AI VIETNAM
All-in-One Course
(TA Session)

Softmax Regression

Exercise

Dinh-Thang Duong – TA

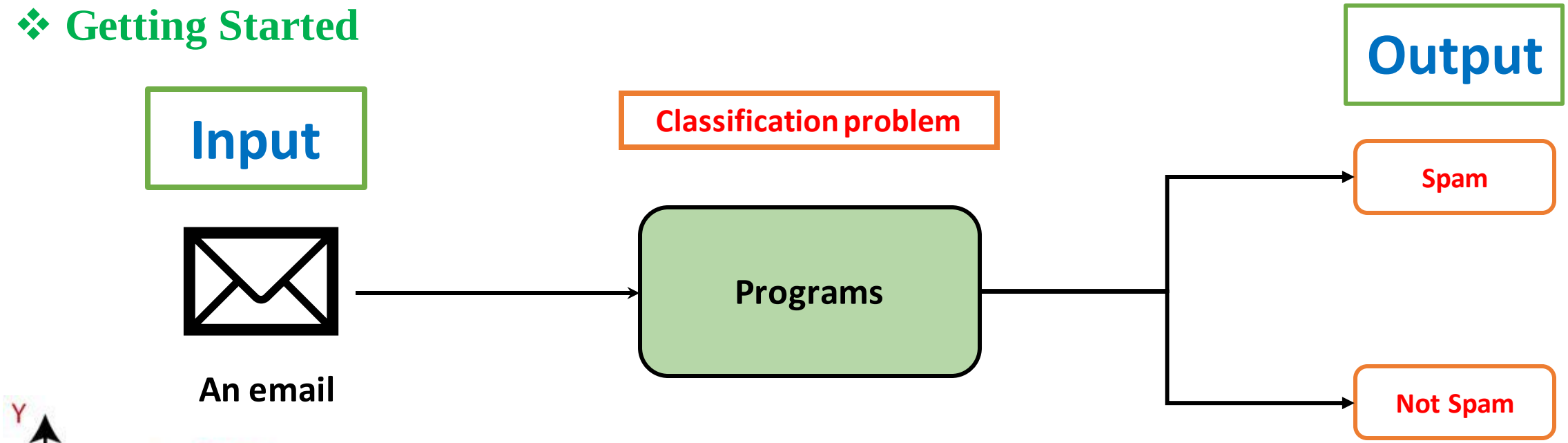
Year 2023

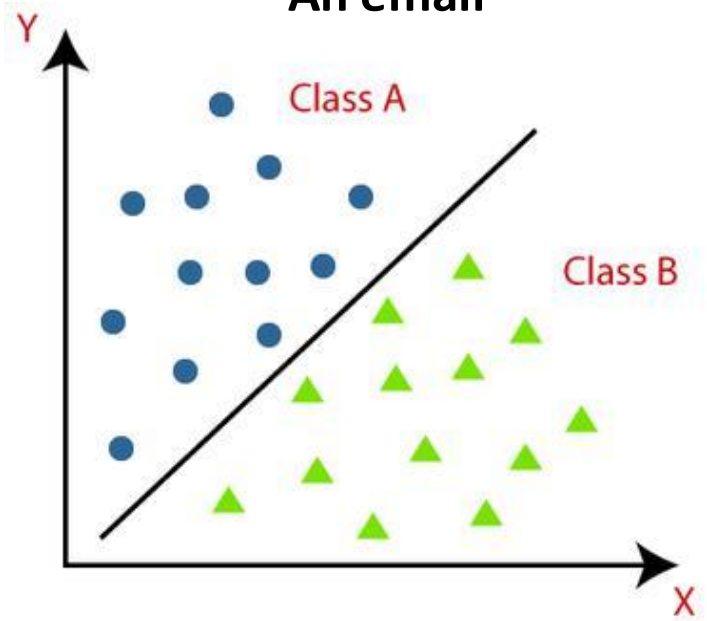
Outline

- Review
- Card Fraud Detection
- Sentiment Analysis
- Question

Review

❖ Getting Started



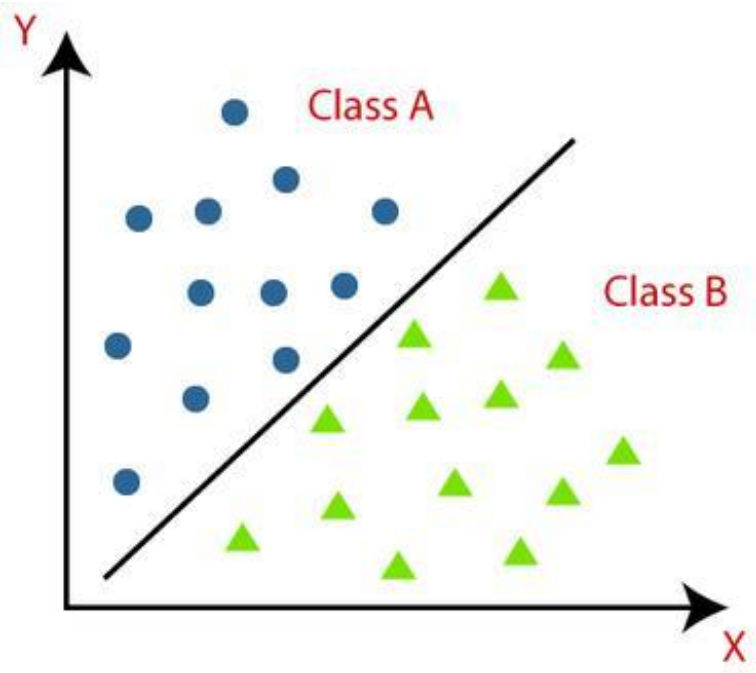


Output is categorical. E.g:

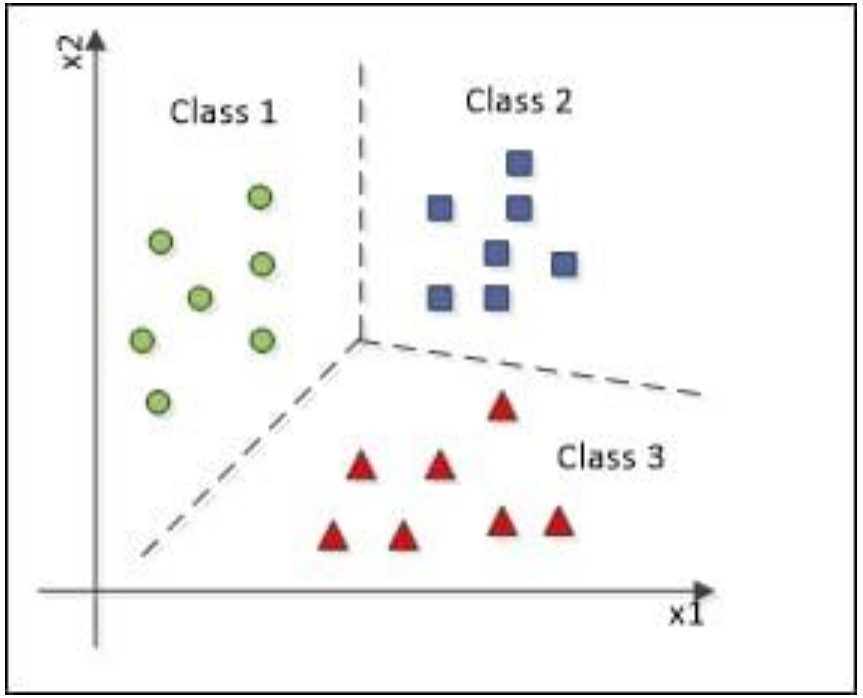
- Spam/Not Spam
- Positive/Negative/Neutral
- Cat/Dog/Bird/Mouse
- 1/2/3/4

Review

❖ Getting Started



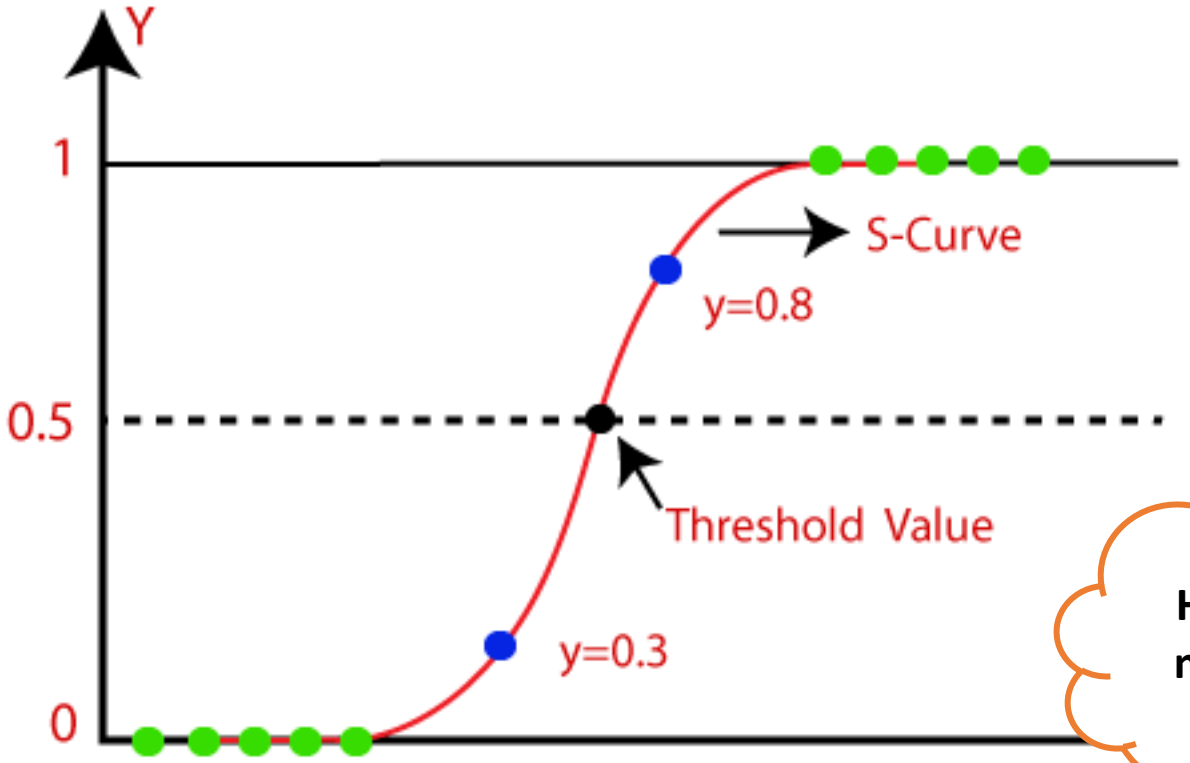
Binary Classification (n_classes=2)



Multi-class Classification (n_classes >= 2)

Review

❖ Logistic Regression for multiclass?



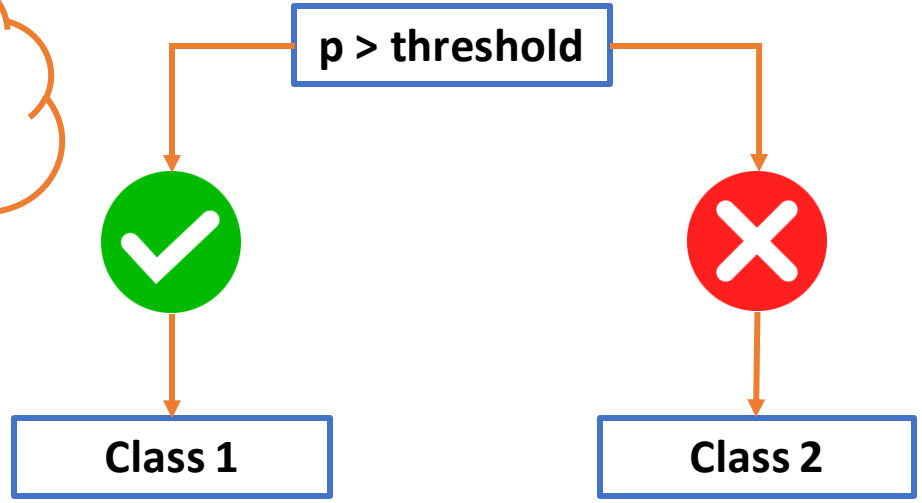
Sigmoid function

$$z = b_0 + \sum_{i=1}^{n\_features} b_i x_i$$

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

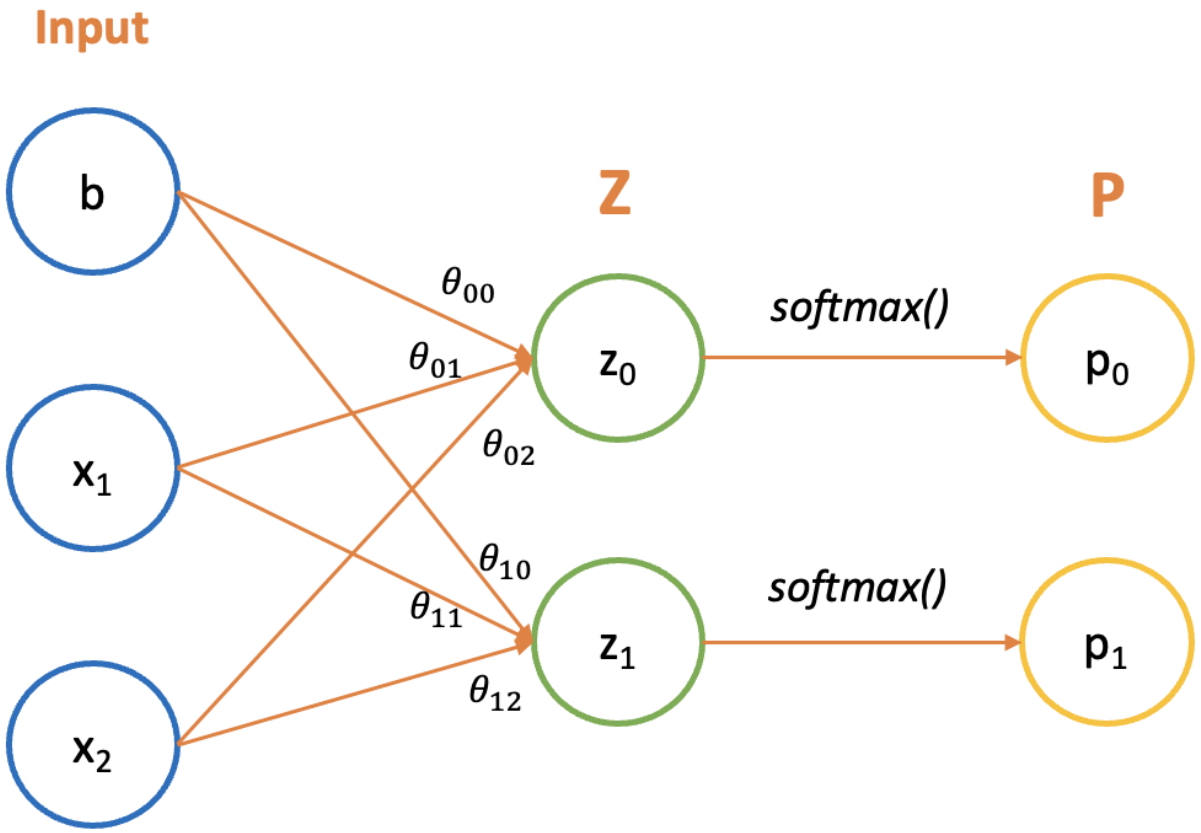
Range value: (0, 1)

How about multiclass?



Review

❖ Introduction



Softmax Regression (Multinomial Logisitic Regression): A supervised ML algorithm used for classification tasks. It is a generalization of Logistic Regression to the case where we want to handle multiple classes.

Review

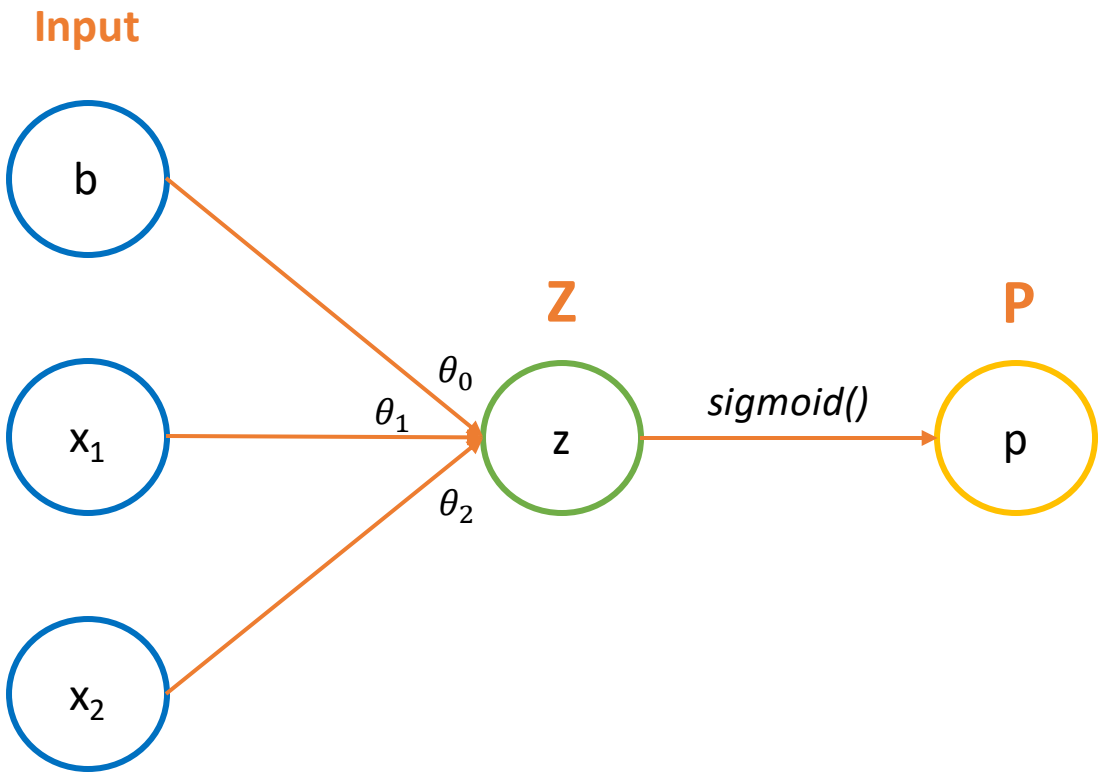
❖ Logistic Regression (Graph)

x : **Input data x_i**

z : **Linear value (z)**

p : **Activation value (p)**

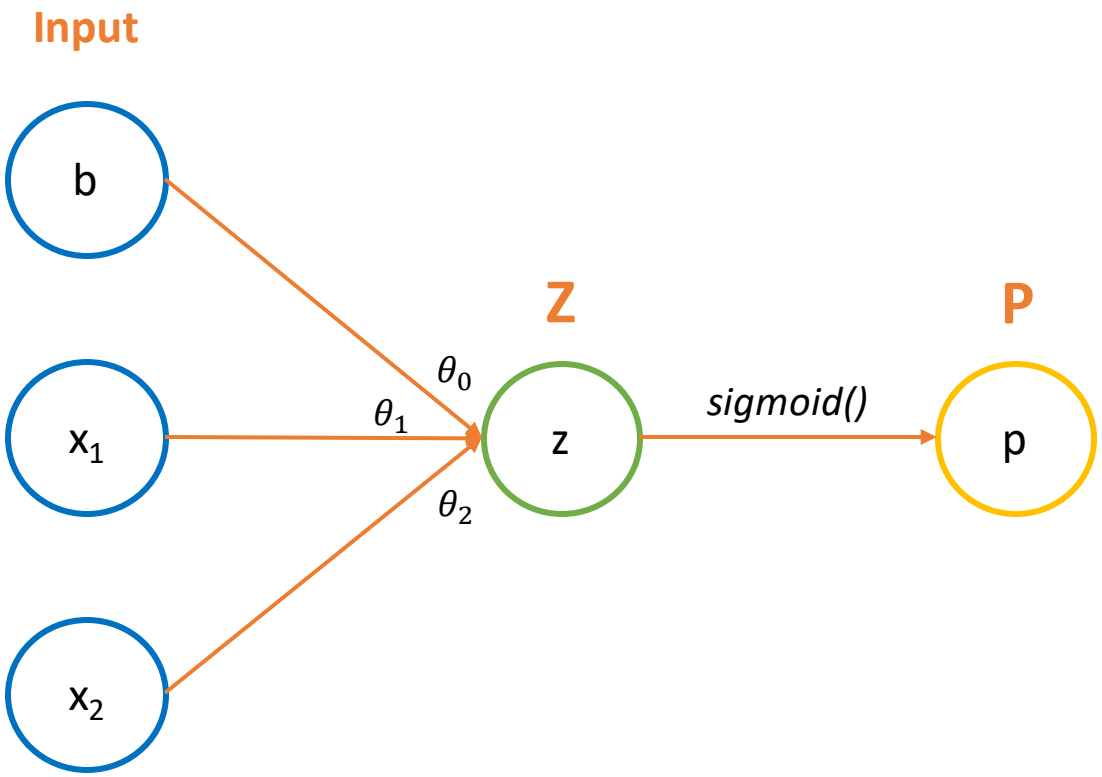
: **Direction**



Logistic Regression Computation

Review

❖ Logistic Regression (Graph)

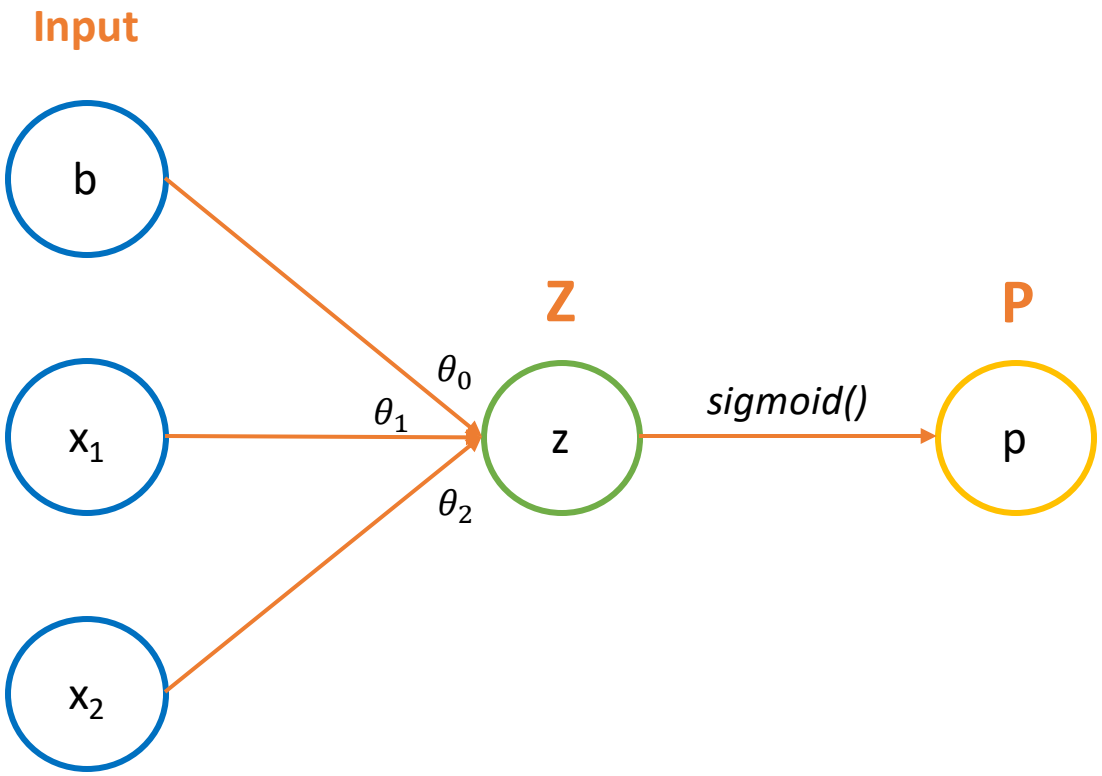


θ → : Direction

Each direction is associated with a unique weight theta.

Review

❖ Logistic Regression (Graph)



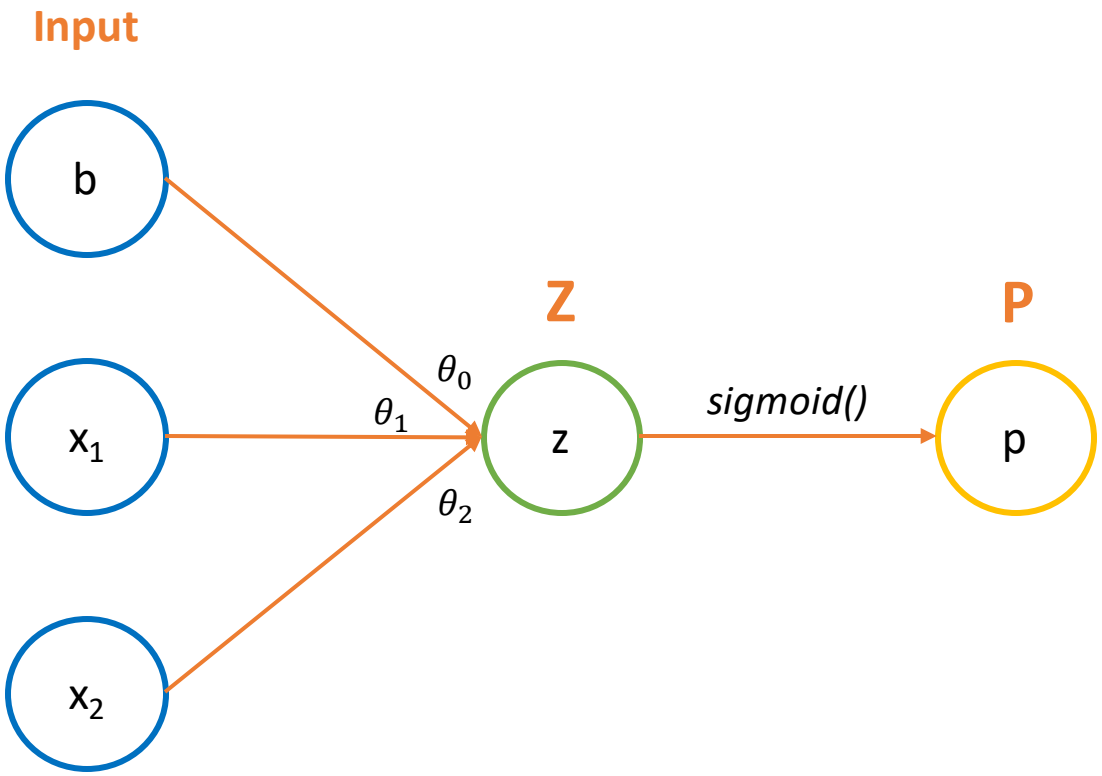
z : **Linear value**

Green node represents the dot product of input features and their weights (including bias):

$$z = X \cdot \theta = \sum_{i=1}^{n} X_i \theta_i$$

Review

❖ Logistic Regression (Graph)



p : **Activation value (p)**

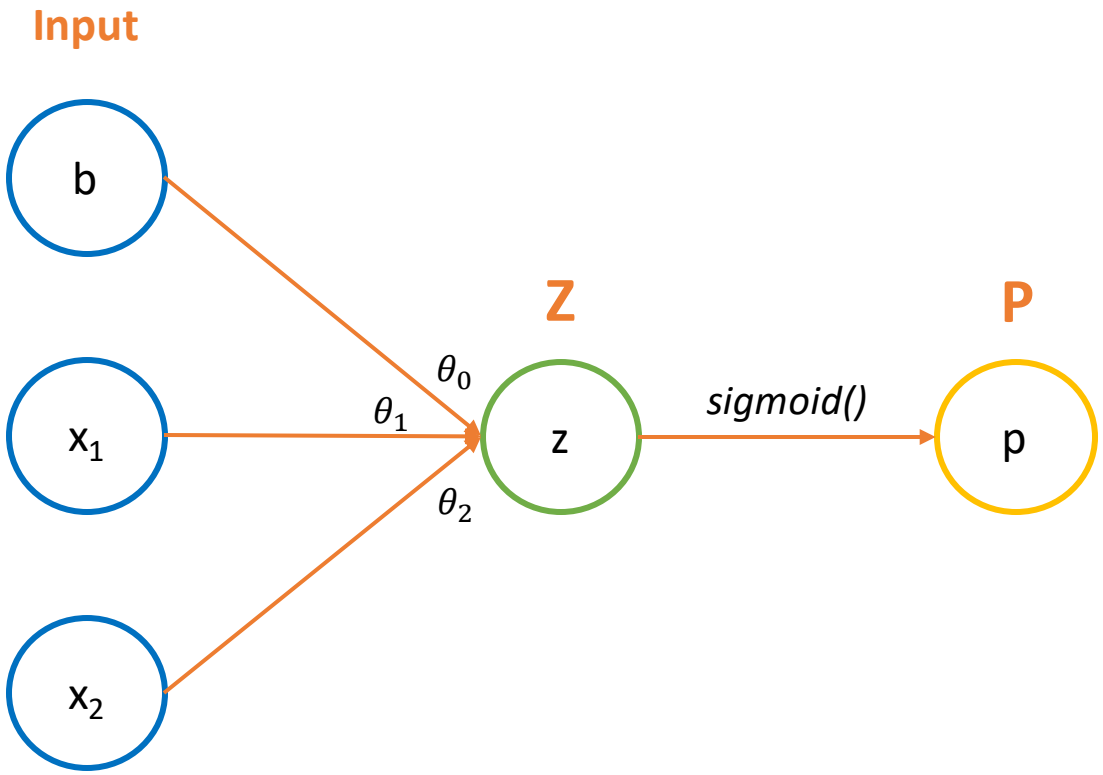
Yellow node represents the output of sigmoid function (activation function) with the input of z:

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

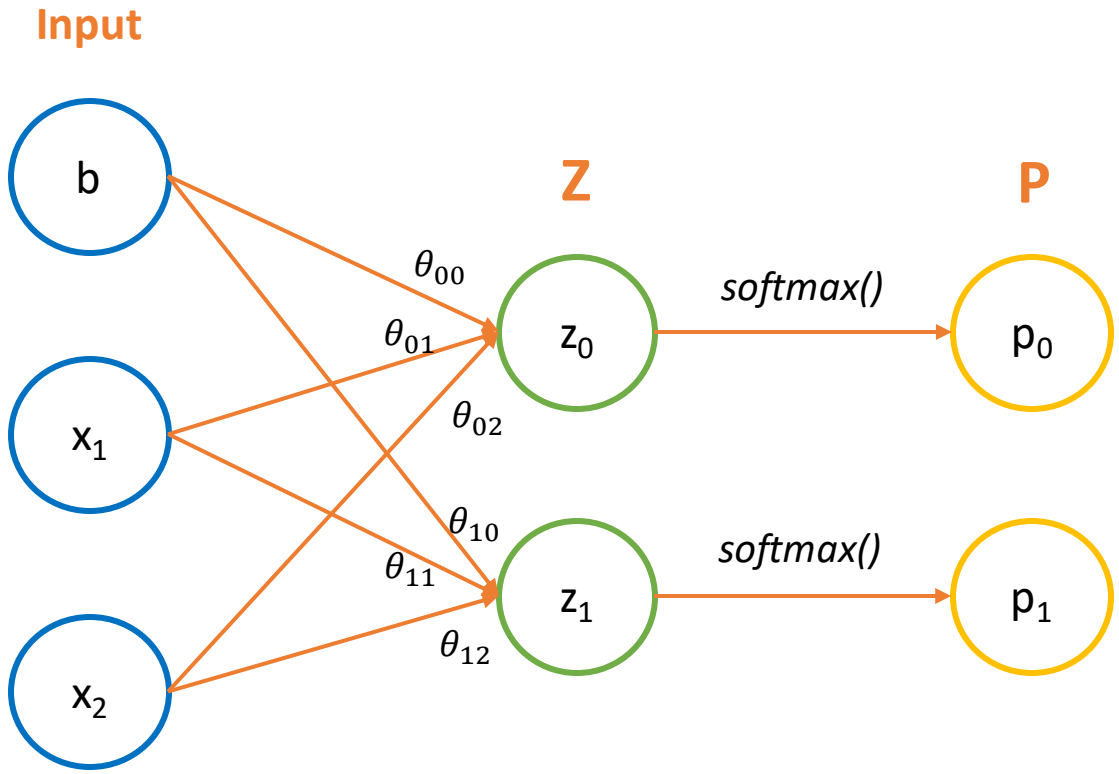
Review

❖ Softmax Regression

n_features = 2, n_classes = 2



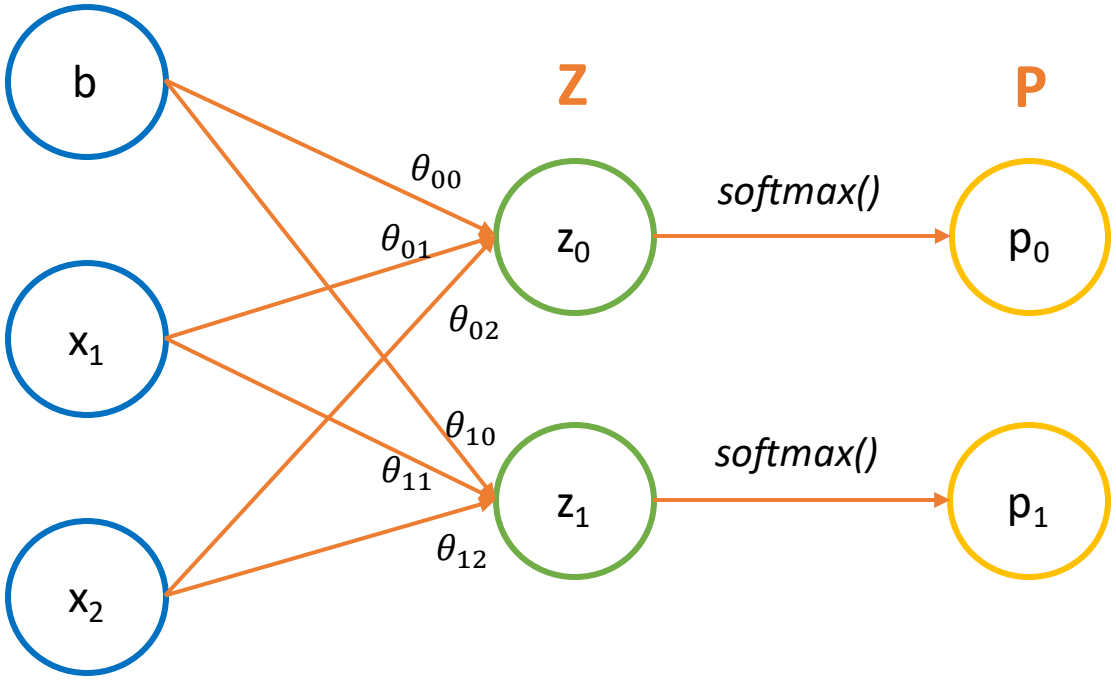
Logistic Regression



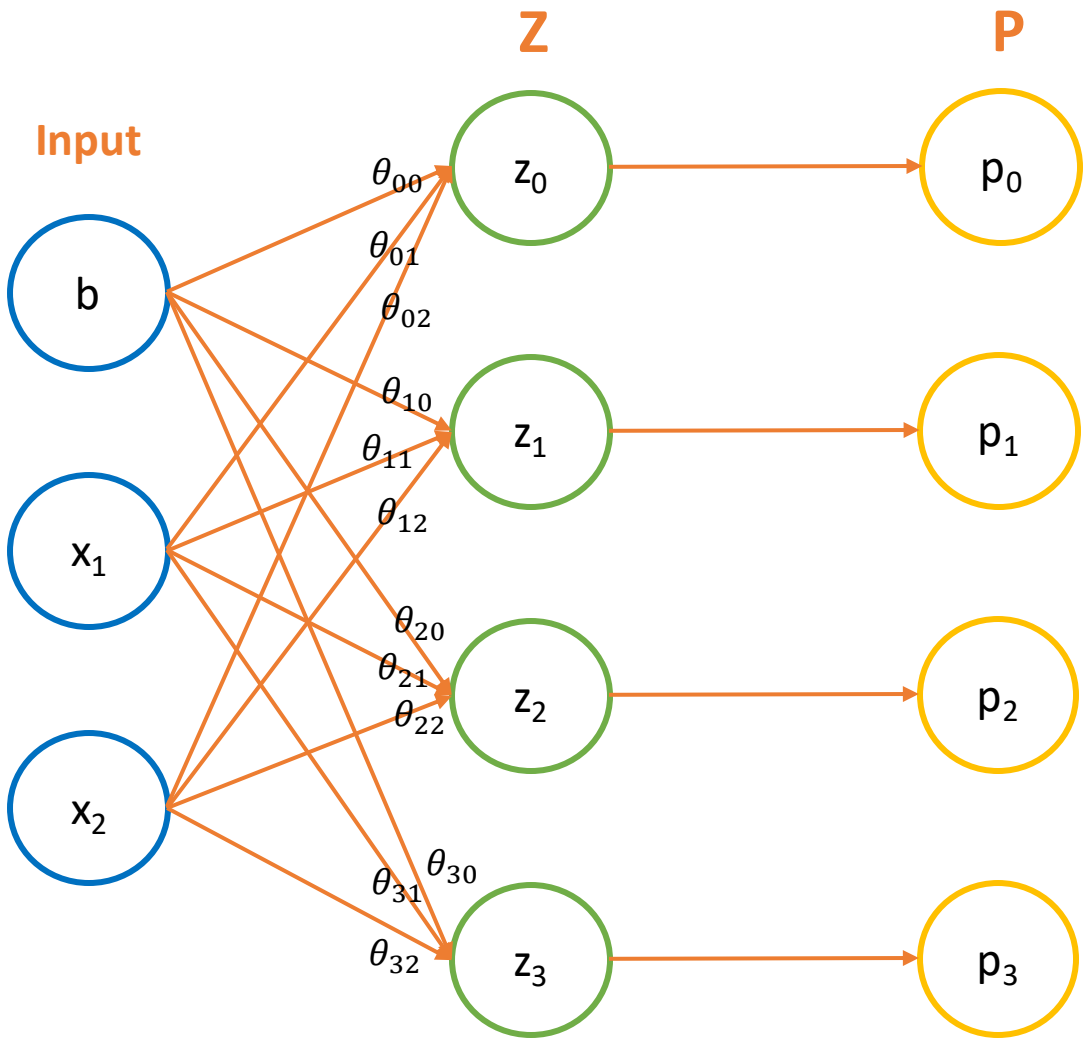
Softmax Regression

Review

❖ Softmax Regression



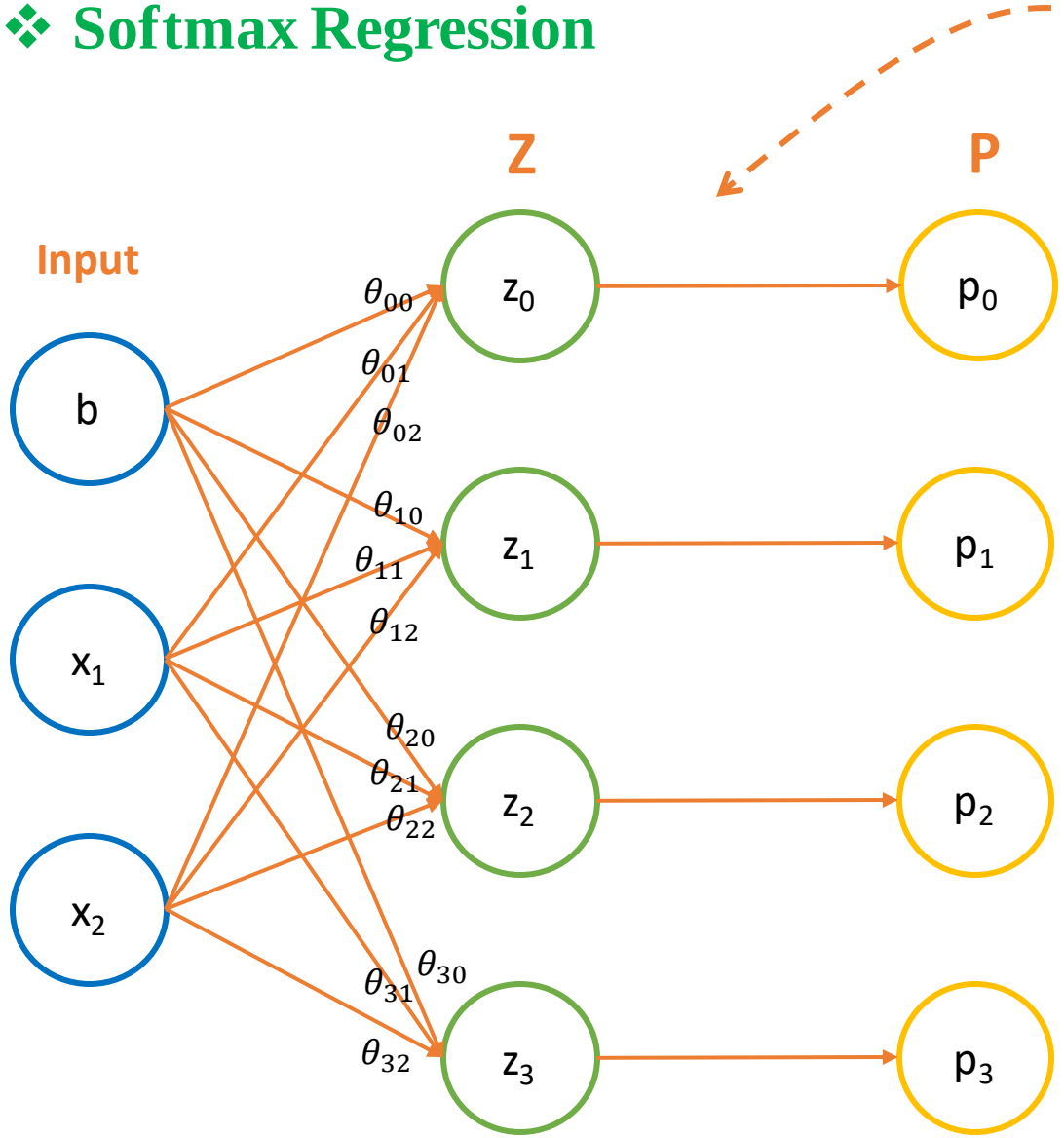
Softmax Regression (2 class)



Softmax Regression (4 class)

Review

❖ Softmax Regression



n_features = 2, n_classes = 4

❖ **Calculate z of class i'th:**

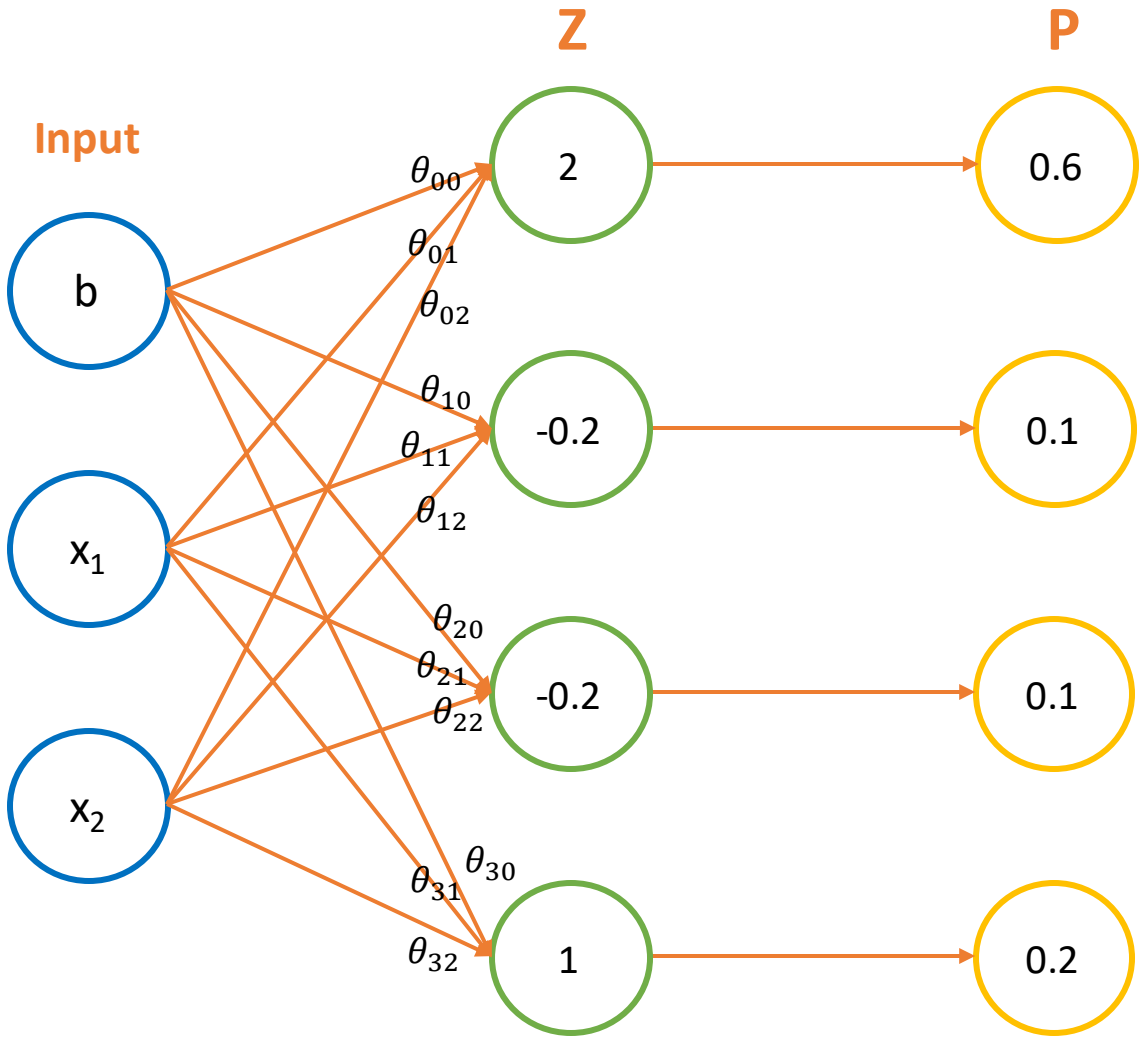
$$z_i = \theta_i^T X = b\theta_{i0} + \sum_{j=1}^{n\_features} \theta_{ij} X_j$$

❖ **Calculate p of class i'th:**

$$p_i = softmax(Z) = \frac{e^{z_i}}{\sum_{j=1}^{n\_classes} e^{z_j}}$$

Review

❖ Properties of Softmax Output



n_features = 2, n_classes = 4

$$1) 0 < p_i < 1$$

$$2) \sum_i^{n\_classes} p_i = 1$$

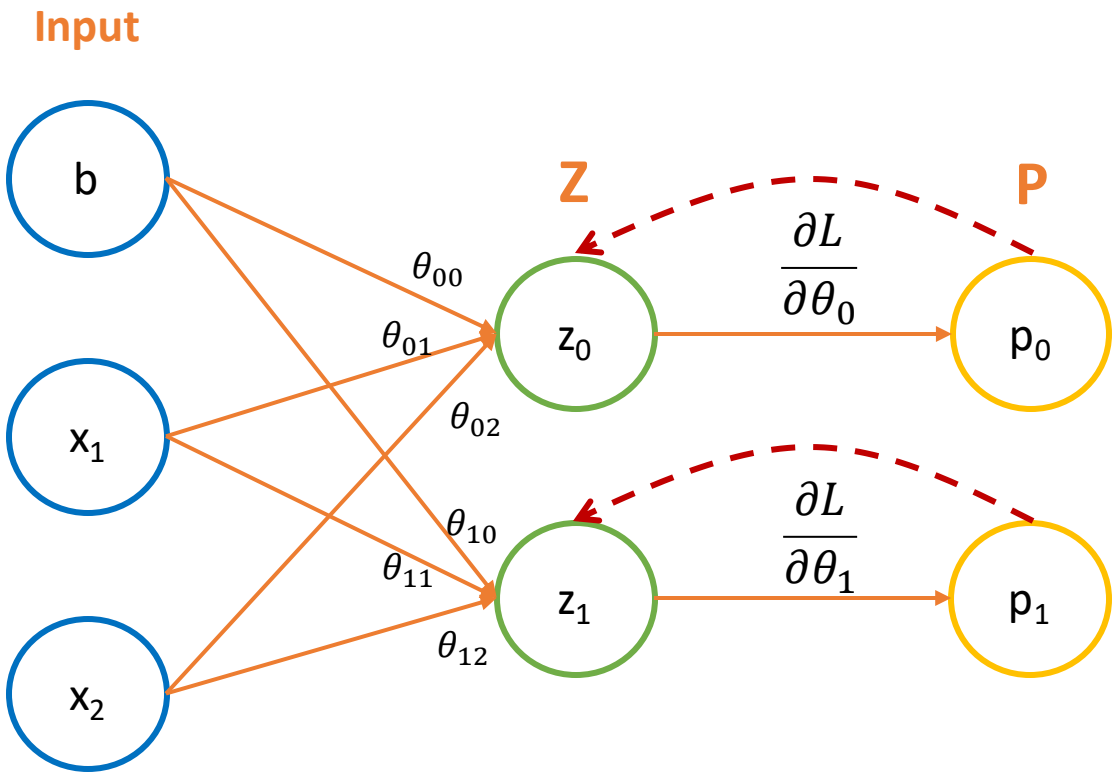
$$3) z_i < 0, p_i > 0$$

$$4) z_i = z_j, p_i = p_j$$

$$5) max = z_i, max = p_i$$

Review

❖ Loss function and derivative



❖ **Cross-entropy loss:**

$$L(\hat{y}, y) = -\log(y^T \cdot \hat{y}) = -y^T \cdot \log(\hat{y})$$

❖ **Derivative:**

$$\nabla_\theta L = \frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta} = X \cdot (\hat{y} - y)^T$$

Review

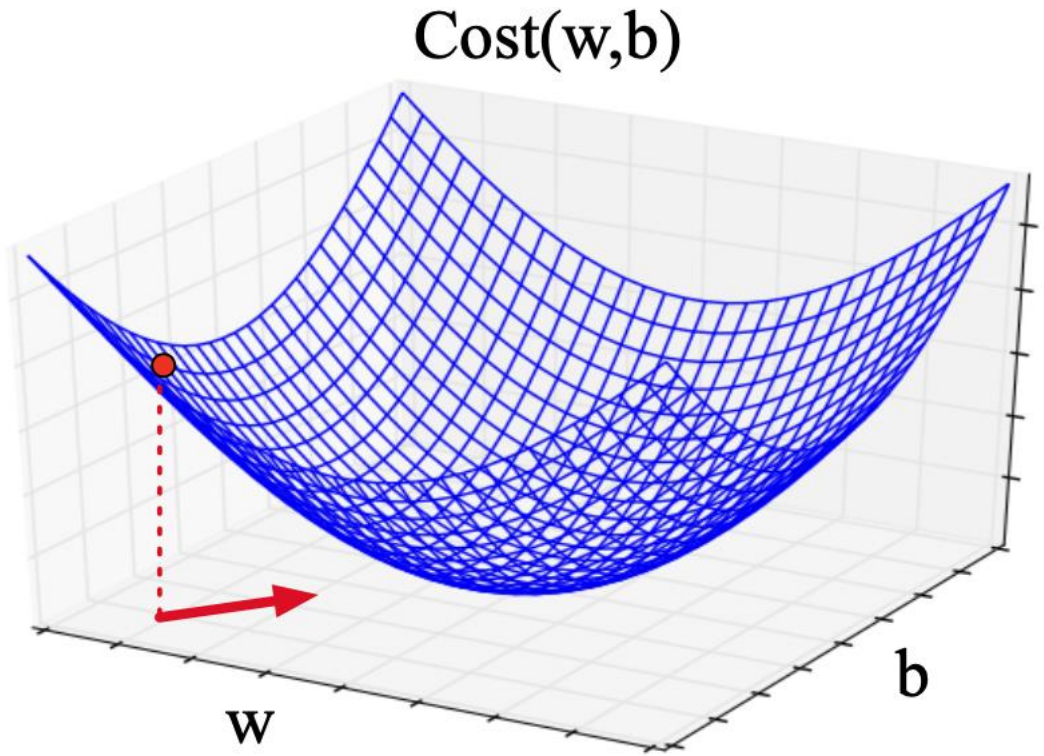
❖ Softmax Regression objective

Our objective is to find the optimal weights and bias (theta) that minimize the loss function:

$$\hat{\theta} = arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}^i, y^i)$$

For each step, compute the gradient and update the current theta:

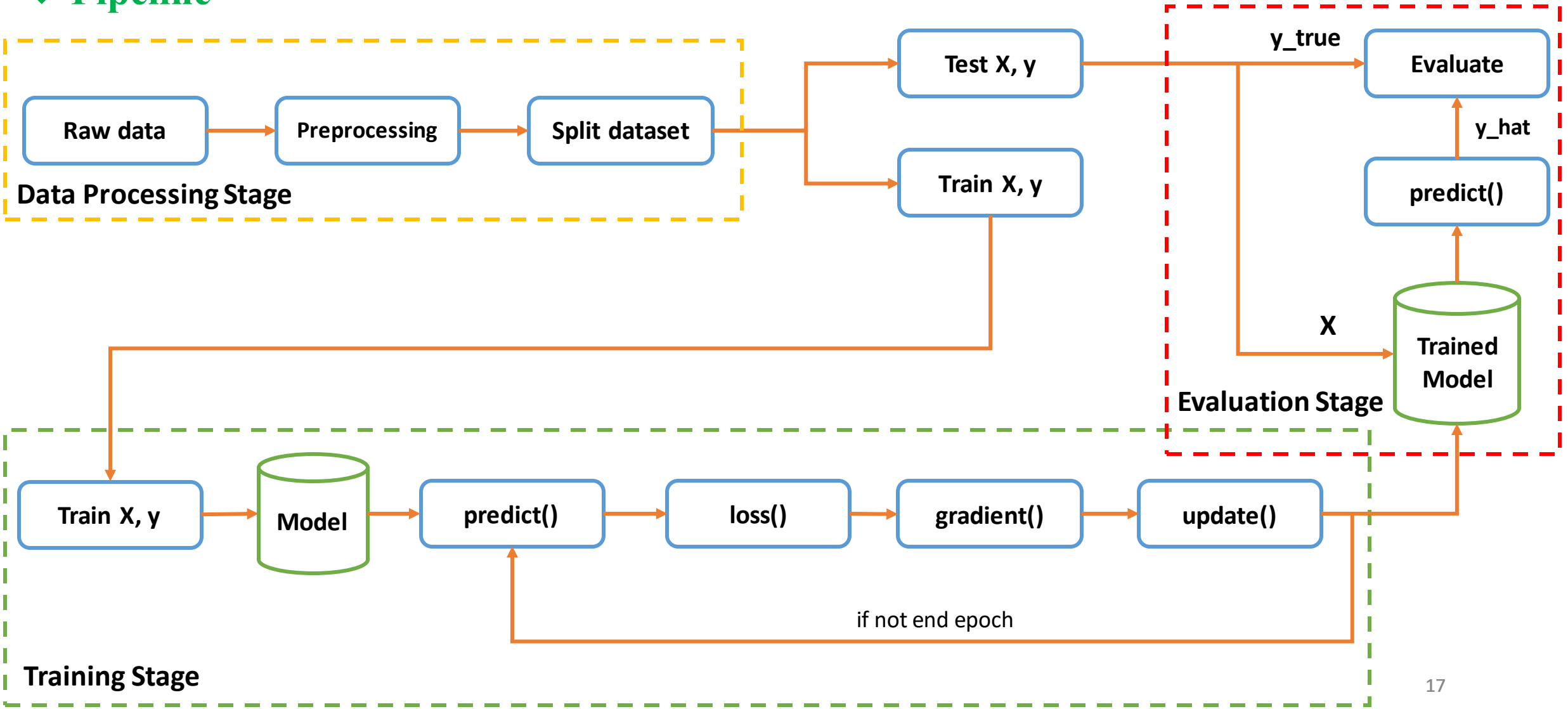
- $\nabla_{\theta_s} L = \frac{1}{N} X^{i^T} (\hat{y}^i - y^i)$
- $\theta_{s+1} = \theta_s - \eta \nabla_{\theta_s} L$



Gradient Descent

Review

❖ Pipeline



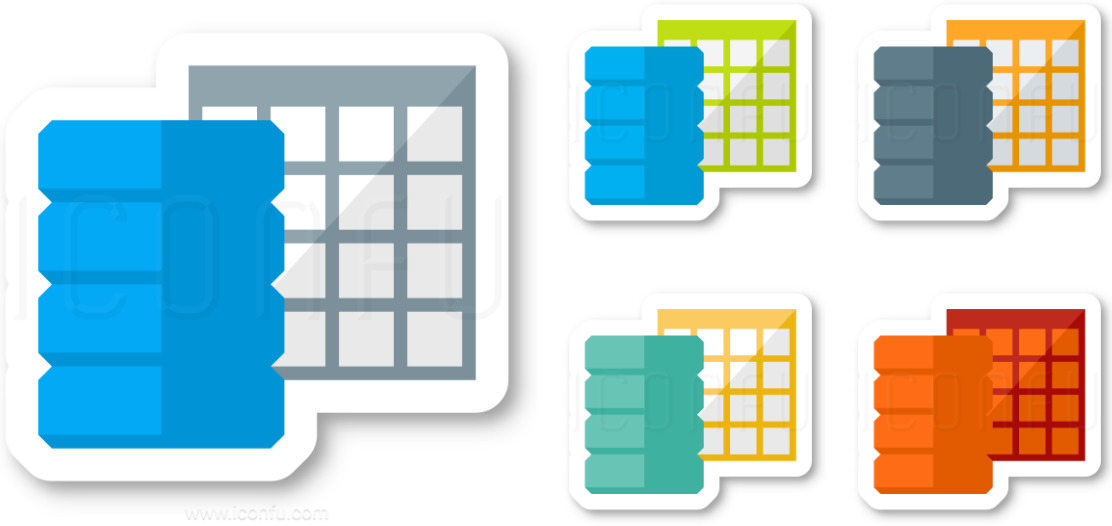
Card Fraud Detection

❖ Introduction

Description: Given card_fraud_detection.csv dataset, build a Card Fraud Detection model using Softmax Regression.

Card Fraud Detection

❖ Step 1: Import libraries and read dataset



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from sklearn.model_selection import train_test_split
6 from sklearn.preprocessing import StandardScaler
```

```
1 dataset_path = 'creditcard.csv'
2 df = pd.read_csv(
3     dataset_path
4 )
5 df
```

Card Fraud Detection

❖ Step 2: Get dataset information

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
```

```
1 df.describe()
```

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

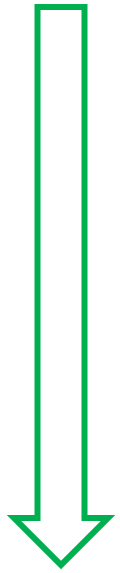
8 rows × 31 columns

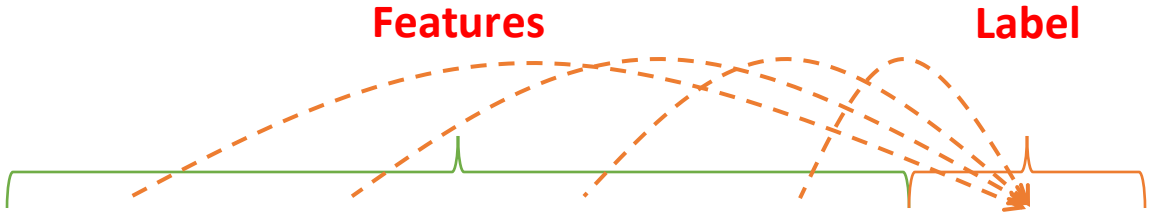
Card Fraud Detection

❖ Step 3: Split X, y

Independent Variable (Cause)

Dependent Variable (Result)





Index	Column 1	Column 2	Column 3	...	Column n
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

In our exericse's dataset:

- ❖ **Independent variables (features):** from column *1* to *n – 1*.
- ❖ **Depedent varaibles (labels):** last column.

Card Fraud Detection

❖ Step 3: Split X, y

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

```
1 dataset_arr = df.to_numpy()
2 X, y = dataset_arr[:, :-1].astype(np.float64), dataset_arr[:, -1].astype(np.uint8)
```

Card Fraud Detection

❖ Step 4: Add the bias term

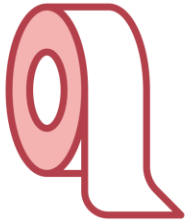
❖ Calculate z of class i'th:

$$z_i = \theta_i^T X = b\theta_{i0} + \sum_{j=1}^{n\_features} \theta_{ij} X_j$$

1. Ones Vector (n_elements = n_samples)

idx	0	1	...	n_samples - 1
value	1	1	1	1

2. Concat by column to independent variable



X variable

Bias
1
1
1
1

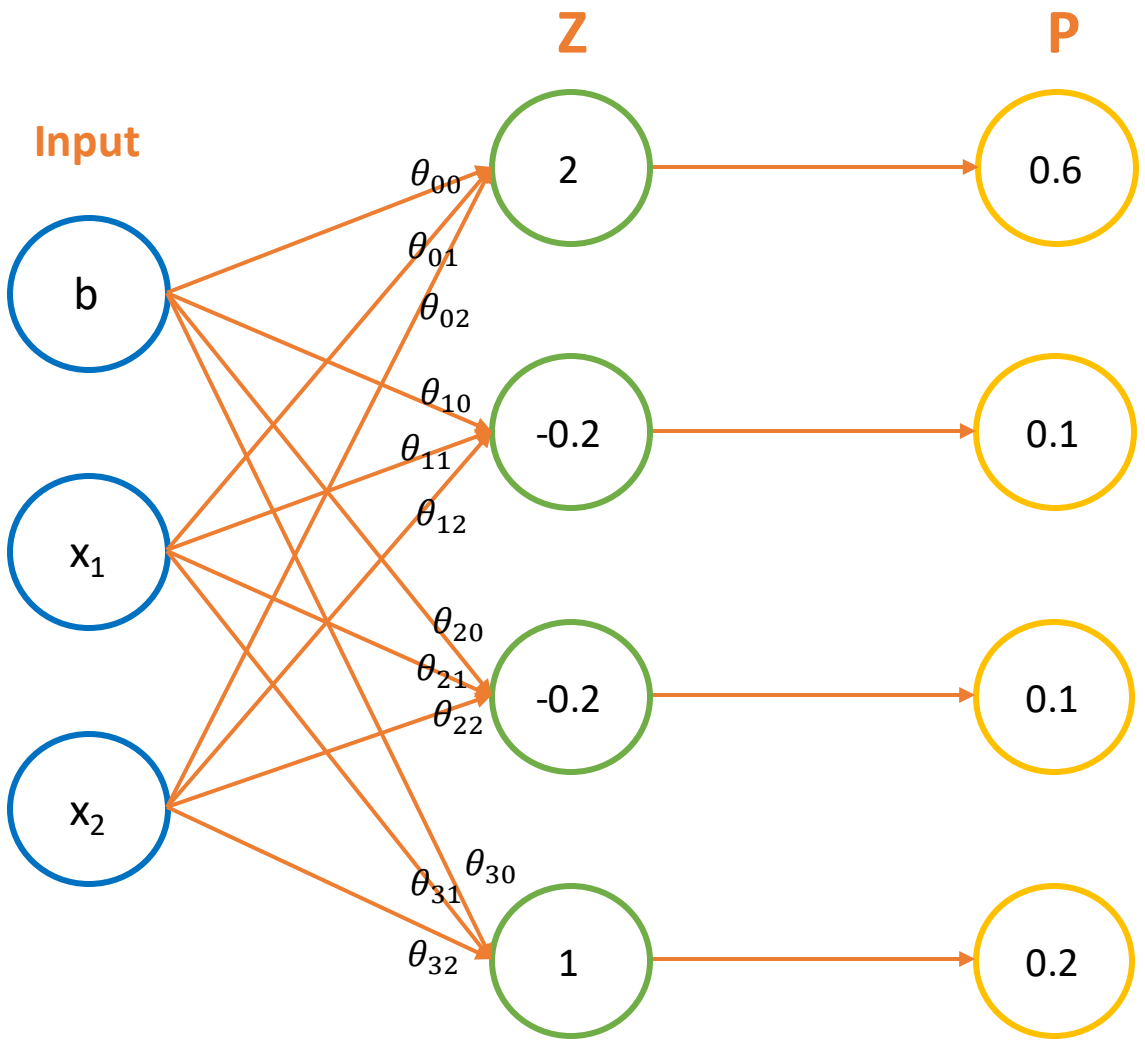
Feature 1	Feature 2	Feature 3	Feature 4
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

```
intercept = np.ones((
    X.shape[0], 1)
)
X_b = np.concatenate(
    (intercept, X),
    axis=1
)
```

Card Fraud Detection

❖ Step 5: One-hot encoding label

❖ **Recall:** Predict on sample i'th output an n elements vector



y_hat_i = [0.6, 0.1, 0.1, 0.2]

y_i = 0

```
1 y
```

```
array([0., 0., 0., ..., 0., 0., 0.])
```

How to calculate loss between y vs y_hat?

One hot encoding label

Card Fraud Detection

❖ Step 5: One-hot encoding label

Definition: Convert categorial data to one-hot vector

n_features = 2, **n_classes = 4**

1. Zeros Vector (n_elements = n_classes)

idx	0	1	2	3
value	0	0	0	0

2. E.g: y = 2

zeros_vector[y] = 1

idx	0	1	2	3
value	0	0	1	0

Label encoding

Label
0
0
1
3
2
1
2

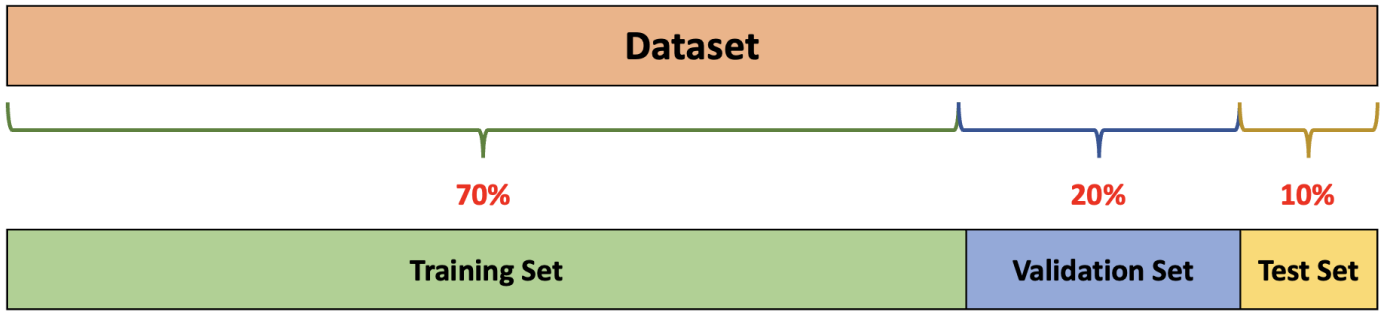
One-hot encoding

Label			
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	1	0	0
0	0	1	0

```
# One hot encoding label
y = np.array([np.zeros(n_classes) for _ in range(y_idx.shape[0])])
y[np.arange(len(y_idx)), y_idx] = 1
```

Card Fraud Detection

❖ Step 6: Split train, val, test set




```python
val_size = 0.3
test_size = 0.1
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X_b, y_encoded,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_val, X_test, y_val, y_test = train_test_split(
    X_val, y_val,
    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: 199364
Number of val samples: 76898
Number of test samples: 8545
```

Card Fraud Detection

❖ Step 7: Normalization

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

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

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

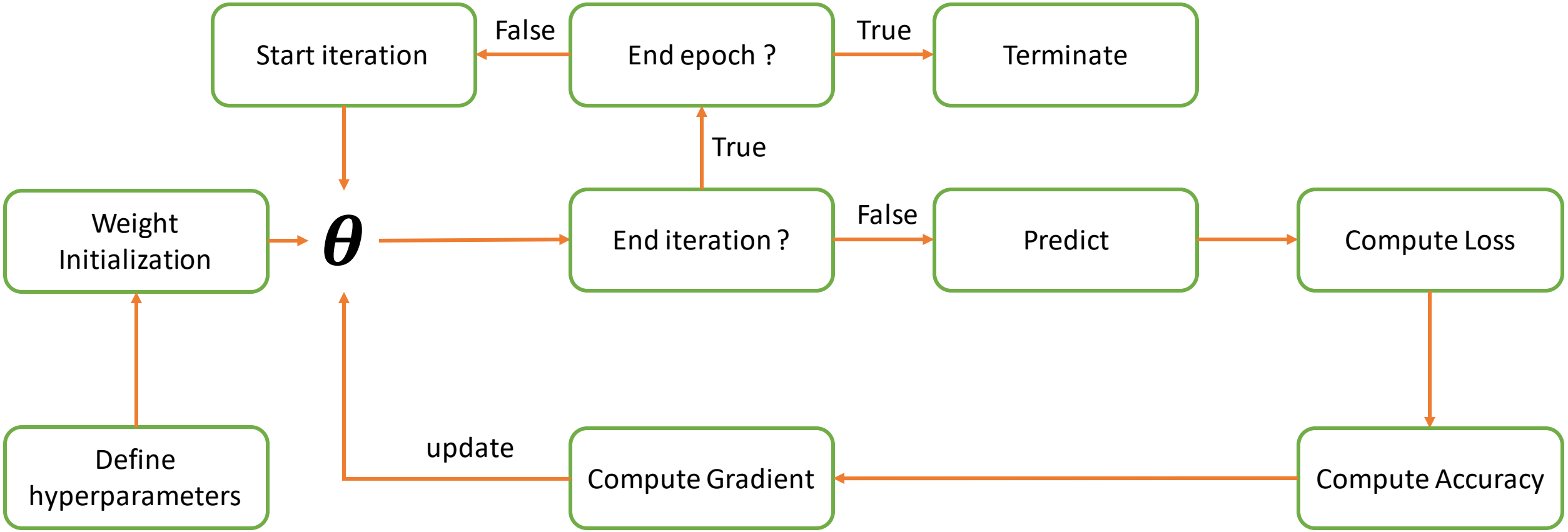
```
1 X_train
```

```
array([[ 1.        , -1.95311891, -0.76256074, ...,  0.79907219,
        -0.37273829, -0.32754448],
       [ 1.        ,  1.49473018,  0.96434558, ...,  0.119096  ,
        -0.11391761,  0.13119296],
       [ 1.        , -0.4733483 , -1.33737659, ...,  1.16380164,
        -0.46691221, -0.34433048],
       ...,
       [ 1.        , -0.61621632, -0.40746973, ..., -0.05352446,
         0.15422491, -0.04989603],
       [ 1.        ,  0.84222097,  1.03085144, ..., -0.02330856,
        -0.18601998, -0.30104658],
       [ 1.        , -0.57043285,  0.66268869, ...,  0.19155844,
         0.08433652, -0.34748785]])
```

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

Card Fraud Detection

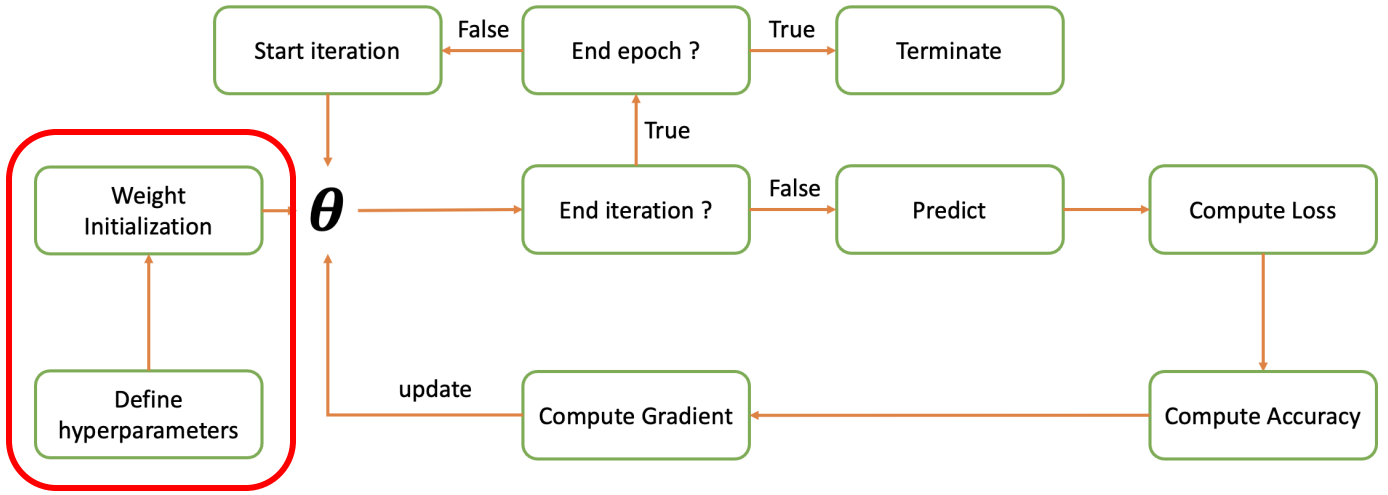
❖ Step 8: Training



Card Fraud Detection

❖ Step 8: Training

```
lr = 0.01
epochs = 30
batch_size = 1024
n_features = X_train.shape[1]

np.random.seed(random_state)
theta = np.random.uniform(
    size=(n_features, n_classes)
)
```



Define essential hyperparameters and initialize weights

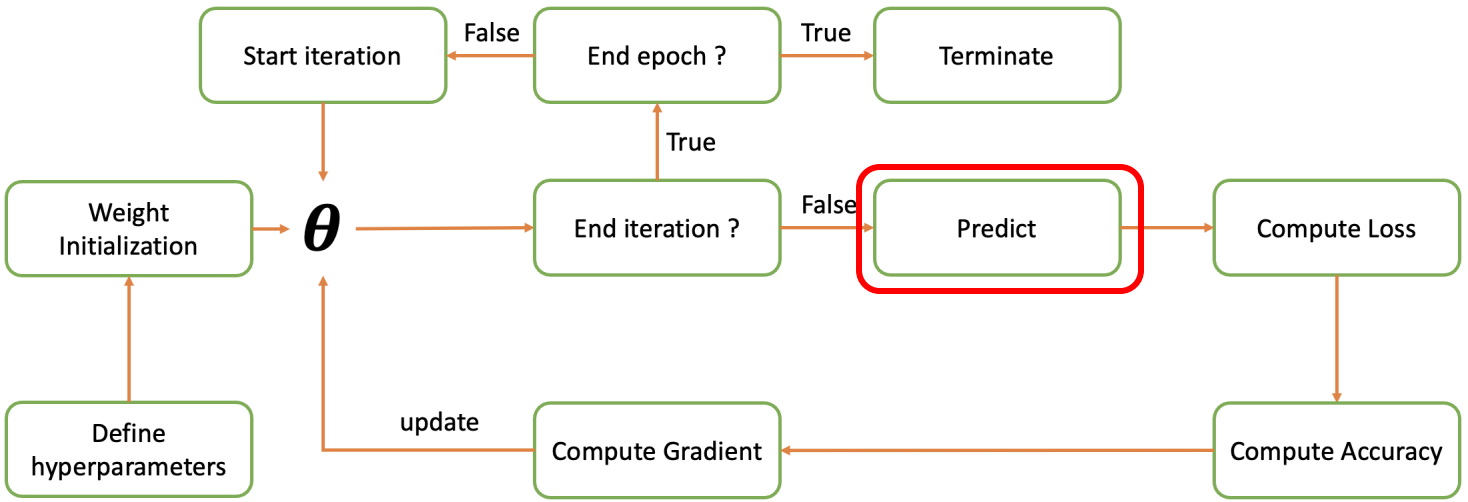
Card Fraud Detection

❖ Step 8: Training

Hypothesis function

- $z = b_0 + \sum_{i=1}^{n\_features} b_i x_i$
- $y_{hat} = softmax(z) = \frac{e^{z_i}}{\sum_{j=1}^{n\_classes} e^{z_j}}$

```
1  def softmax(z):
2      exp_z = np.exp(z)
3
4      return exp_z / exp_z.sum(axis=1)[:, None]
5
6
7  def predict(X, theta):
8      z = np.dot(X, theta)
9      y_hat = softmax(z)
10
11     return y_hat
```



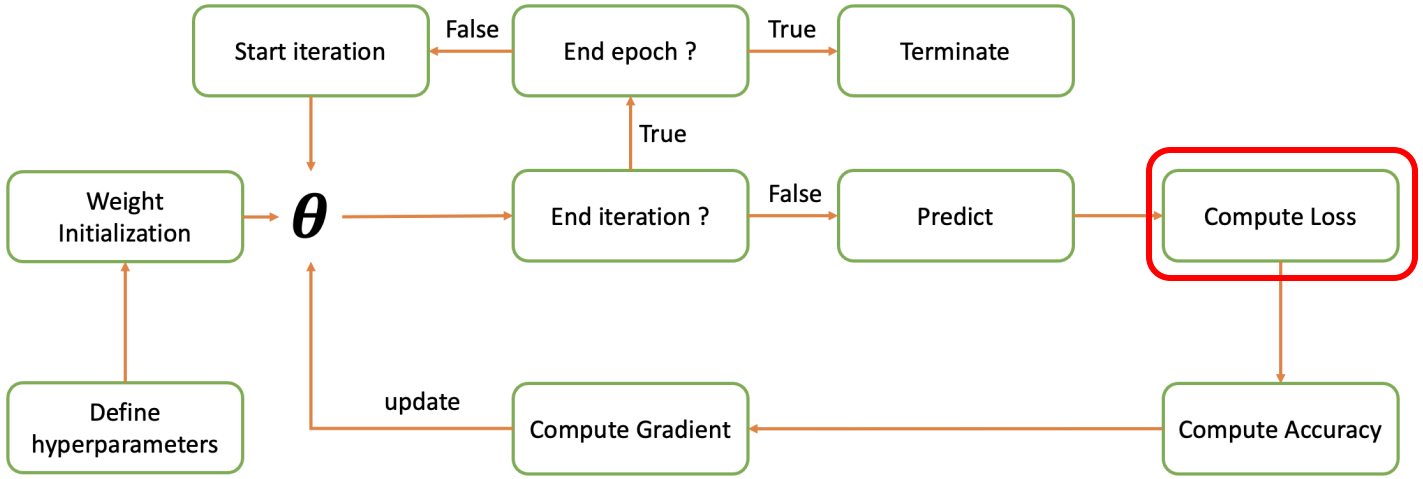
Card Fraud Detection

❖ Step 8: Training

Softmax Regression Loss (Cross-entropy)

$$L(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i log\hat{y}_i$$

```python
def compute_loss(y_hat, y):
    n = y.size

    return (-1 / n) * np.sum(y * np.log(y_hat))
```



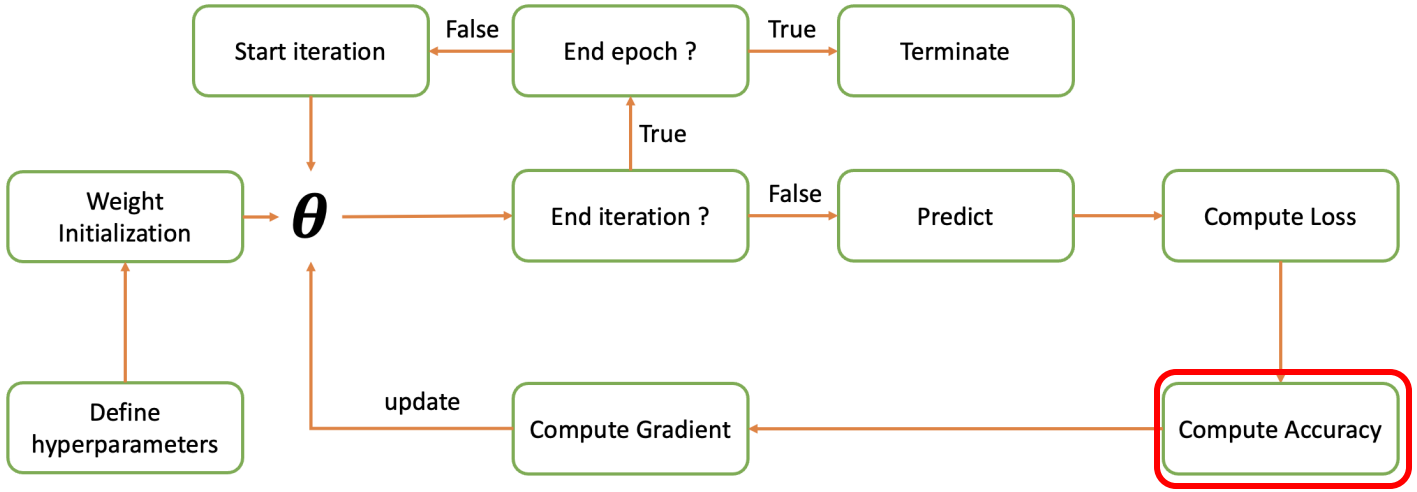
Card Fraud Detection

❖ Step 8: Training

Accuracy Formula

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

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta)
    max_y_hat = np.argmax(y_hat, axis=1)
    max_y = np.argmax(y, axis=1)
    acc = (max_y_hat == max_y).mean()

    return acc
```



Card Fraud Detection

❖ Step 8: Training

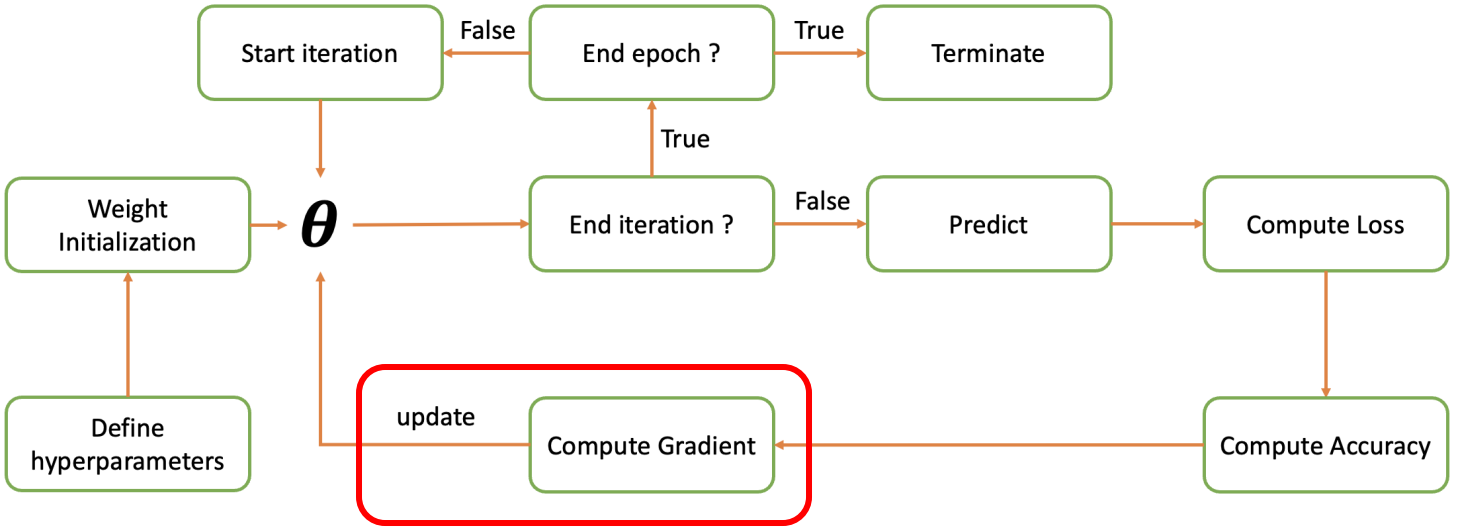
- **Gradient computation formula:**

$$\nabla_\theta L = \frac{1}{N} X^T (\hat{y} - y)$$

- **Weights update formula:**

$$\theta = \theta - \eta \nabla_\theta L$$

```python
def compute_gradient(X, y, y_hat):
    n = y.size

    return np.dot(X.T, (y_hat - y)) / n

def update_theta(theta, gradient, lr):
    return theta - lr * gradient
```



Card Fraud Detection

❖ Step 8: Training

Line 6: Loop over number of epochs.

Line 7, 8, 9, 10: Declare empty lists to stor batche accuracies, losses of train and val sets.

```
6  for epoch in range(epochs):
7      train_batch_losses = []
8      train_batch_accs = []
9      val_batch_losses = []
10     val_batch_accs = []
11
12     for i in range(0, X_train.shape[0], batch_size):
13         X_i = X_train[i:i+batch_size]
14         y_i = y_train[i:i+batch_size]
15
16         y_hat = predict(X_i, theta)
17         train_loss = compute_loss(y_hat, y_i)
18         gradient = compute_gradient(X_i, y_i, y_hat)
19         theta = update_theta(theta, gradient, lr)
20         train_batch_losses.append(train_loss)
21
22         train_acc = compute_accuracy(X_train, y_train, theta)
23         train_batch_accs.append(train_acc)
24
25         y_val_hat = predict(X_val, theta)
26         val_loss = compute_loss(y_val_hat, y_val)
27         val_batch_losses.append(val_loss)
28
29         val_acc = compute_accuracy(X_val, y_val, theta)
30         val_batch_accs.append(val_acc)
31
32     train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33     val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34     train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35     val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Card Fraud Detection

❖ Step 8: Training

Line 12: Loop over number of batches (based on batch size).

Line 13, 14: Get X and y data of current batch.

```
6  for epoch in range(epochs):
7      train_batch_losses = []
8      train_batch_accs = []
9      val_batch_losses = []
10     val_batch_accs = []
11
12     for i in range(0, X_train.shape[0], batch_size):
13         X_i = X_train[i:i+batch_size]
14         y_i = y_train[i:i+batch_size]
15
16         y_hat = predict(X_i, theta)
17         train_loss = compute_loss(y_hat, y_i)
18         gradient = compute_gradient(X_i, y_i, y_hat)
19         theta = update_theta(theta, gradient, lr)
20         train_batch_losses.append(train_loss)
21
22         train_acc = compute_accuracy(X_train, y_train, theta)
23         train_batch_accs.append(train_acc)
24
25         y_val_hat = predict(X_val, theta)
26         val_loss = compute_loss(y_val_hat, y_val)
27         val_batch_losses.append(val_loss)
28
29         val_acc = compute_accuracy(X_val, y_val, theta)
30         val_batch_accs.append(val_acc)
31
32     train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33     val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34     train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35     val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Card Fraud Detection

❖ Step 8: Training

Line 16: Do prediction on X_i and theta.

Line 17: Compute the loss of y_hat and y_i.

Line 18: Compute the gradient.

Line 19: Update theta using the computed gradient.

```
6  for epoch in range(epochs):
7      train_batch_losses = []
8      train_batch_accs = []
9      val_batch_losses = []
10     val_batch_accs = []
11
12     for i in range(0, X_train.shape[0], batch_size):
13         X_i = X_train[i:i+batch_size]
14         y_i = y_train[i:i+batch_size]
15
16         y_hat = predict(X_i, theta)
17         train_loss = compute_loss(y_hat, y_i)
18         gradient = compute_gradient(X_i, y_i, y_hat)
19         theta = update_theta(theta, gradient, lr)
20         train_batch_losses.append(train_loss)
21
22         train_acc = compute_accuracy(X_train, y_train, theta)
23         train_batch_accs.append(train_acc)
24
25         y_val_hat = predict(X_val, theta)
26         val_loss = compute_loss(y_val_hat, y_val)
27         val_batch_losses.append(val_loss)
28
29         val_acc = compute_accuracy(X_val, y_val, theta)
30         val_batch_accs.append(val_acc)
31
32     train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33     val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34     train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35     val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Card Fraud Detection

❖ Step 8: Training

From line 20 to line 30: Compute and store accuracies and losses on train and val sets.

```
6  for epoch in range(epochs):
7      train_batch_losses = []
8      train_batch_accs = []
9      val_batch_losses = []
10     val_batch_accs = []
11
12     for i in range(0, X_train.shape[0], batch_size):
13         X_i = X_train[i:i+batch_size]
14         y_i = y_train[i:i+batch_size]
15
16         y_hat = predict(X_i, theta)
17         train_loss = compute_loss(y_hat, y_i)
18         gradient = compute_gradient(X_i, y_i, y_hat)
19         theta = update_theta(theta, gradient, lr)
20         train_batch_losses.append(train_loss)
21
22         train_acc = compute_accuracy(X_train, y_train, theta)
23         train_batch_accs.append(train_acc)
24
25         y_val_hat = predict(X_val, theta)
26         val_loss = compute_loss(y_val_hat, y_val)
27         val_batch_losses.append(val_loss)
28
29         val_acc = compute_accuracy(X_val, y_val, theta)
30         val_batch_accs.append(val_acc)
31
32     train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33     val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34     train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35     val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Card Fraud Detection

❖ Step 8: Training

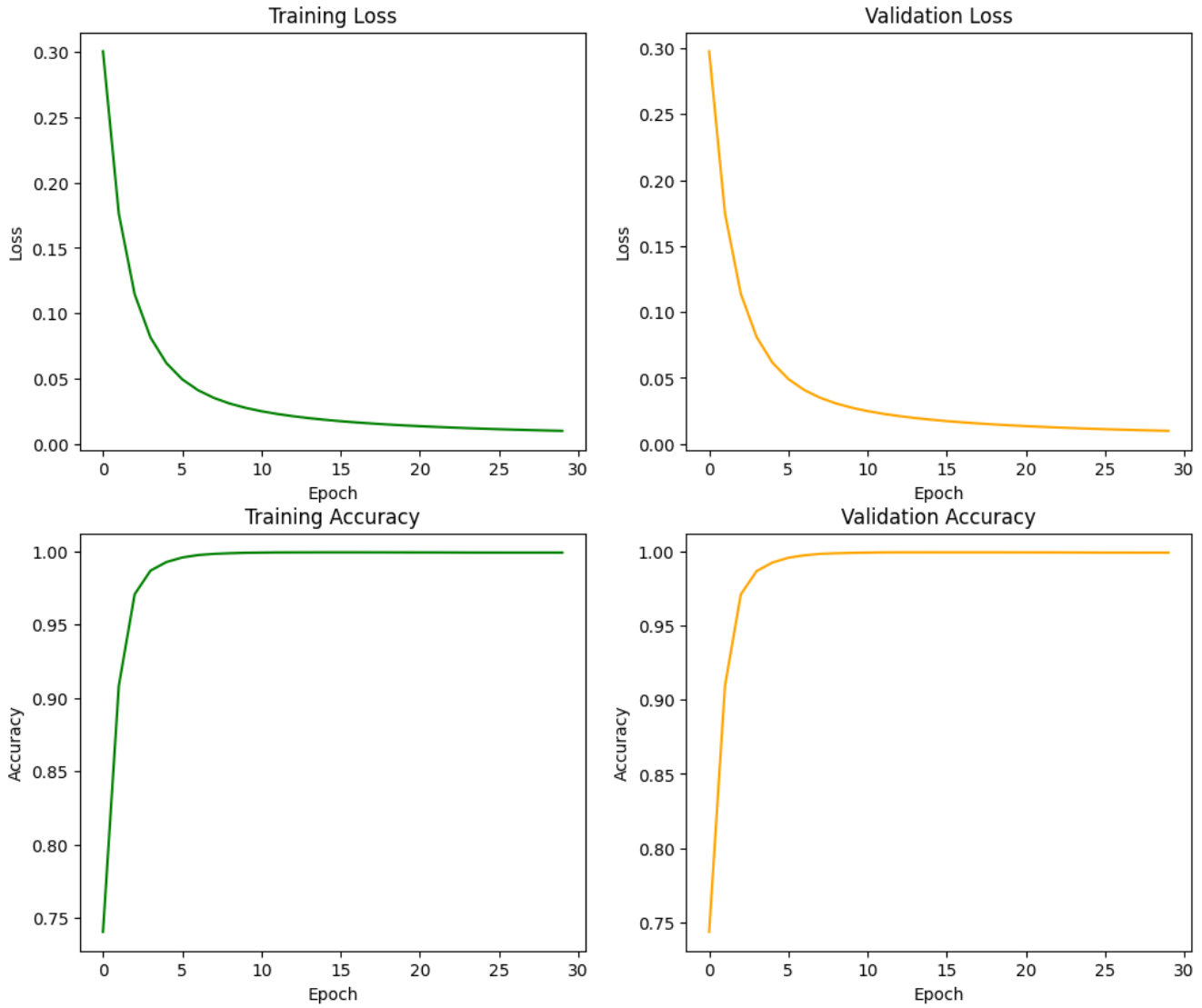
From line 32 to line 35: Compute and store batch accuracies and losses of train and val sets.

```
6  for epoch in range(epochs):
7      train_batch_losses = []
8      train_batch_accs = []
9      val_batch_losses = []
10     val_batch_accs = []
11
12     for i in range(0, X_train.shape[0], batch_size):
13         X_i = X_train[i:i+batch_size]
14         y_i = y_train[i:i+batch_size]
15
16         y_hat = predict(X_i, theta)
17         train_loss = compute_loss(y_hat, y_i)
18         gradient = compute_gradient(X_i, y_i, y_hat)
19         theta = update_theta(theta, gradient, lr)
20         train_batch_losses.append(train_loss)
21
22         train_acc = compute_accuracy(X_train, y_train, theta)
23         train_batch_accs.append(train_acc)
24
25         y_val_hat = predict(X_val, theta)
26         val_loss = compute_loss(y_val_hat, y_val)
27         val_batch_losses.append(val_loss)
28
29         val_acc = compute_accuracy(X_val, y_val, theta)
30         val_batch_accs.append(val_acc)
31
32     train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33     val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34     train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35     val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Card Fraud Detection

❖ Step 8: Training (Visualization)

```
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()
```



Card Fraud Detection

❖ Step 9: Evaluation

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

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta)
    max_y_hat = np.argmax(y_hat, axis=1)
    max_y = np.argmax(y, axis=1)
    acc = (max_y_hat == max_y).mean()

    return acc
```

```python
# Val set
val_set_acc = compute_accuracy(X_val, y_val, theta)
print('Evaluation on validation set:')
print(f'Accuracy: {val_set_acc}')
```

```
Evaluation on validation set:
Accuracy: 0.9992197456370777
```

```python
# Test set
test_set_acc = compute_accuracy(X_test, y_test, theta)
print('Evaluation on test set:')
print(f'Accuracy: {test_set_acc}')
```

```
Evaluation on test set:
Accuracy: 0.9994148624926857
```

Sentiment Analysis

❖ Introduction

Description: Given [Twitter Sentiment Analysis] dataset, build a Logistic Regression model to determine whether a tweet (text) has a positive, neutral or negative sentiment.

Positive

Neutral

Negative

Sentiment Analysis

❖ Introduction

In this exercise, we will implement Softmax Regression with two approaches: **NumPy** and **PyTorch.**

Approach 1: Build with NumPy

Approach 2: Build with PyTorch

Sentiment Analysis

❖ Step 1: Import libraries

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import re
5 import nltk
6 nltk.download('stopwords')
7
8 from sklearn.model_selection import train_test_split
9 from sklearn.feature_extraction.text import TfidfVectorizer
10 from nltk.corpus import stopwords
11 from nltk.stem import SnowballStemmer
```

Sentiment Analysis

❖ Step 2: Read dataset

```
1 dataset_path = 'Twitter_Data.csv'
2 df = pd.read_csv(
3     dataset_path
4 )
5 df
```

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0
...	...	...
162975	why these 456 crores paid neerav modi not reco...	-1.0
162976	dear rss terrorist payal gawar what about modi...	-1.0
162977	did you cover her interaction forum where she ...	0.0
162978	there big project came into india modi dream p...	0.0
162979	have you ever listen about like gurukul where ...	1.0

162980 rows × 2 columns

Sentiment Analysis

❖ Step 3: Get dataset information

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162980 entries, 0 to 162979
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   clean_text  162976 non-null  object
 1   category    162973 non-null  float64
dtypes: float64(1), object(1)
memory usage: 2.5+ MB
```

	category
count	162973.000000
mean	0.225436
std	0.781279
min	-1.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

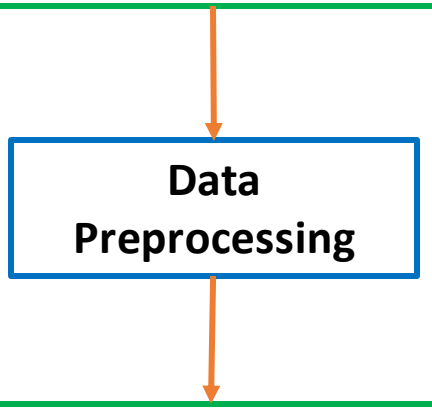
Sentiment Analysis

❖ Step 4: Data preprocessing

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0
...	...	...
162975	why these 456 crores paid neerav modi not reco...	-1.0
162976	dear rss terrorist payal gawar what about modi...	-1.0
162977	did you cover her interaction forum where she ...	0.0
162978	there big project came into india modi dream p...	0.0
162979	have you ever listen about like gurukul where ...	1.0

162980 rows × 2 columns

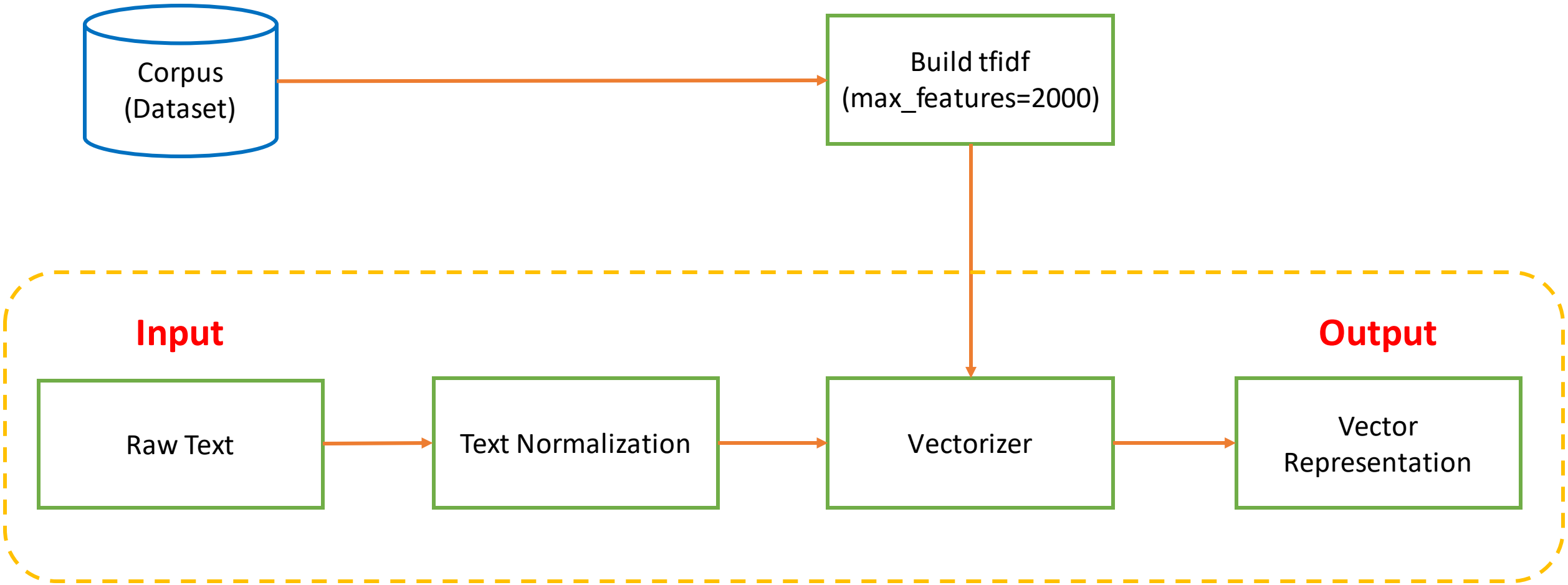
Raw Text: We love this! Would you go? #talk #makememories #unplug #relax #iphone #smartphone #wifi #connect… http://fb.me/6N3LsUpCu



Vector Representation: *array([1.0, 4768.0, 1425.0])*

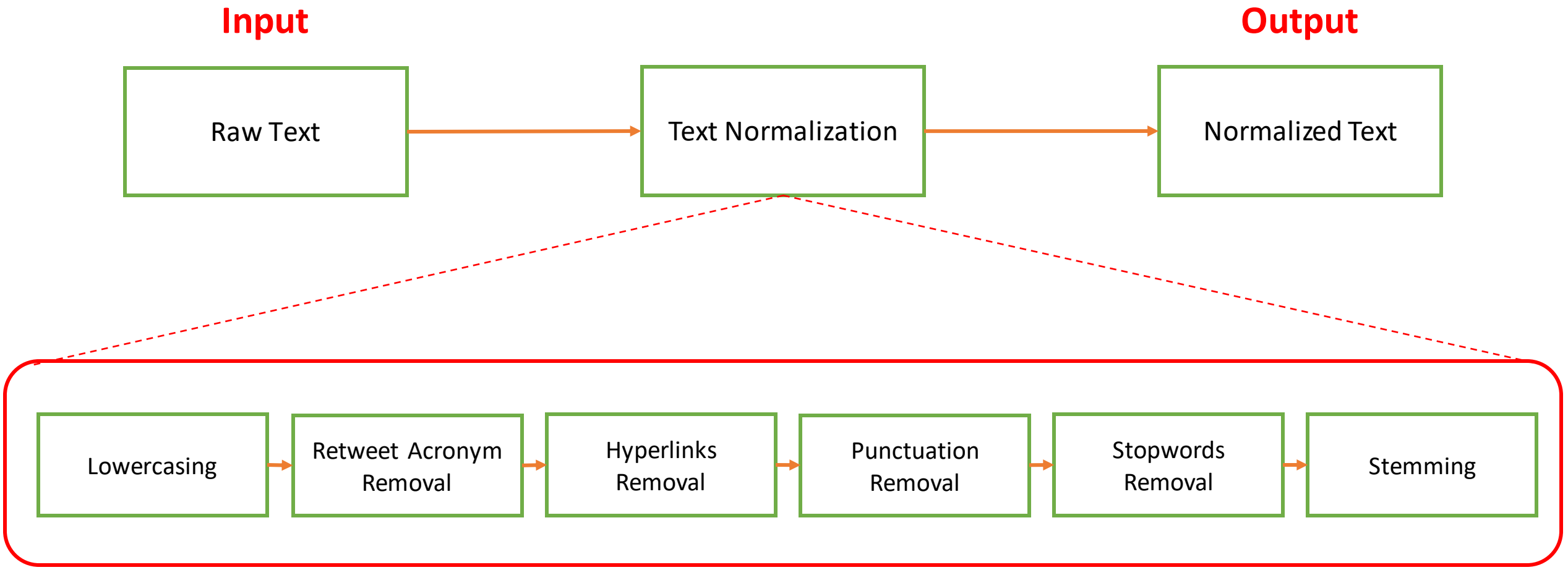
Sentiment Analysis

❖ Step 4: Data preprocessing



Sentiment Analysis

❖ Step 4: Data preprocessing



Sentiment Analysis

❖ Step 4: Data preprocessing

Lowercasing: Convert text into lowercase.

```
1 text = 'Hello WORLD'
2 text = text.lower()
3 print(text)
```

```
hello world
```

Retweet Acronym Removal: Remove "RT" at the start of the string

```
1 text = "RT Hello World"
2 text = re.sub(r'^RT[\s]+', '', text
3 print(text)
```

```
Hello World
```

Punctuation Removal: Remove all punctuations symbols "#".

```
1 text = "#xinchao, #hello, #konnichiwa"
2 text = re.sub(r'[^\w\s]', '', text)
3 print(text)
```

```
xinchao hello konnichiwa
```

Hyperlinks Removal: Remove all hyperlinks in the string.

```
1 text = "http://fb.me/6N3LsUpCu"
2 text = re.sub(
3     r'https?:\/\/.*[\r\n]*',
4     '',
5     text
6 )
7 print(text)
```

Sentiment Analysis

❖ Step 4: Data preprocessing

```
> stopwords("english")
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
```

```python
text = 'Hello world we are one'
stop_words = set(stopwords.words('english'))
words = text.split()
words = [
    word for word in words \
        if word not in stop_words
]
text = ' '.join(words)
print(text)
```

```
Hello world one
```

Stopwords Removal: Remove all “stopwords” which are not necessary to represent in vocabulary.

Sentiment Analysis

❖ Step 4: Data preprocessing

Snowball Stemmer Rules

Ends with		Reduced to		
ILY	→	ILI	easily	→ easili
LY	→		fairly	→ fair
SS	→	SS	caress	→ caress
S	→		bats	→ bat
ING	→		singing	→ sing
ED	→	E	shared	→ share

```
1 text = 'I am eating breakfast'
2 stemmer = SnowballStemmer('english')
3 words = text.split()
4 words = [stemmer.stem(word) for word in words]
5 text = ' '.join(words)
6 print(text)
```

```
i am eat breakfast
```

Stemming: Convert all the words into its root form (only use in English or similar languages).

Sentiment Analysis

❖ Step 4: Data preprocessing

```python
def text_normalize(text):
    # Lowercasing
    text = text.lower()

    # Retweet old acronym "RT" removal
    text = re.sub(r'^rt[\s]+', '', text)

    # Hyperlinks removal
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text)

    # Punctuation removal
    text = re.sub(r'[^\w\s]', '', text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = text.split()
    words = [word for word in words if word not in stop_words]
    text = ' '.join(words)

    # Stemming
    stemmer = SnowballStemmer('english')
    words = text.split()
    words = [stemmer.stem(word) for word in words]
    text = ' '.join(words)

    return text
```

```python
text = """We love this! Would you go?
#talk #makememories #unplug
#relax #iphone #smartphone #wifi #connect...
http://fb.me/6N3LsUpCu
"""
text = text_normalize(text)
text
```

```
'love would go talk makememori unplug relax iphon smartphon wifi connect'
```

Sentiment Analysis

❖ Step 4: Data preprocessing

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

To represent normalized text as vector, we will use **TfIdf vectorizer** from sklearn.

```
vectorizer = TfidfVectorizer(
    max_features=2000
)
X = vectorizer.fit_transform(
    df['clean_text']
).toarray()
```

Sentiment Analysis

❖ Step 4: Data preprocessing

```
intercept = np.ones((
    X.shape[0], 1)
)
X_b = np.concatenate(
    (intercept, X),
    axis=1
)
```

We also add the bias term into each TfIdf vector.

```
1 intercept
```

```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
```

```
1 X
```

```
array([[ 3.   ,  0.
       [ 1.   ,  1.
       [ 3.   ,  1.
       ...,
       [ 3.   ,  1.
       [ 1.   ,  0.
       [ 3.   ,  0.
```

```
1 X_b
```

```
array([[ 1.   ,  3.
       [ 1.   ,  1.
       [ 1.   ,  3.
       ...,
       [ 1.   ,  3.
       [ 1.   ,  1.
       [ 1.   ,  3.
```

Sentiment Analysis

❖ Step 5: One-hot encode label

Label encoding

Label
0
0
1
3
2
1
2

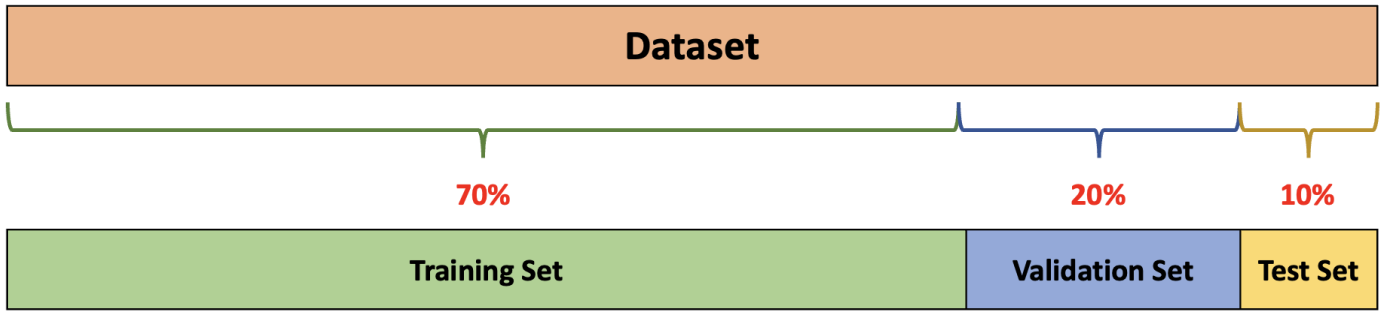
One-hot encoding

Label			
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	1	0	0
0	0	1	0

```python
n_classes = df['category'].nunique()
n_samples = df['category'].size

y = df['category'].to_numpy() + 1
y = y.astype(np.uint8)
y_encoded = np.array(
    [np.zeros(n_classes) for _ in range(n_samples)]
)
y_encoded[np.arange(n_samples), y] = 1
```

Sentiment Analysis

❖ Step 6: Split train, val, test set



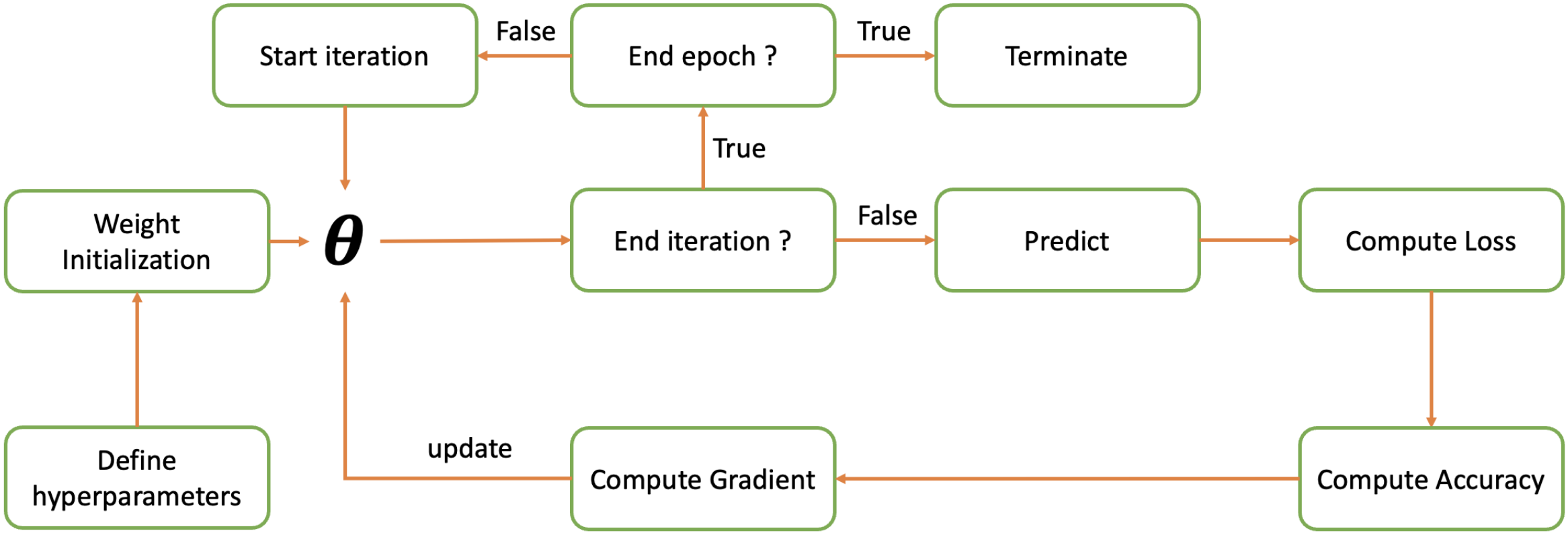

```python
val_size = 0.3
test_size = 0.1
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X_b, y_encoded,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_val, X_test, y_val, y_test = train_test_split(
    X_val, y_val,
    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: 114078
Number of val samples: 44001
Number of test samples: 4890
```

Sentiment Analysis

❖ Step 7: Training



Sentiment Analysis

❖ Step 7: Training

```python
lr = 0.1
epochs = 200
batch_size = X_train.shape[0]
n_features = X_train.shape[1]

np.random.seed(random_state)
theta = np.random.uniform(
    size=(n_features, n_classes)
)
```

Define essential functions and hyperparameters

```python
def softmax(z):
    exp_z = np.exp(z)

    return exp_z / exp_z.sum(axis=1)[:, None]

def compute_loss(y_hat, y):
    n = y.size

    return (-1 / n) * np.sum(y * np.log(y_hat))

def predict(X, theta):
    z = np.dot(X, theta)
    y_hat = softmax(z)

    return y_hat

def compute_gradient(X, y, y_hat):
    n = y.size

    return np.dot(X.T, (y_hat - y)) / n

def update_theta(theta, gradient, lr):
    return theta - lr * gradient

def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta)
    acc = (np.argmax(y_hat, axis=1) == np.argmax(y, axis=1)).mean()

    return acc
```

Sentiment Analysis

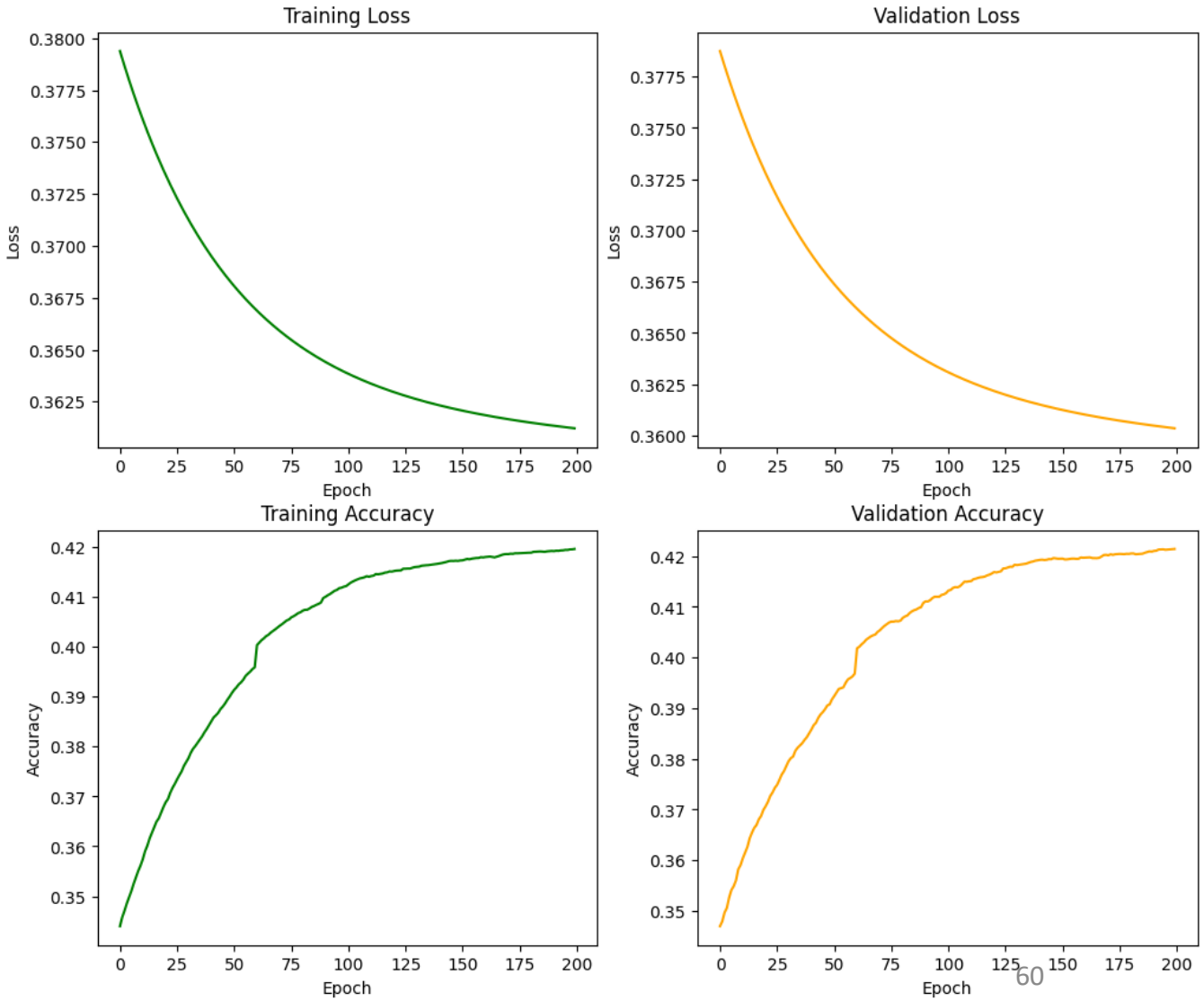
❖ Step 7: Training

```python
for epoch in range(epochs):
    train_batch_losses = []
    train_batch_accs = []
    val_batch_losses = []
    val_batch_accs = []

    for i in range(0, X_train.shape[0], batch_size):
        X_i = X_train[i:i+batch_size]
        y_i = y_train[i:i+batch_size]

        y_hat = predict(X_i, theta)
        train_loss = compute_loss(y_hat, y_i)
        gradient = compute_gradient(X_i, y_i, y_hat)
        theta = update_theta(theta, gradient, lr)
        train_batch_losses.append(train_loss)

        train_acc = compute_accuracy(X_train, y_train, theta)
        train_batch_accs.append(train_acc)

        y_val_hat = predict(X_val, theta)
        val_loss = compute_loss(y_val_hat, y_val)
        val_batch_losses.append(val_loss)

        val_acc = compute_accuracy(X_val, y_val, theta)
        val_batch_accs.append(val_acc)

    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Sentiment Analysis

❖ Step 7: Training (Visualization)

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

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

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

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

plt.show()
```



Sentiment Analysis

❖ Step 8: Evaluation

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

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```

```python
# Val set
val_set_acc = compute_accuracy(X_val, y_val, theta)
print('Evaluation on validation set:')
print(f'Accuracy: {val_set_acc}')
```

```
Evaluation on validation set:
Accuracy: 0.4214222404036272
```

```python
# Test set
test_set_acc = compute_accuracy(X_test, y_test, theta)
print('Evaluation on test set:')
print(f'Accuracy: {test_set_acc}')
```

```
Evaluation on test set:
Accuracy: 0.416359918200409
```

Sentiment Analysis

❖ Implement with PyTorch: Step 1

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import re
5 import torch
6 import torch.nn as nn
7 import torch.optim as optim
8 import nltk
9 nltk.download('stopwords')
10
11 from sklearn.model_selection import train_test_split
12 from sklearn.feature_extraction.text import TfidfVectorizer
13 from nltk.corpus import stopwords
14 from nltk.stem import SnowballStemmer
```

Sentiment Analysis

❖ Implement with PyTorch: Step 2

```
dataset_path = 'Twitter_Data.csv'
df = pd.read_csv(
    dataset_path
)
df
```

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0
...	...	...
162975	why these 456 crores paid neerav modi not reco...	-1.0
162976	dear rss terrorist payal gawar what about modi...	-1.0
162977	did you cover her interaction forum where she ...	0.0
162978	there big project came into india modi dream p...	0.0
162979	have you ever listen about like gurukul where ...	1.0

162980 rows × 2 columns

Sentiment Analysis

❖ Implement with PyTorch: Step 3

```
1 null_rows = df.isnull().any(axis=1)
2 df[null_rows]
```

	clean_text	category
148	NaN	0.0
130448	the foundation stone northeast gas grid inaugu...	NaN
155642	dear terrorists you can run but you cant hide ...	NaN
155698	offense the best defence with mission shakti m...	NaN
155770	have always heard politicians backing out thei...	NaN
158693	modi government plans felicitate the faceless ...	NaN
158694	NaN	-1.0
159442	chidambaram gives praises modinomics	NaN
159443	NaN	0.0
160559	the reason why modi contested from seats 2014 ...	NaN
160560	NaN	1.0

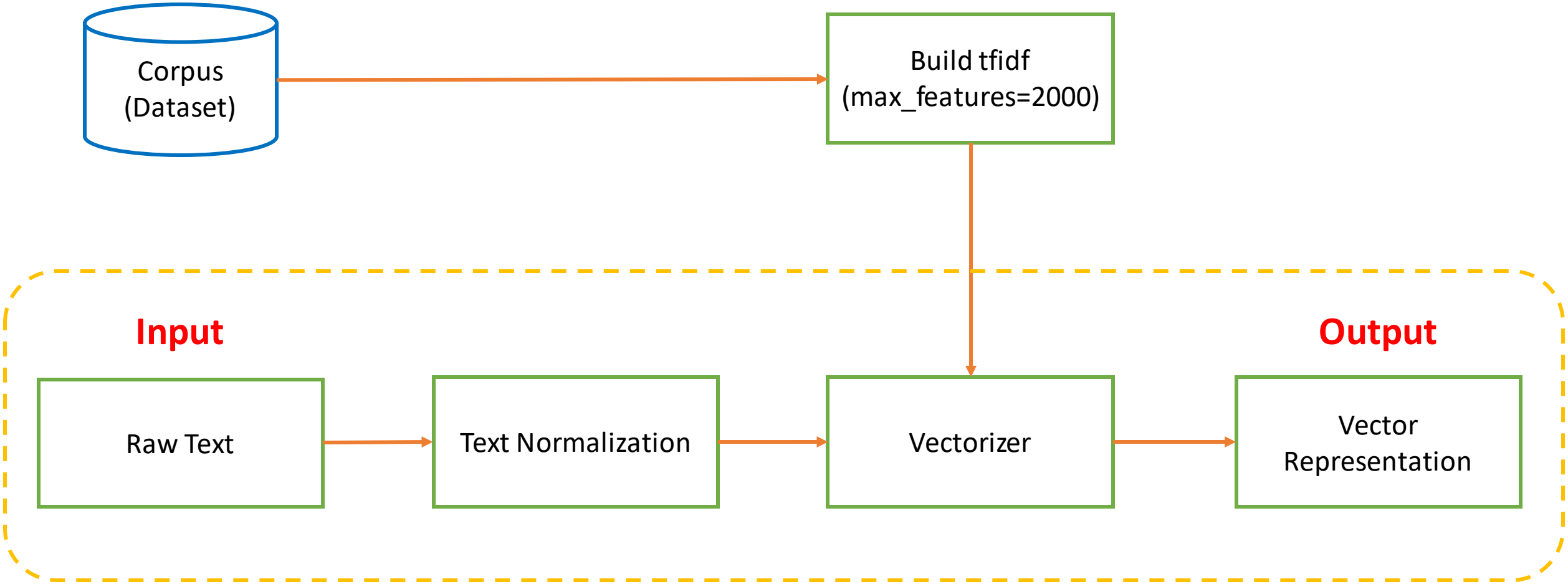
```
1 df = df.dropna()
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 162969 entries, 0 to 162979
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   clean_text  162969 non-null  object
 1   category    162969 non-null  float64
dtypes: float64(1), object(1)
memory usage: 3.7+ MB
```

Sentiment Analysis

❖ Implement with PyTorch: Step 4



Sentiment Analysis

❖ Implement with PyTorch: Step 4

```python
def text_normalize(text):
    # Lowercasing
    text = text.lower()

    # Retweet old acronym "RT" removal
    text = re.sub(r'^rt[\s]+', '', text)

    # Hyperlinks removal
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text)

    # Punctuation removal
    text = re.sub(r'[^\w\s]', '', text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = text.split()
    words = [word for word in words if word not in stop_words]
    text = ' '.join(words)

    # Stemming
    stemmer = SnowballStemmer('english')
    words = text.split()
    words = [stemmer.stem(word) for word in words]
    text = ' '.join(words)

    return text
```

```python
df['clean_text'] = df['clean_text'].apply(
    lambda x: text_normalize(x)
)
```

```python
vectorizer = TfidfVectorizer(
    max_features=2000
)
X = vectorizer.fit_transform(
    df['clean_text']
).toarray()
```

Sentiment Analysis

❖ Implement with PyTorch: Step 5

Label encoding

Label
0
0
1
3
2
1
2

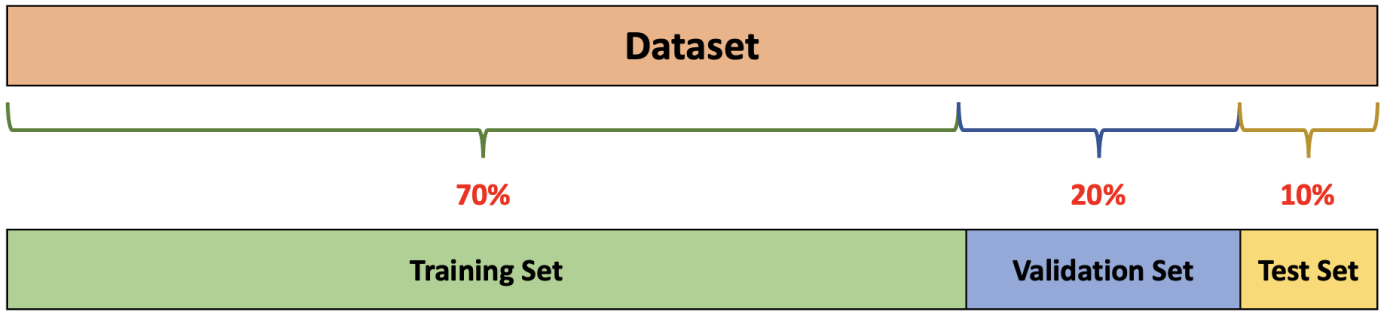
One-hot encoding

Label			
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	1	0	0
0	0	1	0

```python
n_classes = df['category'].nunique()
n_samples = df['category'].size

y = df['category'].to_numpy() + 1
y = y.astype(np.uint8)
y_encoded = np.array(
    [np.zeros(n_classes) for _ in range(n_samples)]
)
y_encoded[np.arange(n_samples), y] = 1
```

Sentiment Analysis

❖ Implement with PyTorch: Step 6




```python
val_size = 0.3
test_size = 0.1
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X_b, y_encoded,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_val, X_test, y_val, y_test = train_test_split(
    X_val, y_val,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

```python
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y_encoded, dtype=torch.float32)
```

```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: 114078
Number of val samples: 44001
Number of test samples: 4890
```

Sentiment Analysis

❖ Implement with PyTorch: Step 7

Create a SoftmaxRegression class that inherits from nn.Module:

```
1  class SoftmaxRegression(nn.Module):
2      def __init__(self, input_dim, output_dim):
3          super(SoftmaxRegression, self).__init__()
4          self.linear = nn.Linear(
5              input_dim,
6              output_dim,
7              bias=True
8          )
9
10     def forward(self, x):
11         return self.linear(x)
```

Applies a linear transformation to the incoming data: $y = xA^T + b$

This module supports TensorFloat32.

On certain ROCm devices, when using float16 inputs this module will use different precision for backward.

Parameters

- **in_features** (*int*) – size of each input sample
- **out_features** (*int*) – size of each output sample
- **bias** (*bool*) – If set to `False`, the layer will not learn an additive bias. Default: `True`

We define a linear transformation, which is just the dot product (remember to set bias=True to include the bias term).

Sentiment Analysis

❖ Implement with PyTorch: Step 8

```
lr = 0.1
epochs = 500
torch.manual_seed(random_state)
if torch.cuda.is_available():
    torch.cuda.manual_seed(random_state)

input_dim = X_train.shape[1]
output_dim = y_train.shape[1]

model = SoftmaxRegression(
    input_dim, output_dim
)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(
    model.parameters(), lr=lr
)
```

We define the loss function (called criterion in PyTorch) using Cross-entropy and optimizer algorithm (in this case is SGD).

Sentiment Analysis

❖ Implement with PyTorch: Step 8

```
1 train_losses = []
2 train_accs = []
3 val_losses = []
4 val_accs = []
5
6 for epoch in range(epochs):
7     model.train()
8
9     # Zero the gradients
10    optimizer.zero_grad()
11
12    # Forward pass
13    y_hat = model(X_train)
14
15    # Compute loss
16    train_loss = criterion(y_hat, y_train)
17
18    train_losses.append(train_loss.item())
19
20    train_acc = compute_accuracy(y_hat, y_train)
21    train_accs.append(train_acc)
22
23    # Backward pass and optimization
24    train_loss.backward()
25    optimizer.step()
26
27    model.eval()
28    # Forward pass for validation data
29    with torch.no_grad():
30        y_val_hat = model(X_val)
31
32        # Compute validation loss
33        val_loss = criterion(y_val_hat, y_val)
34        val_losses.append(val_loss.item())
35
36        val_acc = compute_accuracy(y_val_hat, y_val)
37        val_accs.append(val_acc)
38
39    print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\tValidation loss: {val_loss:.3f}')
```

Line 1, 2, 3, 4: Define empty lists for storing losses and accuracies of train and val sets in each epoch.

Sentiment Analysis

❖ Implement with PyTorch: Step 8

Line 9, 10: Zero out the gradients of previous epoch before any computations.

```
1  train_losses = []
2  train_accs = []
3  val_losses = []
4  val_accs = []
5
6  for epoch in range(epochs):
7      model.train()
8
9      # Zero the gradients
10     optimizer.zero_grad()
11
12     # Forward pass
13     y_hat = model(X_train)
14
15     # Compute loss
16     train_loss = criterion(y_hat, y_train)
17
18     train_losses.append(train_loss.item())
19
20     train_acc = compute_accuracy(y_hat, y_train)
21     train_accs.append(train_acc)
22
23     # Backward pass and optimization
24     train_loss.backward()
25     optimizer.step()
26
27     model.eval()
28     # Forward pass for validation data
29     with torch.no_grad():
30         y_val_hat = model(X_val)
31
32         # Compute validation loss
33         val_loss = criterion(y_val_hat, y_val)
34         val_losses.append(val_loss.item())
35
36         val_acc = compute_accuracy(y_val_hat, y_val)
37         val_accs.append(val_acc)
38
39     print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\tValidation loss: {val_loss:.3f}')
```

Sentiment Analysis

❖ Implement with PyTorch: Step 8

```
1  train_losses = []
2  train_accs = []
3  val_losses = []
4  val_accs = []
5
6  for epoch in range(epochs):
7      model.train()
8
9      # Zero the gradients
10     optimizer.zero_grad()
11
12     # Forward pass
13     y_hat = model(X_train)
14
15     # Compute loss
16     train_loss = criterion(y_hat, y_train)
17
18     train_losses.append(train_loss.item())
19
20     train_acc = compute_accuracy(y_hat, y_train)
21     train_accs.append(train_acc)
22
23     # Backward pass and optimization
24     train_loss.backward()
25     optimizer.step()
26
27     model.eval()
28     # Forward pass for validation data
29     with torch.no_grad():
30         y_val_hat = model(X_val)
31
32         # Compute validation loss
33         val_loss = criterion(y_val_hat, y_val)
34         val_losses.append(val_loss.item())
35
36         val_acc = compute_accuracy(y_val_hat, y_val)
37         val_accs.append(val_acc)
38
39     print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\tValidation loss: {val_loss:.3f}')
```

Line 12, 13: Do prediction on all data points of current epoch.

Sentiment Analysis

❖ Implement with PyTorch: Step 8

Line 15, 16: Calculating the training loss and append the result to train_losses list.
Line 20, 21: Calculate the accuracy on the train set.

```
1  train_losses = []
2  train_accs = []
3  val_losses = []
4  val_accs = []
5
6  for epoch in range(epochs):
7      model.train()
8
9      # Zero the gradients
10     optimizer.zero_grad()
11
12     # Forward pass
13     y_hat = model(X_train)
14
15     # Compute loss
16     train_loss = criterion(y_hat, y_train)
17
18     train_losses.append(train_loss.item())
19
20     train_acc = compute_accuracy(y_hat, y_train)
21     train_accs.append(train_acc)
22
23     # Backward pass and optimization
24     train_loss.backward()
25     optimizer.step()
26
27     model.eval()
28     # Forward pass for validation data
29     with torch.no_grad():
30         y_val_hat = model(X_val)
31
32         # Compute validation loss
33         val_loss = criterion(y_val_hat, y_val)
34         val_losses.append(val_loss.item())
35
36         val_acc = compute_accuracy(y_val_hat, y_val)
37         val_accs.append(val_acc)
38
39     print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\tValidation loss: {val_loss:.3f}')
```

Sentiment Analysis

❖ Implement with PyTorch: Step 8

Line 23, 24, 25: Compute the gradient and update the weights.

```
1  train_losses = []
2  train_accs = []
3  val_losses = []
4  val_accs = []
5
6  for epoch in range(epochs):
7      model.train()
8
9      # Zero the gradients
10     optimizer.zero_grad()
11
12     # Forward pass
13     y_hat = model(X_train)
14
15     # Compute loss
16     train_loss = criterion(y_hat, y_train)
17
18     train_losses.append(train_loss.item())
19
20     train_acc = compute_accuracy(y_hat, y_train)
21     train_accs.append(train_acc)
22
23     # Backward pass and optimization
24     train_loss.backward()
25     optimizer.step()
26
27     model.eval()
28     # Forward pass for validation data
29     with torch.no_grad():
30         y_val_hat = model(X_val)
31
32         # Compute validation loss
33         val_loss = criterion(y_val_hat, y_val)
34         val_losses.append(val_loss.item())
35
36         val_acc = compute_accuracy(y_val_hat, y_val)
37         val_accs.append(val_acc)
38
39     print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\tValidation loss: {val_loss:.3f}')
```

Sentiment Analysis

❖ Implement with PyTorch: Step 8

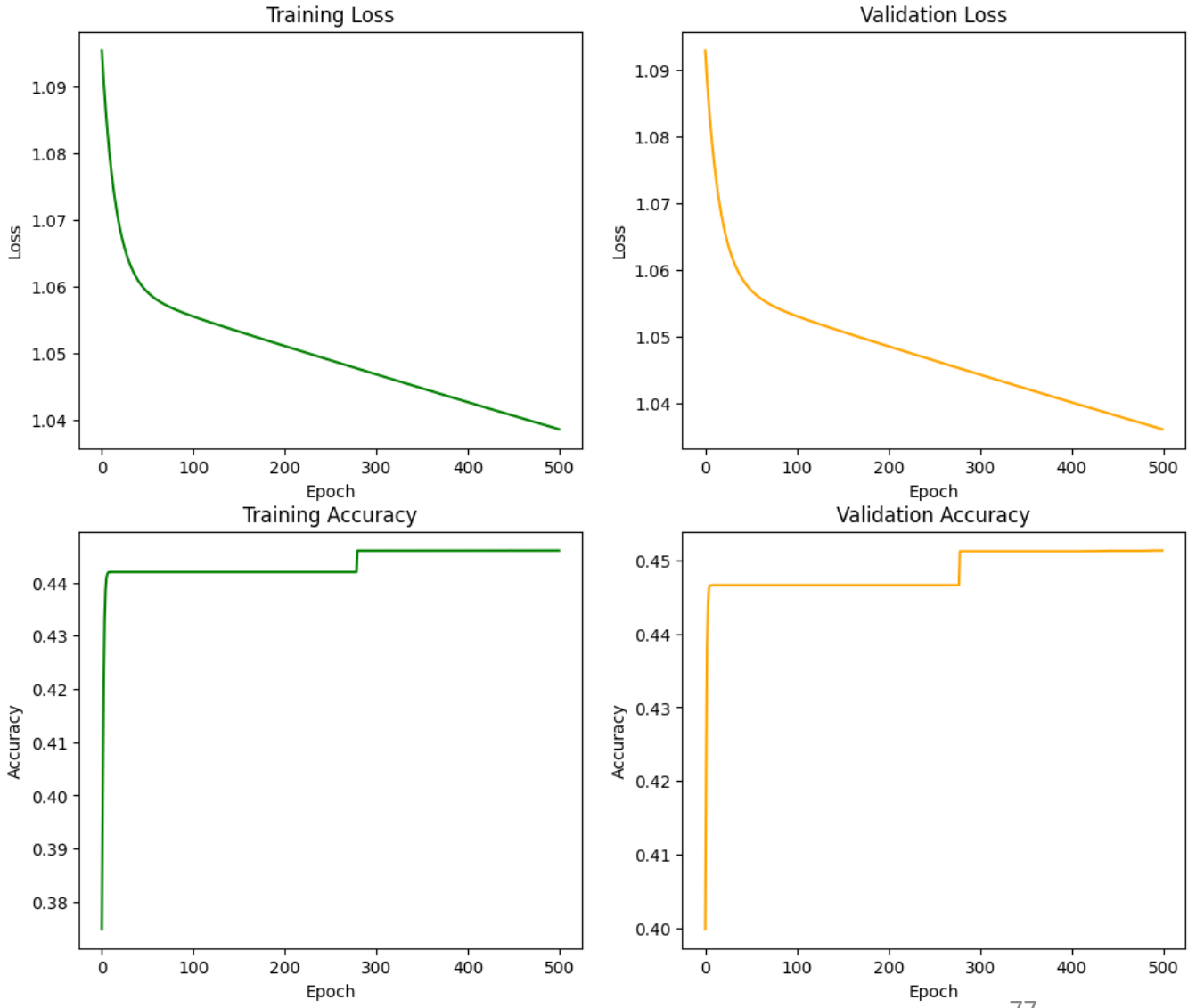
From line 27 to line 37: Change the model to evaluation mode and calculate the loss and accuracy on val set. Then append those scores into according lists.

```
1  train_losses = []
2  train_accs = []
3  val_losses = []
4  val_accs = []
5
6  for epoch in range(epochs):
7      model.train()
8
9      # Zero the gradients
10     optimizer.zero_grad()
11
12     # Forward pass
13     y_hat = model(X_train)
14
15     # Compute loss
16     train_loss = criterion(y_hat, y_train)
17
18     train_losses.append(train_loss.item())
19
20     train_acc = compute_accuracy(y_hat, y_train)
21     train_accs.append(train_acc)
22
23     # Backward pass and optimization
24     train_loss.backward()
25     optimizer.step()
26
27     model.eval()
28     # Forward pass for validation data
29     with torch.no_grad():
30         y_val_hat = model(X_val)
31
32         # Compute validation loss
33         val_loss = criterion(y_val_hat, y_val)
34         val_losses.append(val_loss.item())
35
36         val_acc = compute_accuracy(y_val_hat, y_val)
37         val_accs.append(val_acc)
38
39     print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\tValidation loss: {val_loss:.3f}')
```

Sentiment Analysis

❖ Implement with PyTorch: Step 9

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

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

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

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

plt.show()
```



Sentiment Analysis

❖ Implement with PyTorch: Step 10

```python
def compute_accuracy(y_hat, y_true):
    _, y_hat = torch.max(y_hat, dim=1)
    _, y_true = torch.max(y_true, dim=1)

    correct = (y_hat == y_true).sum().item()

    accuracy = (correct / len(y_true))

    return accuracy
```

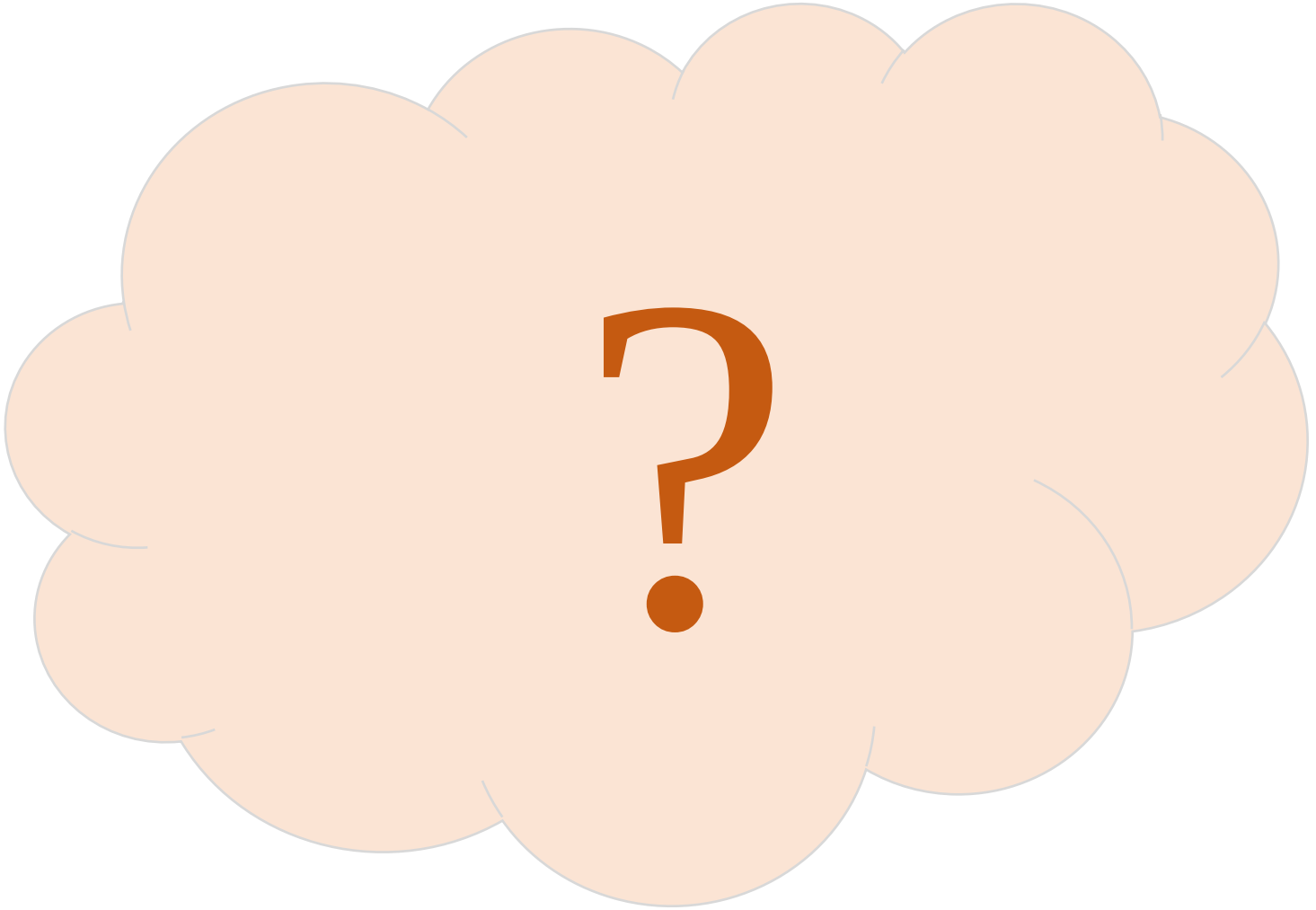
```python
# Val set
model.eval()
with torch.no_grad():
    y_hat = model(X_val)
    val_set_acc = compute_accuracy(y_hat, y_val)
    print('Evaluation on validation set:')
    print(f'Accuracy: {val_set_acc}')
```

```
Evaluation on validation set:
Accuracy: 0.4512851980636804
```

```python
# Test set
model.eval()
with torch.no_grad():
    y_hat = model(X_test)
    test_set_acc = compute_accuracy(y_hat, y_test)
    print('Evaluation on test set:')
    print(f'Accuracy: {test_set_acc}')
```

```
Evaluation on test set:
Accuracy: 0.4494887525562372
```

Question




Thank you!