Math for Computer Science

Logistic Regression, Neural Network & Image Classification

Quan Minh Phan & Ngoc Hoang Luong

University of Information Technology
-
Vietnam National University Ho Chi Minh City

December 3, 2022

Table of contents

What is an Image Classification problem?

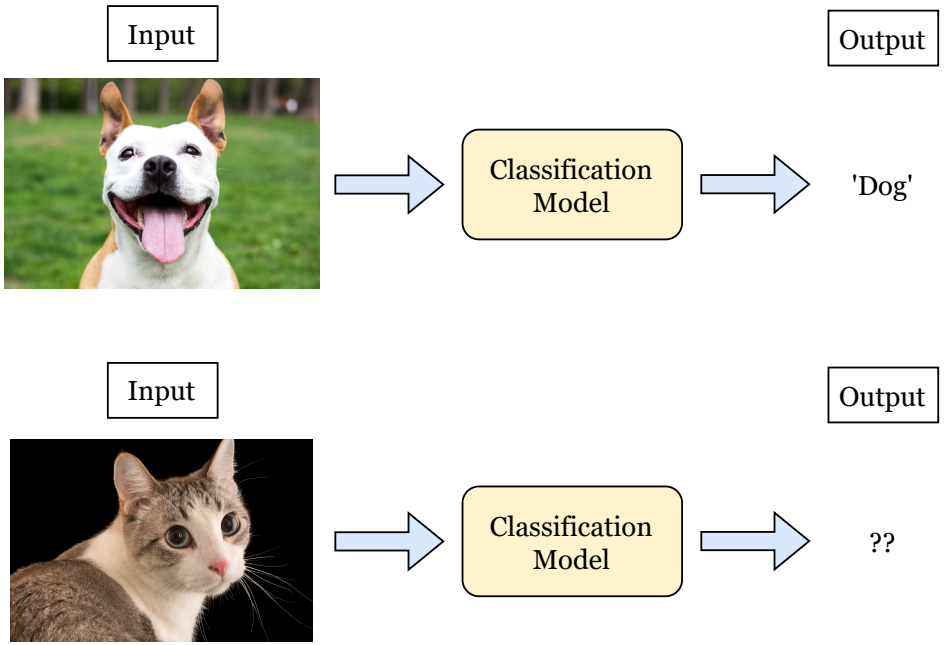


Figure: Example of a 'Dog and Cat' classification problem.

- **Input:** An image contains the object that need to be classified.
- **Output:** The class which the object belongs to.

Handwritten Digits Classification

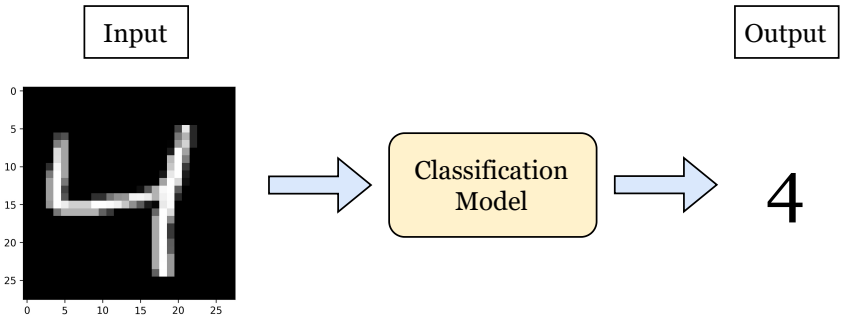


Figure: Example of an image in the MNIST dataset.

MNIST dataset [1]

- Grayscale images
- Size (each image): 28 x 28
- 10 classes (0, 1, 2, ..., 9)
- 60,000 samples in the training set.
- 10,000 samples in the testing set.

[1] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.

Load MNIST dataset

New package: **PyTorch** [2]

Import necessary packages

```
>> import torch
   from torch.utils.data import DataLoader
   import torchvision.datasets as datasets
   import torchvision.transforms as transforms
```

Load MNIST dataset

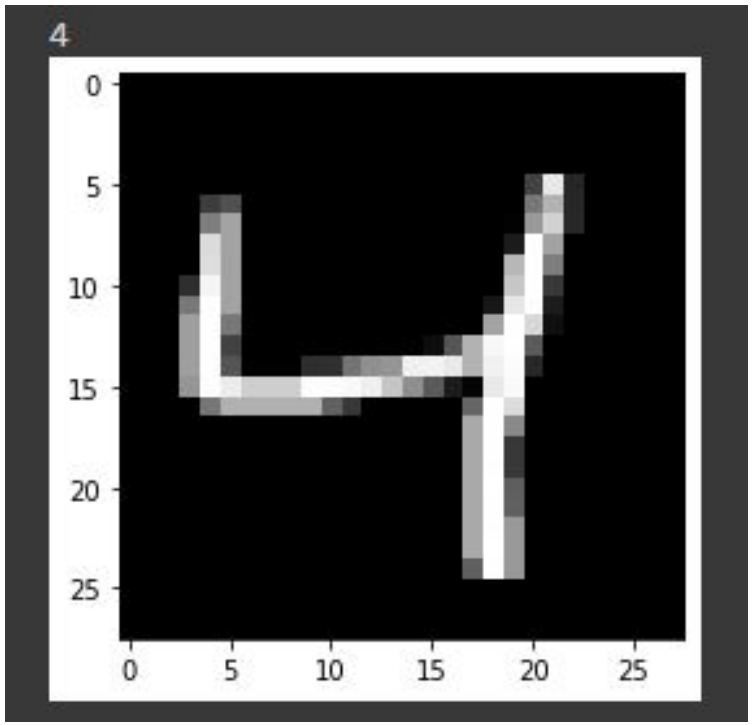
```
>> train_dataset = datasets.MNIST(root='path_to_contain_dataset',
   train=True, transform=transforms.ToTensor(), download=True)
   test_dataset = datasets.MNIST(root='path_to_contain_dataset',
   train=False, transform=transforms.ToTensor(), download=True)
```

When experimenting with PyTorch, almost data is represented in Tensor type.

[2] https://pytorch.org/docs/stable/index.html

Show an example image

```
>> import matplotlib.pyplot as plt
>> plt.imshow(train_dataset[2][0].squeeze(), cmap='gray')
   train_dataset[2][-1]
```



Create train/test data

Setup for getting reproducible results

```
>> import random
   import numpy as np
>> seed = 2411
>> random.seed(seed)
   np.random.seed(seed)
   torch.manual_seed(seed)
   torch.backends.cudnn.deterministic = True
   torch.backends.cudnn.benchmark = False
```

```
>> train_loader = DataLoader(train_dataset, batch_size=64,
   shuffle=True)
   test_loader = DataLoader(test_dataset)
```

Table of contents

Workflow

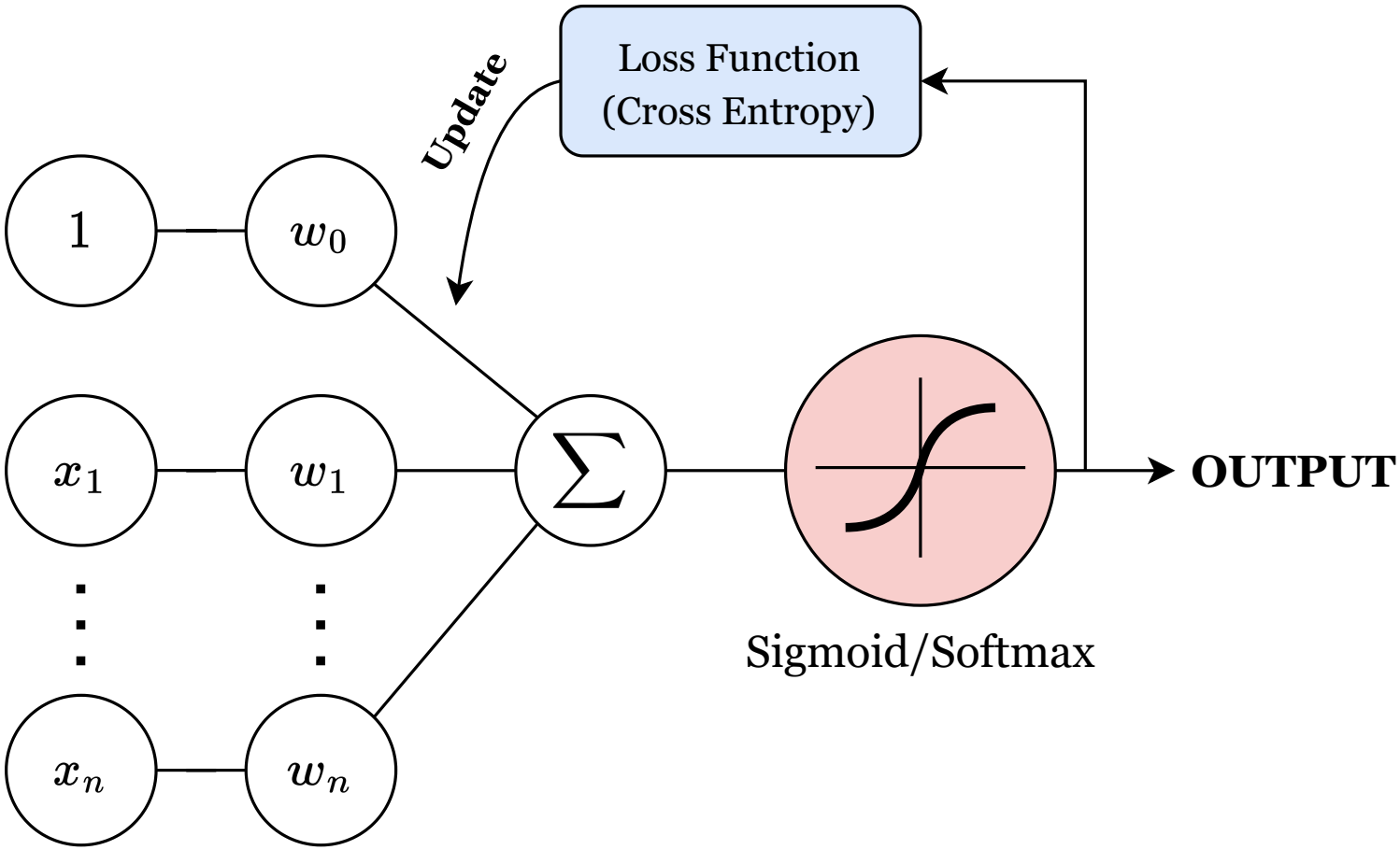


Figure: Workflow of Logistic Regression

Build Logistic Regression model

```
>> import torch.nn as nn
   import torch.nn.functional as F
```

```
class LogisticRegression(nn.Module):
    # input_dim → the number of independent variables (attributes)
    # output_dim → the number of (unique) classes for classification (e.g.,
dog-cat: 2; car-bicycle-plane: 3)
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        logits = self.linear(x)
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

Setup for Training

```
>> import torch.optim as optim ▷ load the training algorithm (e.g., SGD, ADAM)
>> device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
   device
```

Design 'Train Model' Function

- Loss Function: Cross Entropy Loss (Binary Cross Entropy Loss for binary classification)

Cross Entropy Loss

- In case of Binary Classification ($N = 2$), cross-entropy is given by:

$$l = -(y * log(p) + (1 - y) * log(1 - p))$$

where p is the predicted probability (sigmoid), and y is the truth label (0 or 1).

- When $N > 2$, cross-entropy is given by:

$$l = -\sum_{c=1}^{N} y_c * log(p_c)$$

where p_c is the predicted probability (softmax) for the c^{th} class, and y_c is the truth label.

Sigmoid and Softmax Activation Functions

- **Sigmoid Function** (2 classes)

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

- **Softmax Function** (C classes)

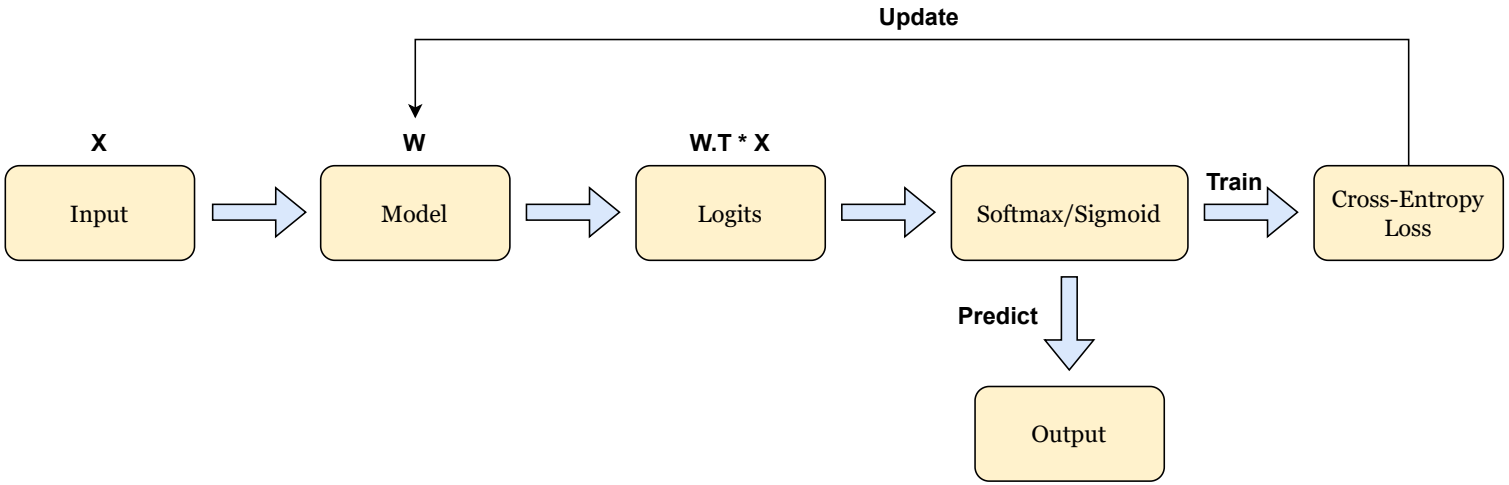
$$softmax(z)_i = \frac{e^{z(i)}}{\sum_{j=1}^{C} e^{z(j)}} \tag{2}$$

for $i = 1, 2, \ldots, C$

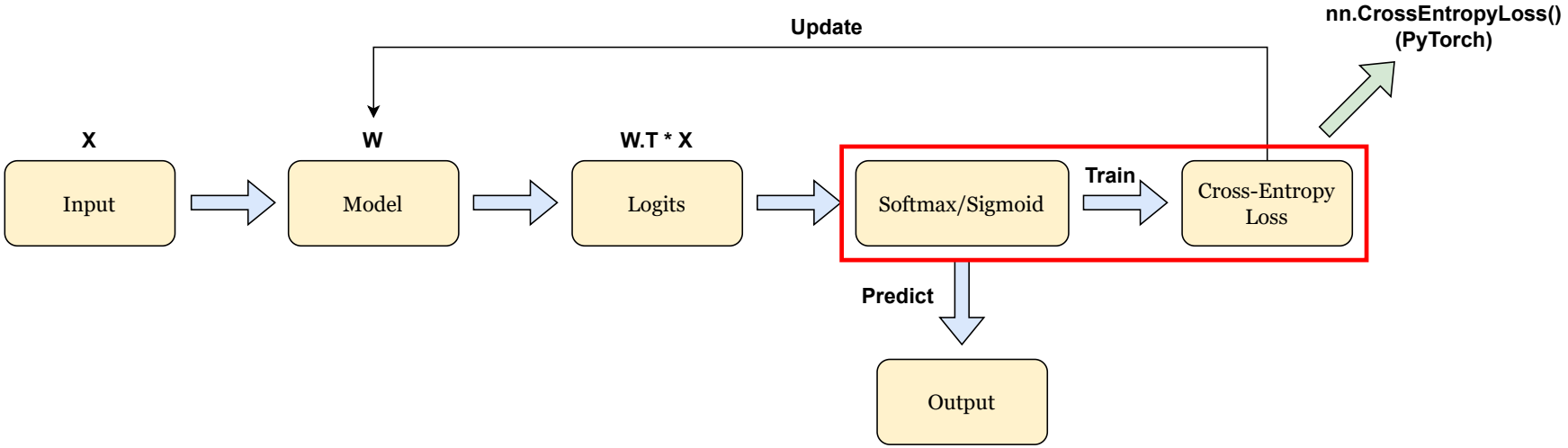
Design 'Train Model' Function (cont.)

- Loss Function: Cross Entropy Loss (Binary Cross Entropy Loss for binary classification)
- Optimizer: Stochastic Gradient Descent

nn.CrossEntropyLoss() (PyTorch)



nn.CrossEntropyLoss() (PyTorch)



Train Logistic Regression model on MNIST dataset

Train model

```
>> clf_LR = LogisticRegression(input_dim= 28 * 28,
   output_dim=10).to(device)
   loss_each_epoch_LR = train_model(data_loader=train_loader,
   model=clf_LR, init_lr=0.001, maxEpoch=20)
```

Visualize the trend of loss value

```
>> plt.plot(range(1, len(loss_each_epoch_LR) + 1), loss_each_epoch_LR)
   plt.xlabel('Epoch')
   plt.ylabel('Loss')
   plt.title('LogisticRegression-MNIST')
   plt.show()
```

Visualize the trend of loss value

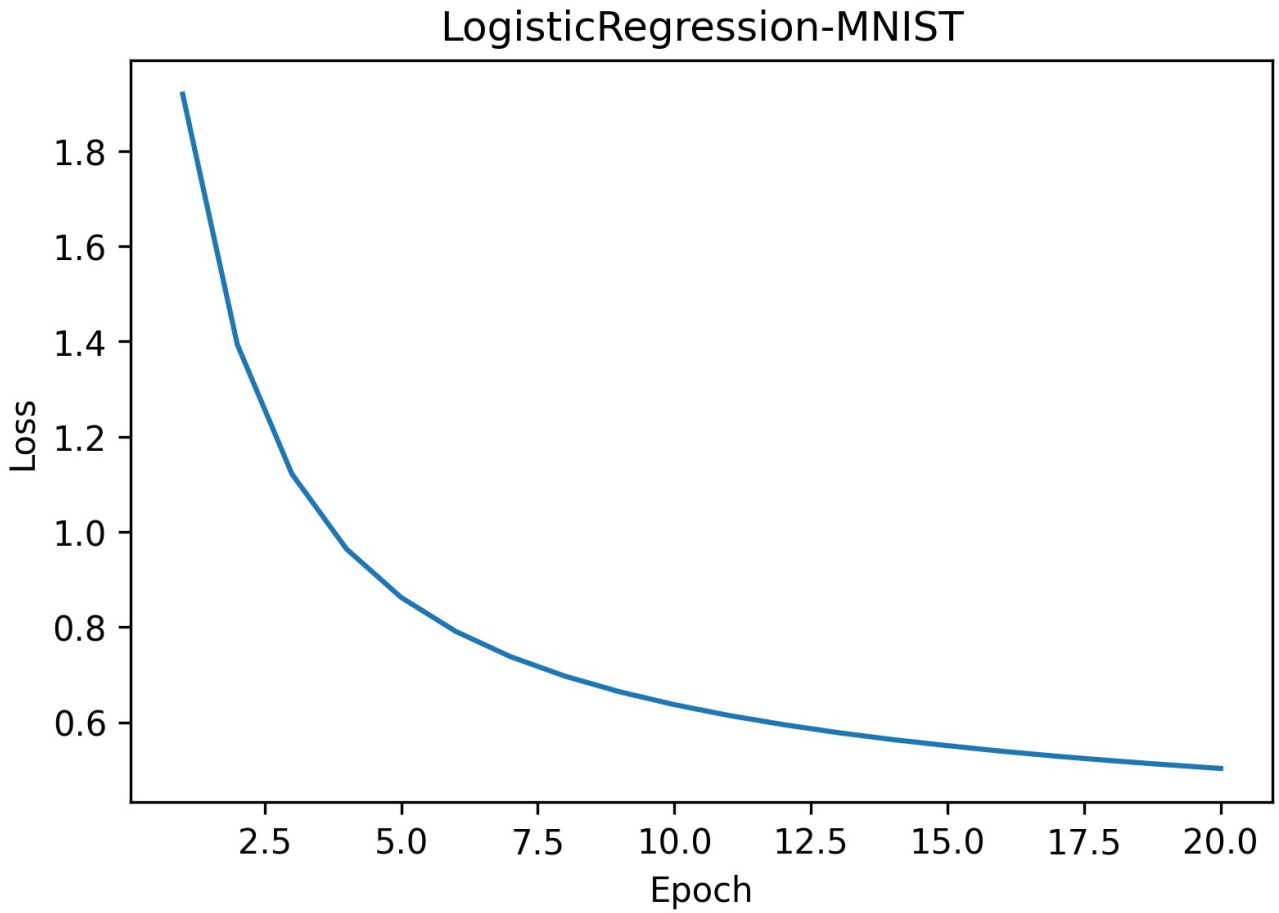


Figure: The loss values of Logistic Regression model over epochs.

Model Evaluation

```
def model_evaluation(data_loader, model):
    nCorrects = 0
    nSamples = 0
    model.eval()    ▷ switch our model to the evaluation mode
    with torch.no_grad():    ▷ ensure that no backprop here.
        for X, y in data_loader:
            X = X.to(device)
            y = y.to(device)
            X = X.reshape(X.shape[0], -1)

            _, probs = model(X)    ▷ get the class probabilities of X
            _, y_pred = probs.max(1)    ▷ X belongs to the class that has the highest
probability
            nCorrects += (y_pred == y).sum()
            nSamples += y_pred.size(0)
        print(f'We got {nCorrects}/{nSamples} correct. Accuracy =
{(nCorrects/nSamples) * 100:.2f} %')
```

Model Evaluation (cont.)

Train Accuracy

```
>> print('Logistic Regression (Train Accuracy)')
   model_evaluation(data_loader=train_loader, model=clf_LR)
```

Test Accuracy

```
>> print('Logistic Regression (Test Accuracy)')
   model_evaluation(data_loader=test_loader, model=clf_LR)
```

Visualize the some predictions

```
>> ROW_IMG, N_ROWS = 10, 5
   fig = plt.figure()
   for index in range(1, ROW_IMG * N_ROWS + 1):
      plt.subplot(N_ROWS, ROW_IMG, index)
      plt.axis('off')
      plt.imshow(test_dataset.data[index], cmap='gray_r')

      with torch.no_grad():
         clf_LR.eval()
         X = test_dataset[index][0].reshape(test_dataset[index][0].shape[0], -1)
         _, probs = clf_LR(X)

      title = f'{torch.argmax(probs)} ({torch.max(probs * 100):.0f}%)'
      plt.title(title, fontsize=7)
   fig.suptitle('Logistic Regression - predictions')
   plt.savefig('LogisticRegession_pred.jpg', bbox_inches='tight',
   pad_inches=0.1, dpi=300)
```

Logistic Regression - predictions

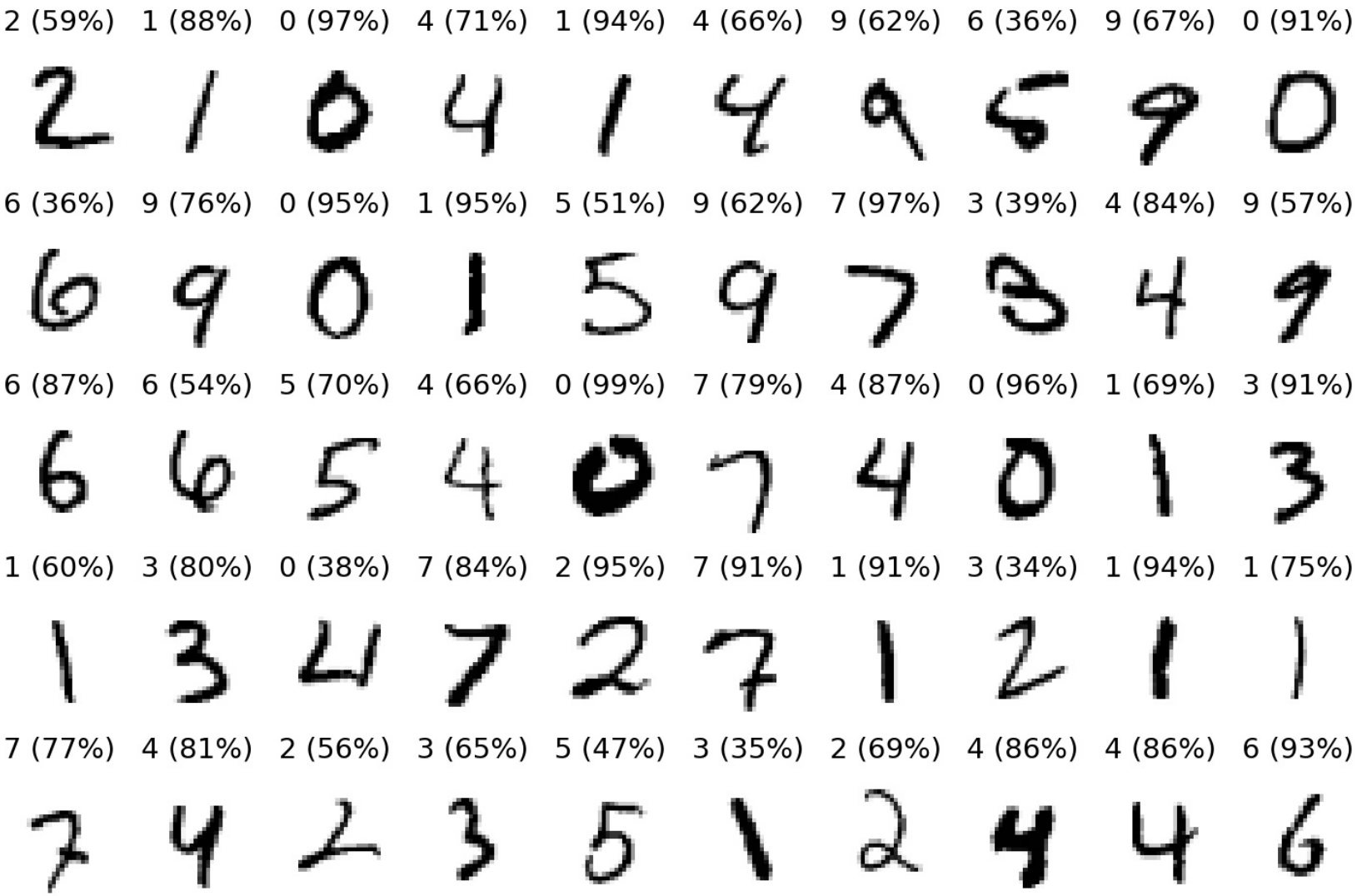


Table of contents

Multi Layer Perceptron

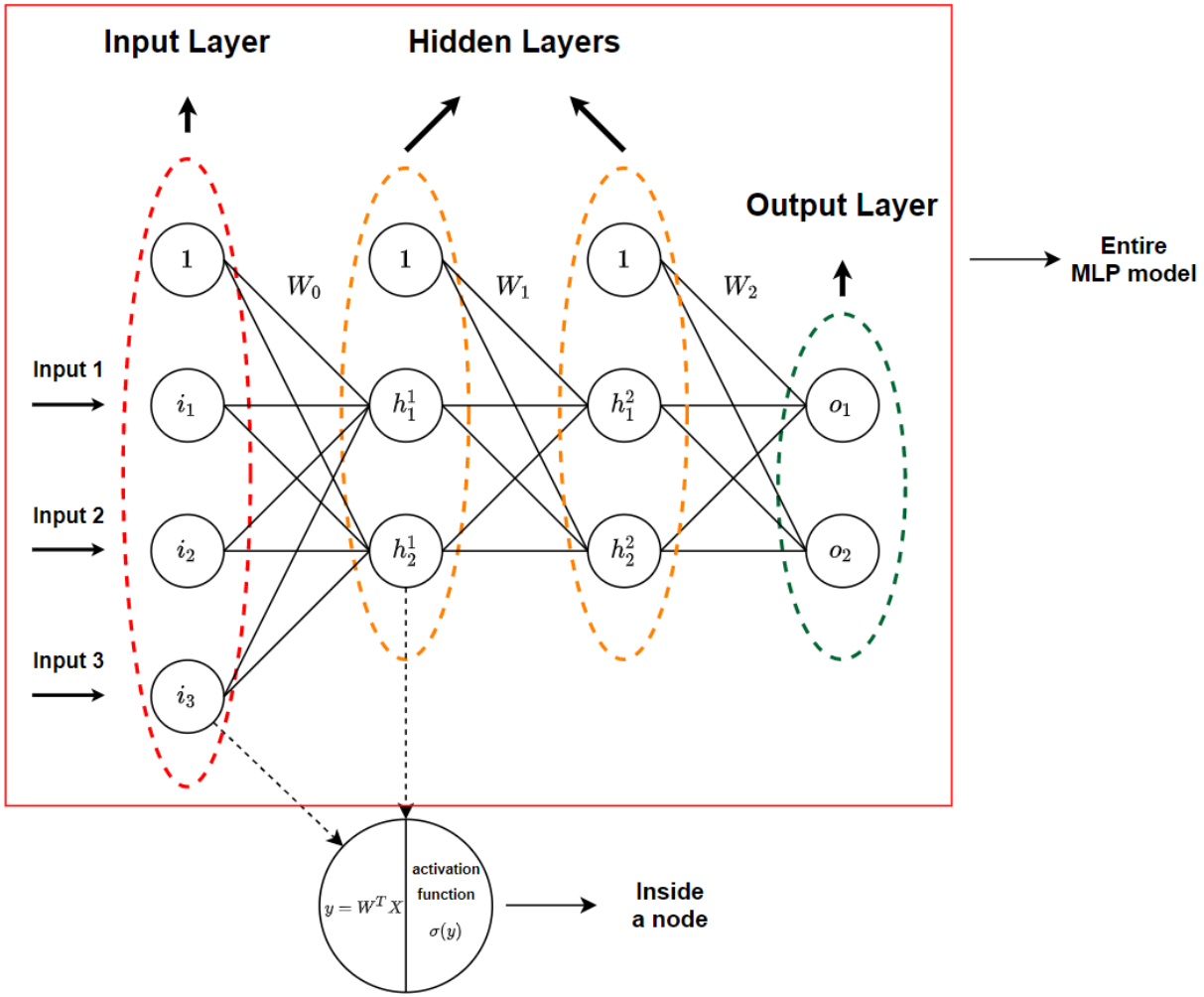


Figure: Example of a Multi Layer Perceptron model.

Build a Multi Layer Perceptron model by using Pytorch

Problem dataset: **MNIST** (10 classes)

Requirements:

- Build an MLP model has 3 layers: 1 'input' layer, 1 'hidden' layer, and 1 'output' layer.
- 'input' layer has 782 nodes (input_size), 'hidden' layer has 100 nodes, and 'output' layer has 25 nodes.
- The activation function of nodes in 'input' and 'hidden' layers is 'ReLU'.
- The activation of nodes in 'output' layer is 'Softmax'.

Build a Multi Layer Perceptron model with PyTorch

```
class MLP(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.classifier = nn.Sequential(
            nn.Linear(input_dim, 100),     ▷ 'input' layer
            nn.ReLU(),    ▷ activation function of the 'input' layer
            nn.Linear(100, 25),    ▷ 'hidden' layer
            nn.ReLU(),    ▷ activation function of the 'hidden' layer
            nn.Linear(25, output_dim)    ▷ 'output' layer
        )

    def forward(self, x):
        logits = self.classifier(x)
        probs = F.softmax(logits, dim=1)    ▷ activation function of 'output' layer
        return logits, probs
```

Train MLP model on MNIST dataset

Train model

```
>> clf_MLP = MLP(input_dim= 28 * 28, output_dim=10).to(device)
   loss_each_epoch_MLP = train_model(data_loader=train_loader,
   model=clf_MLP, init_lr=0.001, maxEpoch=20)
```

Visualize the trend of loss value

```
>> plt.plot(range(1, len(loss_each_epoch_MLP) + 1),
   loss_each_epoch_MLP)
   plt.xlabel('Epoch')
   plt.ylabel('Loss')
   plt.title('MLP-MNIST')
   plt.show()
```

Visualize the trend of loss value

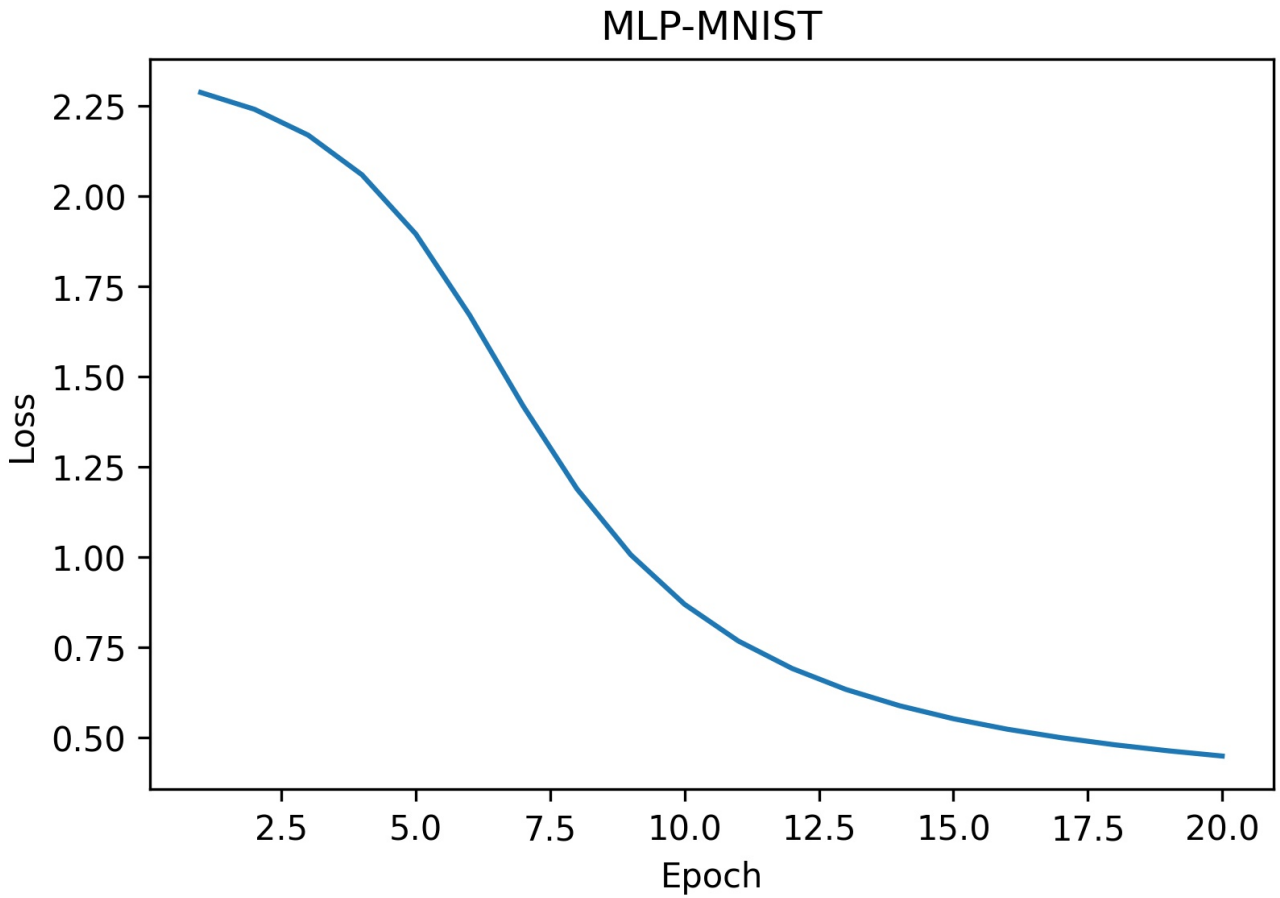


Figure: The loss values of Multi Layer Perceptron model over epochs.

Model Evaluation

Train Accuracy

```
>> print('Multi Layer Perceptron (Train Accuracy)')
   model_evaluation(data_loader=train_loader, model=clf_MLP)
```

Test Accuracy

```
>> print('Multi Layer Perceptron (Test Accuracy)')
   model_evaluation(data_loader=test_loader, model=clf_MLP)
```

Table of contents

Cake Dataset

- 300 images. Each class has 50 images.
- 6 classes: 'banhmi', 'cookie', 'croissant', 'donut', 'pizza', 'pretzel'
- Size (each image): 500 x 500

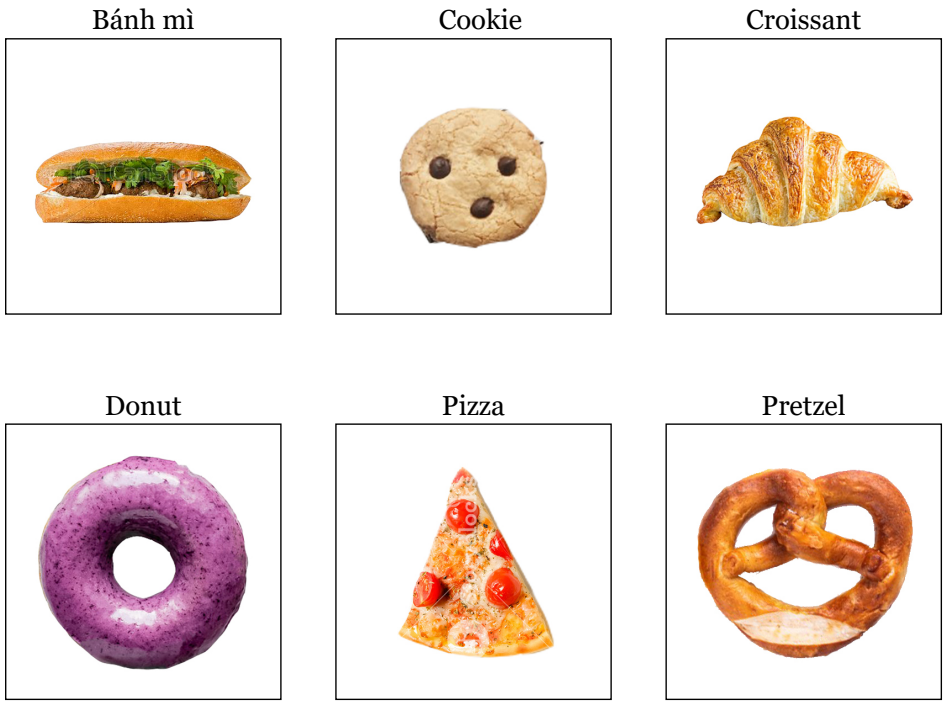


Figure: Example of some images in Cake dataset.

Create train/test data

```
>> import os
   import cv2 ▷ package for image processing (e.g., read, load, save)
```

Load images

```
>> root_dir = 'the storage path of cake dataset'

>> list_folders = os.listdir(root_dir)
   list_folders

>> X_cake, y_cake = [ ], [ ]

>> for folder in list_folders:
       list_image_paths = os.listdir(root_dir + '/' + folder)
       for image_path in list_image_paths:
           img_matrix = cv2.imread(root_dir + '/' + folder + '/' + image_path,
   0)
           X_cake.append([img_matrix])
           y_cake.append(folder)
```

Create train/test data (cont.)

Preprocess Data

```
>> X_cake = np.array(X_cake)/255
>> from sklearn.preprocessing import LabelEncoder
   le = LabelEncoder()
>> y_cake = le.fit_transform(y_cake)
```

Split data into train/test sets

```
>> from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(X_cake, y_cake,
   test_size=0.3, random_state=seed)
```

Create train/test data (cont.)

Convert Numpy to Tensor

```
>> X_train, X_test = np.array(X_train), np.array(X_test)
   y_train, y_test = np.array(y_train, np.array(y_test)

>> X_train = torch.from_numpy(X_train).float()
   X_test = torch.from_numpy(X_test).float()
   y_train = torch.from_numpy(y_train)
   y_test = torch.from_numpy(y_test)
```

Create data loaders

```
>> from torch.utils.data import TensorDataset

>> train_dataset_cake = TensorDataset(X_train, y_train)
   train_loader_cake = DataLoader(train_dataset_cake, batch_size=16,
   shuffle=True)

>> test_dataset_cake = TensorDataset(X_test, y_test)
   test_loader_cake = DataLoader(test_dataset_cake)
```

Train models on Cake Dataset

- input_dim = 500 * 500 = 250000
- output_dim = 6
- init_lr = 0.001
- maxEpoch = 200

Train model

```
>> clf_LR_1 = LogisticRegression(input_dim= 500 * 500,
   output_dim=6).to(device)
   loss_each_epoch_LR_1 = train_model(train_loader_cake, clf_LR_1,
   0.001, 200)
>> clf_MLP_1 = MLP(input_dim= 500 * 500, output_dim=6).to(device)
   loss_each_epoch_MLP_1 = train_model(train_loader_cake, clf_MLP_1,
   0.001, 200)
```

Model Evaluation

Logistic Regression - Train Accuracy

```
>> print('Logistic Regression (Train Accuracy)')
   model_evaluation(data_loader=train_loader_cake, model=clf_LR_1)
```

Logistic Regression - Test Accuracy

```
>> print('Logistic Regression (Test Accuracy)')
   model_evaluation(data_loader=test_loader_cake, model=clf_LR_1)
```

Multi Layer Perceptron - Train Accuracy

```
>> print('Multi Layer Perceptron (Train Accuracy)')
   model_evaluation(data_loader=train_loader_cake, model=clf_MLP_1)
```

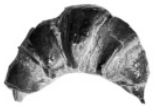
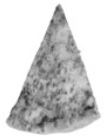
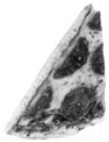
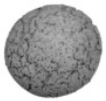
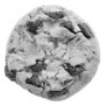
Multi Layer Perceptron - Test Accuracy

```
>> print('Multi Layer Perceptron (Test Accuracy)')
   model_evaluation(data_loader=test_loader_cake, model=clf_MLP_1)
```

Visualize the some predictions

```
>> ROW_IMG, N_ROWS = 5, 5
   fig = plt.figure()
   for index in range(1, ROW_IMG * N_ROWS + 1):
      plt.subplot(N_ROWS, ROW_IMG, index)
      plt.axis('off')
      plt.imshow(test_dataset_cake[index][0].squeeze(), cmap='gray_r')
      with torch.no_grad():
         clf_LR_1.eval()
         X =
test_dataset_cake[index][0].reshape(test_dataset_cake[index][0].shape[0], -1)
         _, probs = clf_LR_1(X)
      title = f'{le.inverse_transform([torch.argmax(probs)])[-1]}
({torch.max(probs * 100):.0f}%)'
      plt.title(title, fontsize=7)
   fig.suptitle('Logistic Regression - predictions')
```

Logistic Regression - predictions

croissaint (100%)	pretzel (100%)	pizza (100%)	cookie (100%)	cookie (100%)
				
pretzel (100%)	croissaint (100%)	pizza (100%)	pretzel (100%)	cookie (100%)
				
donut (100%)	cookie (100%)	pizza (100%)	pretzel (100%)	cookie (100%)
				
croissaint (100%)	cookie (100%)	croissaint (100%)	pretzel (100%)	donut (100%)
				
croissaint (100%)	cookie (100%)	croissaint (100%)	banhmi (100%)	pizza (100%)