AI VIET NAM – AI COURSE 2024

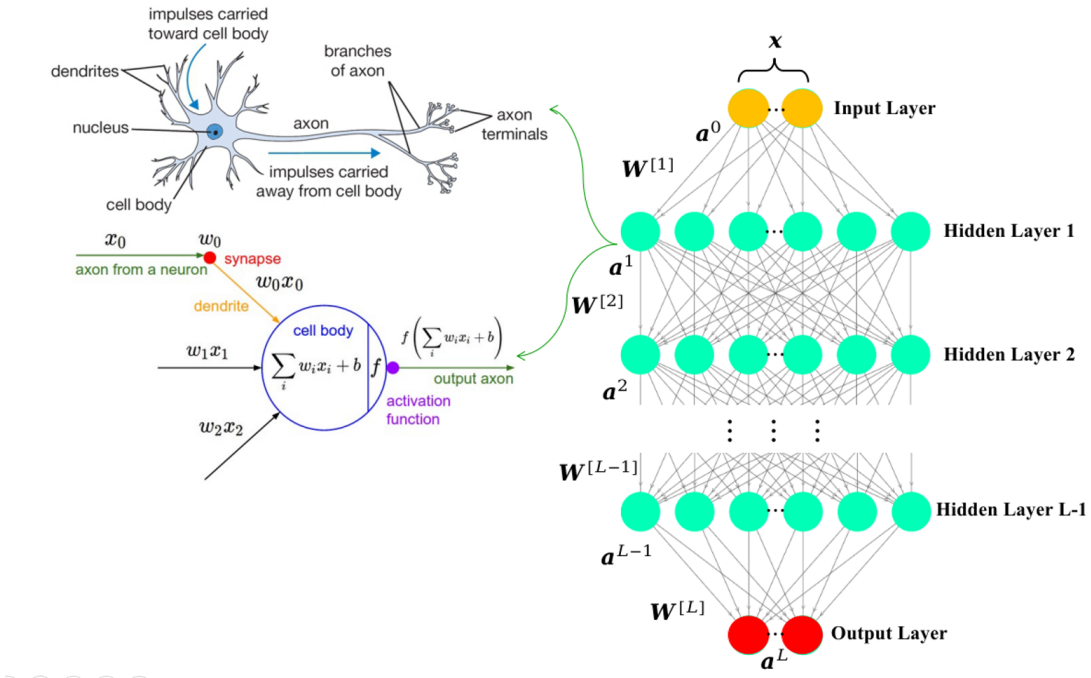
# Exercise: Multilayer Perceptrons (MLPs) and Activation Functions

Dinh-Thang Duong, Yen-Linh Vu, Quang-Vinh Dinh

Ngày 9 tháng 11 năm 2024

# I. Giới thiệu

**Multilayer Perceptrons (MLPs)** là một kiến trúc mạng nơ-ron trong lĩnh vực học sâu (deep learning) với cấu tạo gồm 3 thành phần chính: Lớp đầu vào (Input layer), Lớp ẩn (Hidden layer) và Lớp đầu ra (Output layer). Mỗi lớp ẩn sử dụng hàm kích hoạt (Activation functions) để mô hình có thể học các mối quan hệ phức tạp giữa đầu vào và đầu ra. MLPs thường được dùng trong các bài toán phân loại, hồi quy và các tác vụ học sâu khác trên các bộ dữ liệu có phân bố phức tạp.




Hình 1: Kiến trúc các lớp của Multilayer Perceptrons (MLPs).

Trong bài tập này, chúng ta sẽ tìm hiểu cách cài đặt mạng MLPs để thực hiện các task bao gồm: regression, classification với non-linear data, và classification với dữ liệu ảnh. Dưới đây là tổng hợp một số nội dung cơ bản liên quan đến MLPs:

1. **Một Số Ký Hiệu:**
   (a) $x$: Input vector (features).
   (b) $y$: label, kết quả thực tế.
   (c) $\hat{y}$: Kết quả dự đoán.
   (d) $W$: trọng số của mô hình (nhiệm vụ của huấn luyện là tìm bộ $W$ để $y \approx \hat{y}$).
   (e) Loss: loss function, mục tiêu huấn luyện là tìm loss tối ưu nhất.
   (f) $n$: Số lượng đặc trưng của một mẫu dữ liệu.
   (g) $f(\cdot)$: Activation Function.
   (h) $m$: số lượng mẫu dữ liệu trong một batch.
   (i) $C$: số lượng class cần phân loại.

2. **Linear Regression:** Sử dụng cho regression task (ví dụ dự đoán giá nhà).

$$\hat{y} = W^T x = w_0 + w_1 x_1 + \cdots + w_n x_n$$

   Loss thường được dùng là SE:

$$L(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

3. **Softmax Regression:** Sử dụng cho bài toán phân loại.

$$\hat{y} = f(W^T x) = f(w_0 + w_1 x_1 + \cdots + w_n x_n)$$

   Đối với bài toán binary classification thì dùng Logistic Regression với sigmoid function

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

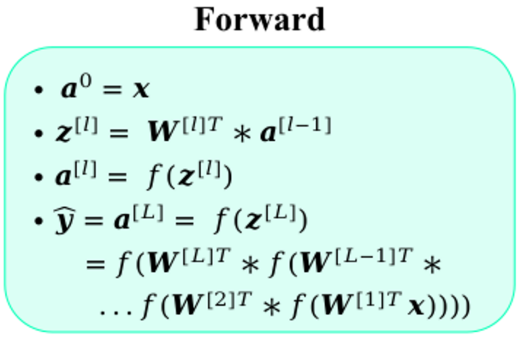
   Nếu có nhiều hơn 2 classes thì dùng Softmax Regression với công thức

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$$

   Loss là cross entropy với các class ở dạng one-hot-encoding, công thức này tính loss cho một mẫu duy nhất (1 sample):

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

4. **MLP:** Như ta đã biết, MLP thông thường có 3 loại layer: Input layer, Output layer, và Hidden layer. Có thể có nhiều Hidden layer và thường được bắt đầu từ 1 (số lượng hidden layer càng nhiều thì model càng deep). Mỗi layer sẽ có nhiều node (hình tròn), với Input layer là số lượng features, Output layer là số lượng node dùng cho output (ví dụ bài toán classification phân loại 3 classes thì cần 3 nodes), và mỗi Hidden layer là số lượng neuron tham gia vào trích xuất đặc trưng của layer đó. Loss sẽ tùy thuộc vào từng bài toán mà sẽ có function cụ thể.




Hình 2: Minh họa giai đoạn Forward (a) của Multi-layer Perceptron (MLPs).

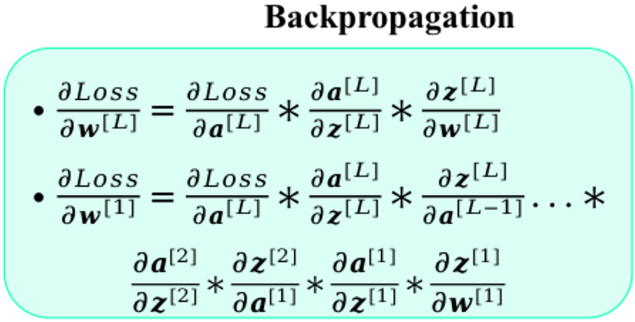
(a) **Forward:** Thông tin sẽ đi theo chiều từ Input layer đến Output layer. Tại mỗi layer $l$ của các Hidden layer, thông tin đầu vào sẽ là output từ layer $l-1$ gọi là $a^{[l-1]}$. Tại đây ta tính được:

$$z^{[l]} = W^{[l]^T} a^{[l-1]}$$

Tiếp theo $z^{[l]}$ sẽ đi qua một activation và thu được:

$$a^{[l]} = f(z^{[l]})$$

Việc này lặp lại cho đến khi đi qua hết các layer và ra output cuối cùng.

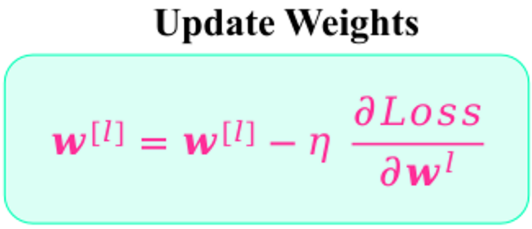



Hình 3: Minh họa giai đoạn Backpropagation (b) của Multi-layer Perceptron (MLPs).

(b) **Backpropagation:** Lúc này hàm Loss sẽ được dùng để tính toán sự sai lệch giữa $y$ và $\hat{y}$. Dựa vào thông tin này để gửi phản hồi theo chiều từ Output layer đến Input layer để điều chỉnh các tham số trong mỗi layer (quá trình này là backpropagation). Thông tin phản hồi về chính là gradient và được tính toán dựa vào đạo hàm và chain rule:

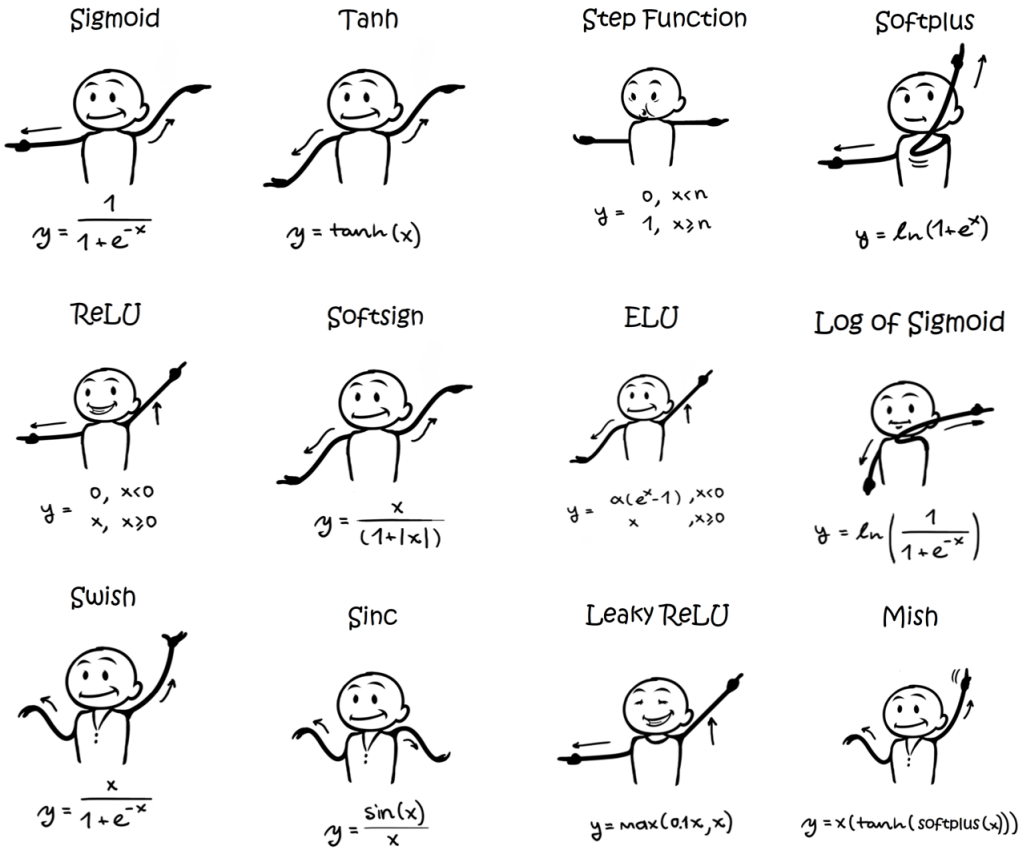
$$\frac{\partial \text{Loss}}{\partial W} = \frac{\partial \text{Loss}}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial W}$$

Sau đó sử dụng thuật toán gradient descent để update weights.




Hình 4: Minh họa quá trình cập nhật trọng số (b) của Multi-layer Perceptron (MLPs).

(c) **Hàm kích hoạt:** Là các hàm toán học được áp dụng cho đầu ra của các nơ-ron trong MLP, với ý nghĩa chính là giúp MLP đưa ra quyết định có nên kích hoạt nơ-ron hay không. Nó giúp chuyển đổi đầu ra tuyến tính của một lớp thành một dạng phi tuyến, cho phép mạng nơ-ron học được các quan hệ phức tạp hơn.




Hình 5: Một số hàm kích hoạt thông dụng. Nguồn: link.

# II. Bài tập

## A. Phần lập trình

**Giới thiệu sơ lược**: Bài tập tuần này sẽ bao gồm 3 bài toán sau đây:

- **Bài toán 1**: Task về regression cho một tập data **Auto_MPG_data.csv** dựa vào 9 features của xe ô tô để dự đoán năng lượng tiêu thụ (MPG)
- **Bài toán 2**: Task về classification (nonlinear data). Cho 1 tập data **NonLinear_data.npy** dựa vào 2 features để phân loại 3 classes
- **Bài toán 3**: Task về classification ảnh. Cho 1 tập data **FER-2013.zip** gồm các ảnh thể hiện 7 (classes) cảm xúc khác nhau trên khuôn mặt con người

- **Bài toán 1: Dự đoán hiệu suất xe hơi.**

1. **Tải bộ dữ liệu:** Bước đầu tiên là tải dữ liệu cho bài bằng lệnh "!gdown":

```
!gdown --id 1qiUDDoYyRLBiKOoYWdFl_5WByHE8Cugu
```

   Nếu không thể tải bằng cách gdown do giới hạn số lượt tải, các bạn có thể thủ công bằng cách tải file **Auto_MPG_data.csv** trong đường dẫn thuộc mục Datasets tại phần III.

2. **Import các thư viện cần thiết:** Trong bài tập này, chúng ta cần import thư viện PyTorch để cài đặt mô hình:

```
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
```

3. **Cài đặt giá trị ngẫu nhiên cố định:** Để kết quả trả về là cố định với mỗi lần chúng ta thực hiện chạy lại từ đầu file notebook, thama số ngẫu nhiên (seed) cần phải được cài đặt với tất cả các thư viện và hàm có liên quan đến phép ngẫu nhiên. Trong bài này, các thư viện NumPy, PyTorch, scikit-learn sẽ có một vài hàm có sử dụng tính ngẫu nhiên, vì vậy ta sẽ cài đặt chung một giá trị ngẫu nhiên như sau:

```
random_state = 59
np.random.seed(random_state)
torch.manual_seed(random_state)
if torch.cuda.is_available():
    torch.cuda.manual_seed(random_state)
```

4. **Cài đặt thiết bị tính toán:** Đối với PyTorch, để tận dụng sức mạnh của GPU nếu có sẵn (trong colab các bạn cần điều chỉnh runtime sang GPU để có thể kích hoạt được GPU), ta cần chỉ định thiết bị mà mô hình sẽ được chạy. Đoạn code dưới đây sẽ tự động chọn GPU (nếu khả dụng), hoặc quay về CPU khi GPU không được hỗ trợ:

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

5. **Đọc bộ dữ liệu:** Bộ dữ liệu trong bài được lưu dưới dạng file .csv, vì vậy chúng ta sẽ sử dụng thư viện pandas để đọc data này lên:

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

6. **Tiền xử lý bộ dữ liệu:** Với DataFrame vừa đọc lên, ta sẽ thực hiện một số bước tiền xử lý dữ liệu để có thể sử dụng chúng trong việc huấn luyện mô hình.

   (a) **Tách đặc trưng X và nhãn y:**

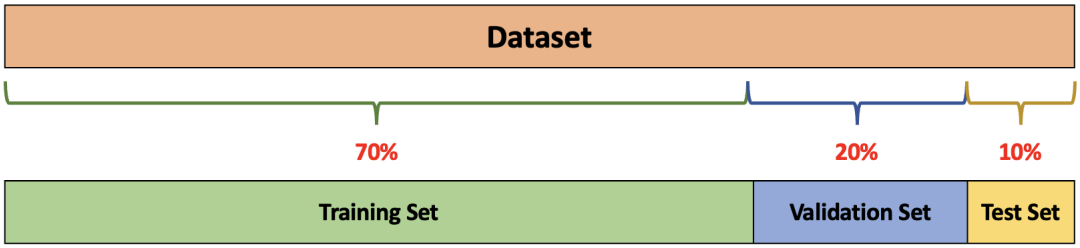
```
X = dataset.drop(columns='MPG').values
y = dataset['MPG'].values
```

   (b) **Chia bộ dữ liệu train/val/test:** Với toàn bộ dữ liệu hiện tại, ta sẽ tách thành 3 bộ dữ liệu riêng biệt cho việc huấn luyện và đánh giá mô hình. Trong bài này, ta sẽ chia ba bộ train/val/test theo tỉ lệ 7:2:1 như sau:

```
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
)
```

   (c) **Chuẩn hóa đặc trưng đầu vào:**

```
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.float32)
y_val = torch.tensor(y_val, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
```




Hình 6: Minh họa việc chia bộ dữ liệu thành ba bộ train, val, test theo tỉ lệ 7:2:1.

7. **Xây dựng DataLoader:** Để có thể duyệt qua bộ dữ liệu một cách hiệu quả cho việc huấn luyện cũng như đánh giá mô hình trong PyTorch, chúng ta cần xây dựng PyTorch DataLoader. Để xây dựng DataLoader, chúng ta cần phải xây dựng PyTorch Dataset nhằm lưu dữ liệu đầu vào đầu ra trong bài toán. Theo đó, ta sẽ xây dựng một PyTorch Dataset đơn giản như sau:

```
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]
```

   Sau khi khai báo class PyTorch Dataset, ta chỉ việc gói triển khai của các dataset này vào DataLoader là hoàn tất. Lưu ý rằng, ở đây ta sẽ khai báo luôn tham số batch size, vì DataLoader sẽ hỗ trợ chúng ta lấy batch một cách tối ưu hơn:

```
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)
```

8. **Xây dựng mạng MLP:** Ta triển khai một mạng MLP gồm 2 lớp ẩn, theoo sau mỗi lớp ẩn là hàm kích hoạt ReLU. Ta cài đặt cấu trúc mạng này trong PyTorch như sau:

```
class MLP(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims):
        super().__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)
```

Sau khi đã có cài đặt class cho kiến trúc MLP, ta sẽ khai báo biến mô hình bằng cách khai báo một đối tượng của class trên đi kèm với các tham số phù hợp với bài toán đang giải:

```
input_dims = X_train.shape[1]
output_dims = 1
hidden_dims = 64

model = MLP(input_dims=input_dims,
            hidden_dims=hidden_dims,
            output_dims=output_dims).to(device)
```

Theo đó, chúng ta cần điều chỉnh các tham số về số lượng node dữ liệu đầu vào (input_dims), số node trong lớp ẩn (hidden_dims) và số node đầu ra (output_dims). Trong bài này, input_dims chính là số lượng đặc trưng của biến X, output_dims bằng 1 vì bài toán ta làm là bài regression nên chỉ có một giá trị dự đoán. Số node trong hidden_dims ta có thể tùy chỉnh, ở đây lấy ví dụ cài đặt = 64

9. **Khai báo hàm loss và optimizer:** Đối với PyTorch, ta có thể dễ dàng cài đặt hàm loss cũng như thuật toán tối ưu bằng class có sẵn. Trong bài này, ta sử dụng hàm Mean Squared Error Loss (MSELoss) cũng như thuật toán tối ưu Stochastic Gradient Descent (SGD):

```
lr = 1e-2
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

10. **Xây dựng hàm tính điểm R2:** Trong bài này, ta sẽ sử dụng thêm một độ đo khác cho bài regression là điểm R2 (Coefficient of determination). Cách triển khai như sau:

```
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
```

11. **Huấn luyện mô hình:** Khi đã đầy đủ tất cả gồm mô hình, dataloader, hàm loss và optimizer, chúng ta đã có thể tiến hành huấn luyện mô hình trong PyTorch. Cách cài đặt như sau:

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

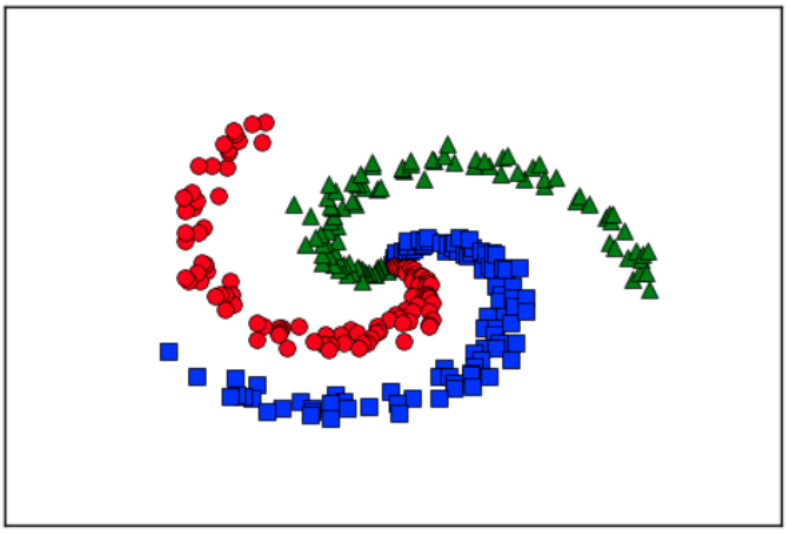
```
6
7  for epoch in range(epochs):
8      train_loss = 0.0
9      train_target = []
10     val_target = []
11     train_predict = []
12     val_predict = []
13     model.train()
14     for X_samples, y_samples in train_loader:
15         X_samples = X_samples.to(device)
16         y_samples = y_samples.to(device)
17         optimizer.zero_grad()
18         outputs = model(X_samples)
19         train_predict += outputs.tolist()
20         train_target += y_samples.tolist()
21         loss = criterion(outputs, y_samples)
22         loss.backward()
23         optimizer.step()
24         train_loss += loss.item()
25     train_loss /= len(train_loader)
26     train_losses.append(train_loss)
27     train_r2.append(r_squared(train_target, train_predict))
28     model.eval()
29     val_loss = 0.0
30     with torch.no_grad():
31         for X_samples, y_samples in val_loader:
32             X_samples = X_samples.to(device)
33             y_samples = y_samples.to(device)
34             outputs = model(X_samples)
35             val_predict += outputs.tolist()
36             val_target += y_samples.tolist()
37             loss = criterion(outputs, y_samples)
38             val_loss += loss.item()
39     val_loss /= len(val_loader)
40     val_losses.append(val_loss)
41     val_r2.append(r_squared(val_target, val_predict))
42     print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\
   tValidation loss: {val_loss:.3f}')
```

12. **Đánh giá mô hình:** Sau khi quá trình huấn luyện hoàn tất, ta có thể đánh giá hiệu suất của mô hình trên tập test như sau:

```
model.eval()
with torch.no_grad():
    y_hat = model(X_test)
    test_set_r2 = r_squared(y_hat, y_test)
    print('Evaluation on test set:')
    print(f'R2: {test_set_r2}')
```

Theo đó, kết quả ta nhận được sẽ là điểm R2 của mô hình trên tập test.

- **Bài toán 2: Phân loại với dữ liệu phi tuyến:** Đây là một bài toán đơn giản về classification trên dữ liệu phi tuyến (nonlinear data), bài tập này nhằm mục đích mô tả về khả năng của một mạng Multilayer Perceptron với những dữ liệu phân bố phức tạp hơn.



Hình 7: Trực quan hóa trên không gian 2D bộ dữ liệu phi tuyến.

1. **Tải bộ dữ liệu:** Bước đầu tiên là tải dữ liệu cho bài bằng lệnh "!gdown"

```
!gdown --id 1SqSn_8rxkk-Qvu4JLMcN_3ZFGDNa6P_V
```

   Nếu không thể tải xuống bằng gdown do giới hạn số lượt tải, hãy tải thủ công bộ dữ liệu **NonLinear_data.npy** trong đường dẫn thuộc mục Datasets tại phần III.

2. **Import các thư viện cần thiết:**

```
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
```

3. **Cài đặt giá trị ngẫu nhiên cố định và thiết bị tính toán:**

```
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)
```

4. **Đọc bộ dữ liệu:** Dữ liệu cho bài toán phân loại được lưu dưới dạng `.npy` (Python NumPy Array File). Ta sử dụng `np.load` để tải dữ liệu từ file này, thu được một

dictionary với hai phần: `X` là ma trận đặc trưng có kích thước $(300, 2)$ (300 mẫu, mỗi mẫu có 2 đặc trưng) và `y` là vector chứa 300 nhãn tương ứng.

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

```
print(X.shape, y.shape)
```

(300, 2) (300,)

5. **Chia bộ dữ liệu train/val/test:** Trong bài này, ta cũng chia dữ liệu thành ba bộ train/val/test theo tỉ lệ 7:2:1 tương tự như bài 1.

```
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
)
```

6. **Chuẩn hóa đặc trưng đầu vào:**

```
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)
```

7. **Xây dựng DataLoader:** Phần này được thực hiện tương tự như bài 1. Chúng ta xây dựng class CustomDataset để quản lý dữ liệu và sử dụng DataLoader cho các tập train, val, và test, với batch_size là 32.

```
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]
```

```
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)
```

8. **Xây dựng mạng MLP:** Để giải quyết bài toán phân loại phi tuyến với dữ liệu có 2 đặc trưng và 3 lớp, chúng ta xây dựng một mạng MLP gồm 2 lớp ẩn với hàm kích hoạt ReLU sau mỗi lớp. Cấu trúc này giúp mô hình học các đặc trưng phi tuyến trong dữ liệu.

```
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)
```

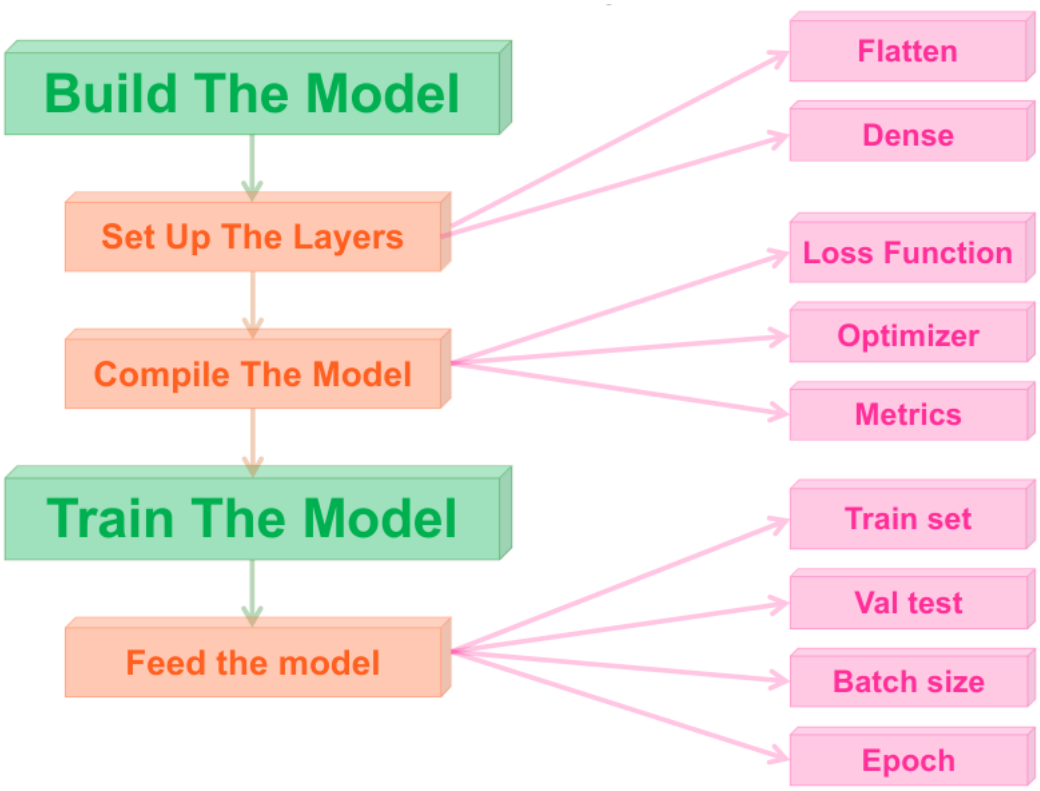
Tiếp theo, chúng ta thiết lập các tham số cho mạng MLP, bao gồm số lượng node cho đầu vào (input_dims), số node trong lớp ẩn (hidden_dims), và số node cho đầu ra (output_dims).

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

model = MLP(input_dims=input_dims,
            hidden_dims=hidden_dims,
            output_dims=output_dims).to(device)
```

Để xác định output_dims - tức số lớp cần phân loại, tương ứng với số nhãn khác nhau có trong y_train, chúng ta sử dụng cách đếm số nhãn duy nhất trong y_train bằng hàm torch.unique, giúp mô hình tự động thích ứng với các bài toán có số lớp khác nhau mà không cần phải thay đổi thủ công.

Trong bài 1, hidden_dims được chọn là 64 vì bài 1 là bài toán hồi quy đơn giản, chỉ cần dự đoán một giá trị đầu ra liên tục (output_dims = 1), nên không đòi hỏi nhiều

node trong lớp ẩn. Còn đối với bài 2, ta đặt hidden_dims là 128 vì đây là bài toán phân loại phi tuyến với 3 lớp, đòi hỏi mô hình có khả năng học các đặc trưng phức tạp để phân biệt giữa các lớp. Đặt số lượng node cao hơn sẽ giúp mạng đạt đủ độ phức tạp để học các đặc trưng mà vẫn đảm bảo hiệu quả tính toán.




Hình 8: Minh họa các bước xây dựng và huấn luyện mô hình trong PyTorch.

9. **Khai báo hàm loss và optimizer:** Ta sử dụng hàm mất mát Cross Entropy cho bài toán phân loại đa lớp, cùng với thuật toán tối ưu SGD (Stochastic Gradient Descent). Về lý thuyết, Cross Entropy Loss được định nghĩa cho từng mẫu dữ liệu riêng lẻ (1 sample). Tuy nhiên, trong thực tế huấn luyện, dữ liệu thường được chia thành các batch nhỏ thông qua DataLoader, do đó ta thường tính loss cho một batch (nhiều samples cùng lúc) để tăng hiệu suất tính toán. Công thức chính xác của hàm mất mát này cho một batch dữ liệu là:

$$\text{CrossEntropyLoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})\,,$$

trong đó:

- $N$ là số mẫu trong batch,
- $C$ là số lớp,
- $y_{i,c}$ là nhãn thực tế của mẫu $i$ dưới dạng one-hot vector,
- $\hat{y}_{i,c}$ là xác suất dự đoán cho lớp $c$ của mẫu $i$.

```
lr = 1e-1
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

10. **Xây dựng hàm tính độ chính xác:** Để đánh giá hiệu suất của mô hình, chúng ta định nghĩa hàm compute_accuracy, tính độ chính xác phân loại (classification accuracy) bằng cách chia số dự đoán đúng cho tổng số mẫu, công thức chung có dạng:

$$\text{Accuracy} = \frac{\text{Số lượng dự đoán đúng}}{\text{Tổng số mẫu}}$$

```
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
```

11. **Huấn luyện mô hình:**

```
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)

    val_loss = 0.0
    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_loss += criterion(outputs, y_samples).item()

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

    val_loss /= len(val_loader)
    val_losses.append(val_loss)

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

    print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\
tValidation loss: {val_loss:.3f}')
```

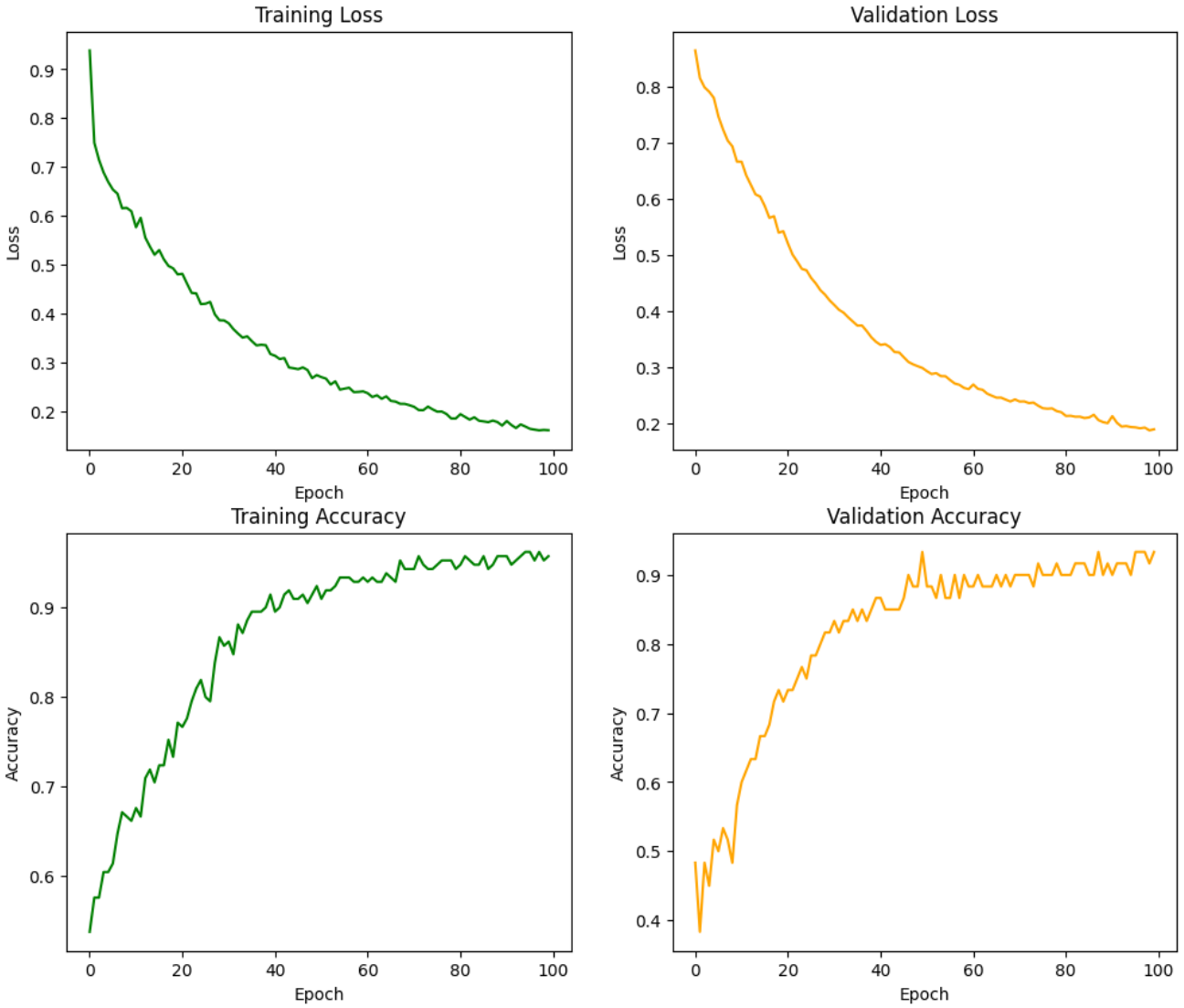
```
EPOCH 92:       Training loss: 0.172    Validation loss: 0.201

EPOCH 93:       Training loss: 0.166    Validation loss: 0.195

EPOCH 94:       Training loss: 0.173    Validation loss: 0.196

EPOCH 95:       Training loss: 0.169    Validation loss: 0.194

EPOCH 96:       Training loss: 0.165    Validation loss: 0.193

EPOCH 97:       Training loss: 0.163    Validation loss: 0.192

EPOCH 98:       Training loss: 0.161    Validation loss: 0.193

EPOCH 99:       Training loss: 0.162    Validation loss: 0.188

EPOCH 100:      Training loss: 0.162    Validation loss: 0.190
```

Hình 9: Kết quả huấn luyện in trên màn hình ở những epoch cuối cùng.

Chúng ta có thể trực quan hóa kết quả huấn luyện lên đồ thị như sau:

```
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()
```




Hình 10: Hình ảnh trực quan kết quả huấn luyện trên tập train và val cho bài toán nonlinear data classification.

12. **Đánh giá mô hình:** Sau khi quá trình huấn luyện hoàn tất, chúng ta đánh giá mô hình trên tập test.

```
test_target = []
test_predict = []
model.eval()
```

```
with torch.no_grad():
    for X_samples, y_samples in test_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        outputs = model(X_samples)

        test_predict.append(outputs.cpu())
        test_target.append(y_samples.cpu())

    test_predict = torch.cat(test_predict)
    test_target = torch.cat(test_target)
    test_acc = compute_accuracy(test_predict, test_target)

    print('Evaluation on test set:')
    print(f'Accuracy: {test_acc}')
```

- **Bài toán 3: Phân loại cảm xúc trên ảnh.**

1. **Tải bộ dữ liệu:** Tải dữ liệu cho bài bằng lệnh "!gdown".

```
!gdown --id 1GaTMURqIQTjtalbNVAyVgPIEis21AOr8
```

Bài tập 3 sử dụng data dạng ảnh. Khi download data về sẽ là một file nén dạng .zip, do đó chúng ta cần giải nén ra như bên dưới

```
!unzip -q './FER-2013.zip'
```

```
FER-2013
├── test
│   ├── angry
│   ├── disgust
│   ├── fear
│   ├── happy
│   ├── neutral
│   ├── sad
│   └── surprise
└── train
    ├── angry
    ├── disgust
    ├── fear
    ├── happy
    ├── neutral
    ├── sad
    └── surprise
```

Hình 11: Cấu trúc thư mục của tập data FER-2013 sau khi được giải nén ra. Các tên của thư mục con cũng chính là class tương ứng của ảnh.

2. **Import các thư viện cần thiết:**

```
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
```

3. **Cài đặt giá trị ngẫu nhiên cố định và thiết bị tính toán:**

```
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)
```

4. **Đọc số lượng class trong dataset:** Thiết lập đường dẫn đến thư mục train và test. Chúng ta lấy danh sách tên các thư mục con trong thư mục train, đại diện cho các lớp cảm xúc khác nhau. Mỗi thư mục con này chứa các ảnh thuộc về một lớp cụ thể.

```
train_dir = '/content/train'
test_dir = '/content/test'

classes = os.listdir(train_dir)

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

   Sau đó, chúng ta xây dựng hai dict là label2idx và idx2label để ánh xạ giữa tên và chỉ số lớp, nhằm thuận tiện cho việc xử lý dữ liệu.

5. **Xây dựng DataLoader:** Xây dựng DataLoader cho bài toán phân loại cảm xúc ảnh từ tập FER-2013. Trước tiên, lớp tùy chỉnh ImageDataset được tạo ra để quản lý và xử lý dữ liệu ảnh. Lớp này sẽ cung cấp các phương thức để lấy danh sách đường dẫn ảnh và nhãn, tính số lượng ảnh, cũng như chuẩn hóa từng ảnh đầu vào (nếu cần) trước khi đưa vào mô hình.

```
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)

    def read_img_files(self):
        img_paths = []
        img_labels = []
        for cls in self.label2idx.keys():
            for img in os.listdir(os.path.join(self.img_dir, cls)):
                img_paths.append(os.path.join(self.img_dir, cls, img)
    )
                img_labels.append(cls)

        return img_paths, img_labels

    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
```

Sau đó, DataLoader được khởi tạo cho các tập train, val, và test với batch size là 256.

```
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)
```

Để trực quan hóa dữ liệu, chúng ta lấy một batch ảnh từ train_loader và hiển thị 9 hình đầu tiên. Các ảnh này được gắn nhãn cảm xúc tương ứng dựa trên idx2label, giúp ta có cái nhìn trực quan về dữ liệu mà mô hình sẽ được học.

```
image_batch, label_batch = next(iter(train_loader))

plt.figure(figsize=(10, 10))
for i in range(9):
```

```
  ax = plt.subplot(3, 3, i + 1)
  minv = image_batch[i].numpy().min()
  maxv = image_batch[i].numpy().max()
  plt.imshow(np.squeeze(image_batch[i].numpy()), vmin=minv, vmax=maxv
   , cmap="gray")
  label = label_batch[i]
  plt.title(idx2label[label.item()])
  plt.axis("off")
```


Hình 12: Minh họa trực quan dữ liệu với các nhãn cảm xúc.

## 6. Xây dựng mạng MLP:

```
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)

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)
```

7. **Khai báo hàm loss và optimizer:**

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

8. **Xây dựng hàm tính độ chính xác:**

```
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
```

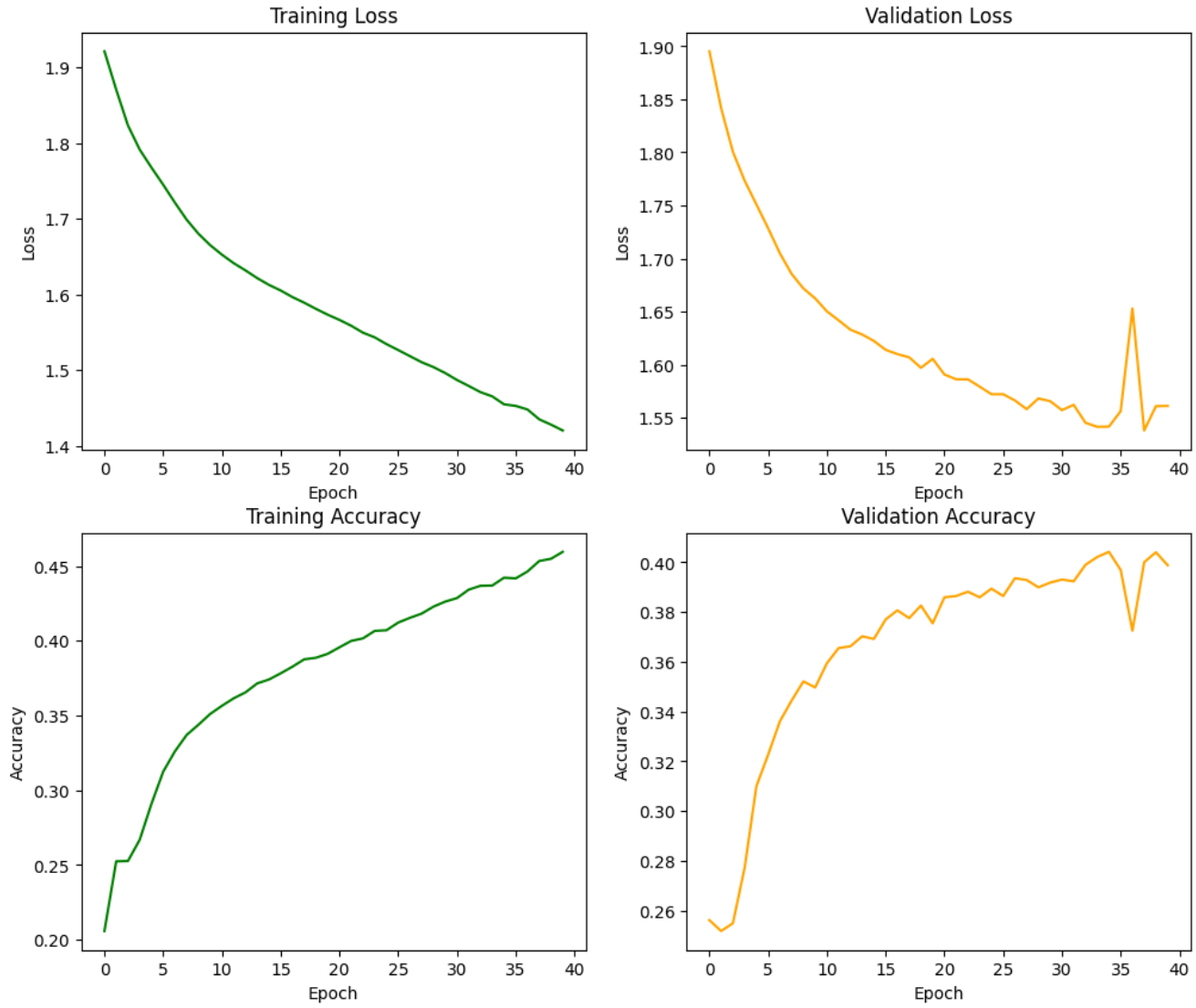
9. **Huấn luyện mô hình:**

```
epochs = 40
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)

    val_loss = 0.0
    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_loss += criterion(outputs, y_samples).item()

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

    val_loss /= len(val_loader)
    val_losses.append(val_loss)

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

    print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_loss:.3f}\
tValidation loss: {val_loss:.3f}')
```

Kết quả huấn luyện được trực quan hóa lên đồ thị như sau:

```
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()
```




Hình 13: Hình ảnh trực quan kết quả huấn luyện trên tập train và val cho bài toán phân loại cảm xúc trên ảnh.

10. **Đánh giá mô hình:**

```
test_target = []
test_predict = []
model.eval()
with torch.no_grad():
    for X_samples, y_samples in test_loader:
        X_samples = X_samples.to(device)
        y_samples = y_samples.to(device)
        outputs = model(X_samples)

        test_predict.append(outputs.cpu())
        test_target.append(y_samples.cpu())

    test_predict = torch.cat(test_predict)
    test_target = torch.cat(test_target)
```

```
    val_acc = compute_accuracy(test_predict, test_target)

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

# B. Phần trắc nghiệm

1. Multilayer perceptron model có các loại layer cơ bản nào:
   (*a*) Input layer.
   (*b*) Hidden layer.
   (*c*) Output layer.
   (*d*) Tất cả đều đúng.

2. Khi một nơ-ron trong mạng MLP có đầu vào $\mathbf{x}$ là $[2, 3]$ và trọng số $\mathbf{w}$ là $[0.5, -0.5]$, bias $\mathbf{b}$ bằng 1 và hàm activation là hàm ReLU, đầu ra là:
   (*a*) 0.5.
   (*b*) 1.0.
   (*c*) 1.5.
   (*d*) 0.0.

3. Trong context của neural networks, **back-propagation** được sử dụng để:
   (*a*) Tính toán đầu vào cho mạng.
   (*b*) Tính toán loss output layer.
   (*c*) Cập nhật trọng số dựa trên loss.
   (*d*) Tăng số lượng hidden layers.

4. Nếu bạn có một MLP model với một hidden layer chứa 5 neurons và output layer có 2 neurons, số trọng số tối thiểu bạn cần là:
   (*a*) 5.
   (*b*) 7.
   (*c*) 10.
   (*d*) 12.

5. Trong quá trình backward, đạo hàm của hàm kích hoạt sigmoid $\sigma(x)$ là:
   (*a*) $\sigma(x)(1 - \sigma(x))$.
   (*b*) $1 - \sigma(x)$.
   (*c*) $\sigma(x) + 1$.
   (*d*) $\sigma(x) - 1$.

6. Phân loại hoa dựa trên chiều dài cánh bằng cách sử dụng MLP model có 1 hidden layer chứa 2 neuron (trọng số $w_{hidden} = [0.5, -0.5]$, và bias $b_{hidden} = [1, -1]$) đi kèm với hàm ReLU và output layer chứa 1 neuron (trọng số $w_{output} = [0.3, -0.2]$, và bias $b_{output} = 0.5$) đi kèm với hàm Sigmoid. Nếu cho input x = 5.1 thì model phân loại thuộc loại nào ($> 0.5$ được xem như loại 1):

| Chiều dài cánh | Loại cánh |
|---|---|
| 5.1 | 0 |
| 6.0 | 1 |
| 5.7 | 0 |

Bảng 1: Bảng dữ liệu phân loại cánh hoa theo chiều dài cánh.

(*a*) Loại 0.

(*b*) Loại 1.

7. Tính kết quả dự đoán giá nhà dựa vào diện tích bằng cách sử dụng MLP model có 1 hidden layer chứa 2 neuron (trọng số $w_{hidden} = [0.5, -0.5]$, và bias $b_{hidden} = [1, -1]$)đi kèm với hàm ReLU và output layer chứa 1 neuron (trọng số $w_{output} = [1.5, 2]$, và bias $b_{output} = 0$). Nếu cho input x = 50 thì model dự đoán giá nhà là bao nhiêu:

| Diện tích | Giá nhà |
|---|---|
| 50 | 100 |
| 60 | 120 |
| 70 | 140 |

Bảng 2: Bảng dữ liệu mô phỏng giá nhà theo diện tích.

(*a*) 39.

(*b*) 390.

(*c*) 93.

(*d*) 930.

8. Dựa vào cài đặt ở **Bài toán 1** trong phần II., hãy điều chỉnh cho phù hợp với nội dung của các câu hỏi dưới đây và chọn câu trả lời chính xác nhất (nội dung các câu hỏi là độc lập với nhau):

   8.1 Khi điều chỉnh kiến trúc class "MLP" thành dạng **Linear Regression** và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có điểm R2 xấp xỉ bằng:

   (*a*) 0.773.

   (*b*) 0.664.

   (*c*) 0.555.

   (*d*) 0.666.

   8.2 Khi điều chỉnh các hàm kích hoạt được sử dụng trong class "MLP" thành hàm **Sigmoid** và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có điểm R2 xấp xỉ bằng:

(*a*) 0.84.
(*b*) 0.85.
(*c*) 0.86.
(*d*) 0.87.

8.3 Khi điều chỉnh các hàm kích hoạt được sử dụng trong class "MLP" thành hàm **Tanh** và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có điểm R2 xấp xỉ bằng:
(*a*) 0.84.
(*b*) 0.85.
(*c*) 0.86.
(*d*) 0.87.

9. Dựa vào cài đặt ở **Bài toán 2** trong phần II., hãy điều chỉnh cho phù hợp với nội dung của các câu hỏi dưới đây và chọn câu trả lời chính xác nhất (nội dung các câu hỏi là độc lập với nhau):

9.1 Khi điều chỉnh kiến trúc class "MLP" thành dạng **Softmax Regression** và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:
(*a*) 0.60.
(*b*) 0.61.
(*c*) 0.62.
(*d*) 0.63.

9.2 Khi điều chỉnh các hàm kích hoạt được sử dụng trong class "MLP" thành hàm **Sigmoid** và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:
(*a*) 0.65.
(*b*) 0.66.
(*c*) 0.67.
(*d*) 0.68.

9.3 Khi điều chỉnh các hàm kích hoạt được sử dụng trong class "MLP" thành hàm **Tanh** và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:
(*a*) 0.90.
(*b*) 0.91.
(*c*) 0.92.
(*d*) 0.93.

9.4 Khi điều chỉnh các hàm kích hoạt được sử dụng trong class "MLP" thành hàm **Tanh**, hidden_dims = 512 và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:
(*a*) 0.90.
(*b*) 0.91.

(*c*) 0.92.

(*d*) 0.93.

10. Dựa vào cài đặt ở **Bài toán 3** trong phần II., hãy điều chỉnh cho phù hợp với nội dung của các câu hỏi dưới đây và chọn câu trả lời chính xác nhất (nội dung các câu hỏi là độc lập với nhau):

    10.1 Khi bỏ phép normalization trên ảnh và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:

    (*a*) 0.9.

    (*b*) 0.11.

    (*c*) 0.13.

    (*d*) 0.15.

    10.2 Khi bỏ hidden layer thứ 3 trong class "MLP" và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:

    (*a*) 0.41.

    (*b*) 0.42.

    (*c*) 0.43.

    (*d*) 0.44.

    10.3 Khi điều chỉnh số hidden_dims của toàn bộ hidden layer trong class "MLP" bằng nhau và với cùng một cài đặt tham số như phiên bản gốc, kết quả của mô hình đã huấn luyện được trên tập test có độ chính xác xấp xỉ bằng:

    (*a*) 0.41.

    (*b*) 0.42.

    (*c*) 0.43.

    (*d*) 0.44.

# III. Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại đây.

2. **Hint:** Các file code gợi ý có thể được tải tại đây.

3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

4. **Rubric:**

| Mục | Kiến Thức | Đánh Giá |
|---|---|---|
| 1 | - Bài toán về regression.<br>- Làm việc với data dạng bảng được lưu dưới dạng file .csv.<br>- Chuẩn hóa data theo kiểu Standardization.<br>- Dùng PyTorch để build, compile, train và evaluate model dự đoán theo regresion task.<br>- Kiến thức về xây dựng MLP cho regression task.<br>- Metric để đánh giá regression task. | - Hiểu và review lại kiến thức về bài toán regression.<br>- Biết cách thao tác với data dạng bảng và lưu dạng file csv bằng thư viện Pandas.<br>- Thực hiện code được chuẩn hóa theo kiểu Standardization.<br>- Đã có thể dùng được PyTorch để thực hiện các bước build, compile, train và evaluate data cho regression task với MLP model. |
| 2 | - Bài toán về classification.<br>- Làm việc với data được lưu dưới dạng file .npy.<br>- Sử dụng NumPy để load và xử lý data.<br>- Dùng pytorch để build, compile, train và evaluate model phân loại.<br>- Ôn tập về Softmax Regression.<br>- Kiến thức về xây dựng MLP cho classification task. | - Hiểu và review lại kiến thức về bài toán classification.<br>- Biết cách thao tác với data dạng file npy bằng thư viện NumPy.<br>- Đã có thể dùng được PyTorch để thực hiện các bước build, compile, train và evaluate data cho classification task với MLP model. |
| 3 | - Bài toán về image classification.<br>- Làm việc với data dạng ảnh.<br>- Load và xử lý data dạng ảnh với PyTorch.<br>- Dùng PyTorch để build, compile, train và evaluate model phân loại ảnh.<br>- Kiến thức về xây dưng MLP cho image classification. | - Hiểu và review lại kiến thức về bài toán classification.<br>- Biết cách thao tác với data dạng ảnh bằng PyTorch.<br>- Đã có thể dùng được PyTorch để thực hiện các bước build, compile, train và evaluate data cho classification ảnh với MLP model |

***- Hết -***