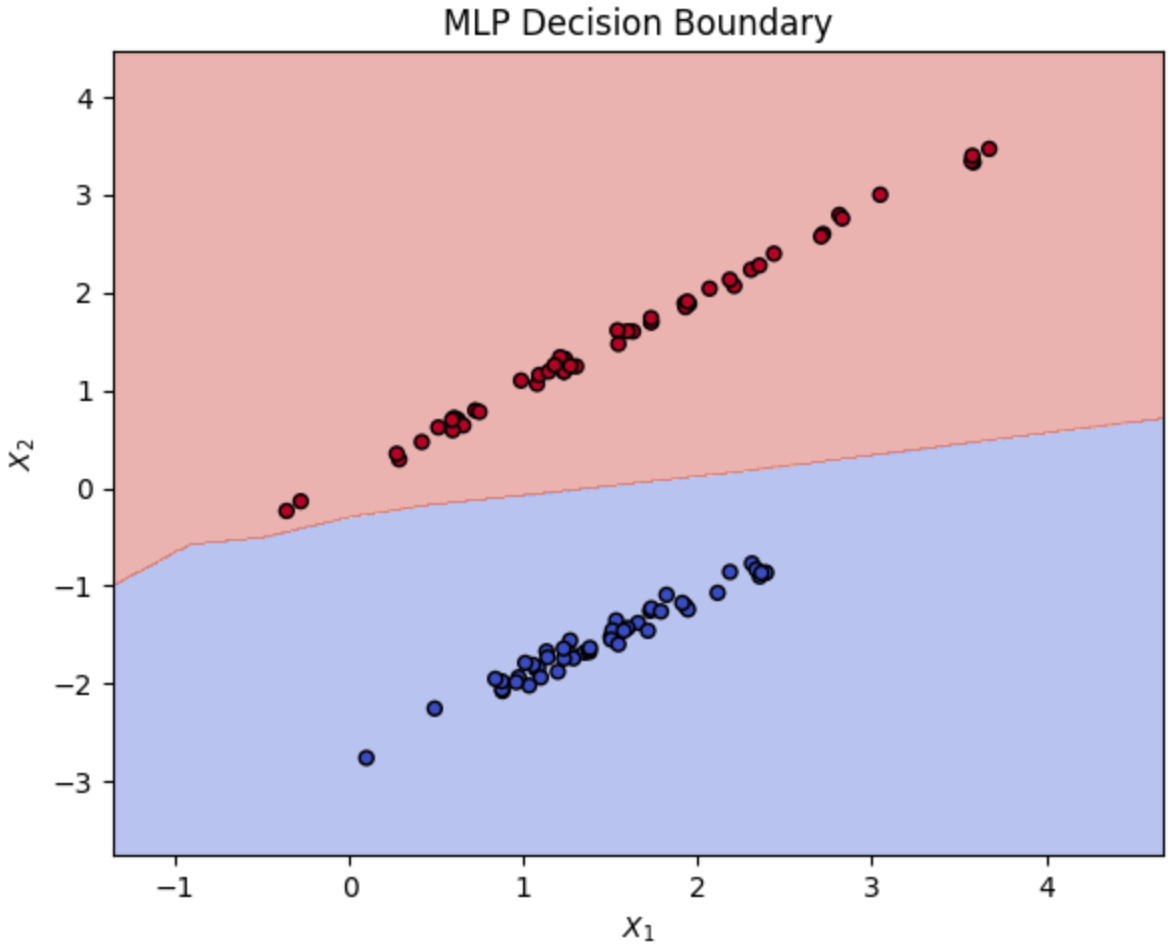


A simple (linear) classification dataset.

Review

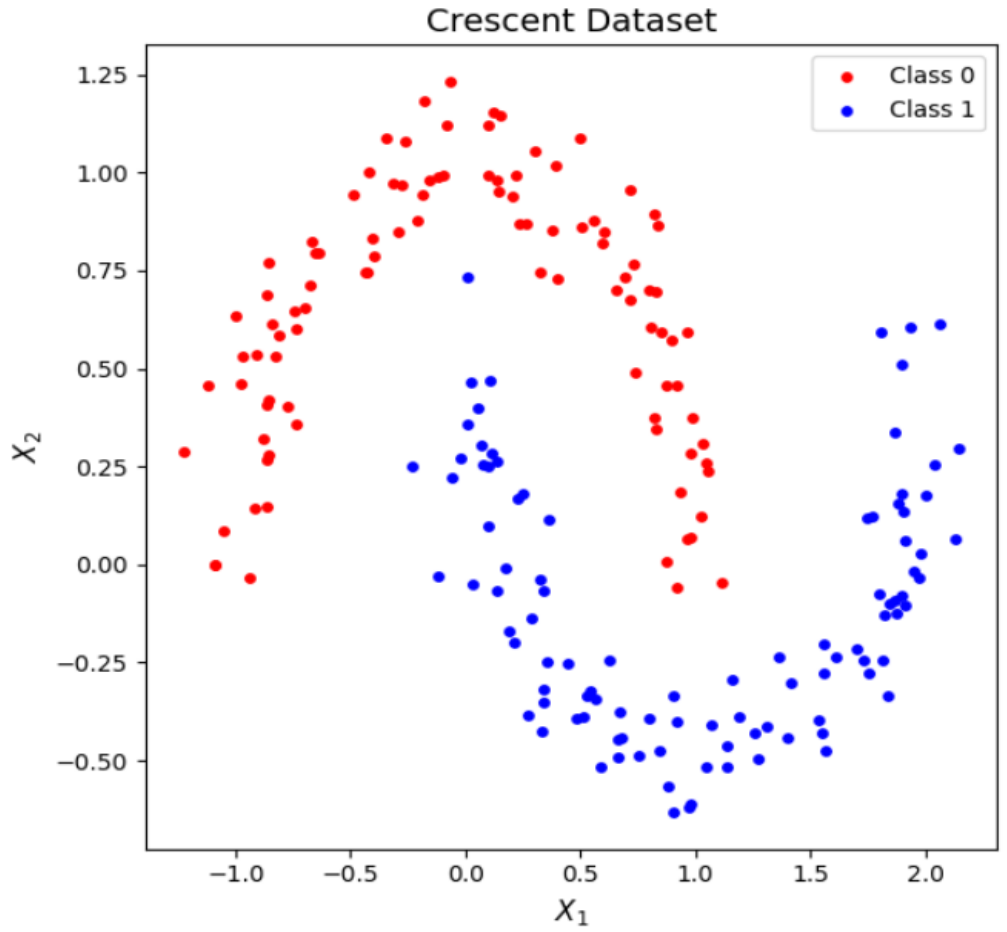
❖ Why we need MLP?





Review

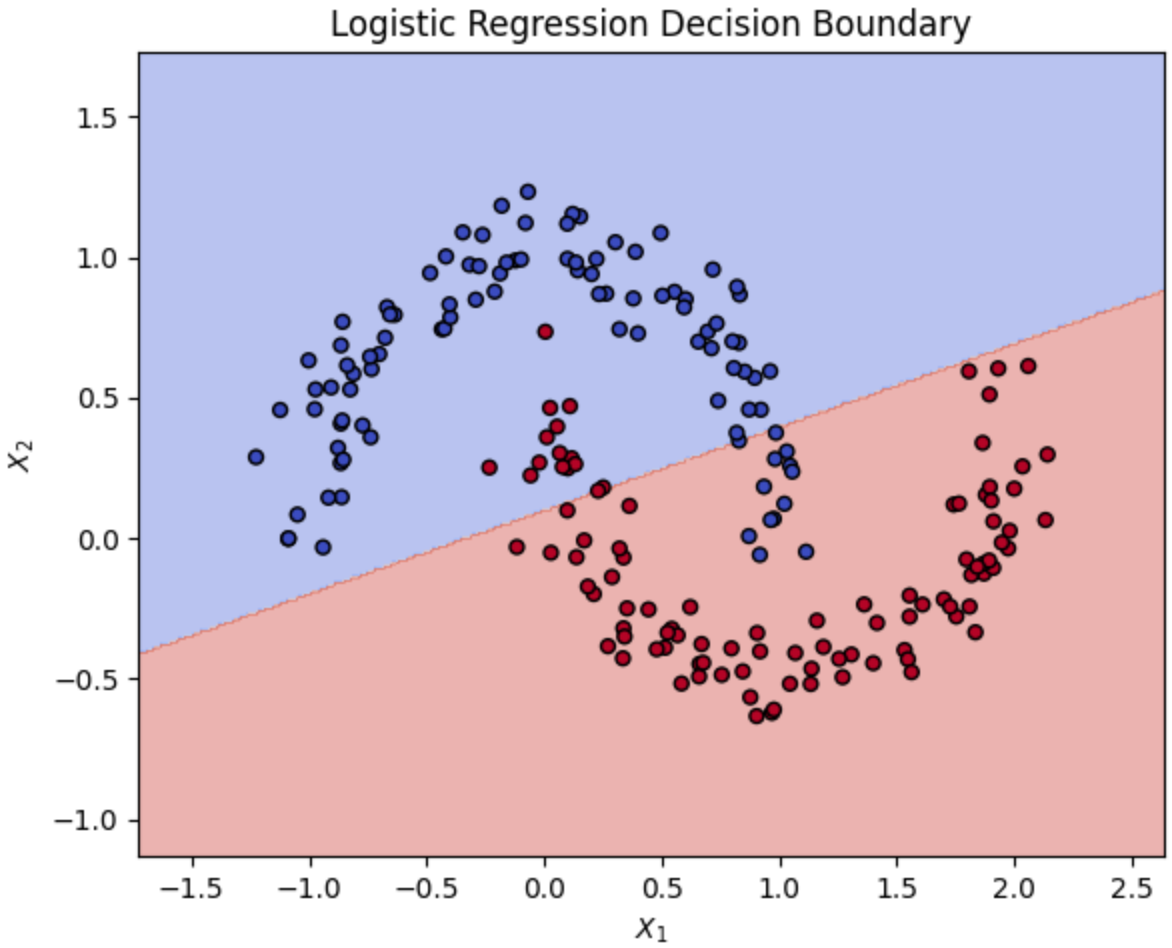
❖ Why we need MLP?

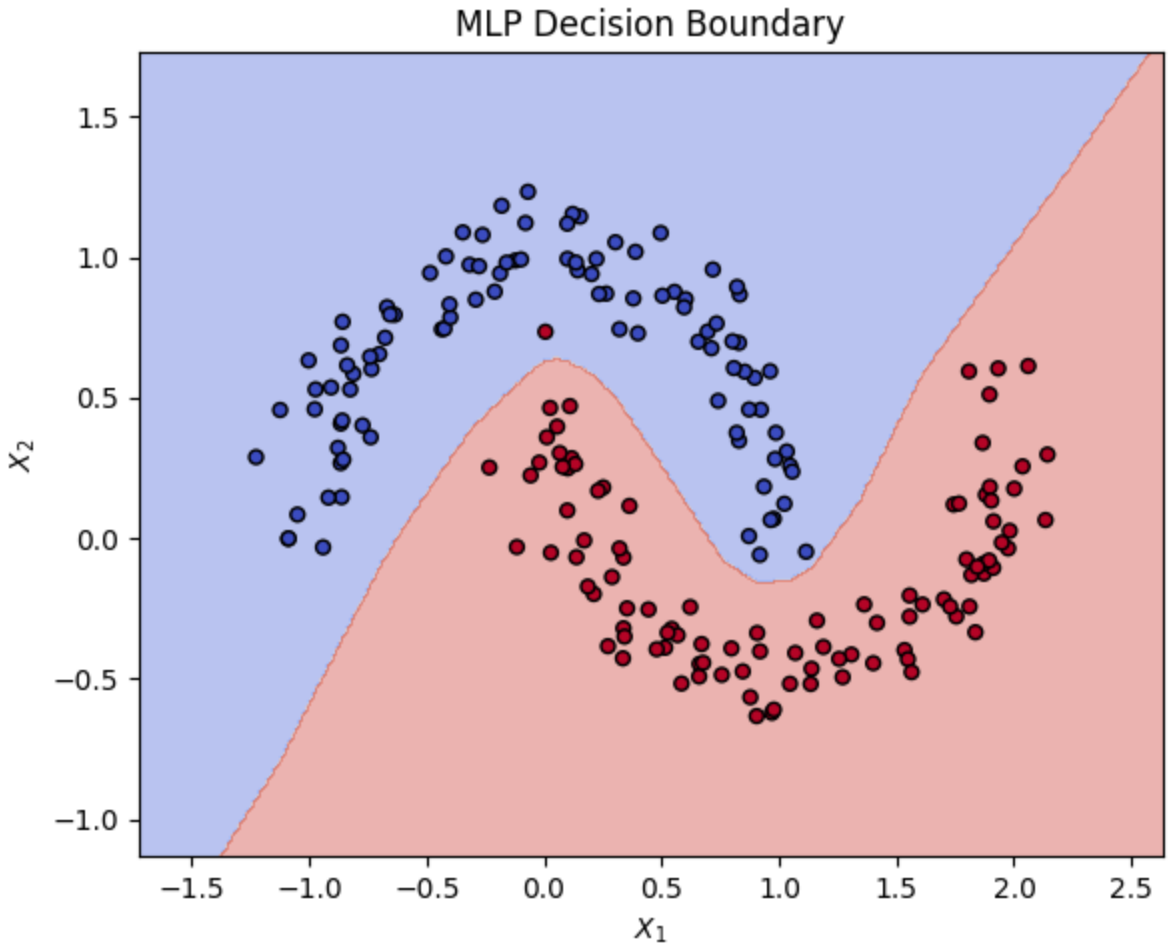


A more complex (non-linear) classification dataset.

Review

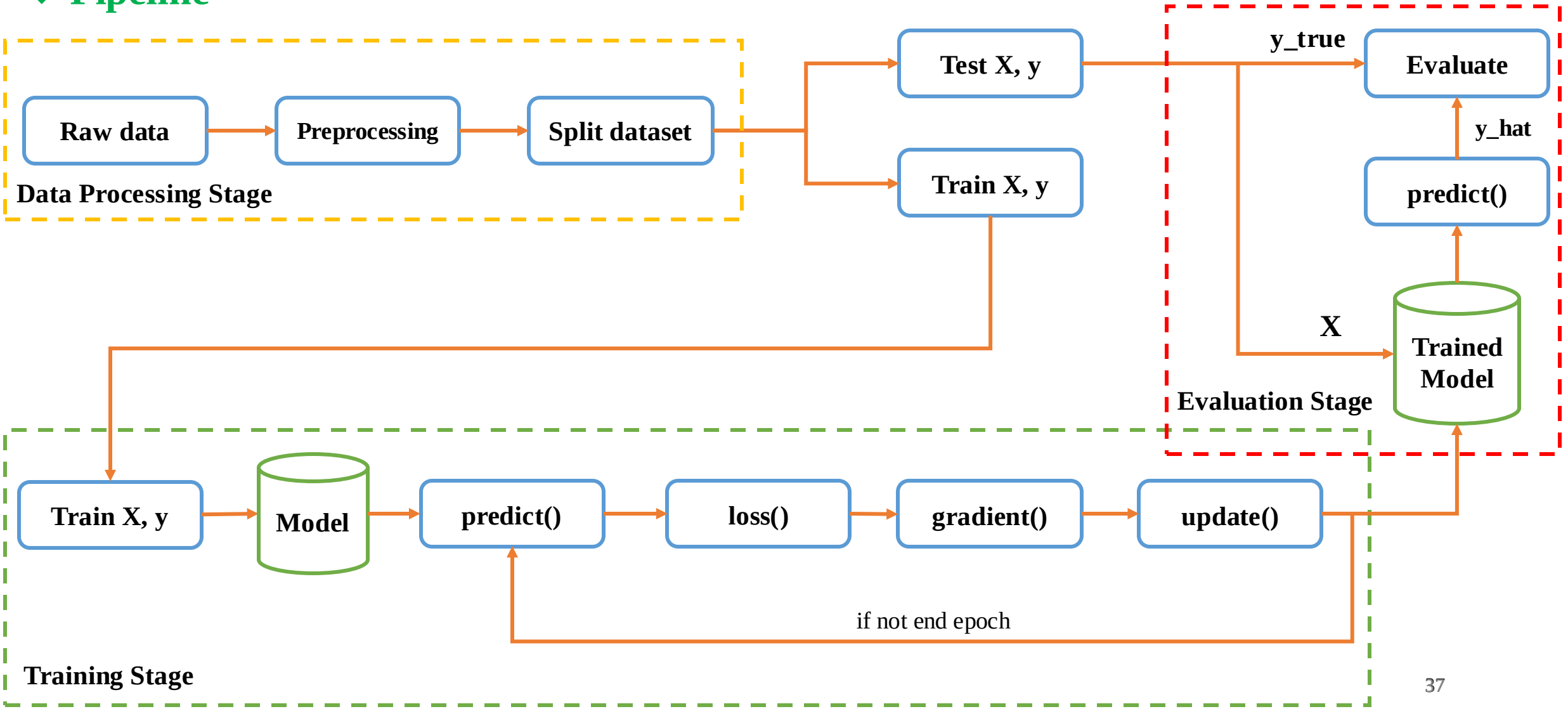
❖ Why we need MLP?





Review

❖ Pipeline



AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Introduction

Description: Given a dataset about car performance, build a Multi-Layer Perceptron (MLP) model capable of accurately predicting the class labels based on the given features.

https://www.goodautos.co.uk/blog/can-a-car-service-improve-performance

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 1: Import libraries and set random seed

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
random_state = 59
np.random.seed(random_state)
torch.manual_seed(random_state)
if torch.cuda.is_available():
    torch.cuda.manual_seed(random_state)
```

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 2: Read dataset

```
1 dataset_path = '/content/Auto_MPG_data.csv'
2 dataset = pd.read_csv(dataset_path)
3 dataset
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Europe	Japan	USA
0	18.0	8	307.0	130.0	3504.0	12.0	70	0	0	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	0	0	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	0	0	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	0	0	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	0	0	1
...	...	...	...	...	...	...	...	...	...	...
387	27.0	4	140.0	86.0	2790.0	15.6	82	0	0	1
388	44.0	4	97.0	52.0	2130.0	24.6	82	1	0	0
389	32.0	4	135.0	84.0	2295.0	11.6	82	0	0	1
390	28.0	4	120.0	79.0	2625.0	18.6	82	0	0	1
391	31.0	4	119.0	82.0	2720.0	19.4	82	0	0	1

392 rows × 10 columns

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 3: Split X, y

Independent Variable (Cause)

Dependent Variable (Result)

Label

Features

Index	Column 1	Column 2	Column 3	…	Column n
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…

In our exericse's dataset:

- ❖ **Independent variables (features):** from column *2* to *n*.
- ❖ **Depedent variables (labels):** first column.

Car Performance Prediction

❖ Step 3: Split X, y

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Europe	Japan	USA
0	18.0	8	307.0	130.0	3504.0	12.0	70	0	0	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	0	0	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	0	0	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	0	0	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	0	0	1
...	...	...	...	...	...	...	...	...	...	...
387	27.0	4	140.0	86.0	2790.0	15.6	82	0	0	1
388	44.0	4	97.0	52.0	2130.0	24.6	82	1	0	0
389	32.0	4	135.0	84.0	2295.0	11.6	82	0	0	1
390	28.0	4	120.0	79.0	2625.0	18.6	82	0	0	1
391	31.0	4	119.0	82.0	2720.0	19.4	82	0	0	1

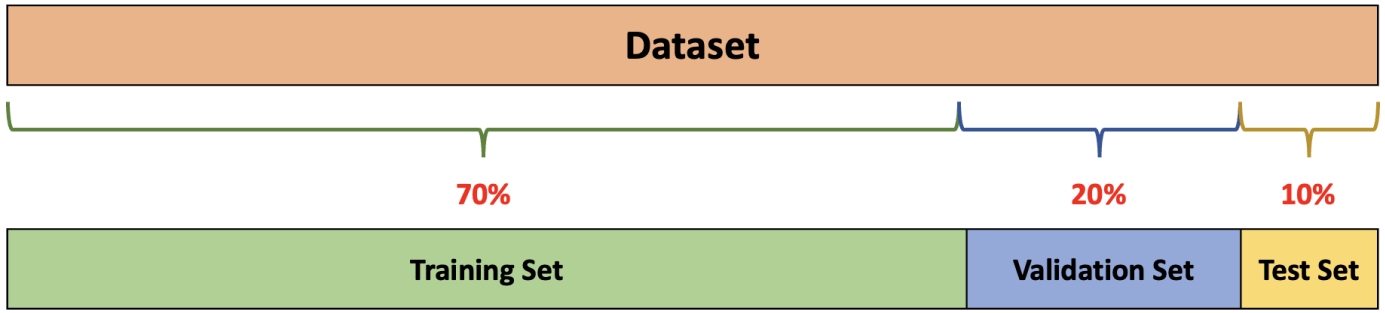
392 rows × 10 columns

```
1 X = dataset.drop(columns='MPG').values
2 y = dataset['MPG'].values
```

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 4: Split train, val, test set




```python
val_size = 0.2
test_size = 0.125
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

```python
print(f'Number of training samples: {X_train.shape[0]}')
print(f'Number of val samples: {X_val.shape[0]}')
print(f'Number of test samples: {X_test.shape[0]}')
```

```
Number of training samples: 273
Number of val samples: 79
Number of test samples: 40
```

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 5: Normalization

Using sklearn.preprocessing.StandardScaler() to scale all values in dataset.

$$z = \frac{x_i - \mu}{\sigma}$$

```
1 normalizer = StandardScaler()
2 X_train = normalizer.fit_transform(X_train)
3 X_val = normalizer.transform(X_val)
4 X_test = normalizer.transform(X_test)
```

```
1 X_train
```

```
array([[-0.84144139, -1.03791712, -1.03812675, ..., -0.44426166,
         1.90449958, -1.27475488],
       [ 0.36757703,  0.36824297,  0.22296166, ..., -0.44426166,
        -0.52507231,  0.78446454],
       [-0.84144139, -0.916522  , -0.89800582, ..., -0.44426166,
        -0.52507231,  0.78446454],
       ...,
       [ 1.57659545,  1.30905511,  1.34392915, ..., -0.44426166,
        -0.52507231,  0.78446454],
       [ 0.36757703,  0.43905678, -0.33752208, ..., -0.44426166,
        -0.52507231,  0.78446454],
       [ 0.36757703,  0.62114946, -0.05728021, ..., -0.44426166,
        -0.52507231,  0.78446454]])
```

Note: We only use the train set to fit the scaler.

Car Performance Prediction

❖ Step 5: Normalization

```
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)

print("First 5 rows of X:")
print(X[:5])
print("First 5 labels:")
print(y[:5])

```

```
Shape of X: (300, 2)
Shape of y: (300,)
First 5 rows of X:
[[0.         0.        ]
 [0.00096008 0.01005528]
 [0.01045864 0.01728405]
 [0.00087922 0.03029027]
 [0.00991727 0.03916803]]
First 5 labels:
[0 0 0 0 0]
```

```
normalizer = StandardScaler()
X_train = normalizer.fit_transform(X_train)
X_val = normalizer.transform(X_val)
X_test = normalizer.transform(X_test)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_val = torch.tensor(y_val, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

Since the original has Numpy arrays form, we need to convert the datasets (features) into tensors of float32 data type as PyTorch libraries require the input data to be tensors.

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 5: Normalization

```
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)

print("First 5 rows of X:")
print(X[:5])
print("First 5 labels:")
print(y[:5])

```

```
Shape of X: (300, 2)
Shape of y: (300,)
First 5 rows of X:
[[0.         0.        ]
 [0.00096008 0.01005528]
 [0.01045864 0.01728405]
 [0.00087922 0.03029027]
 [0.00991727 0.03916803]]
First 5 labels:
[0 0 0 0 0]
```

```
normalizer = StandardScaler()
X_train = normalizer.fit_transform(X_train)
X_val = normalizer.transform(X_val)
X_test = normalizer.transform(X_test)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_val = torch.tensor(y_val, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

- Then convert the label datasets into tensors of long type, commonly used for classification tasks because labels are typically integers (e.g., 0, 1, 2).
- As PyTorch also requires labels in classification tasks to be tensors of type long to be compatible with loss functions (CrossEntropyLoss)

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 6: Build Dataloader

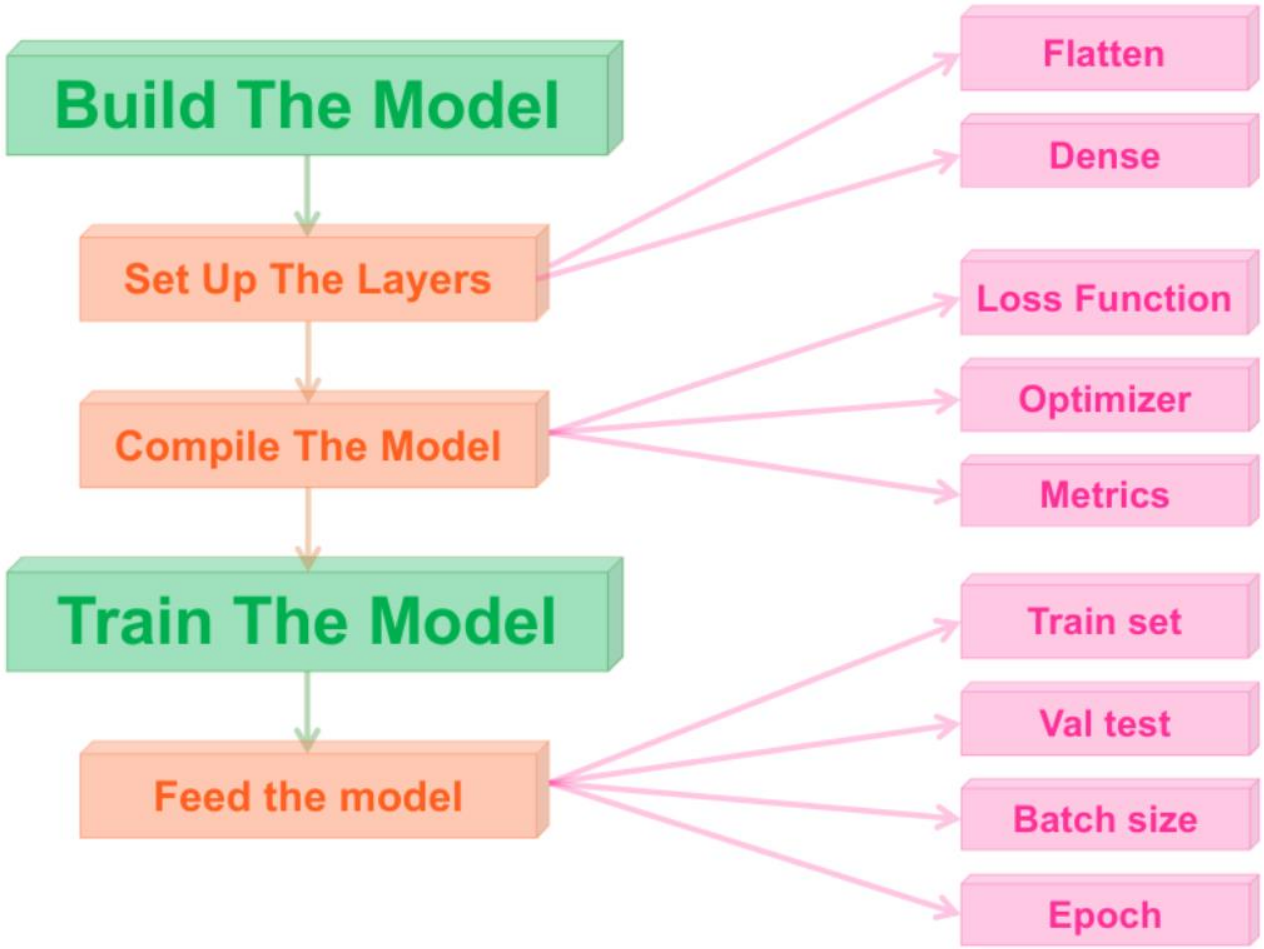
```python
class CustomDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

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

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

```python
batch_size = 32
train_dataset = CustomDataset(X_train, y_train)
val_dataset = CustomDataset(X_val, y_val)
train_loader = DataLoader(train_dataset,
                          batch_size=batch_size,
                          shuffle=True)
val_loader = DataLoader(val_dataset,
                        batch_size=batch_size,
                        shuffle=False)
```

To efficiently handle data for training and evaluation in PyTorch, we use **DataLoader**, which requires a custom **Dataset** to manage inputs and labels.

Car Performance Prediction

❖ Step 7: Build MLP model

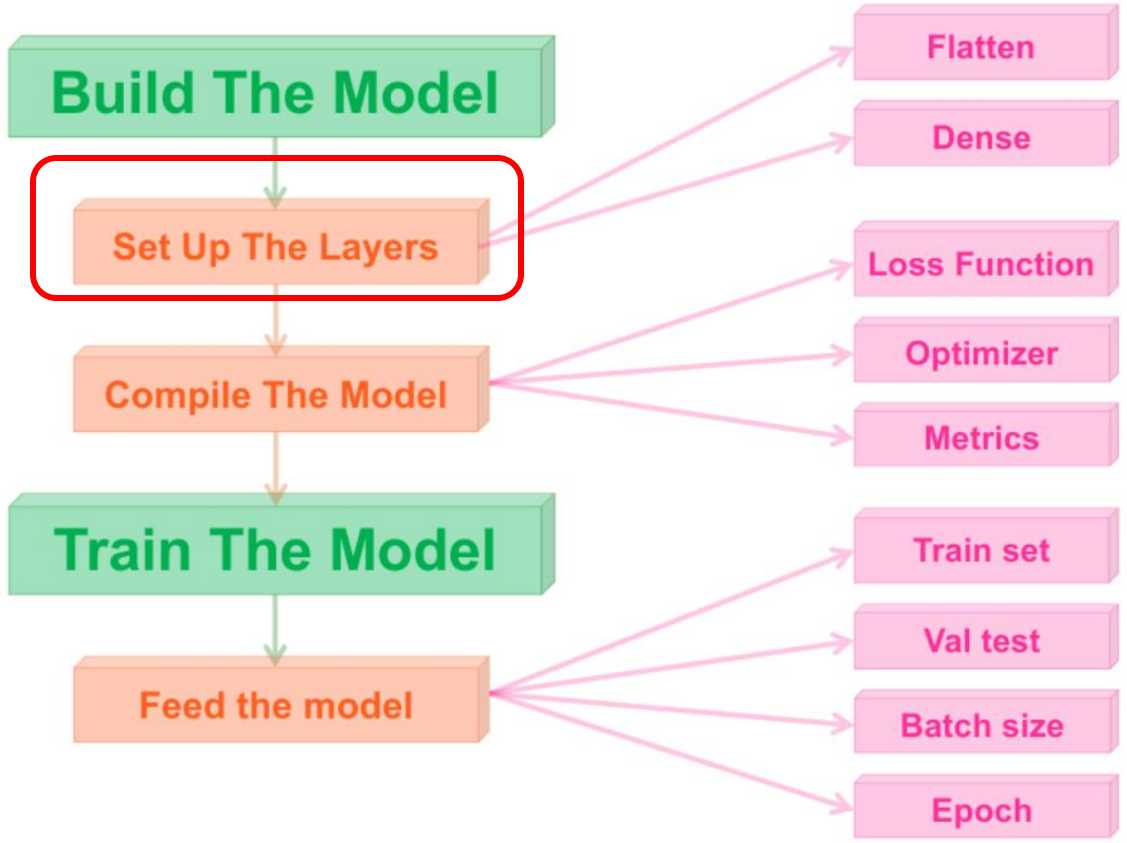


Pipeline to build and train a model in PyTorch.

Car Performance Prediction

❖ Step 7: Build MLP model

```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims)
        self.linear2 = nn.Linear(hidden_dims, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```

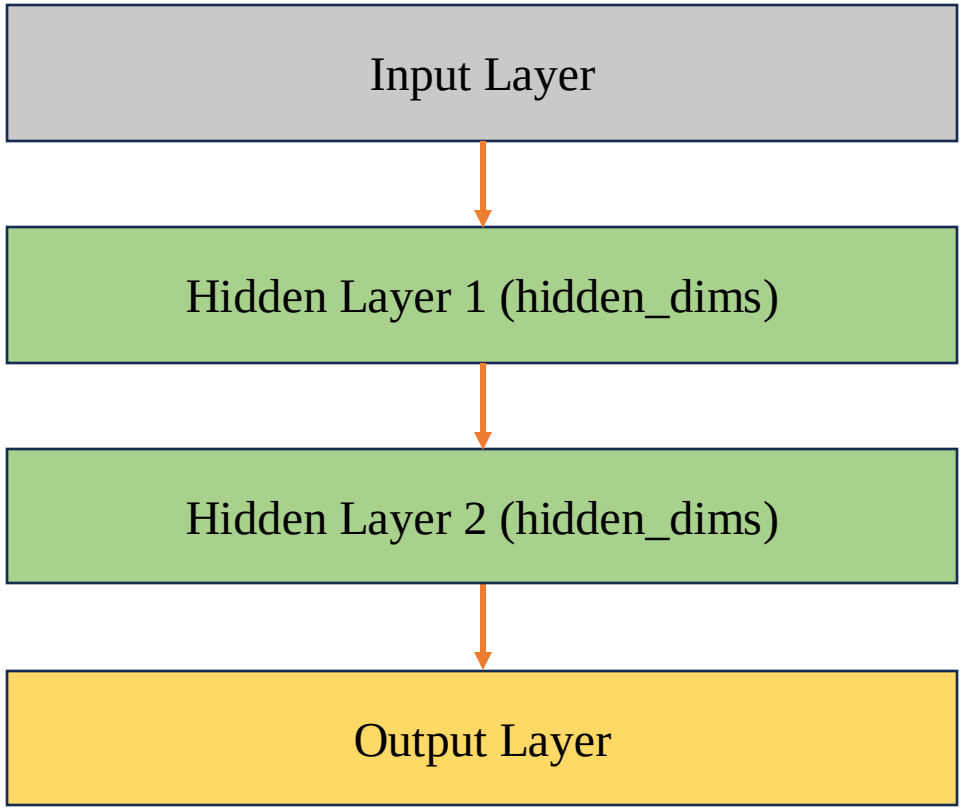


AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 7: Build MLP model

```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims)
        self.linear2 = nn.Linear(hidden_dims, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```



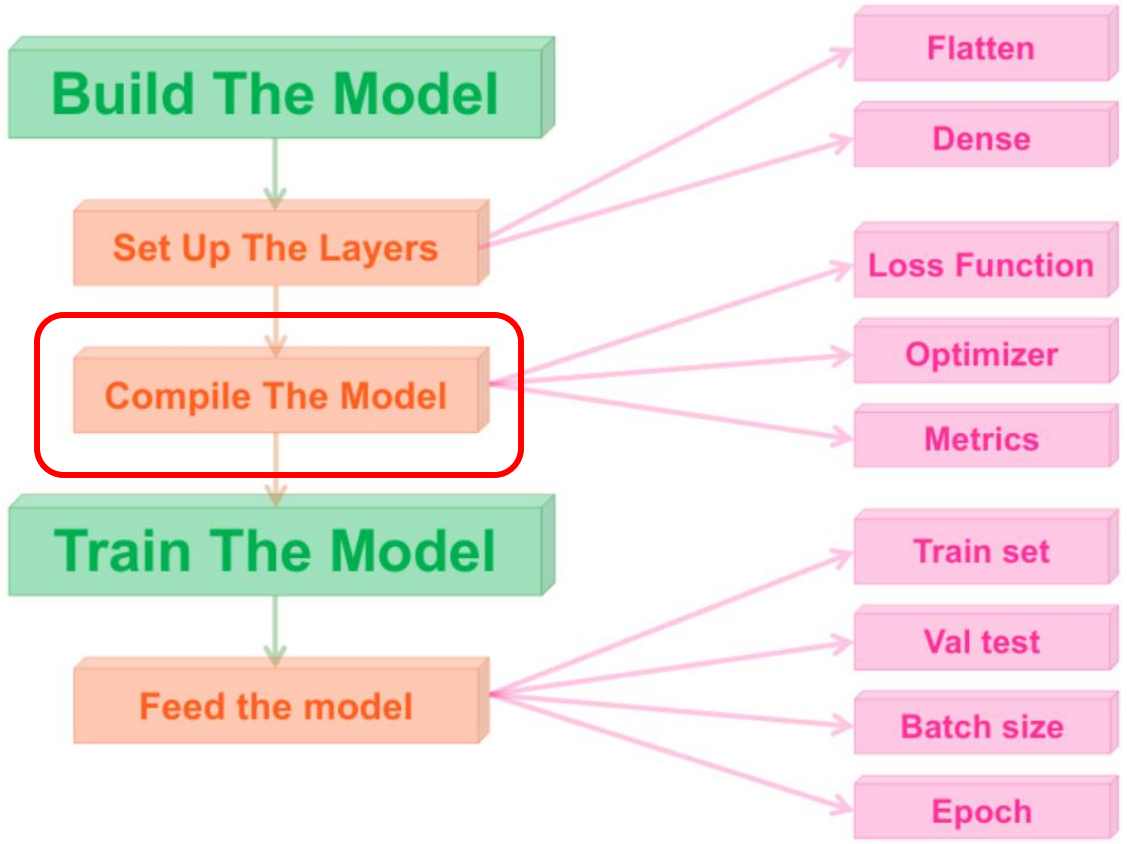
AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 7: Declare model, optimizer and criterion

```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims)
        self.linear2 = nn.Linear(hidden_dims, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```

```python
input_dims = X_train.shape[1]
output_dims = 1
hidden_dims = 64
lr = 1e-2

model = MLP(input_dims=input_dims,
            hidden_dims=hidden_dims,
            output_dims=output_dims).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

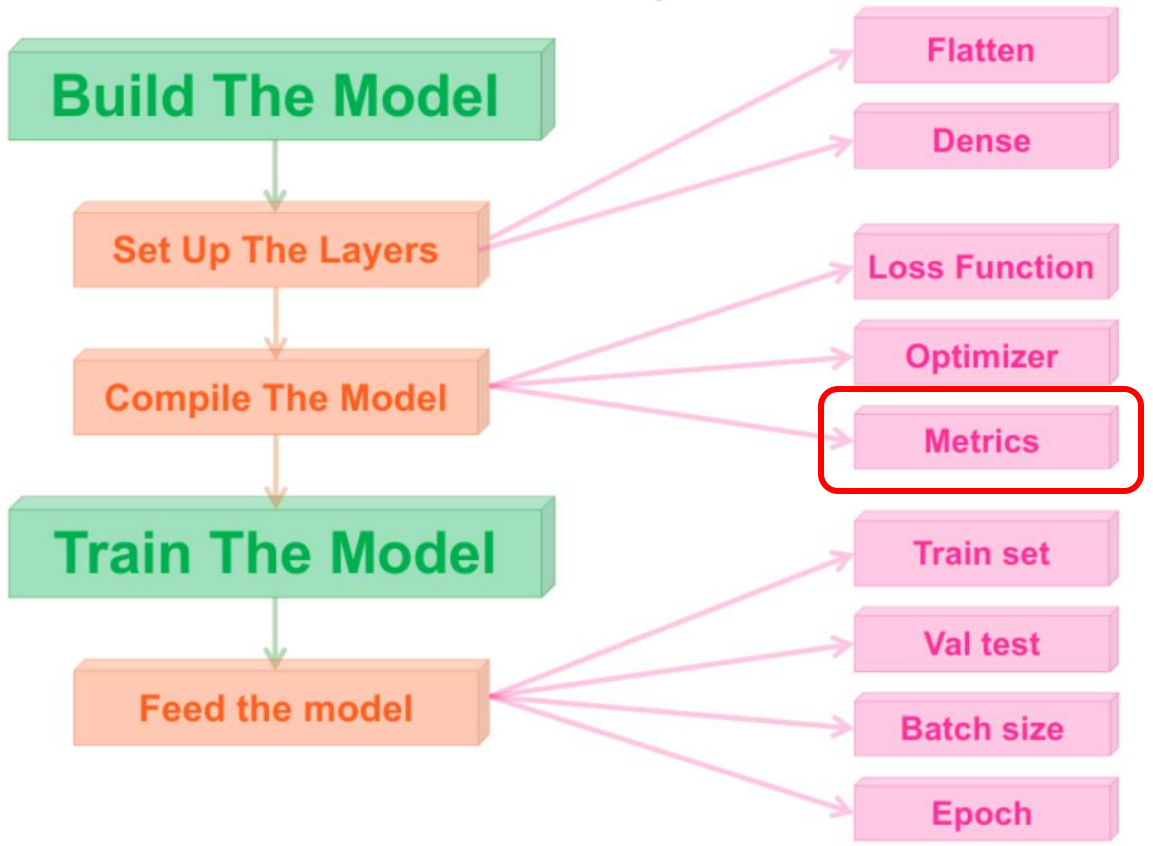


Car Performance Prediction

❖ Step 8: Create R2 Score function

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)}$$

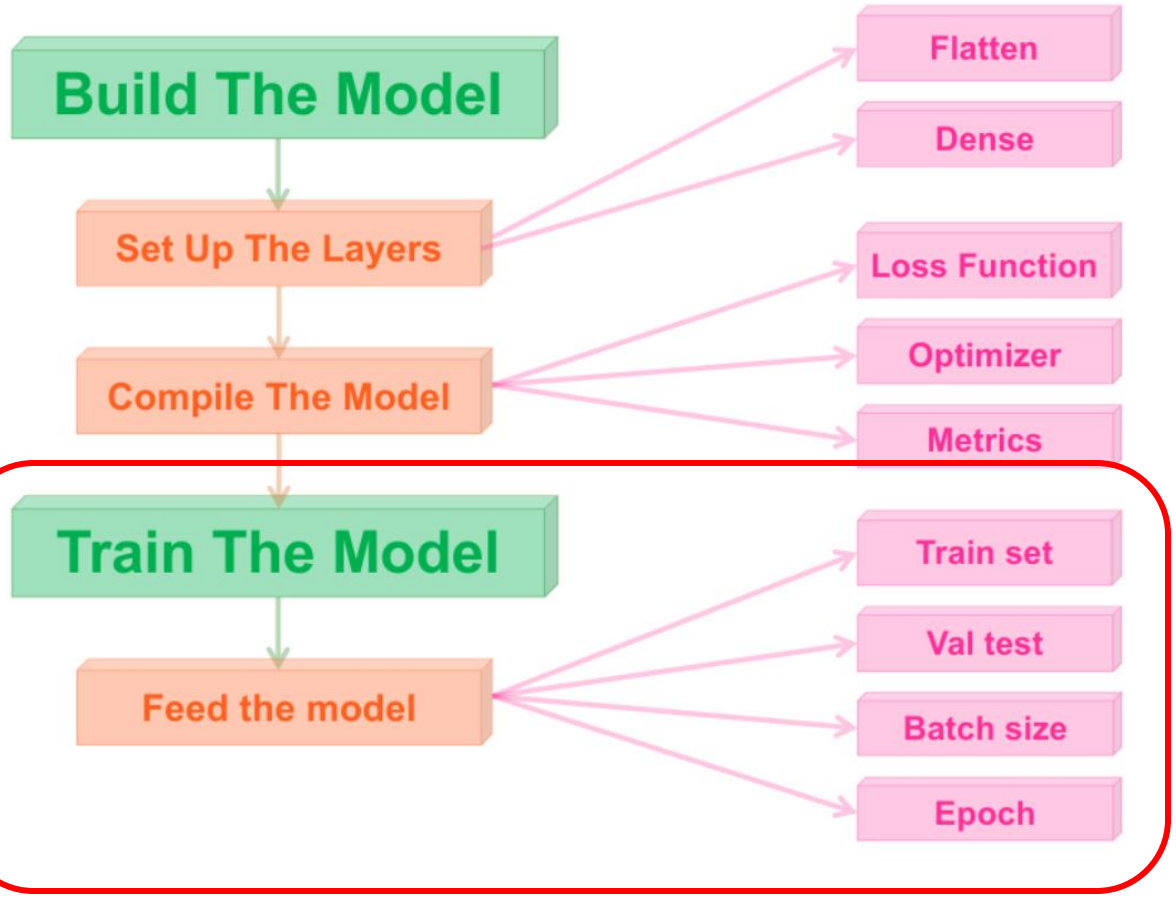
```python
def r_squared(y_true, y_pred):
    y_true = torch.Tensor(y_true).to(device)
    y_pred = torch.Tensor(y_pred).to(device)
    mean_true = torch.mean(y_true)
    ss_tot = torch.sum((y_true - mean_true) ** 2)
    ss_res = torch.sum((y_true - y_pred) ** 2)
    r2 = 1 - (ss_res / ss_tot)
    return r2
```



AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

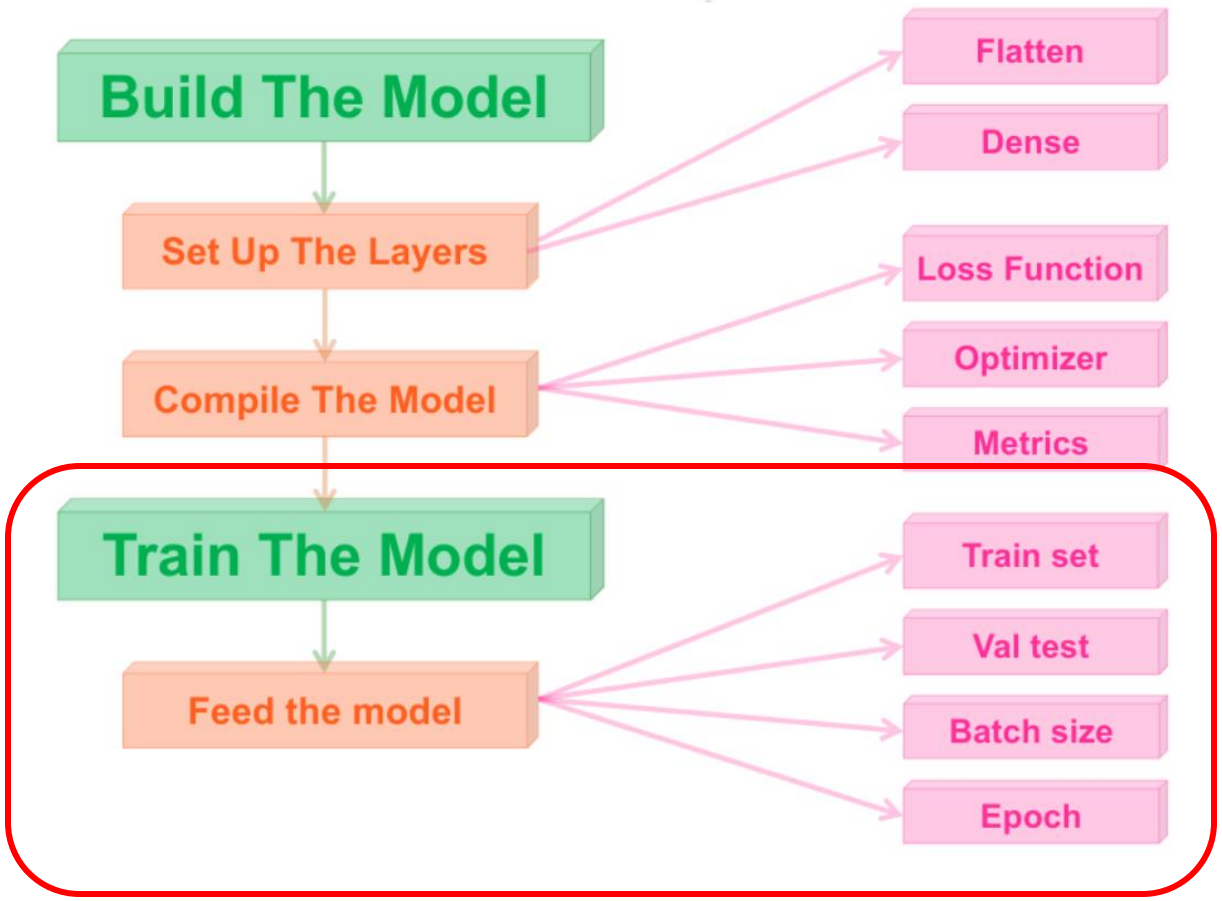
❖ Step 9: Training




```python
epochs = 100
train_losses = []
val_losses = []
train_r2 = []
val_r2 = []

for epoch in range(epochs):
    train_loss = 0.0
    train_target = []
    val_target = []
    train_predict = []
    val_predict = []
    model.train()
    for X_samples, y_samples in train_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        optimizer.zero_grad()
        outputs = model(X_samples)
        train_predict += outputs.tolist()
        train_target += y_samples.tolist()
        loss = criterion(outputs, y_samples)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    train_loss /= len(train_loader)
    train_losses.append(train_loss)
    train_r2.append(r_squared(train_target, train_predict))
```

Car Performance Prediction

❖ Step 9: Training (Evaluation)



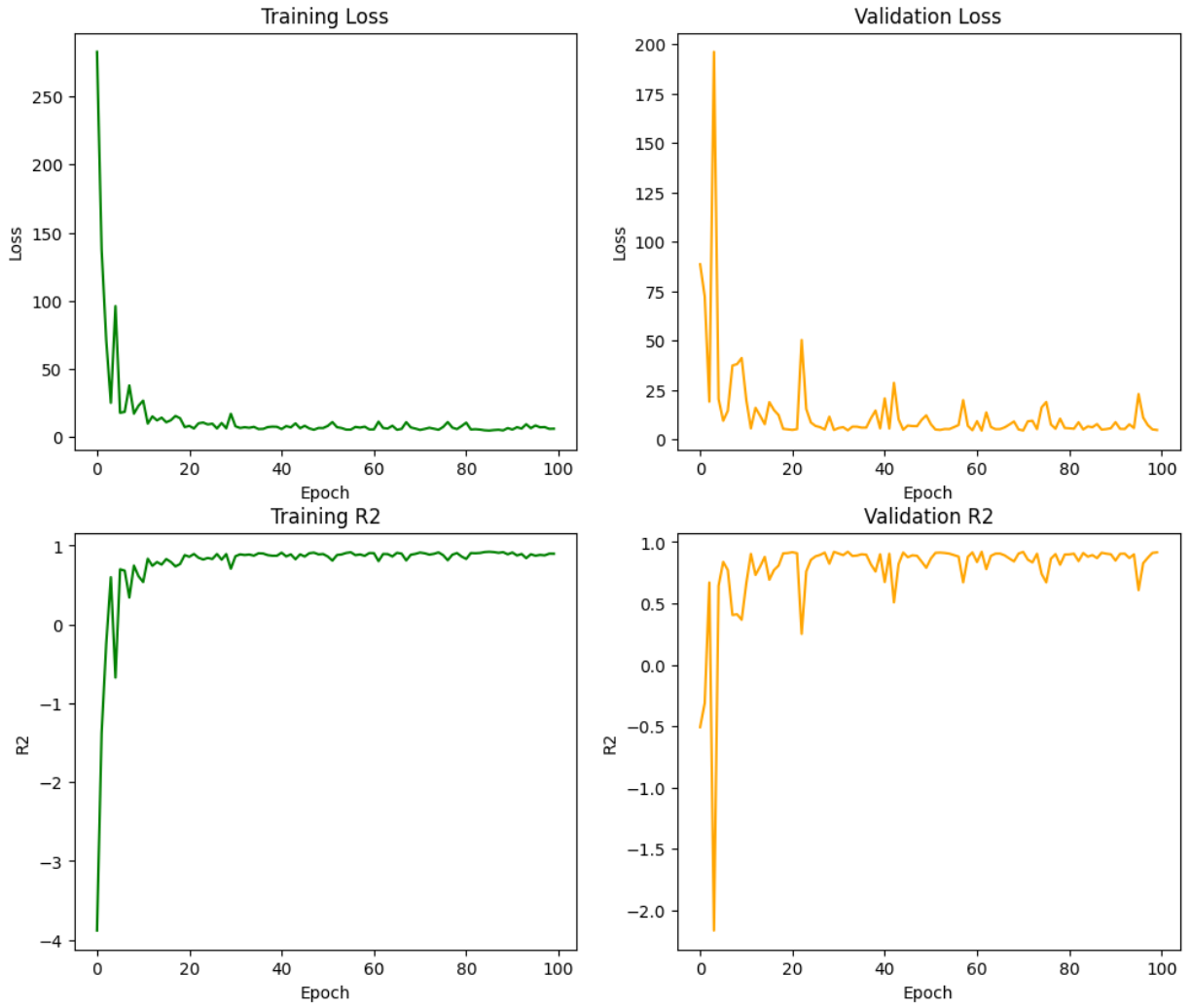

```
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for X_samples, y_samples in val_loader:
            X_samples = X_samples.to(device)
            y_samples = y_samples.to(device)
            outputs = model(X_samples)
            val_predict += outputs.tolist()
            val_target += y_samples.tolist()
            loss = criterion(outputs, y_samples)
            val_loss += loss.item()
    val_loss /= len(val_loader)
    val_losses.append(val_loss)
    val_r2.append(r_squared(val_target, val_predict))
```

AI VIET NAM
@aivietnam.edu.vn

Car Performance Prediction

❖ Step 10: Evaluation

```
fig, ax = plt.subplots(2, 2, figsize=(12, 10))
ax[0, 0].plot(train_losses, color='green')
ax[0, 0].set(xlabel='Epoch', ylabel='Loss')
ax[0, 0].set_title('Training Loss')

ax[0, 1].plot(val_losses, color='orange')
ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
ax[0, 1].set_title('Validation Loss')

ax[1, 0].plot(train_r2, color='green')
ax[1, 0].set(xlabel='Epoch', ylabel='R2')
ax[1, 0].set_title('Training R2')

ax[1, 1].plot(val_r2, color='orange')
ax[1, 1].set(xlabel='Epoch', ylabel='R2')
ax[1, 1].set_title('Validation R2')

plt.show()
```



Car Performance Prediction

❖ Step 10: Evaluation

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)}$$

```python
def r_squared(y_true, y_pred):
    y_true = torch.Tensor(y_true).to(device)
    y_pred = torch.Tensor(y_pred).to(device)
    mean_true = torch.mean(y_true)
    ss_tot = torch.sum((y_true - mean_true) ** 2)
    ss_res = torch.sum((y_true - y_pred) ** 2)
    r2 = 1 - (ss_res / ss_tot)
    return r2
```

```python
model.eval()
with torch.no_grad():
    y_hat = model(X_val)
    val_set_r2 = r_squared(y_hat, y_val)
    print('Evaluation on validation set:')
    print(f'R2: {val_set_r2}')
```

```
Evaluation on validation set:
R2: 0.893627941608429
```

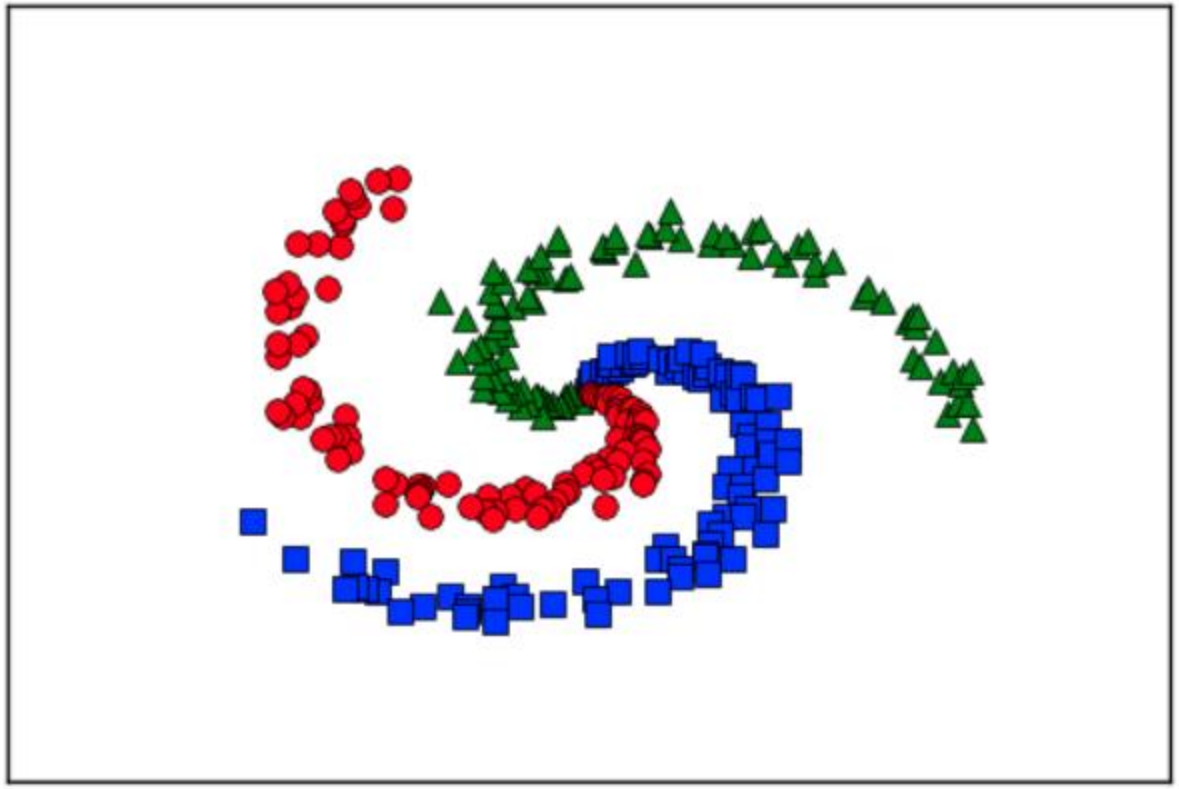
AI VIET NAM
@aivietnam.edu.vn

Non-linear Data Classification

Non-linear Data Classification

❖ Introduction

Description: Classify non-linear data using the dataset NonLinear_data.npy, build a Multi-Layer Perceptron (MLP) model capable of accurately predicting the class labels based on the given features.



Non-linear Data Classification

❖ Step 1: Import libraries and set random seed

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
random_state = 59
np.random.seed(random_state)
torch.manual_seed(random_state)
if torch.cuda.is_available():
    torch.cuda.manual_seed(random_state)
```

Non-linear Data Classification

❖ Step 2: Read Dataset and Visualize

```
data_path = '/content/NonLinear_data.npy'
data = np.load(data_path, allow_pickle=True).item()
X, y = data['X'], data['labels']
```

numpy.load

numpy.load(*file*, *mmap_mode=None*, *allow_pickle=False*, *fix_imports=True*, *encoding='ASCII'*, ***, *max_header_size=10000*) [source]

Load arrays or pickled objects from `.npy`, `.npz` or pickled files.

> **Warning**
>
> Loading files that contain object arrays uses the `pickle` module, which is not secure against erroneous or maliciously constructed data. Consider passing `allow_pickle=False` to load data that is known not to contain object arrays for the safer handling of untrusted sources.

```
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)

print("First 5 rows of X:")
print(X[:5])
print("First 5 labels:")
print(y[:5])
```

```
Shape of X: (300, 2)
Shape of y: (300,)
First 5 rows of X:
[[0.         0.        ]
 [0.00096008 0.01005528]
 [0.01045864 0.01728405]
 [0.00087922 0.03029027]
 [0.00991727 0.03916803]]
First 5 labels:
[0 0 0 0 0]
```

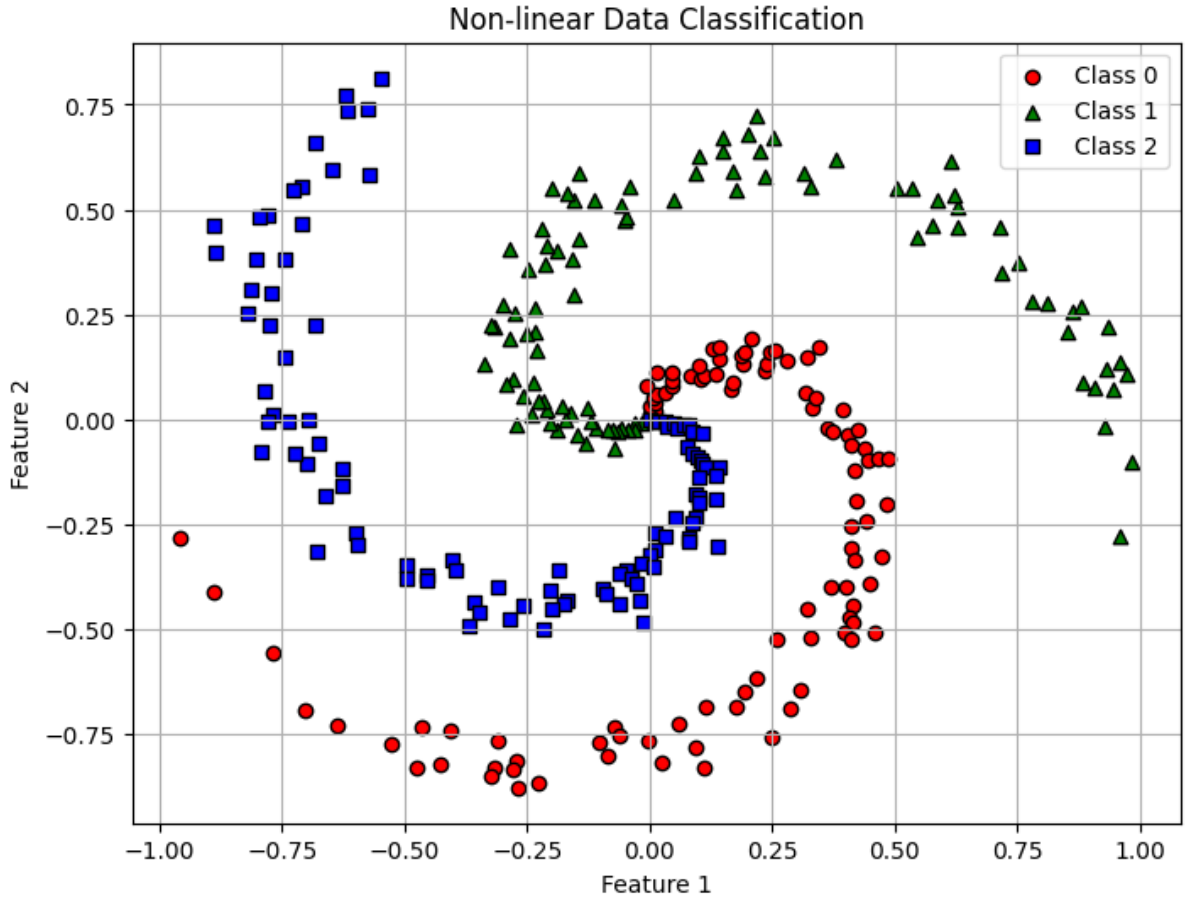

https://numpy.org/doc/stable/reference/generated/numpy.load.html

Non-linear Data Classification

❖ Step 2: Read Dataset and Visualize

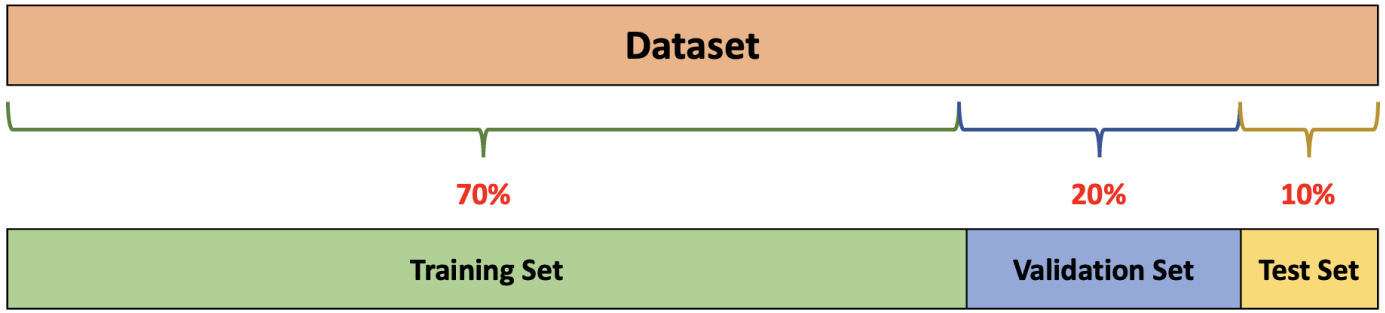
```
colors = ['red', 'green', 'blue']
markers = ['o', '^', 's']

plt.figure(figsize=(8, 6))
for i, class_label in enumerate(np.unique(y)):
    plt.scatter(
        X[y == class_label, 0],
        X[y == class_label, 1],
        c=colors[i],
        marker=markers[i],
        label=f'Class {class_label}',
        edgecolor='k'
    )
```

```
plt.title("Non-linear Data Classification")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```



Non-linear Data Classification

❖ Step 3: Split train, val, test set




```python
val_size = 0.2
test_size = 0.125
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

```python
print(f'Number of training samples: {X_train.shape[0]}')
print(f'Number of val samples: {X_val.shape[0]}')
print(f'Number of test samples: {X_test.shape[0]}')
```

```
Number of training samples: 210
Number of val samples: 60
Number of test samples: 30
```

Non-linear Data Classification

❖ Step 4: Normalization

Using sklearn.preprocessing.StandardScaler() to scale all values in dataset. Only use the train set to fit the scaler.

$$z = \frac{x_i - \mu}{\sigma}$$

```
normalizer = StandardScaler()
X_train = normalizer.fit_transform(X_train)
X_val = normalizer.transform(X_val)
X_test = normalizer.transform(X_test)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_val = torch.tensor(y_val, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

AI VIET NAM
@aivietnam.edu.vn

Non-linear Data Classification

❖ Step 5: Build Dataloader

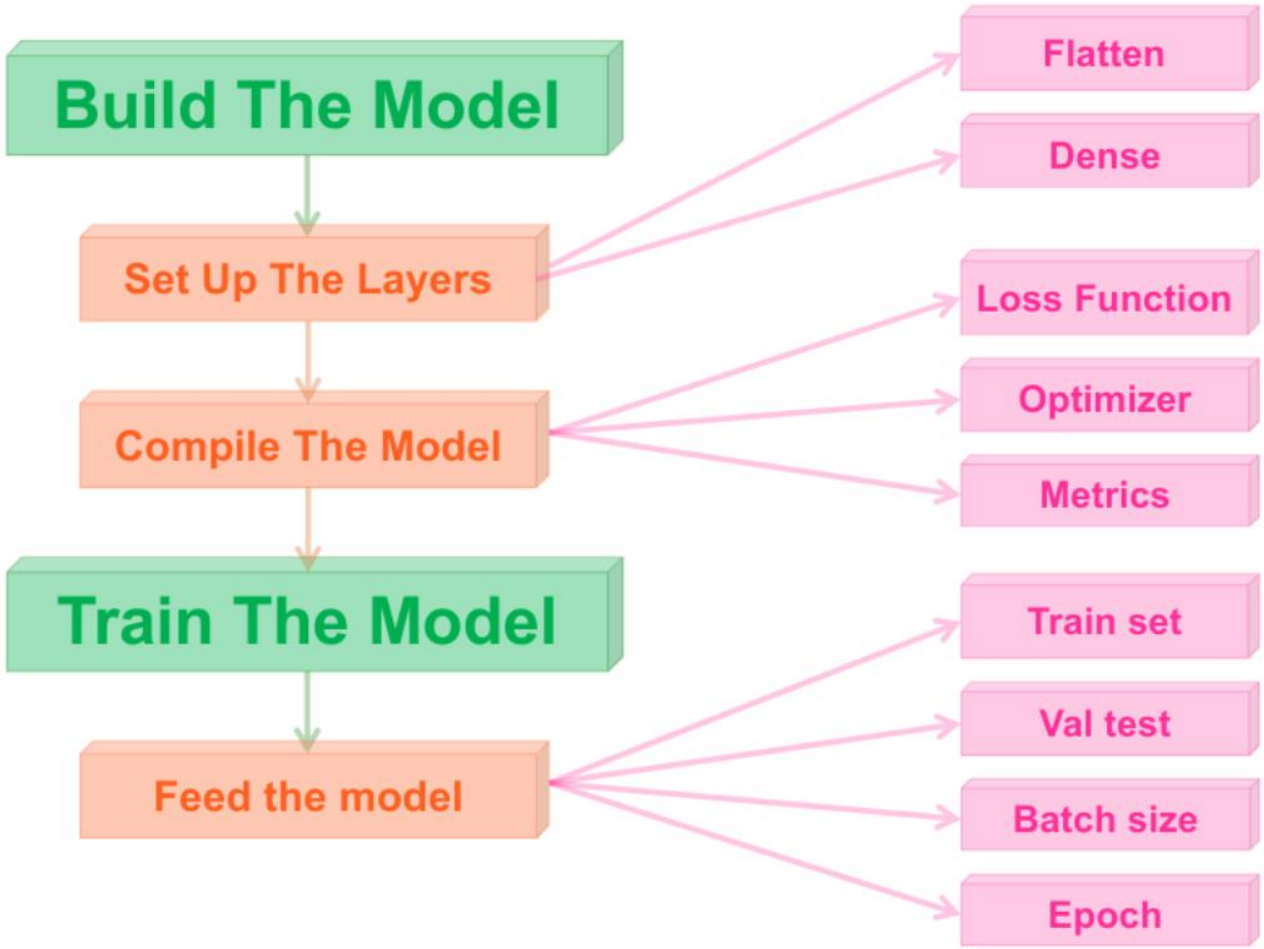
```python
class CustomDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

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

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

```python
batch_size = 32
train_dataset = CustomDataset(X_train, y_train)
val_dataset = CustomDataset(X_val, y_val)
test_dataset = CustomDataset(X_test, y_test)
train_loader = DataLoader(train_dataset,
                          batch_size=batch_size,
                          shuffle=True)
val_loader = DataLoader(val_dataset,
                        batch_size=batch_size,
                        shuffle=False)
test_loader = DataLoader(test_dataset,
                         batch_size=batch_size,
                         shuffle=False)
```

To efficiently handle data for training and evaluation in PyTorch, we use **DataLoader**, which requires a custom **Dataset** to manage inputs and labels.

Non-linear Data Classification

❖ Step 6: Build MLP model

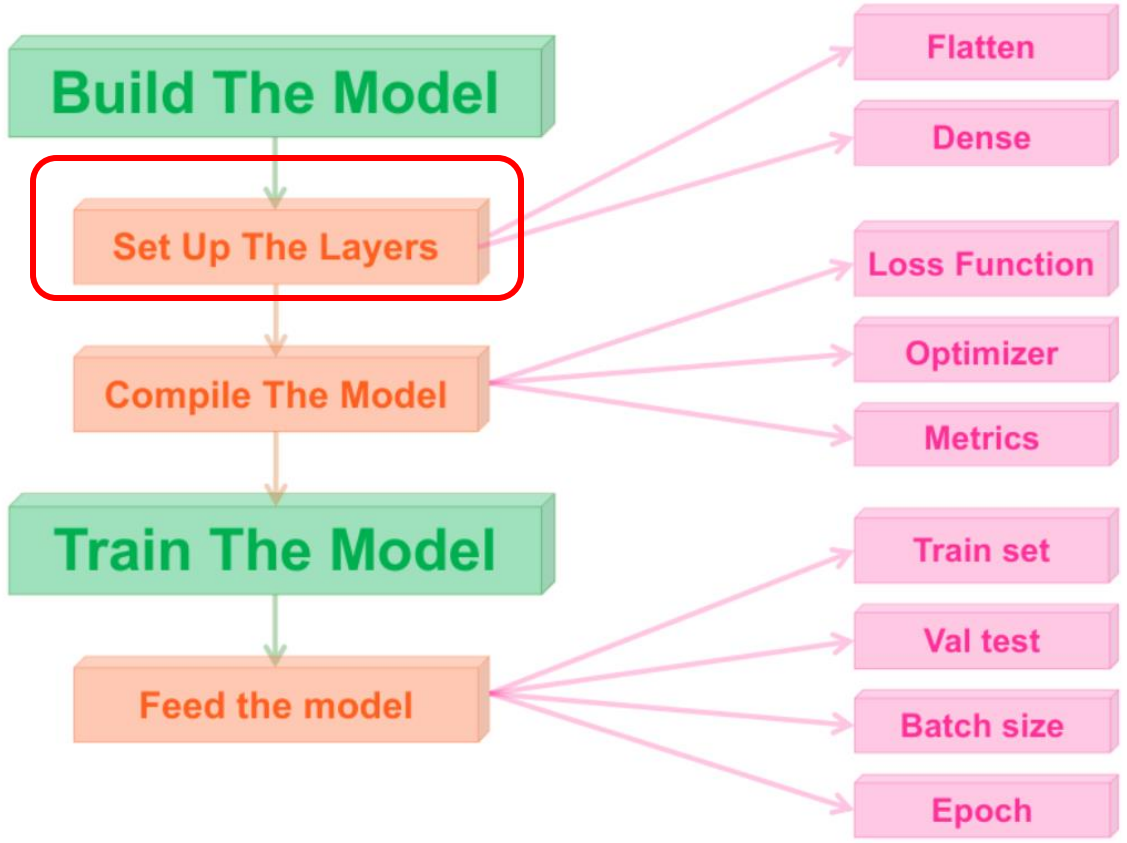


Pipeline to build and train a model in PyTorch.

Non-linear Data Classification

❖ Step 6: Build MLP model

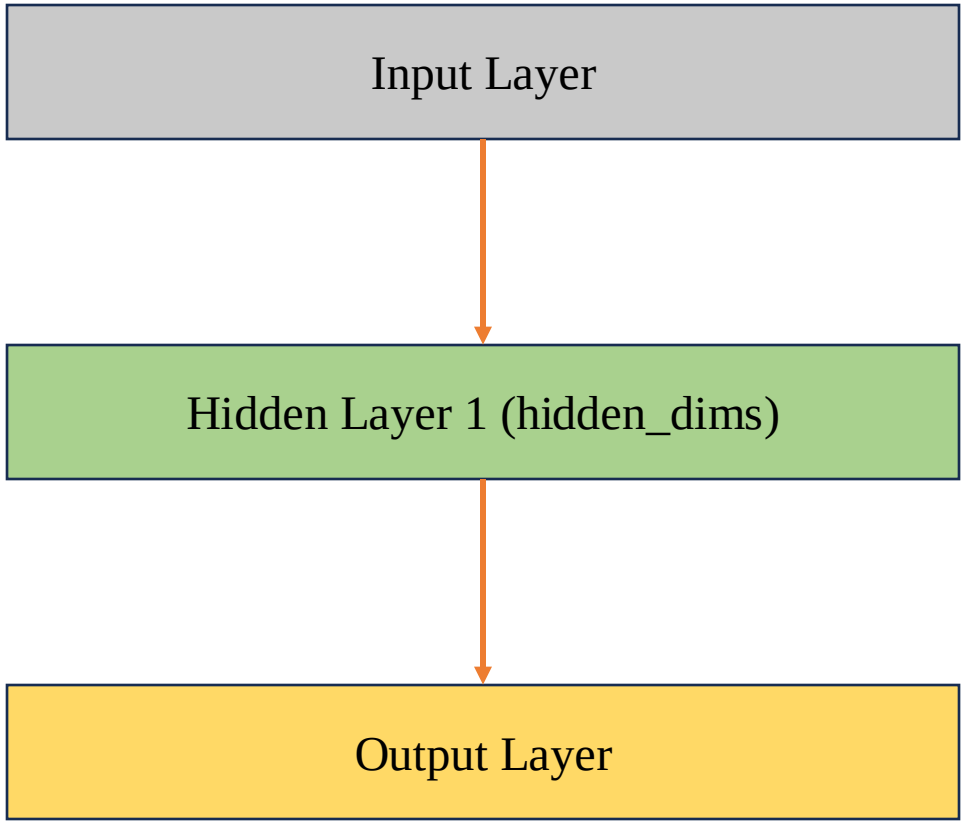
```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.linear1(x)
        x = self.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```



Non-linear Data Classification

❖ Step 6: Build MLP model

```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.linear1(x)
        x = self.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```

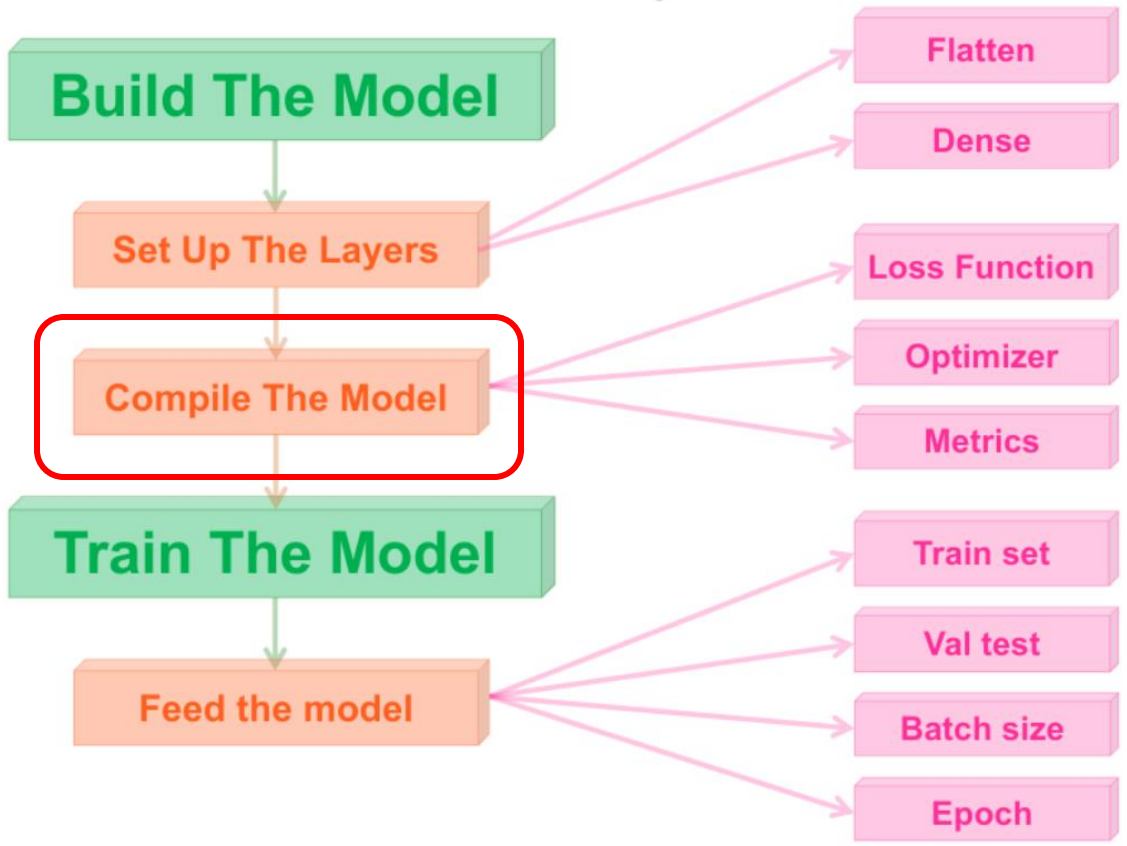


Non-linear Data Classification

❖ Step 6: Declare model, optimizer and criterion

```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = self.linear1(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```

```python
input_dims = X_train.shape[1]
output_dims = torch.unique(y_train).shape[0]
hidden_dims = 128
lr = 1e-1

model = MLP(input_dims=input_dims,
            hidden_dims=hidden_dims,
            output_dims=output_dims).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

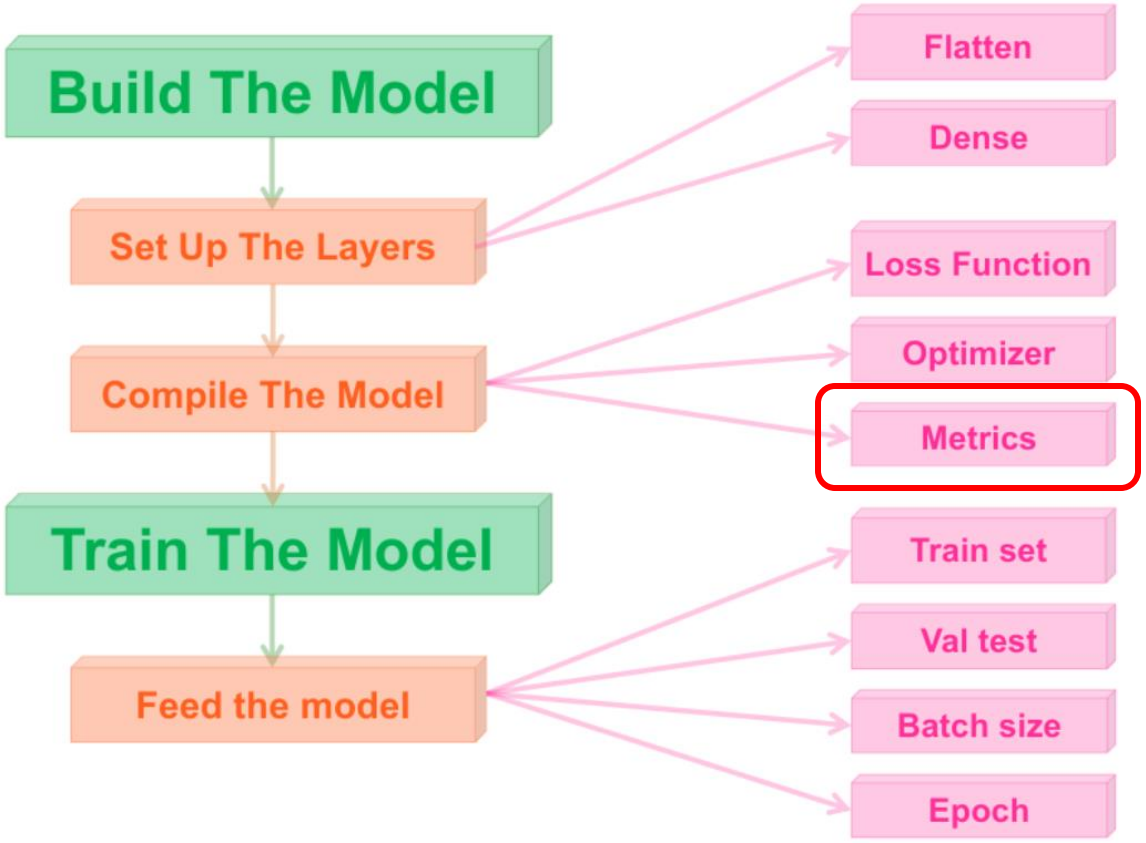


Non-linear Data Classification

❖ Step 7: Create accuracy function

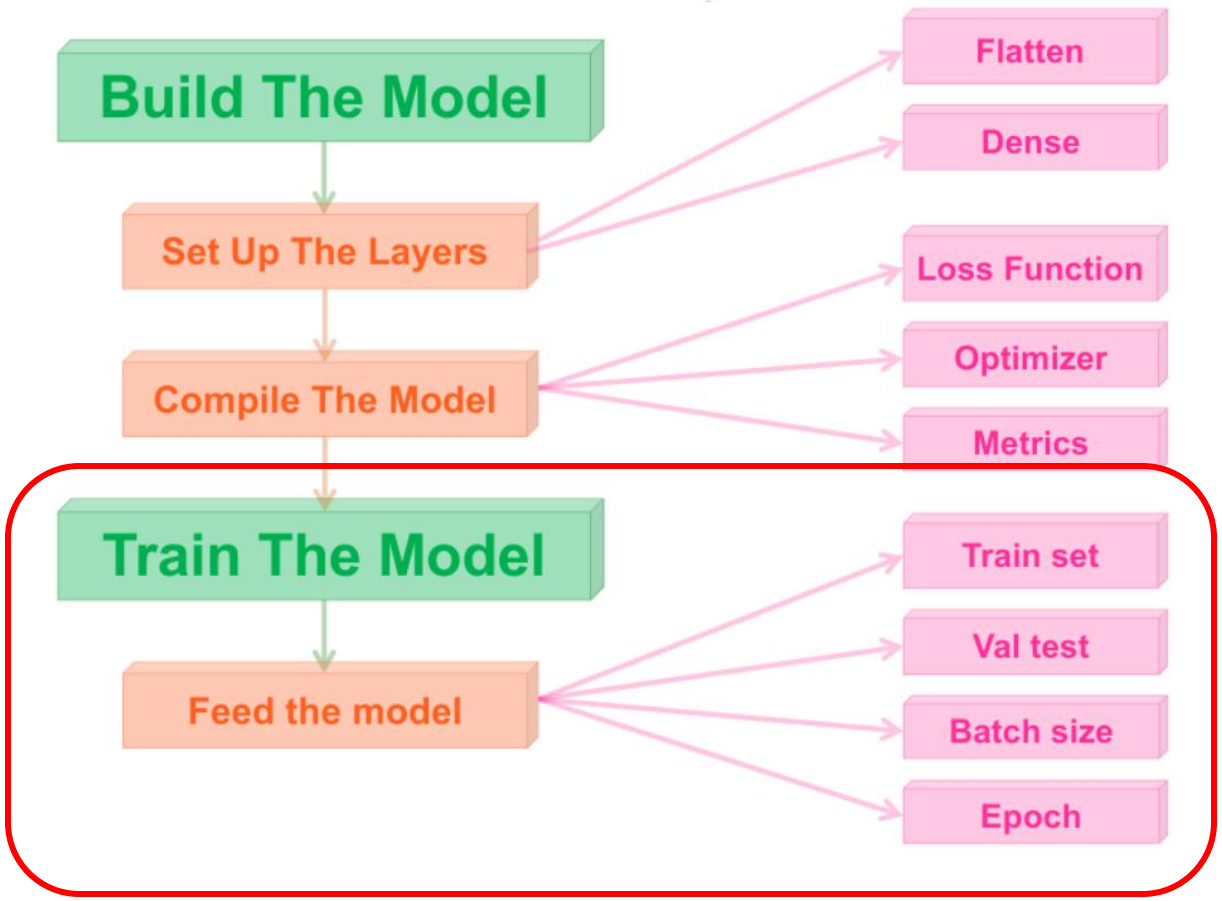
$$accuracy = \frac{true\_predictions}{n\_samples}$$

```
def compute_accuracy(y_hat, y_true):
    _, y_hat = torch.max(y_hat, dim=1)
    correct = (y_hat == y_true).sum().item()
    accuracy = correct / len(y_true)
    return accuracy
```



Non-linear Data Classification

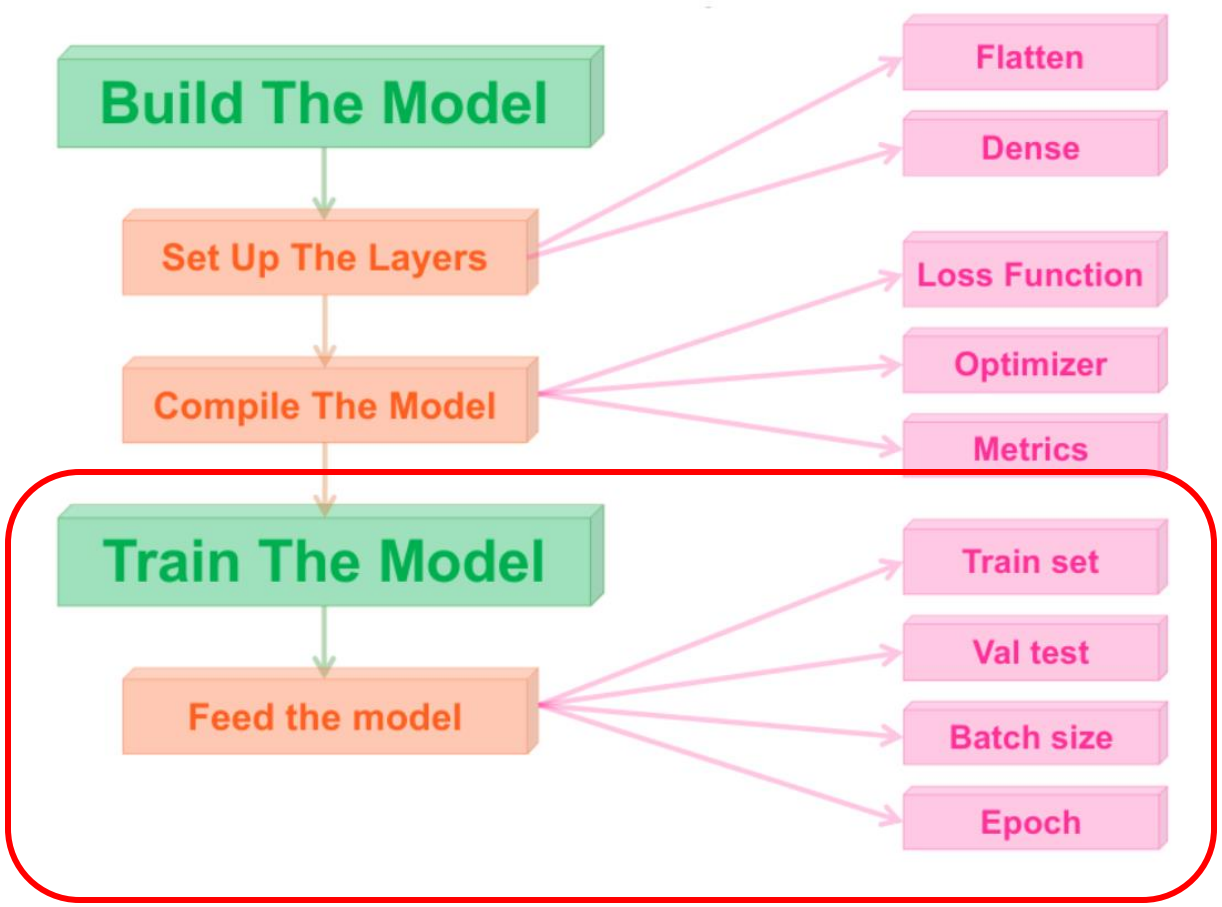
❖ Step 8: Training




```python
epochs = 100
train_losses = []
val_losses = []
train_accs = []
val_accs = []

for epoch in range(epochs):
    train_loss = 0.0
    train_target = []
    train_predict = []
    model.train()
    for X_samples, y_samples in train_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        optimizer.zero_grad()
        outputs = model(X_samples)
        loss = criterion(outputs, y_samples)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        train_predict.append(outputs.detach().cpu())
        train_target.append(y_samples.cpu())

    train_loss /= len(train_loader)
    train_losses.append(train_loss)

    train_predict = torch.cat(train_predict)
    train_target = torch.cat(train_target)
    train_acc = compute_accuracy(train_predict, train_target)
    train_accs.append(train_acc)
```

Non-linear Data Classification

❖ Step 8: Training (Evaluation)

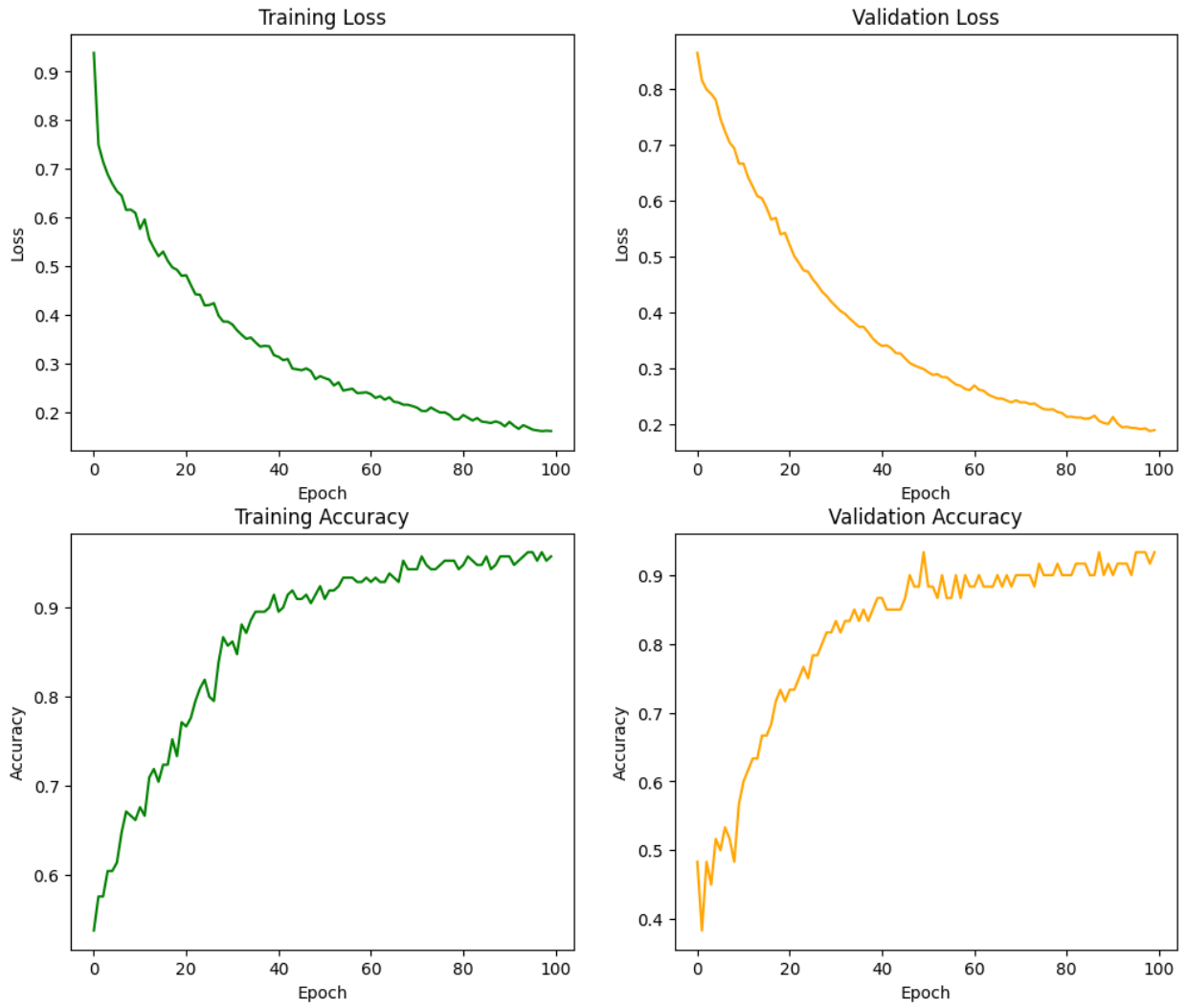



```
33      val_loss = 0.0
34      val_target = []
35      val_predict = []
36      model.eval()
37      with torch.no_grad():
38          for X_samples, y_samples in val_loader:
39              X_samples = X_samples.to(device)
40              y_samples = y_samples.to(device)
41              outputs = model(X_samples)
42              val_loss += criterion(outputs, y_samples).item()
43
44              val_predict.append(outputs.cpu())
45              val_target.append(y_samples.cpu())
46
47      val_loss /= len(val_loader)
48      val_losses.append(val_loss)
49
50      val_predict = torch.cat(val_predict)
51      val_target = torch.cat(val_target)
52      val_acc = compute_accuracy(val_predict, val_target)
53      val_accs.append(val_acc)
```

Non-linear Data Classification

❖ Step 9: Evaluation

```
fig, ax = plt.subplots(2, 2, figsize=(12, 10))
ax[0, 0].plot(train_losses, color='green')
ax[0, 0].set(xlabel='Epoch', ylabel='Loss')
ax[0, 0].set_title('Training Loss')

ax[0, 1].plot(val_losses, color='orange')
ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
ax[0, 1].set_title('Validation Loss')

ax[1, 0].plot(train_accs, color='green')
ax[1, 0].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 0].set_title('Training Accuracy')

ax[1, 1].plot(val_accs, color='orange')
ax[1, 1].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 1].set_title('Validation Accuracy')

plt.show()
```



Non-linear Data Classification

❖ Step 9: Evaluation

$$accuracy = \frac{true\_predictions}{n\_samples}$$

```python
def compute_accuracy(y_hat, y_true):
    _, y_hat = torch.max(y_hat, dim=1)
    correct = (y_hat == y_true).sum().item()
    accuracy = correct / len(y_true)
    return accuracy
```

```python
val_target = []
val_predict = []
model.eval()
with torch.no_grad():
    for X_samples, y_samples in val_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        outputs = model(X_samples)

        val_predict.append(outputs.cpu())
        val_target.append(y_samples.cpu())

    val_predict = torch.cat(val_predict)
    val_target = torch.cat(val_target)
    val_acc = compute_accuracy(val_predict, val_target)

    print('Evaluation on val set:')
    print(f'Accuracy: {val_acc}')
```

```
Evaluation on val set:
Accuracy: 0.9333333333333333
```

AI VIET NAM
@aivietnam.edu.vn

QUIZ

AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Introduction

Description: Given an image dataset about face emotion, build a Multi-Layer Perceptron (MLP) model capable of accurately predicting the class labels based on the given image.

Sentiment Image Analysis

❖ Step 1: Import libraries and set random seed

```python
import cv2
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import Resize
from torchvision.io import read_image

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
random_state = 59
np.random.seed(random_state)
torch.manual_seed(random_state)
if torch.cuda.is_available():
    torch.cuda.manual_seed(random_state)
```

AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Step 2: Get number of classes in dataset

Because of the dataset structure, we can get all classnames by reading the name of each subfolder in a set.

```
1 train_dir = '/content/train'
2 test_dir = '/content/test'
3
4 classes = os.listdir(train_dir)
5 classes
```

```
['happy', 'angry', 'sad', 'neutral', 'surprise', 'disgust', 'fear']
```

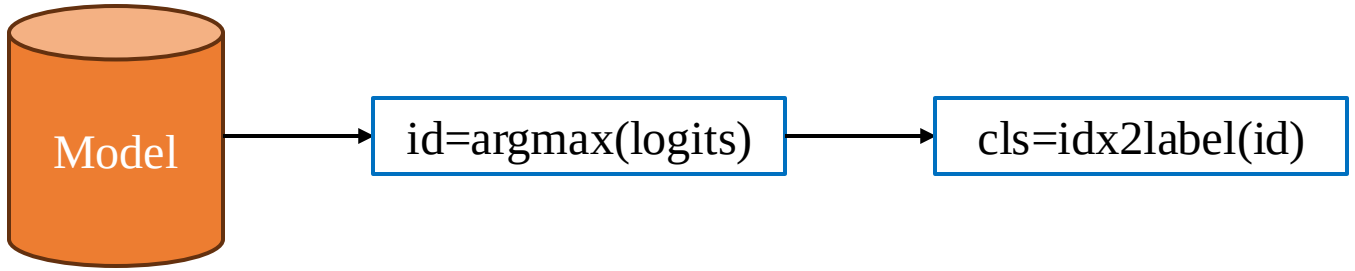
AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Step 3: Create label-idx mapping

In practice, given a list of classnames, we will implement LabelEncoder in PyTorch as follows:

```
1 label2idx = {cls:idx for idx, cls in enumerate(classes)}
2 idx2label = {idx:cls  for cls, idx in label2idx.items()}
```




```
1 label2idx
```

```
{'happy': 0,
 'angry': 1,
 'sad': 2,
 'neutral': 3,
 'surprise': 4,
 'disgust': 5,
 'fear': 6}
```

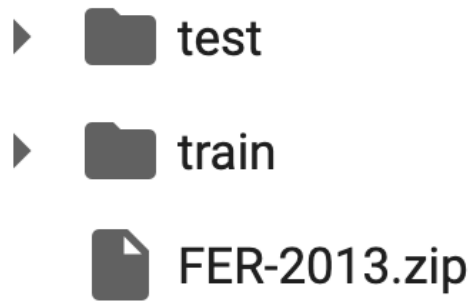
```
1 idx2label
```

```
{0: 'happy',
 1: 'angry',
 2: 'sad',
 3: 'neutral',
 4: 'surprise',
 5: 'disgust',
 6: 'fear'}
```

Sentiment Image Analysis

❖ Step 4: Create PyTorch Dataset: __init__()

```python
class ImageDataset(Dataset):
    def __init__(self, img_dir, norm, label2idx,
                 split='train', train_ratio=0.8):
        self.resize = Resize((img_height, img_width))
        self.norm = norm
        self.split = split
        self.train_ratio = train_ratio
        self.img_dir = img_dir
        self.label2idx = label2idx
        self.img_paths, self.img_labels = self.read_img_files()

        if split in ['train', 'val'] and 'train' in img_dir.lower():
            train_data, val_data = train_test_split(
                list(zip(self.img_paths, self.img_labels)),
                train_size=train_ratio,
                random_state=random_state,
                stratify=self.img_labels
            )

            if split == 'train':
                self.img_paths, self.img_labels = zip(*train_data)
            elif split == 'val':
                self.img_paths, self.img_labels = zip(*val_data)
```



Because we currently have train and test set, we will split validation set from train set.

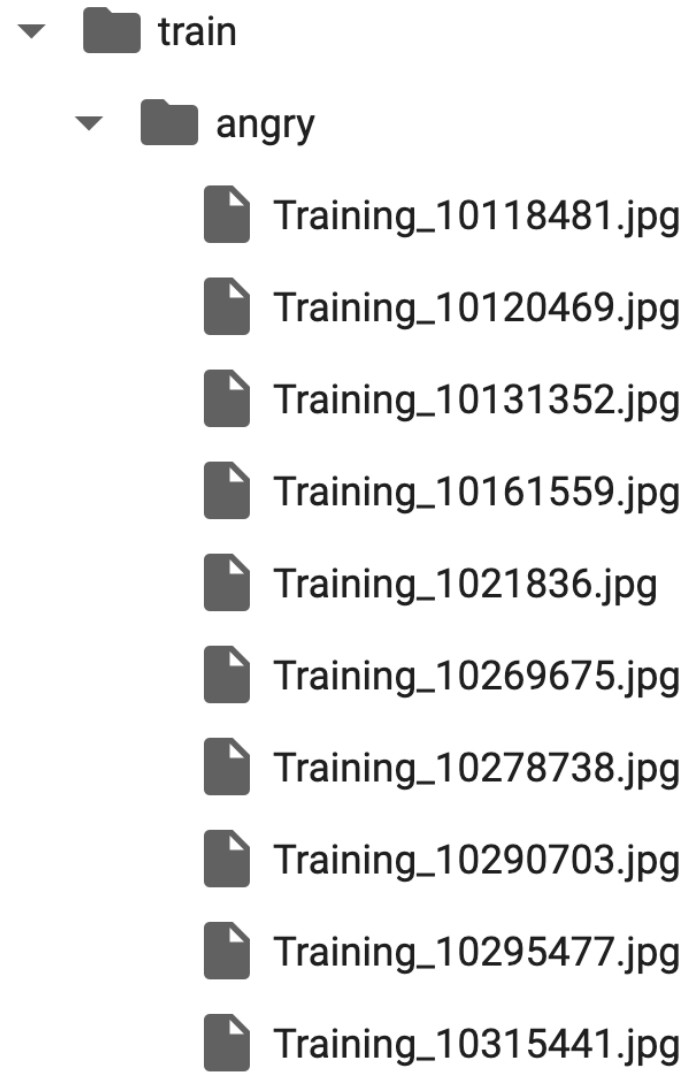
AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Step 4: Create PyTorch Dataset: __read_img_files__()

We extract pairs of (img_path, img_label) from dataset folder:

```
25      def read_img_files(self):
26          img_paths = []
27          img_labels = []
28          for cls in self.label2idx.keys():
29              for img in os.listdir(os.path.join(self.img_dir, cls)):
30                  img_paths.append(os.path.join(self.img_dir, cls, img))
31                  img_labels.append(cls)
32
33          return img_paths, img_labels
```



AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Step 4: Create PyTorch Dataset: __len__() and __getitem__()

```python
def __len__(self):
    return len(self.img_paths)

def __getitem__(self, idx):
    img_path = self.img_paths[idx]
    cls = self.img_labels[idx]
    img = self.resize(read_image(img_path))
    img = img.type(torch.float32)
    label = self.label2idx[cls]
    if self.norm:
        img = (img/127.5) - 1
    return img, label
```

read_image

torchvision.io.read_image(*path: str*, *mode: ImageReadMode = ImageReadMode.UNCHANGED*, *apply_exif_orientation: bool = False*) → Tensor [SOURCE]

Reads a JPEG, PNG or GIF image into a 3 dimensional RGB or grayscale Tensor. Optionally converts the image to the desired format. The values of the output tensor are uint8 in [0, 255].

Resize

CLASS torchvision.transforms.Resize(*size*, *interpolation=InterpolationMode.BILINEAR*, *max_size=None*, *antialias=True*) [SOURCE]

Resize the input image to the given size. If the image is torch Tensor, it is expected to have [..., H, W] shape, where ... means a maximum of two leading dimensions

https://pytorch.org/vision/0.19/generated/torchvision.io.read_image.html
https://pytorch.org/vision/main/generated/torchvision.transforms.Resize.html

AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Step 4: Create PyTorch Dataset: __len__() and __getitem__()

```python
def __len__(self):
    return len(self.img_paths)

def __getitem__(self, idx):
    img_path = self.img_paths[idx]
    cls = self.img_labels[idx]
    img = self.resize(read_image(img_path))
    img = img.type(torch.float32)
    label = self.label2idx[cls]
    if self.norm:
        img = (img/127.5) - 1
    return img, label
```

read_image

torchvision.io.read_image(*path: str*, *mode: ImageReadMode = ImageReadMode.UNCHANGED*, *apply_exif_orientation: bool = False*) → Tensor [SOURCE]

Reads a JPEG, PNG or GIF image into a 3 dimensional RGB or grayscale Tensor. Optionally converts the image to the desired format. The values of the output tensor are uint8 in [0, 255].

Resize

CLASS torchvision.transforms.Resize(*size*, *interpolation=InterpolationMode.BILINEAR*, *max_size=None*, *antialias=True*) [SOURCE]

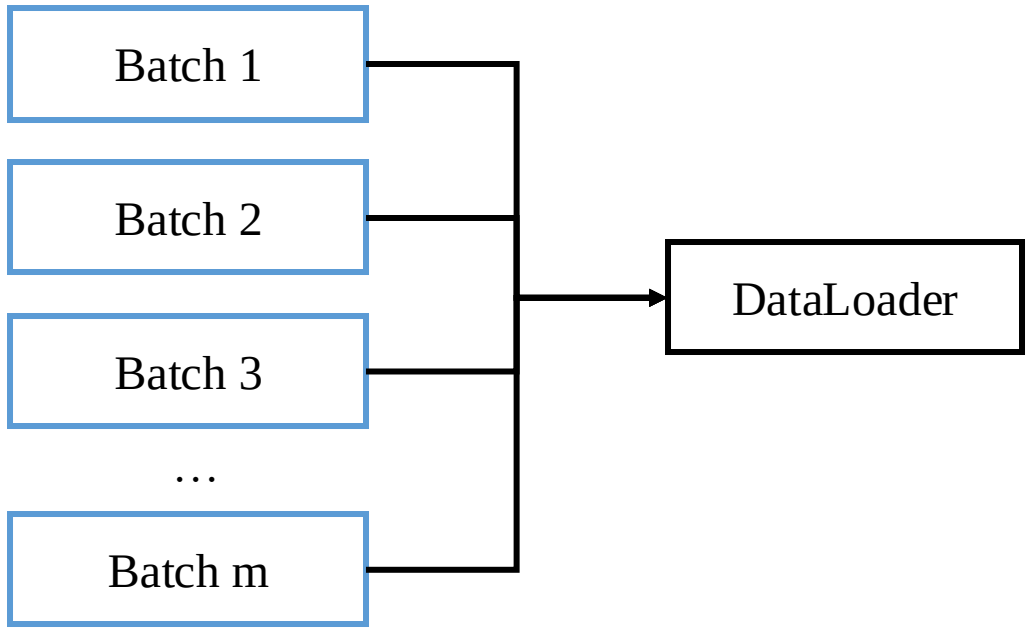
Resize the input image to the given size. If the image is torch Tensor, it is expected to have [..., H, W] shape, where ... means a maximum of two leading dimensions

https://pytorch.org/vision/0.19/generated/torchvision.io.read_image.html
https://pytorch.org/vision/main/generated/torchvision.transforms.Resize.html

Sentiment Image Analysis

❖ Step 5: Create PyTorch Dataloader

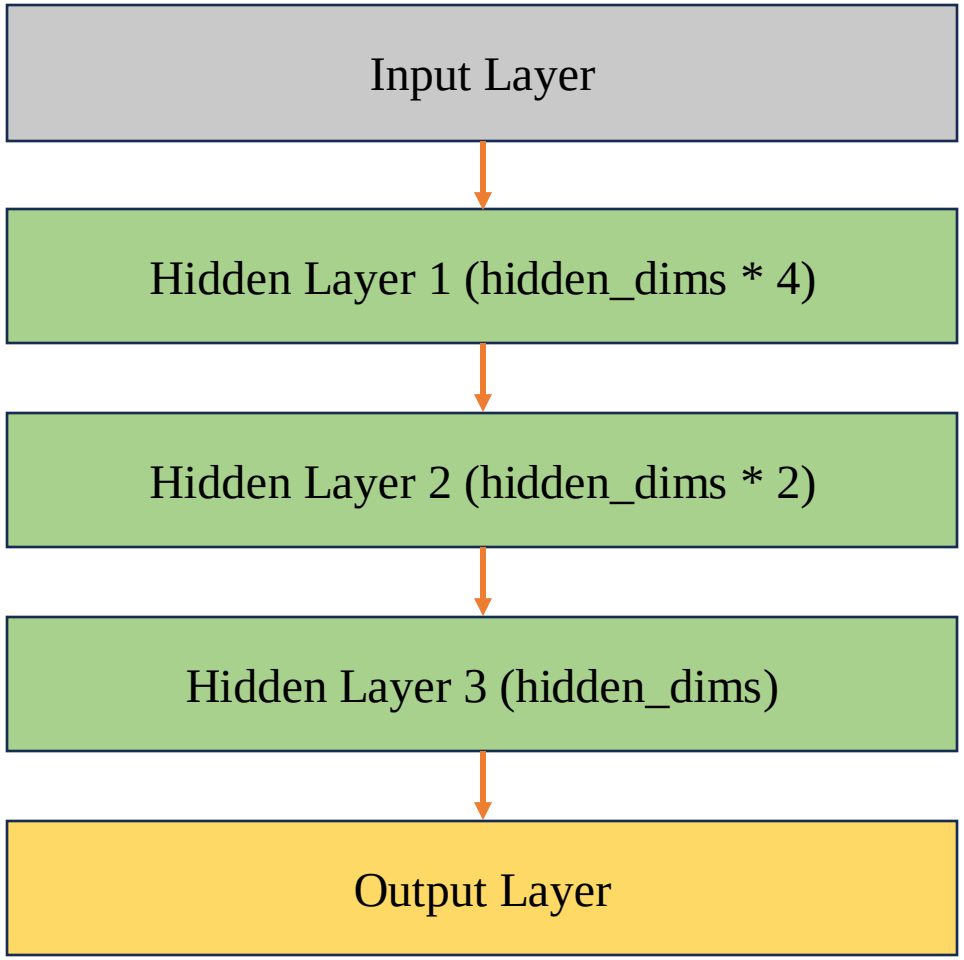
```
batch_size = 256

train_dataset = ImageDataset(train_dir, True,
                             label2idx, split='train')
train_loader = DataLoader(train_dataset,
                          batch_size=batch_size,
                          shuffle=True)

val_dataset = ImageDataset(train_dir, True,
                           label2idx, split='val')
val_loader = DataLoader(val_dataset,
                        batch_size=batch_size,
                        shuffle=False)

test_dataset = ImageDataset(test_dir, True,
                            label2idx, split='test')
test_loader = DataLoader(test_dataset,
                         batch_size=batch_size,
                         shuffle=False)
```



Sentiment Image Analysis

❖ Step 6: Build MLP

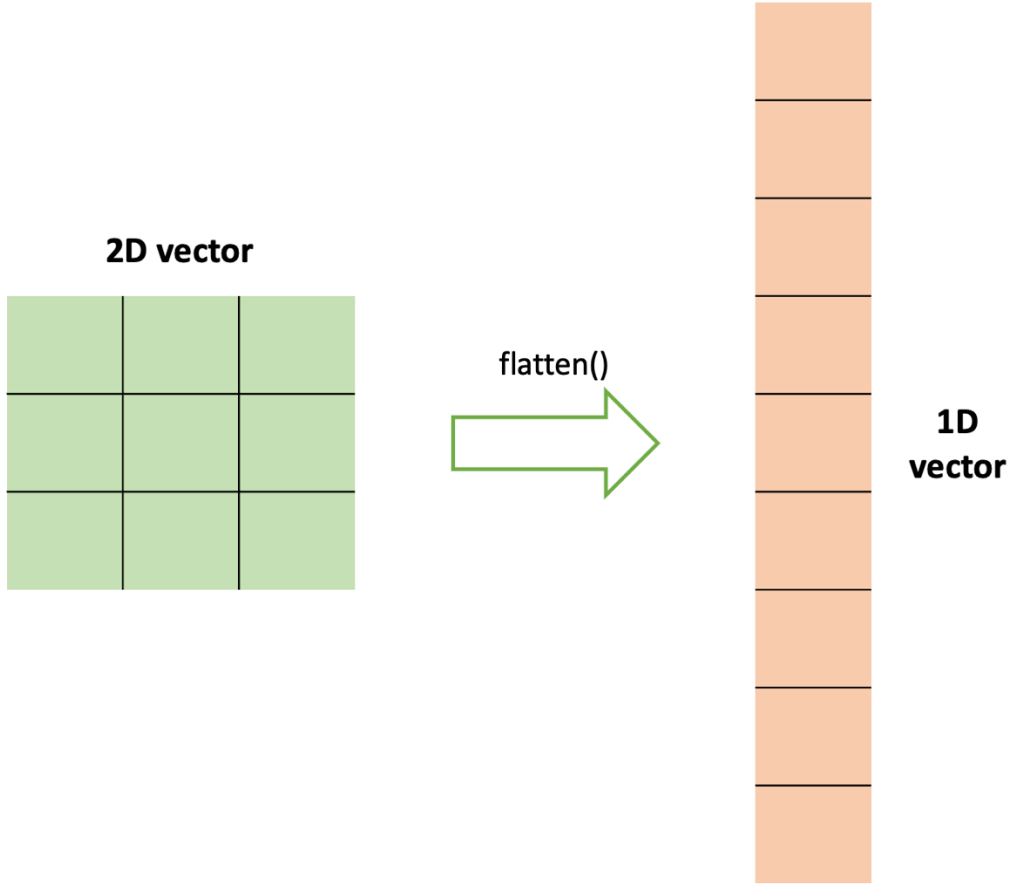
```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims*4)
        self.linear2 = nn.Linear(hidden_dims*4, hidden_dims*2)
        self.linear3 = nn.Linear(hidden_dims*2, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = nn.Flatten()(x)
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        x = self.linear3(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```



Sentiment Image Analysis

❖ Step 6: Build MLP: Flatten()

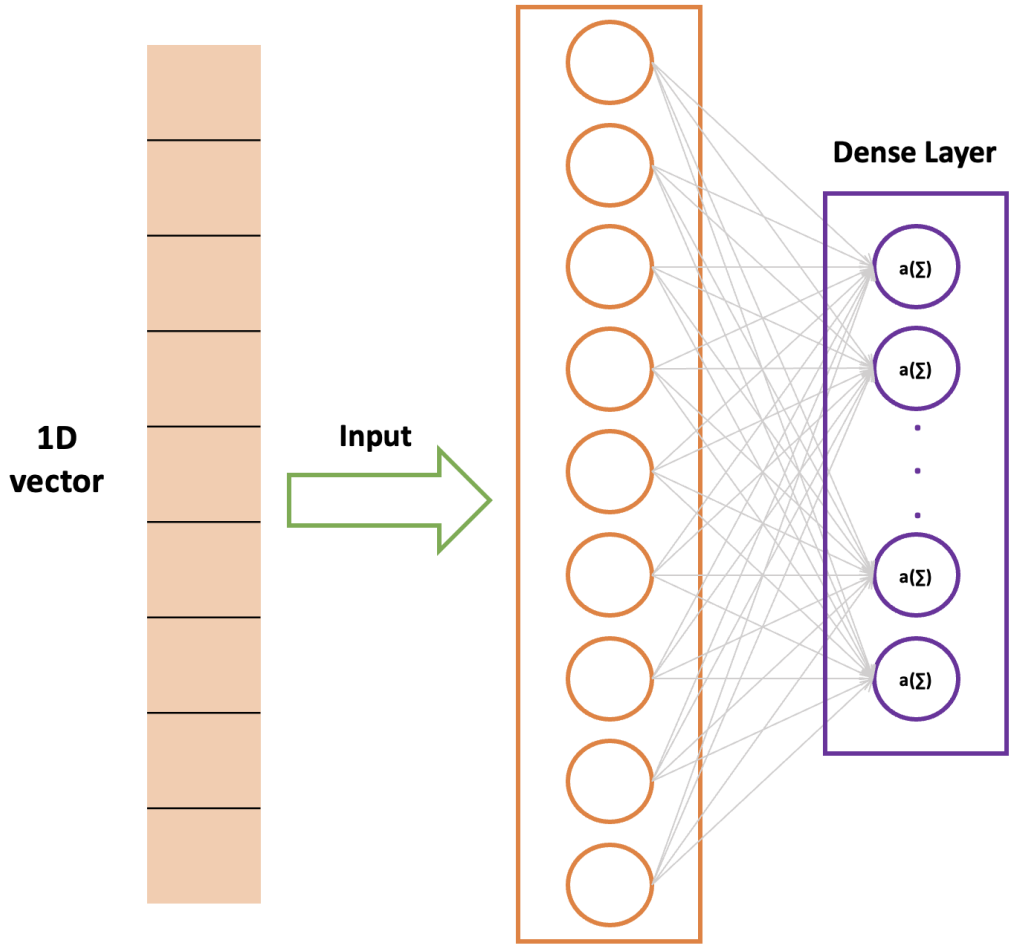
```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims*4)
        self.linear2 = nn.Linear(hidden_dims*4, hidden_dims*2)
        self.linear3 = nn.Linear(hidden_dims*2, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = nn.Flatten()(x)
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        x = self.linear3(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```



Sentiment Image Analysis

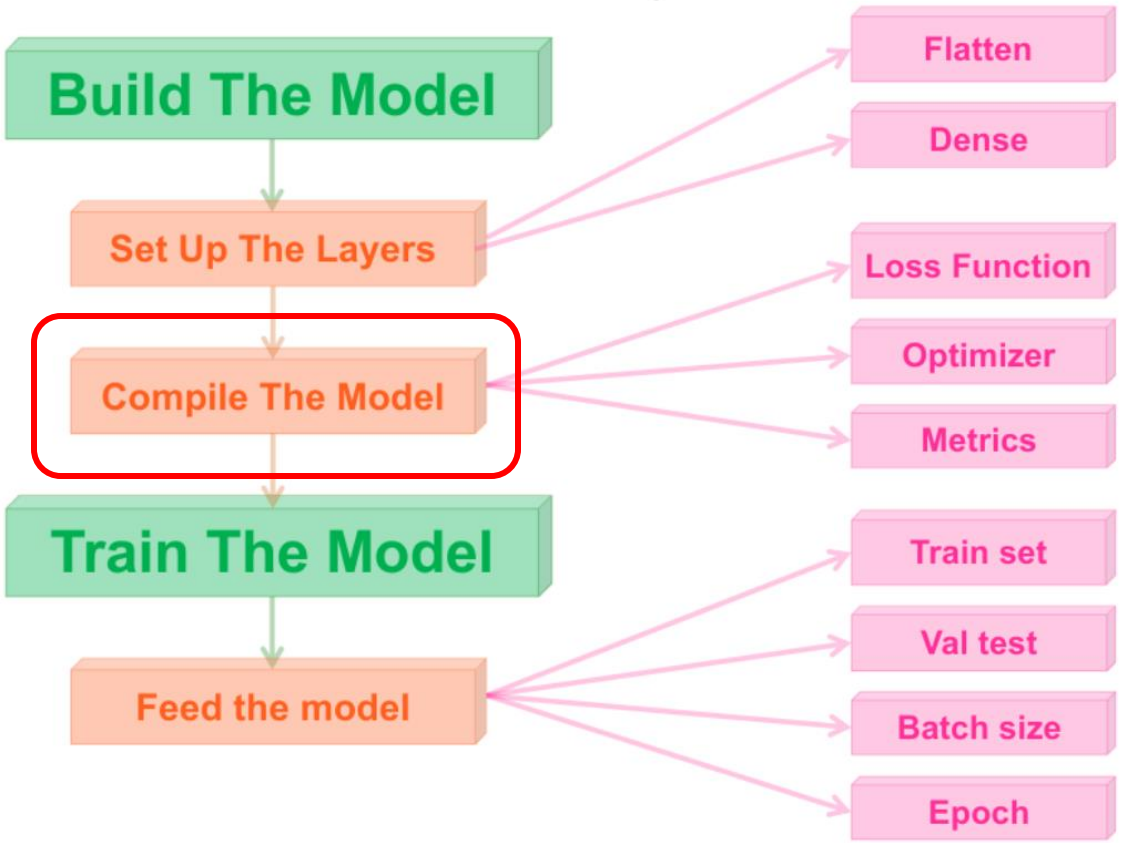
❖ Step 6: Build MLP: Feeding 1D vector image into the model

```python
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dims, hidden_dims*4)
        self.linear2 = nn.Linear(hidden_dims*4, hidden_dims*2)
        self.linear3 = nn.Linear(hidden_dims*2, hidden_dims)
        self.output = nn.Linear(hidden_dims, output_dims)

    def forward(self, x):
        x = nn.Flatten()(x)
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        x = self.linear3(x)
        x = F.relu(x)
        out = self.output(x)
        return out.squeeze(1)
```



Sentiment Image Analysis

❖ Step 7: Declare the model, optimizer and criterion



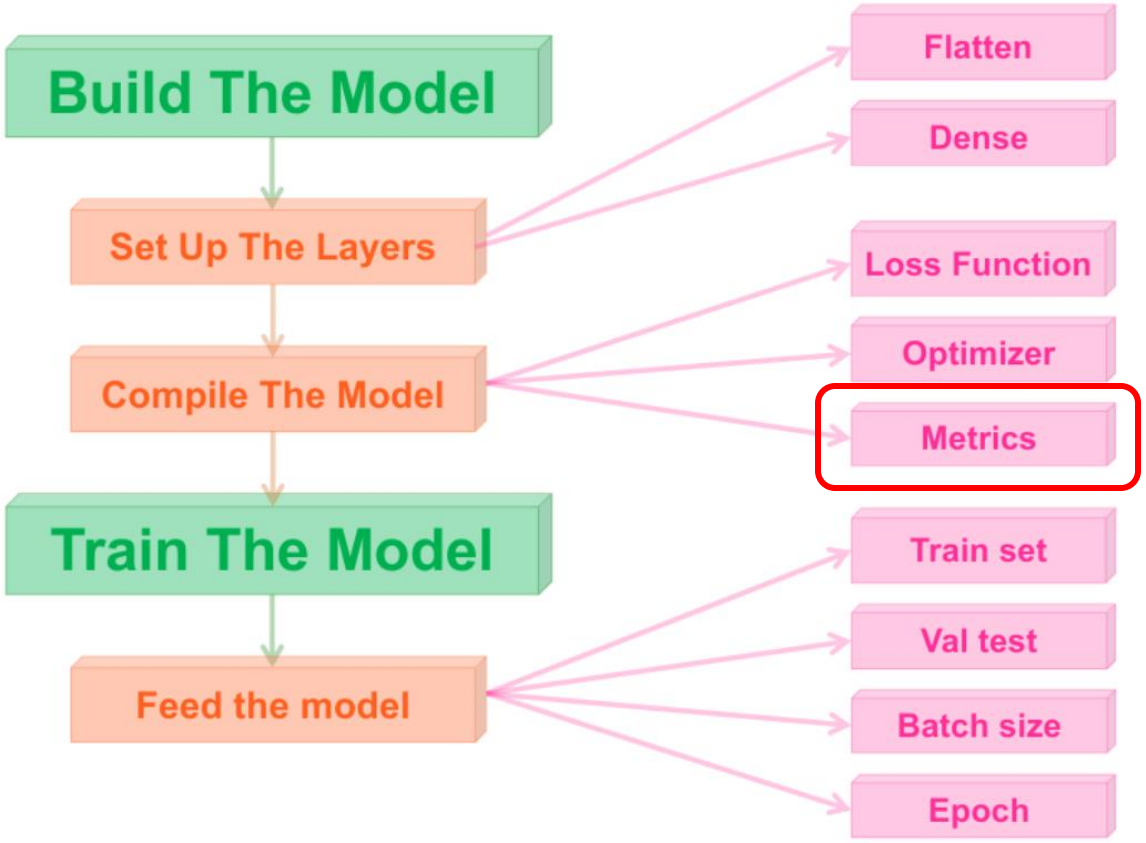

```
input_dims = img_height * img_width
output_dims = len(classes)
hidden_dims = 64
lr = 1e-2

model = MLP(input_dims=input_dims,
            hidden_dims=hidden_dims,
            output_dims=output_dims).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

Sentiment Image Analysis

❖ Step 8: Create accuracy function

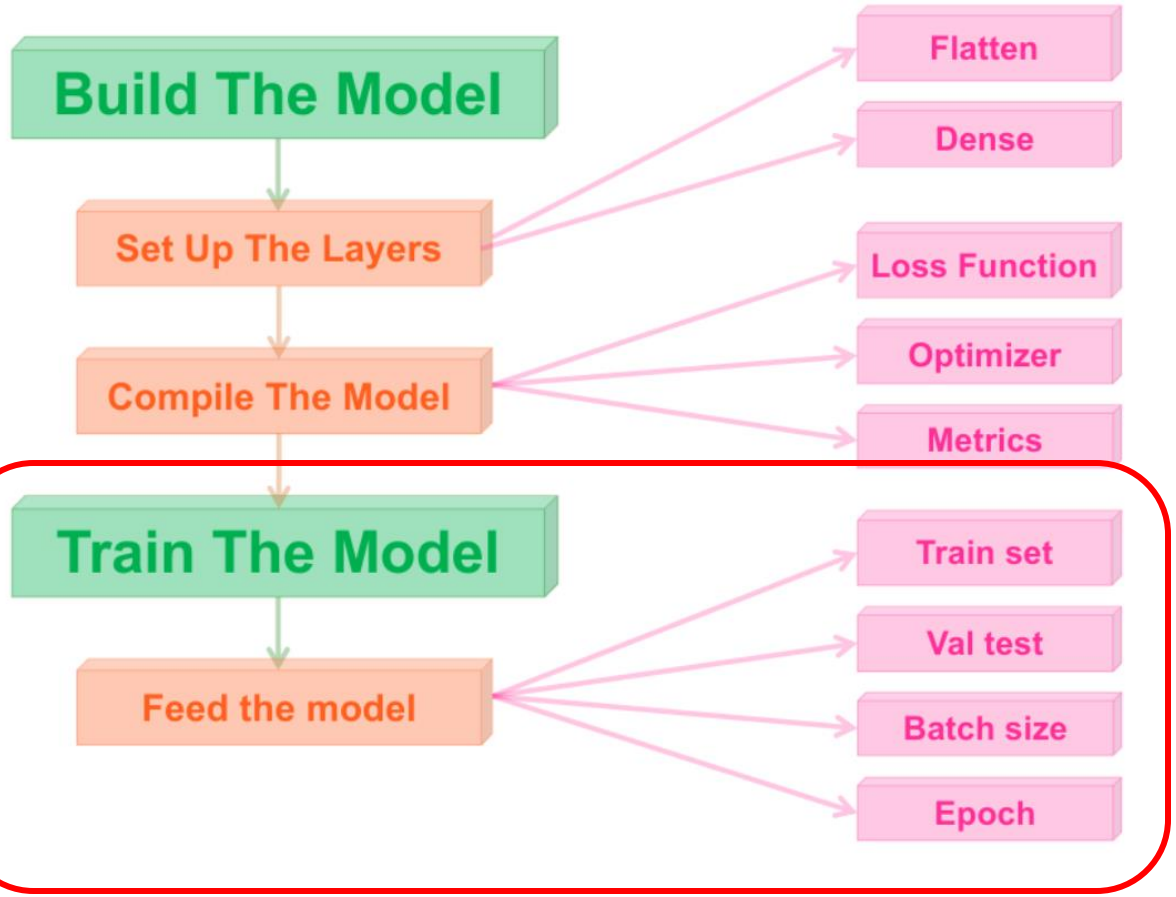
$$accuracy = \frac{true\_predictions}{n\_samples}$$

```
def compute_accuracy(y_hat, y_true):
    _, y_hat = torch.max(y_hat, dim=1)
    correct = (y_hat == y_true).sum().item()
    accuracy = correct / len(y_true)
    return accuracy
```



Sentiment Image Analysis

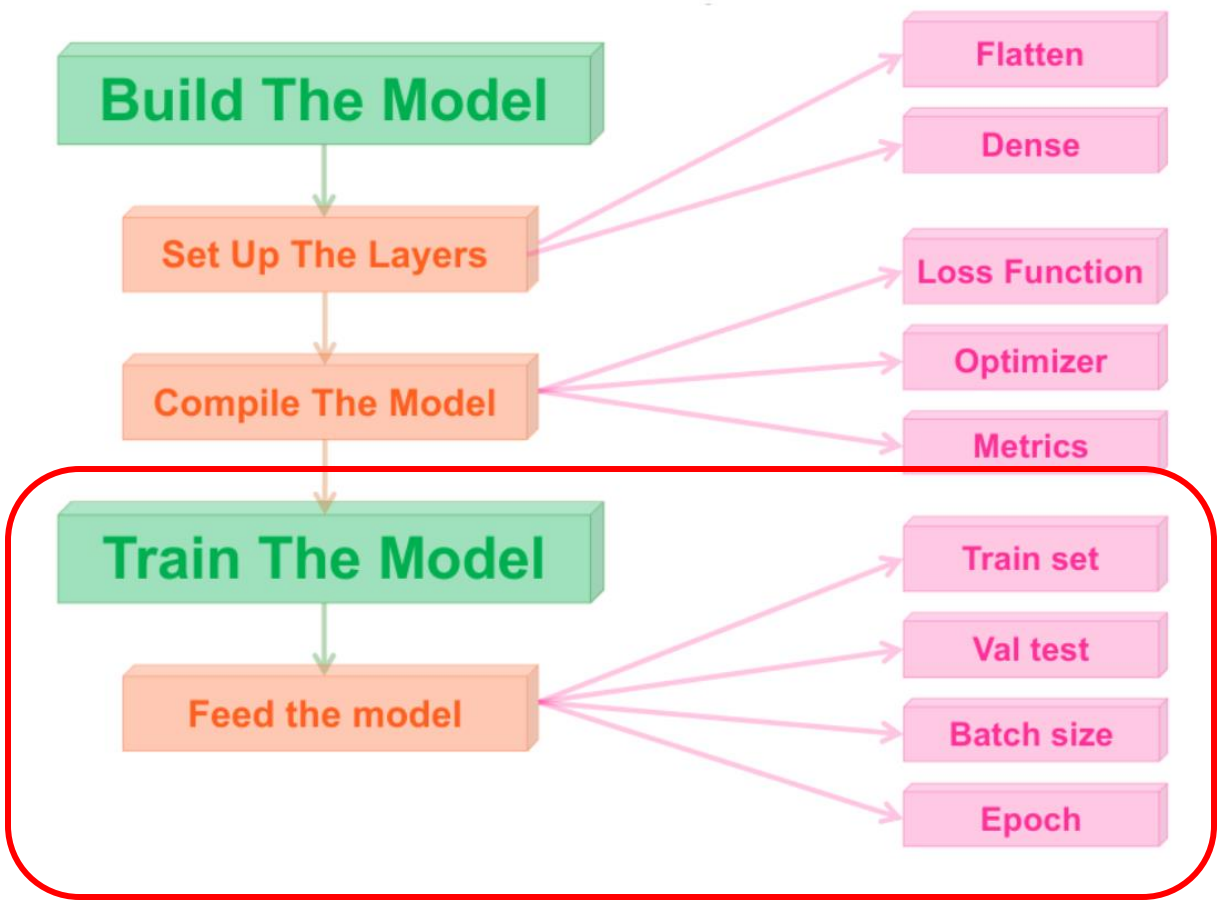
❖ Step 9: Training




```python
epochs = 100
train_losses = []
val_losses = []
train_accs = []
val_accs = []

for epoch in range(epochs):
    train_loss = 0.0
    train_target = []
    train_predict = []
    model.train()
    for X_samples, y_samples in train_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        optimizer.zero_grad()
        outputs = model(X_samples)
        loss = criterion(outputs, y_samples)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        train_predict.append(outputs.detach().cpu())
        train_target.append(y_samples.cpu())

    train_loss /= len(train_loader)
    train_losses.append(train_loss)

    train_predict = torch.cat(train_predict)
    train_target = torch.cat(train_target)
    train_acc = compute_accuracy(train_predict, train_target)
    train_accs.append(train_acc)
```

Sentiment Image Analysis

❖ Step 9: Training (Evaluation)



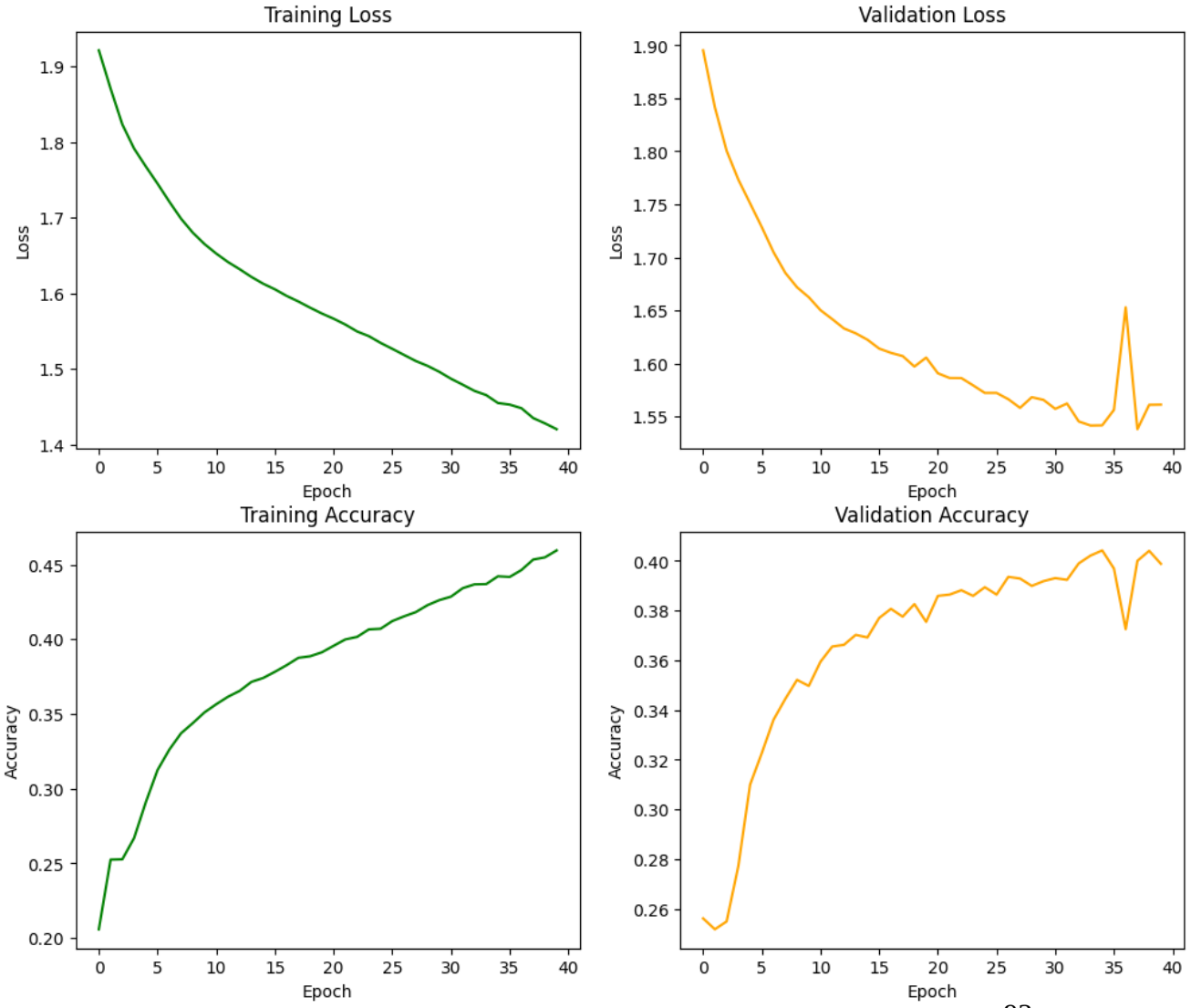

```
33      val_loss = 0.0
34      val_target = []
35      val_predict = []
36      model.eval()
37      with torch.no_grad():
38          for X_samples, y_samples in val_loader:
39              X_samples = X_samples.to(device)
40              y_samples = y_samples.to(device)
41              outputs = model(X_samples)
42              val_loss += criterion(outputs, y_samples).item()
43
44              val_predict.append(outputs.cpu())
45              val_target.append(y_samples.cpu())
46
47      val_loss /= len(val_loader)
48      val_losses.append(val_loss)
49
50      val_predict = torch.cat(val_predict)
51      val_target = torch.cat(val_target)
52      val_acc = compute_accuracy(val_predict, val_target)
53      val_accs.append(val_acc)
```

AI VIET NAM
@aivietnam.edu.vn

Sentiment Image Analysis

❖ Step 10: Evaluation

```
fig, ax = plt.subplots(2, 2, figsize=(12, 10))
ax[0, 0].plot(train_losses, color='green')
ax[0, 0].set(xlabel='Epoch', ylabel='Loss')
ax[0, 0].set_title('Training Loss')

ax[0, 1].plot(val_losses, color='orange')
ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
ax[0, 1].set_title('Validation Loss')

ax[1, 0].plot(train_accs, color='green')
ax[1, 0].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 0].set_title('Training Accuracy')

ax[1, 1].plot(val_accs, color='orange')
ax[1, 1].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 1].set_title('Validation Accuracy')

plt.show()
```



Sentiment Image Analysis

❖ Step 10: Evaluation

$$accuracy = \frac{true\_predictions}{n\_samples}$$

```python
def compute_accuracy(y_hat, y_true):
    _, y_hat = torch.max(y_hat, dim=1)
    correct = (y_hat == y_true).sum().item()
    accuracy = correct / len(y_true)
    return accuracy
```

```python
val_target = []
val_predict = []
model.eval()
with torch.no_grad():
    for X_samples, y_samples in val_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        outputs = model(X_samples)

        val_predict.append(outputs.cpu())
        val_target.append(y_samples.cpu())

    val_predict = torch.cat(val_predict)
    val_target = torch.cat(val_target)
    val_acc = compute_accuracy(val_predict, val_target)

    print('Evaluation on val set:')
    print(f'Accuracy: {val_acc}')
```

```
Evaluation on val set:
Accuracy: 0.39881574364332983
```

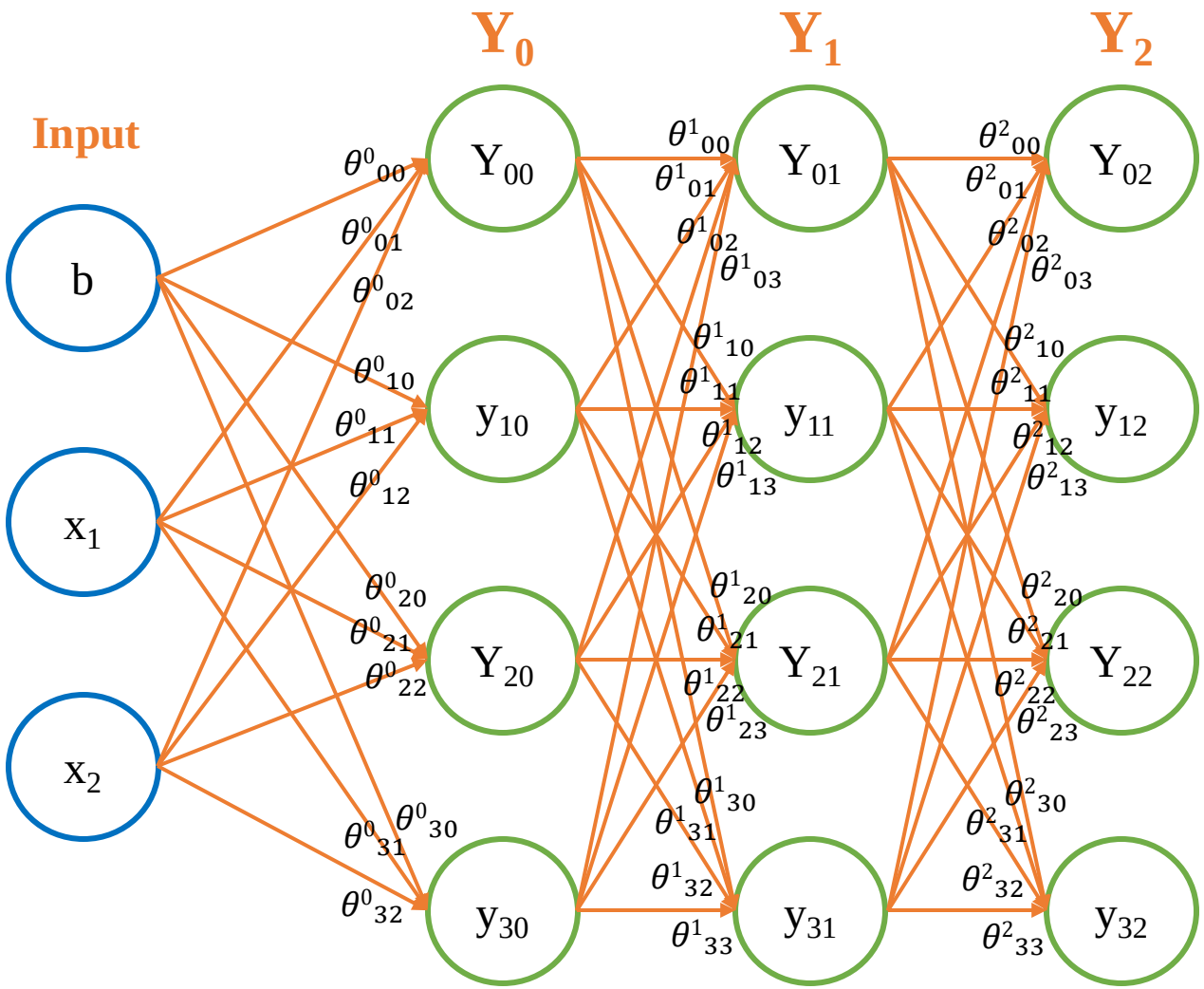
AI VIET NAM
@aivietnam.edu.vn

Summarization and Q&A

AI VIET NAM
@aivietnam.edu.vn

Summarization and Q&A

❖ Summarization

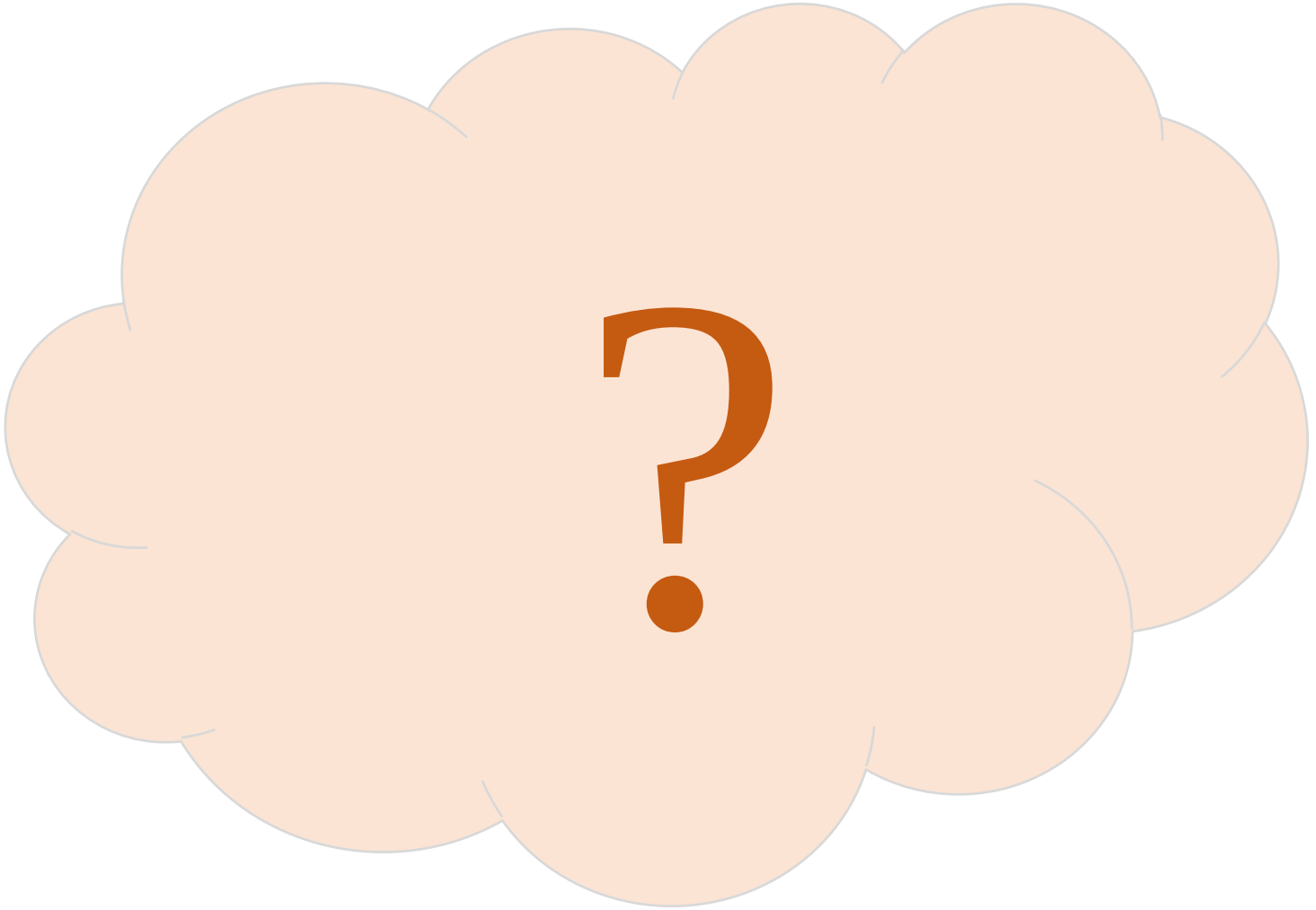


In this lecture, we have discussed about:

- Multilayer Perceptrons (MLPs).
- Activation functions.
- Practice how to implement and train MLP model on tabular data, non-linear data and image data using PyTorch.

Summarization and Q&A

❖ Q&A




Thank you!