# Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU[1]

[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

1 Evaluating and Using Models on New Data

2 Essential Deep Learning Tips & Tricks

3 Improving Convergence with Batch Normalization

1 Evaluating and Using Models on New Data

2 Essential Deep Learning Tips & Tricks

3 Improving Convergence with Batch Normalization

# Choosing Activation Functions

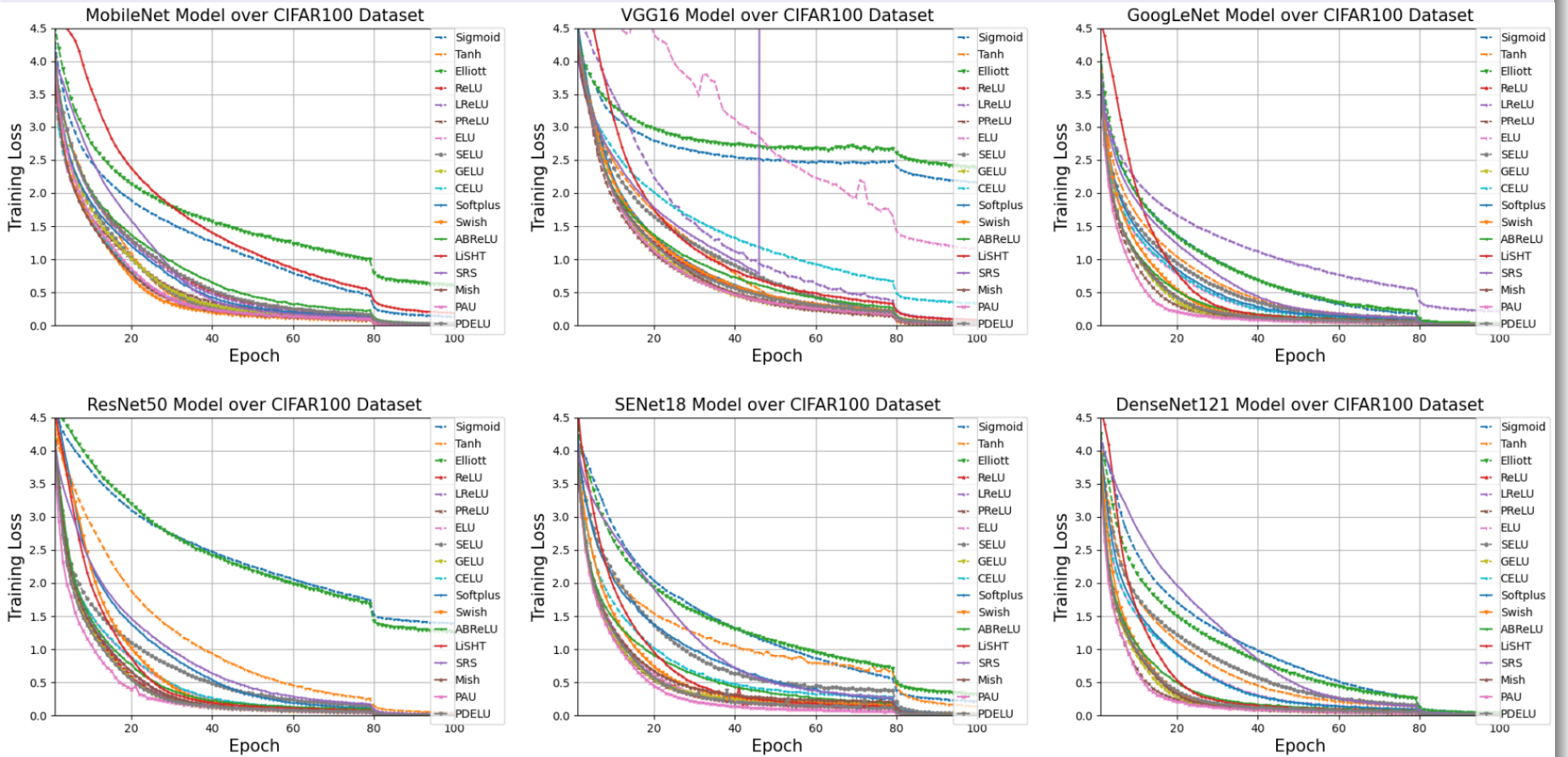
Hiện tại, có rất nhiều hàm kích hoạt có thể được sử dụng trong lớp ẩn của mạng nơ-ron. Tuy nhiên, không có hàm kích hoạt nào là tốt nhất cho tất cả tác vụ.

```
class PyTorchMLP(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super().__init__()

        self.all_layers = torch.nn.Sequential(

            # 1st hidden layer
            torch.nn.Linear(num_features, 100),
            torch.nn... # ACTIVATION FUNCTION

            # 2nd hidden layer
            torch.nn.Linear(100, 50),
            torch.nn... # ACTIVATION FUNCTION

            # output layer
            torch.nn.Linear(50, num_classes),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

Chúng ta có thể thử nghiệm nhiều hàm kích hoạt khác nhau và chọn hàm kích hoạt tốt nhất cho tác vụ cụ thể.

# Choosing Activation Functions

Đồ thị hội tụ của các hàm kích hoạt khác nhau trên tập dữ liệu CIFAR100:



# Choosing Activation Functions

## Logistic sigmoid function

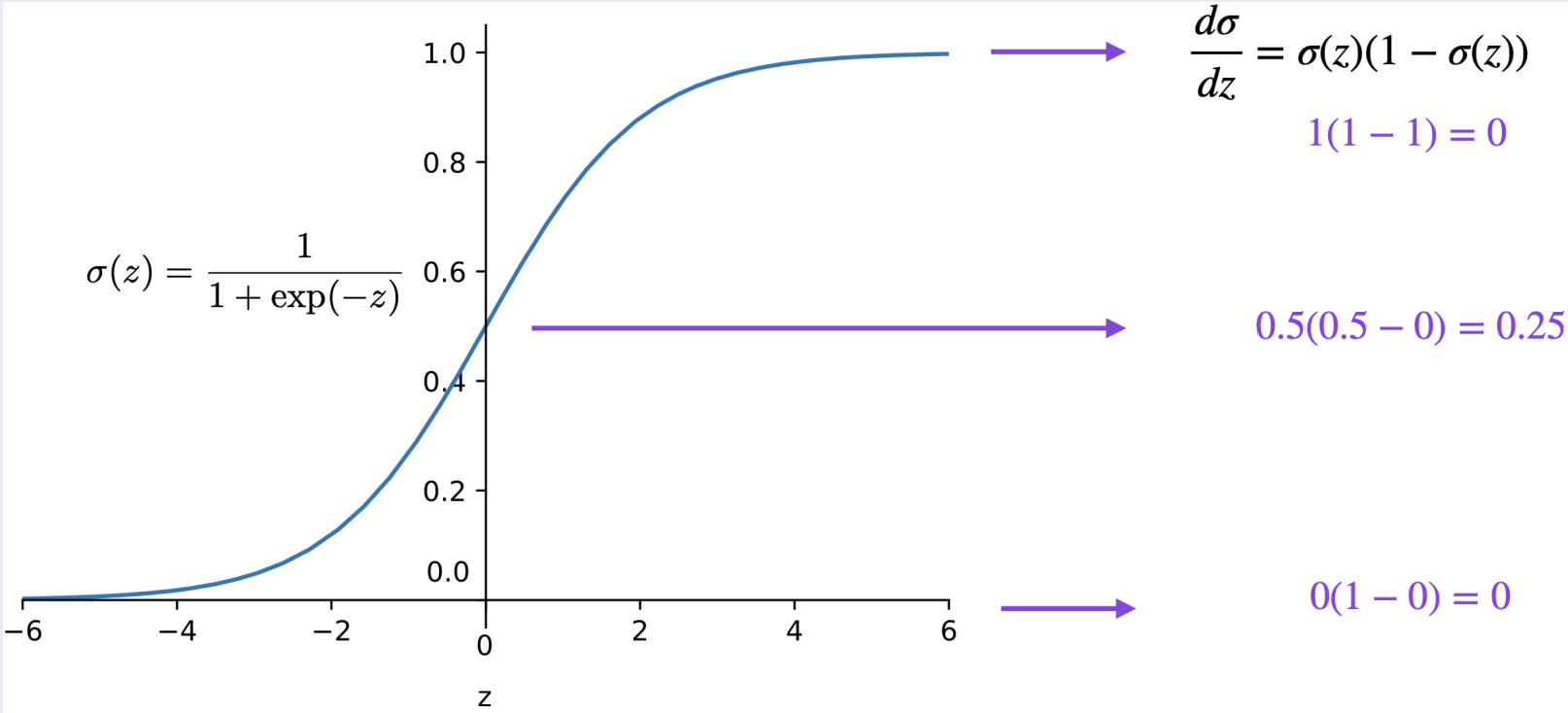
Hàm kích hoạt quen thuộc trong các chương trước là *Logistic sigmoid function* nhưng chúng ta không nên sử dụng nó trong **lớp ẩn** của các mạng nơ-ron sâu vì quá trình huấn luyện chậm hội tụ do:

- **Vấn đề biến mất gradient**: Đạo hàm của hàm sigmoid rất nhỏ khi đầu vào xa 0, dẫn đến việc gradient biến mất.
- **Không tốt cho việc khởi tạo trọng số**: Hàm sigmoid tạo ra gradient nhỏ khi đầu vào xa 0, dẫn đến việc khởi tạo trọng số không tốt.
- **Không zero-centered**: Đầu ra của hàm sigmoid không zero-centered, dẫn đến việc các trọng số cập nhật không hiệu quả.

# Choosing Activation Functions

## Logistic sigmoid function

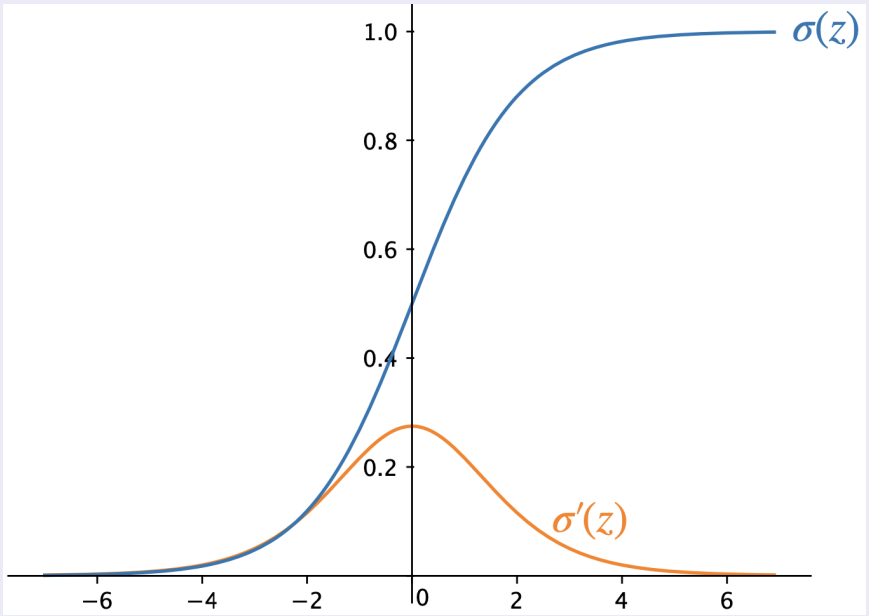
Hàm kích hoạt *Logistic sigmoid function* không nên sử dụng nó trong **lớp ẩn** của các mạng nơ-ron sâu vì quá trình huấn luyện chậm hội tụ



# Choosing Activation Functions

## Logistic sigmoid function

Hàm kích hoạt *Logistic sigmoid function* không nên sử dụng nó trong **lớp ẩn** của các mạng nơ-ron sâu vì quá trình huấn luyện chậm hội tụ

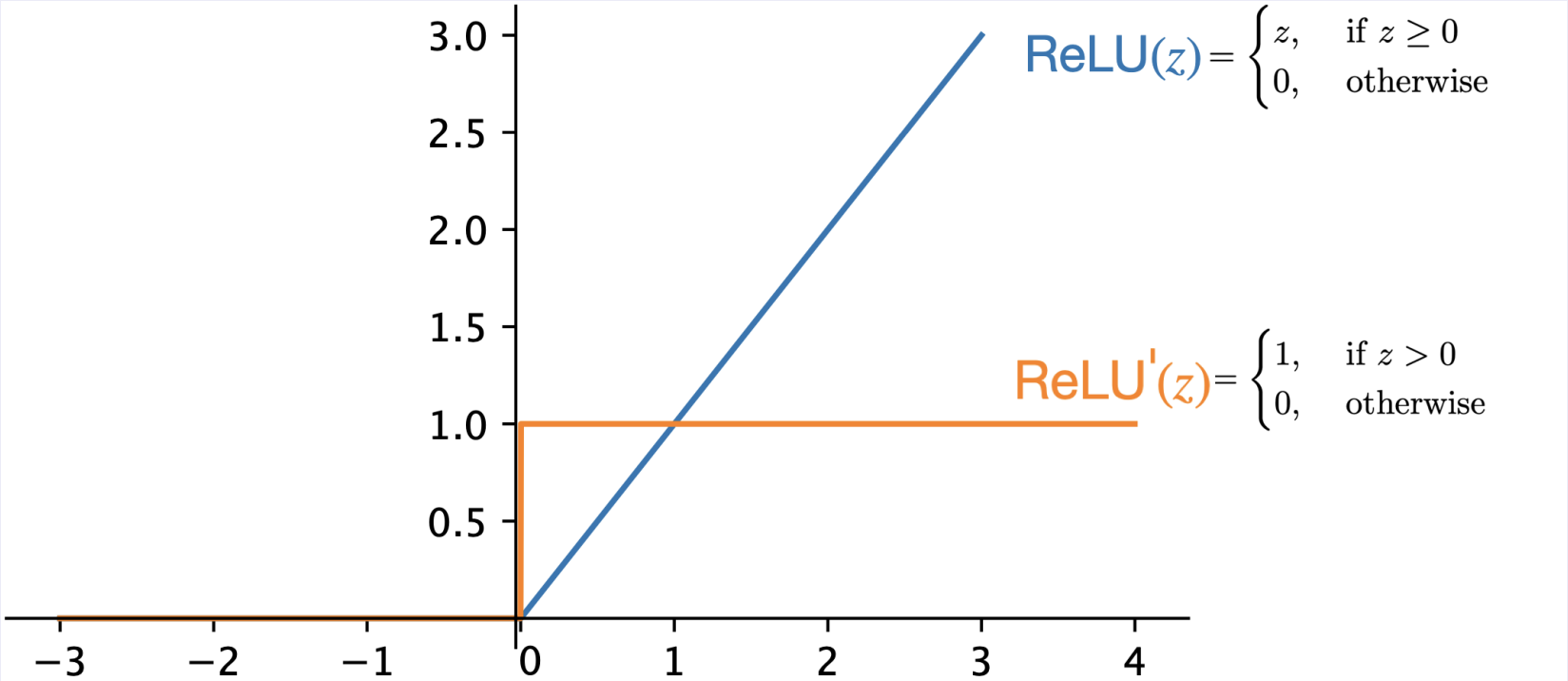



The biggest sigmoid derivative is **0.25**. This can mess up the error signal in backpropagation.

# Choosing Activation Functions

## Rectified Linear Unit-ReLU

Thay vào đó, chúng ta có thể sử dụng hàm kích hoạt khác trong lớp ẩn như *ReLU*




ReLU is a simple & robust choice for hidden layers.

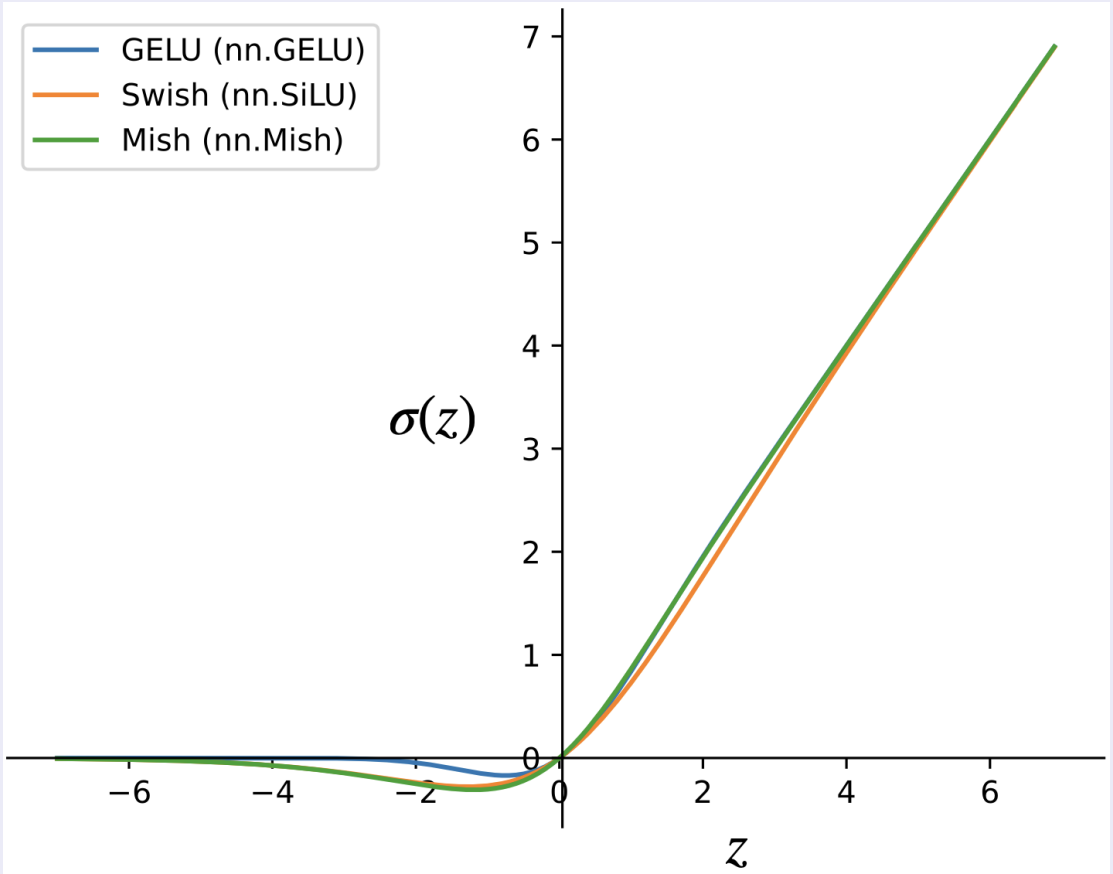
# Choosing Activation Functions

## Rectified Linear Unit-ReLU

```
class PyTorchMLP(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super().__init__()

        self.all_layers = torch.nn.Sequential(

            # 1st hidden layer
            torch.nn.Linear(num_features, 100),
            torch.nn.ReLU() # ACTIVATION FUNCTION

            # 2nd hidden layer
            torch.nn.Linear(100, 50),
            torch.nn.ReLU() # ACTIVATION FUNCTION

            # output layer
            torch.nn.Linear(50, num_classes),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

# Choosing Activation Functions

Ngoài ra còn có một số hàm kích hoạt khác đang là xu thế như *Leaky ReLU, Parametric ReLU, Exponential Linear Unit (ELU), Swish, GELU, Softplus, Softsign, Tanh, Mish ...*



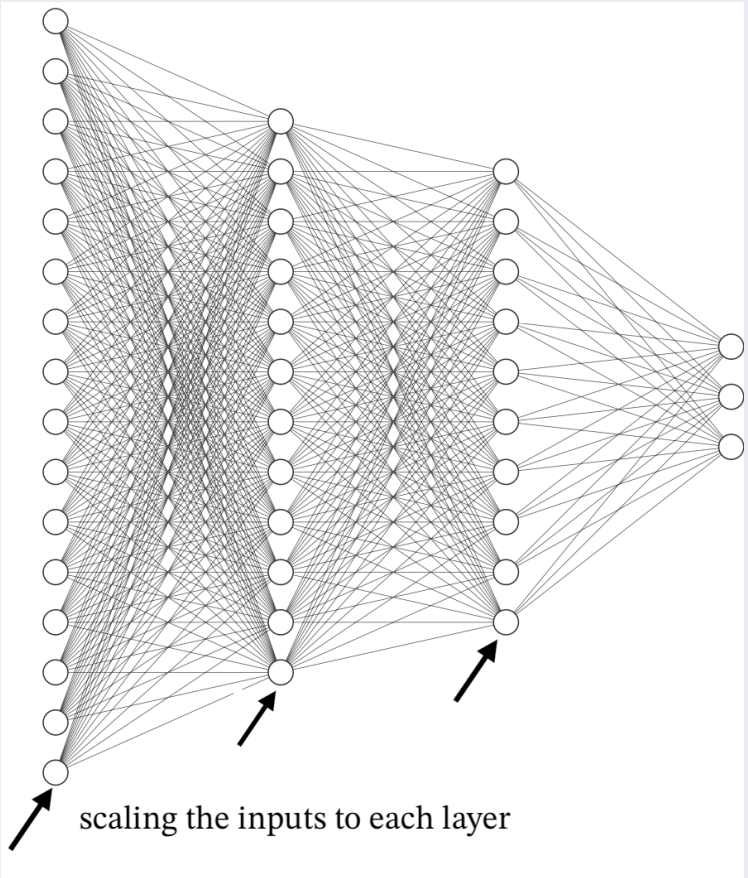
1 Evaluating and Using Models on New Data

2 Essential Deep Learning Tips & Tricks

3 Improving Convergence with Batch Normalization

# Scaling Layer Inputs Using BatchNorm

*Batch normalization (BatchNorm)* can improve convergence and reduce overfitting by scaling the inputs to each layer so that they have *a mean of 0* and *a standard deviation of 1*.



# Scaling Layer Inputs Using BatchNorm

## Standardizing the input features

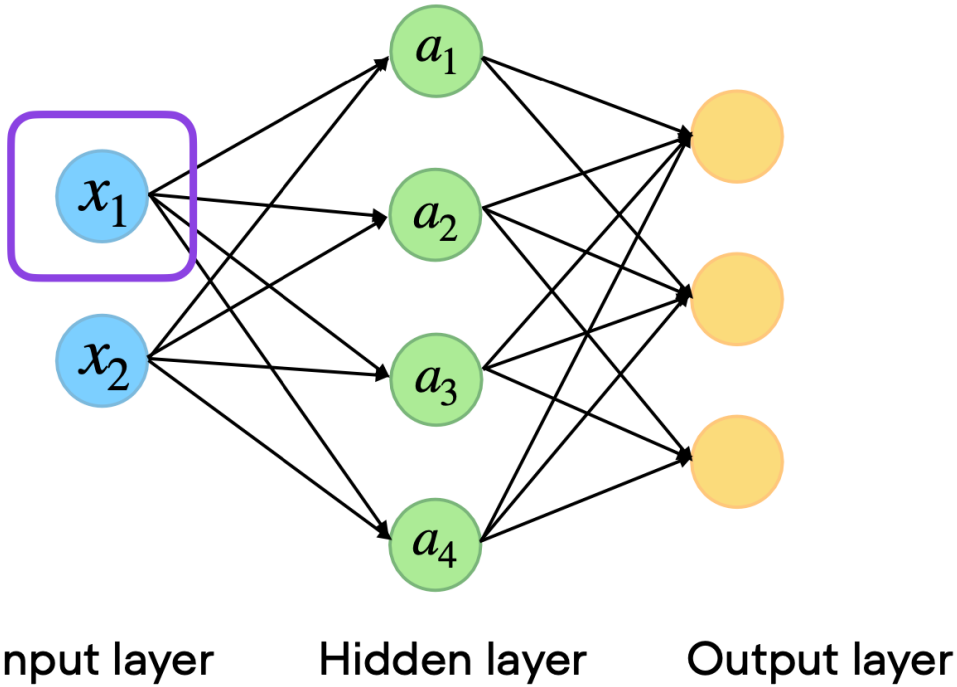
Trong các chương trước, chúng ta đã nhắc đến tầm quan trọng của việc chuẩn hoá cho các đặc trưng đầu vào. Đối với mạng nơ-ron nhiều lớp, ta thường phải áp dụng chuẩn hoá đầu vào từng lớp ẩn dựa trên công thức ở Chuong_3.3.pdf.

$$standar(x_j^{[i]}) = \frac{x_j^{[i]} - mean(x_j)}{std(x_j)}.$$

# Scaling Layer Inputs Using BatchNorm

## Các bước chuẩn hoá dữ liệu trong mạng nhiều lớp

Bước 1: chuẩn hoá đặc trưng đầu vào $x_1, x_2$.



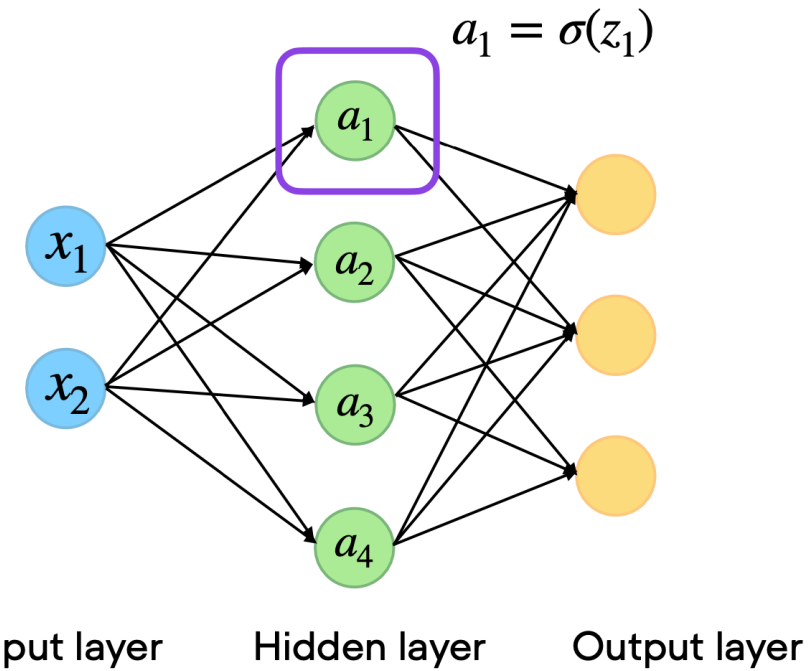

Hình 1: Giả sử xem xét cột đặc trưng đầu vào $x_1$, $x_1^{'[i]} = \frac{x_1^{[}i] - mean(x_1)}{std(x_1)}$

# Scaling Layer Inputs Using BatchNorm

## Các bước chuẩn hoá dữ liệu trong mạng nhiều lớp

Bước 2: chuẩn hoá hidden layer activations.

1. Standardize net inputs $z_1, z_2, z_3, z_4$
2. Pre-activation scaling



# Scaling Layer Inputs Using BatchNorm

## Các bước chuẩn hoá dữ liệu trong mạng nhiều lớp

Bước 2: chuẩn hoá hidden layer activations.

1. Standardize net inputs $z_1, z_2, z_3, z_4$

$$z_1^{'[i]} = \frac{z_1^[i] - mean(z_1)}{std(z_1) + \epsilon},$$

where $\epsilon$ is a small constant to avoid division by zero.

# Scaling Layer Inputs Using BatchNorm

## Các bước chuẩn hoá dữ liệu trong mạng nhiều lớp

Bước 2: chuẩn hoá hidden layer activations.

1. Standardize net inputs $z_1, z_2, z_3, z_4$

$$z_1^{'[i]} = \frac{z_1^[i] - mean(z_1)}{std(z_1) + \epsilon}.$$

2. Pre-activation scaling

$$a_1^{'[i]} = \gamma_1 z_1^{'[i]} + \beta_1,$$

where $\gamma_1$ and $\beta_1$ are learnable parameters. If $\gamma_1 = 1$ and $\beta_1 = 0$, the hidden layer activation is the same as the standardized net input.

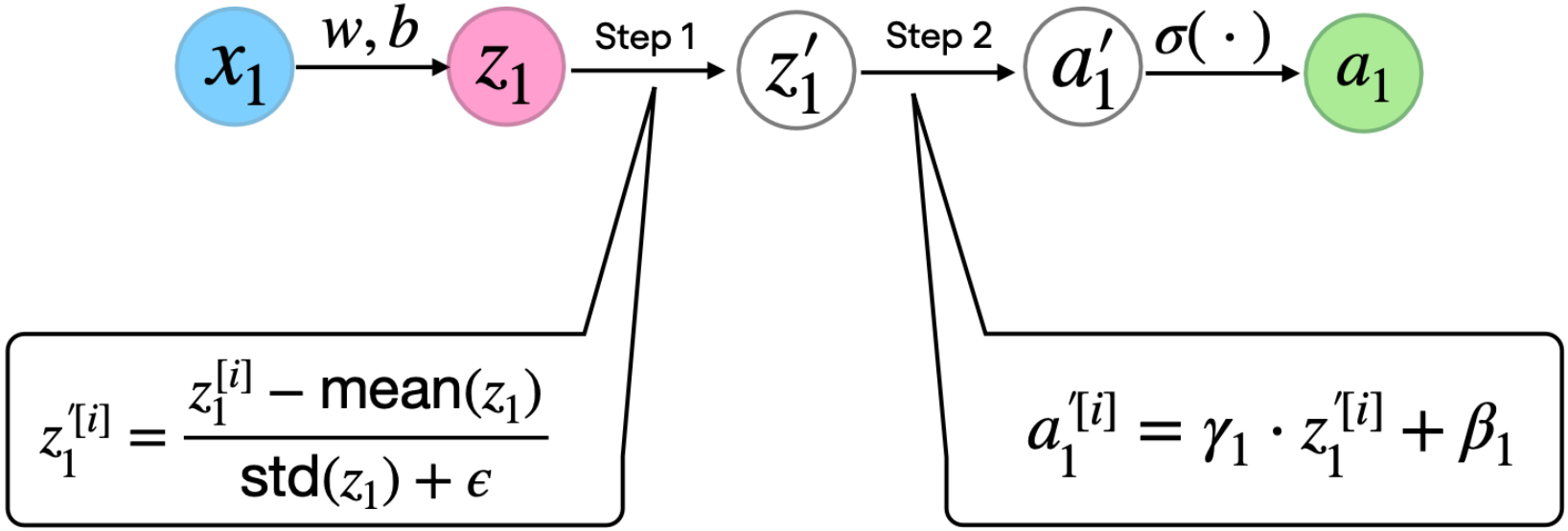
# Scaling Layer Inputs Using BatchNorm

## Các bước chuẩn hoá dữ liệu trong mạng nhiều lớp

Bước 3: tính giá trị hàm kích hoạt:

$$a_1 = \sigma(a_1^{'[i]}).$$

Áp dụng tương tự cho $a_2, a_3, a_4$ và các lớp ẩn khác.



# BatchNorm in PyTorch

Trong PyTorch, chúng ta có thể sử dụng các lớp BatchNorm như:

- **BatchNorm1d**: BatchNorm cho dữ liệu 1 chiều (dữ liệu đầu vào dạng flatten 1 chiều).
- **BatchNorm2d**: BatchNorm cho dữ liệu 2 chiều (thường dữ liệu là ảnh).
- **BatchNorm3d**: BatchNorm cho dữ liệu 3 chiều (thường dữ liệu đầu vào là video-tập hợp các frame).

# BatchNorm in PyTorch

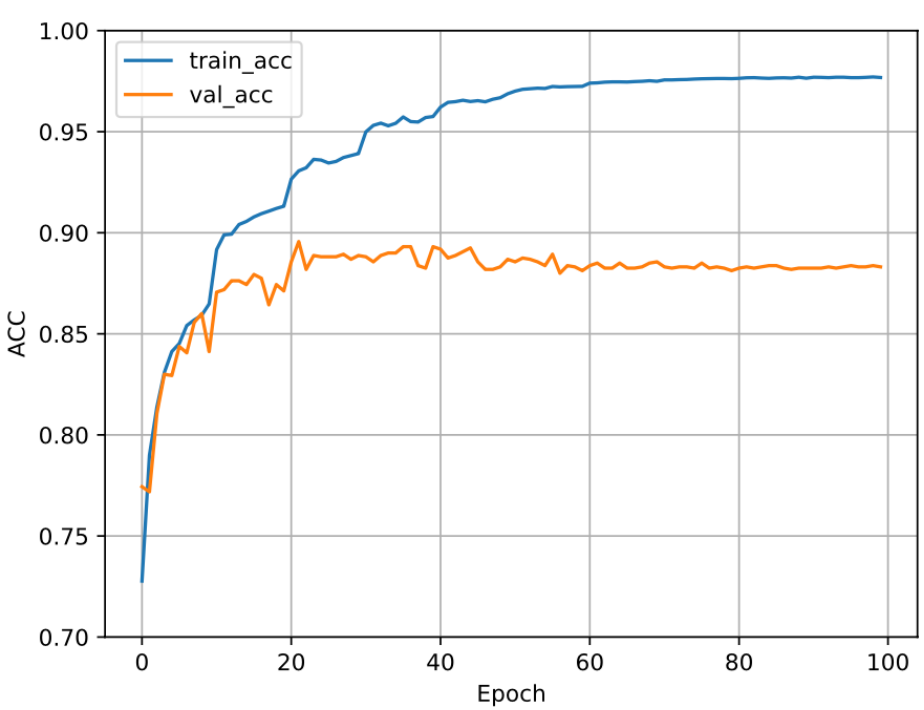
100 because needs
100 $\gamma$ and 100 $\beta$ values

```python
class PyTorchMLP(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(num_features, 100),
            torch.nn.BatchNorm1d(100),
            torch.nn.ReLU(),

            # 2nd hidden layer
            torch.nn.Linear(100, 50),
            torch.nn.BatchNorm1d(50),
            torch.nn.ReLU(),

            # output layer
            torch.nn.Linear(50, num_classes),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

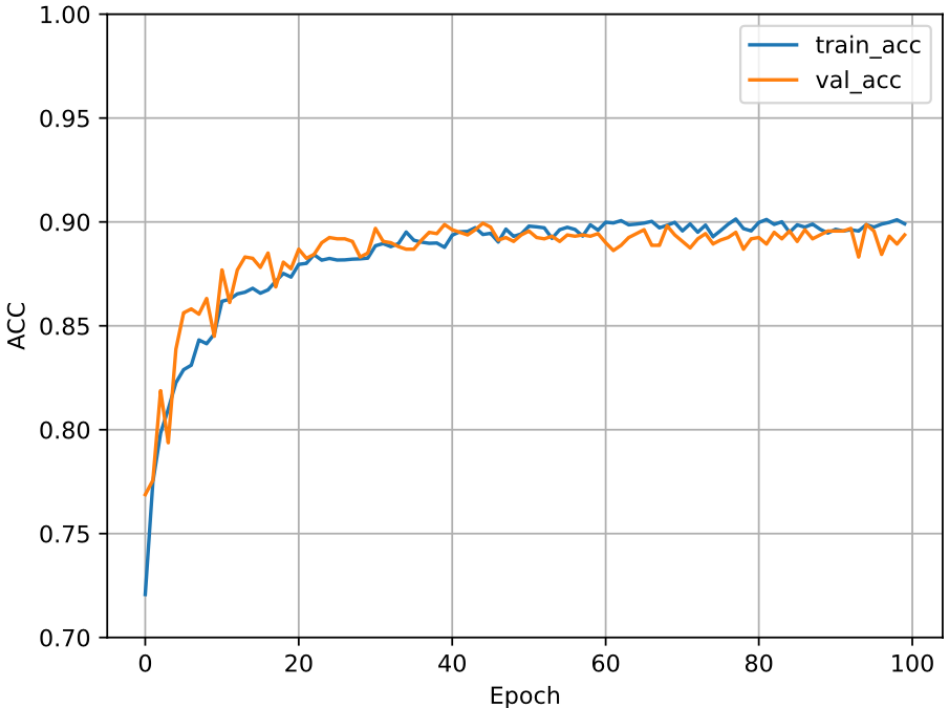
Hình 2: Ví dụ BatchNorm1D trong PyTorch

# BatchNorm in PyTorch

BatchNorm có thể giúp cải thiện quá trình hội tụ và giảm overfitting. Ví dụ như trường hợp bên dưới




(a) without BatchNorm




(b) with BatchNorm

Hình 3: Sử dụng BatchNorm để giảm overfitting

# BatchNorm in PyTorch

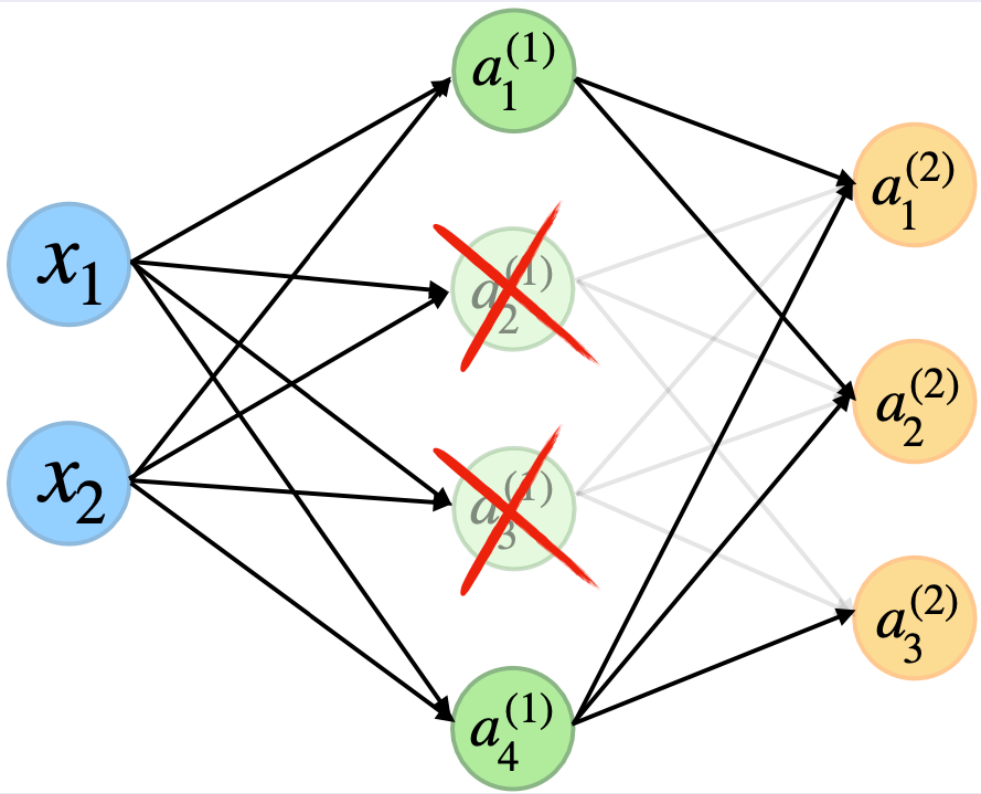
So sánh BatchNorm trong *quá trình huấn luyện* và *inference*

- BatchNorm trong quá trình huấn luyện sử dụng thống kê (mean, variance) của mỗi batch và đồng thời cập nhật *moving average*.
- BatchNorm trong quá trình inference *sử dụng giá trị trung bình có trọng số của moving average* để chuẩn hoá dữ liệu do quá trình dự đoán thường không có batch hoặc có batch khác xa với quá trình huấn luyện.

PyTorch duy trì moving average của mean và variance của tất cả các batch, khi dự đoán ta sử dụng *model.eval()*

# Reducing overfitting with Dropout

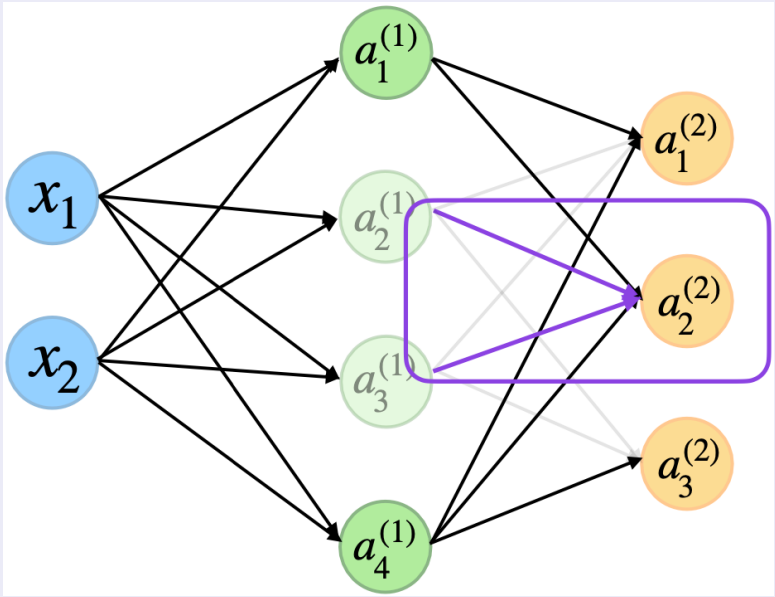
Dropout về cơ bản là một kỹ thuật bỏ ngẫu nhiên một số lượng các nơ-ron trong quá trình huấn luyện để giảm overfitting.




*Lưu ý: Dropout chỉ được sử dụng trong quá trình huấn luyện, không được sử dụng trong quá trình inference.*

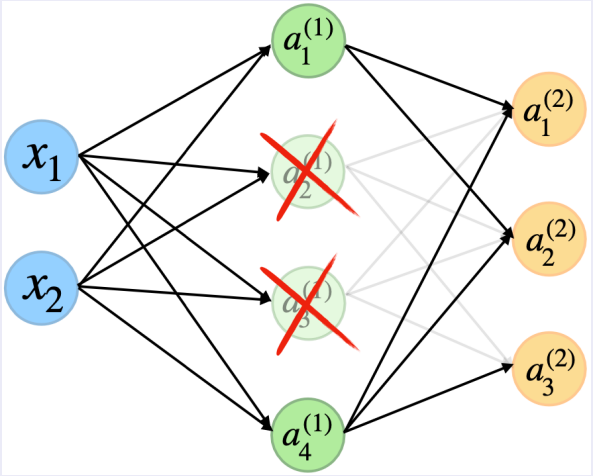
# Reducing overfitting with Dropout

Do có sự khác nhau trong sử dụng dropout giữa quá trình huấn luyện và inference, chúng ta cần phải thay đổi giá trị của các nodes còn lại khi sử dụng dropout. Ví dụ: giả sử ta có *drop probability* $p = 0.5$ (chỉ có $1 - p = 50\%$ nodes hoạt động), nhưng trong quá trình dự đoán vẫn đủ các nodes đóng góp vào mạng, điều này dẫn đến giá trị của các nodes (*node activations*) tăng lên gấp 2 lần trong quá trình dự đoán.



# Reducing overfitting with Dropout

*Để khắc phục sự chênh lệch của các node activations*, đầu ra của mỗi neuron trong quá trình huấn luyện được scale với $1/p$ nhằm đảm bảo tính *consistent* của mạng.

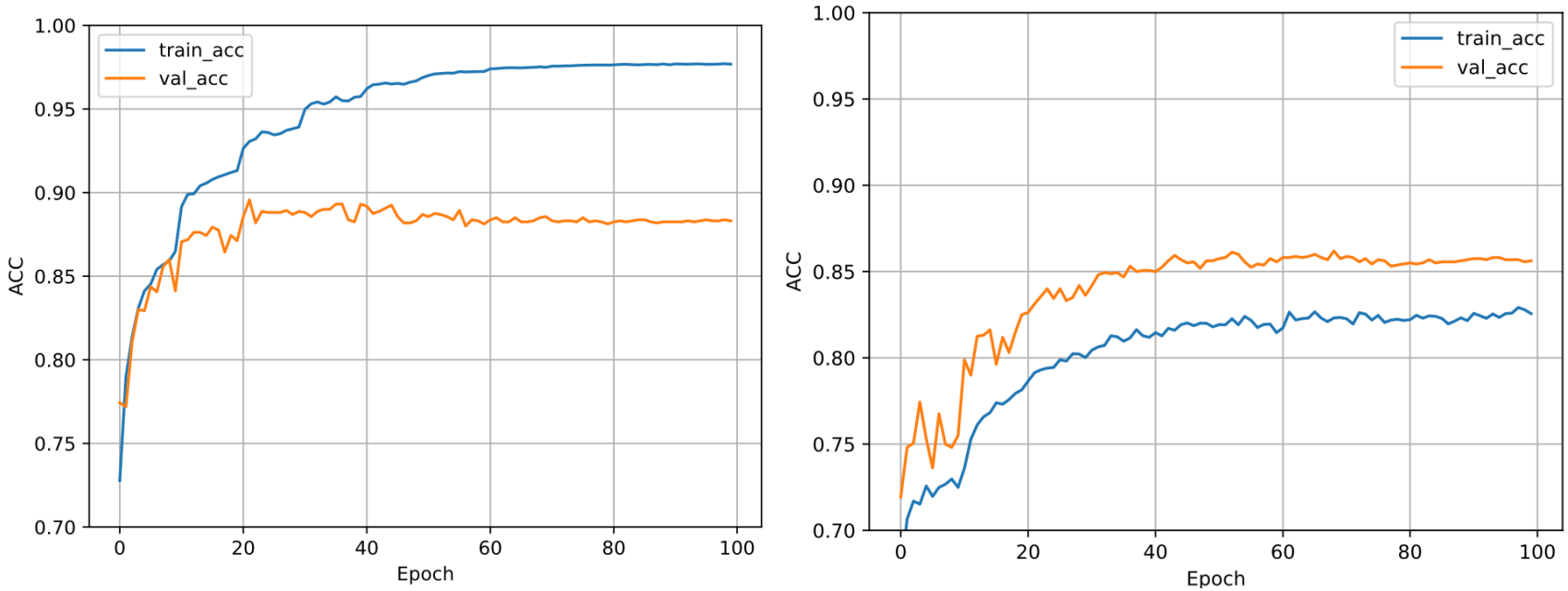



Giả sử ta có $p = 0.5$, $a_1 = 1, a_2 = 2, a_3 = 3, a_4 = 4$, nếu drop $a_2, a_3$, thì đầu ra của quá trình huấn luyện là $a = 5/p = 10$. Quá trình dự đoán, tất cả node sẽ đóng góp vào đầu ra $a = a_1 + a_2 + a_3 + a_4 = 10$. Ngoài ra, ta cũng có thể dùng cách khác bằng cách scale quá trình dự đoán với $1 - p$.

# Dropout in PyTorch

```
class PyTorchMLP(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(num_features, 100),
            torch.nn.ReLU(),
            torch.nn.Dropout(0.2),

            # 2nd hidden layer
            torch.nn.Linear(100, 50),
            torch.nn.ReLU(),
            torch.nn.Dropout(0.5),

            # output layer
            torch.nn.Linear(50, num_classes),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits
```

# Dropout in PyTorch




(a) without Dropout (b) with Dropout

Hình 4: Sử dụng Dropout để giảm overfitting

# Tài liệu tham khảo


Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.

Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions and Answers on Machine Learning and AI (2024). ISBN-13: 978-1-7185-0377-9 (ebook).

LightningAI
LightningAI: PyTorch Lightning (2024) .