Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

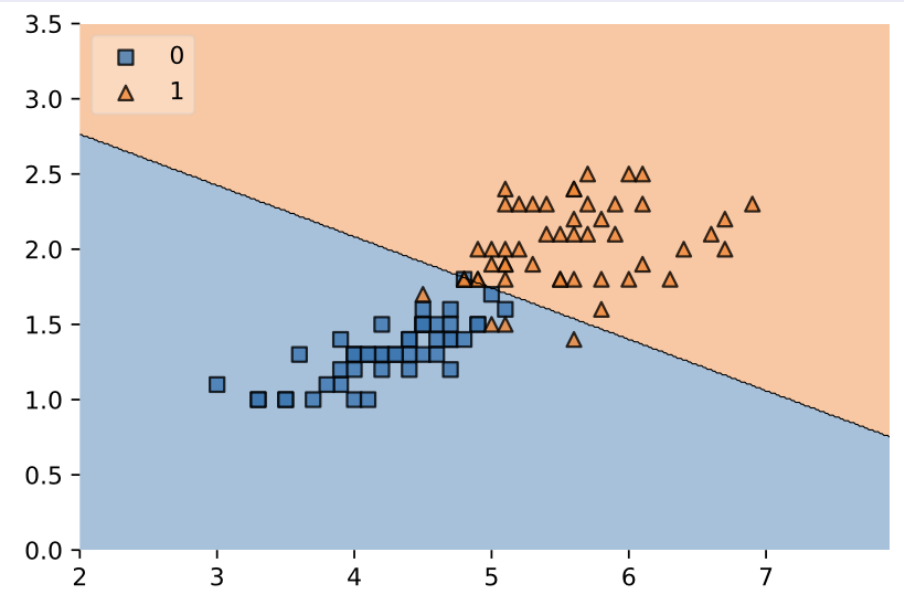
NGUYỄN HẢI TRIỀU[1]

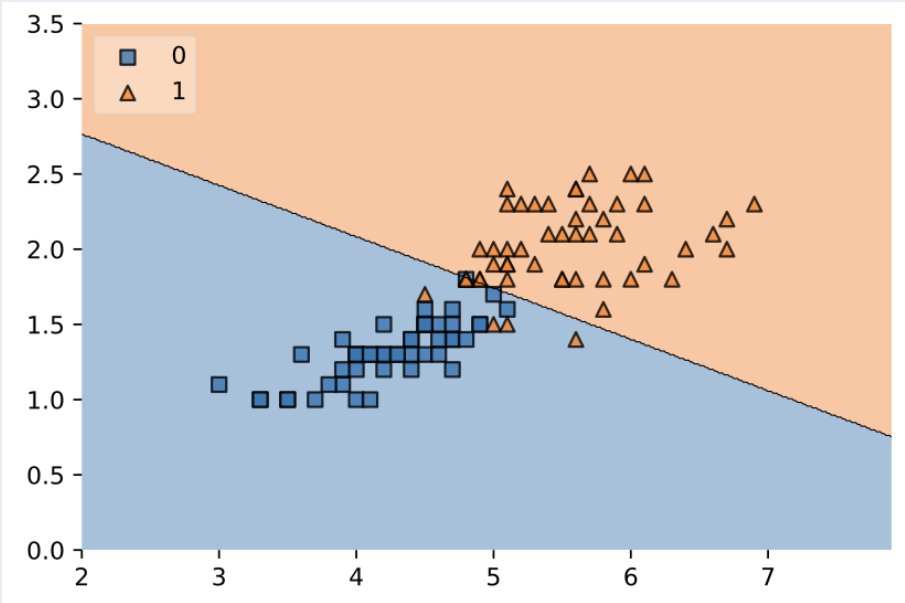
[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

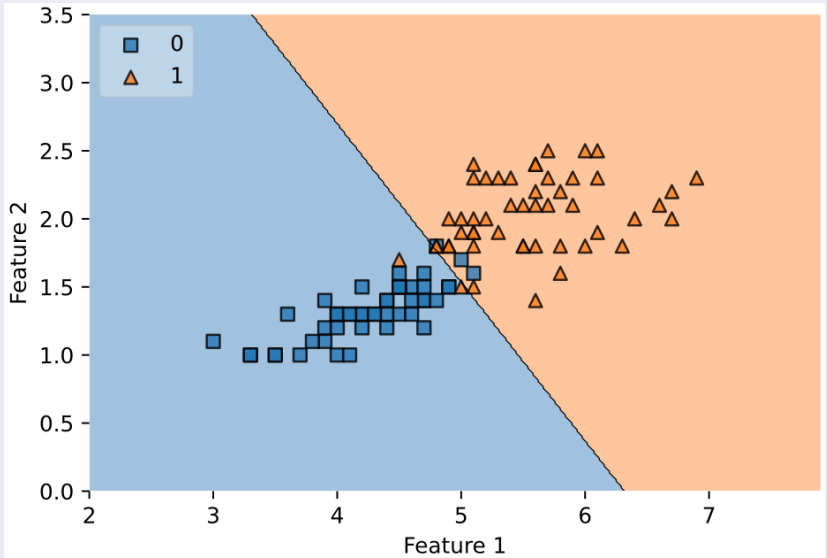
Trong phần này chúng ta sẽ tìm hiểu tại sao cần sử dụng Multilayer Neural Networks. Quay lại bài toán phân loại sử dụng Perceptron có nhược điểm rất lớn là việc *xác định Decision boundary sẽ không bao giờ hội tụ trên tập dữ liệu huấn luyện không phân tách tuyến tính*.



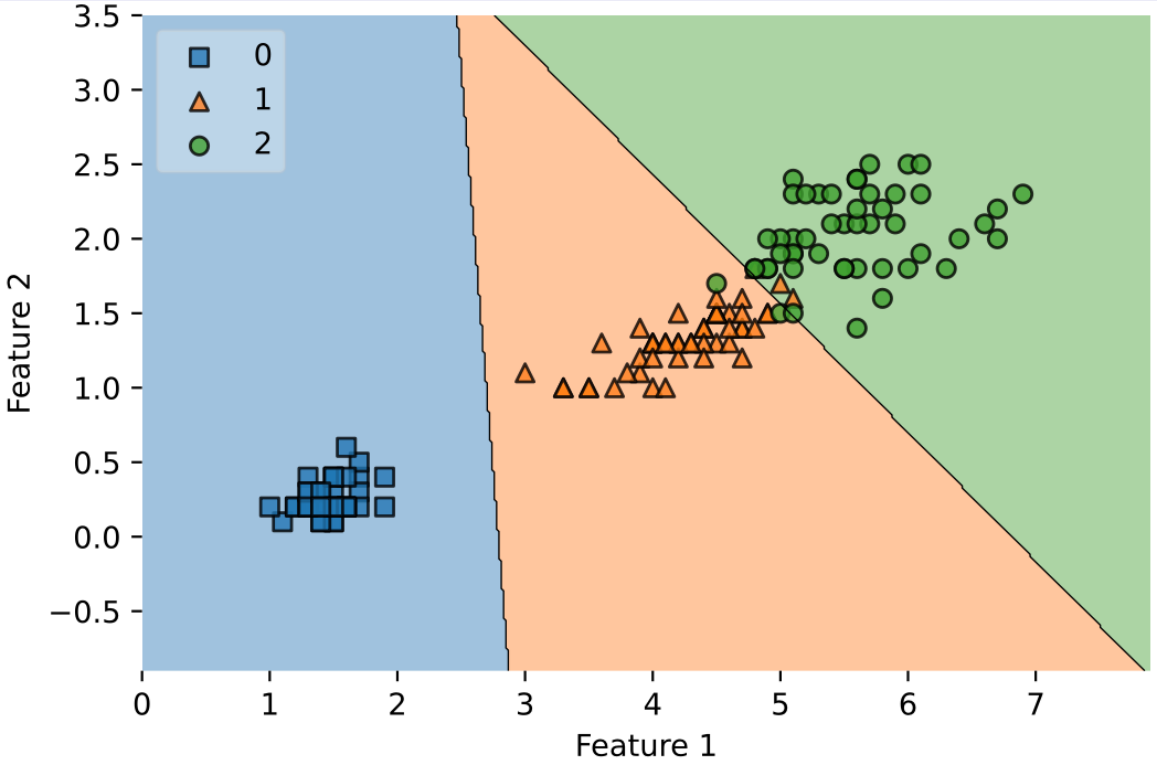


Để khắc phục *nhược điểm của Perceptron*, chúng ta sử dụng Logistic regression bằng cách *thêm vào Differentiable loss + SGD*

Mặc dù Logistic regression luôn hội tụ về loss minimum, nhưng nó cũng có nhược điểm là xử lý được 2 lớp.

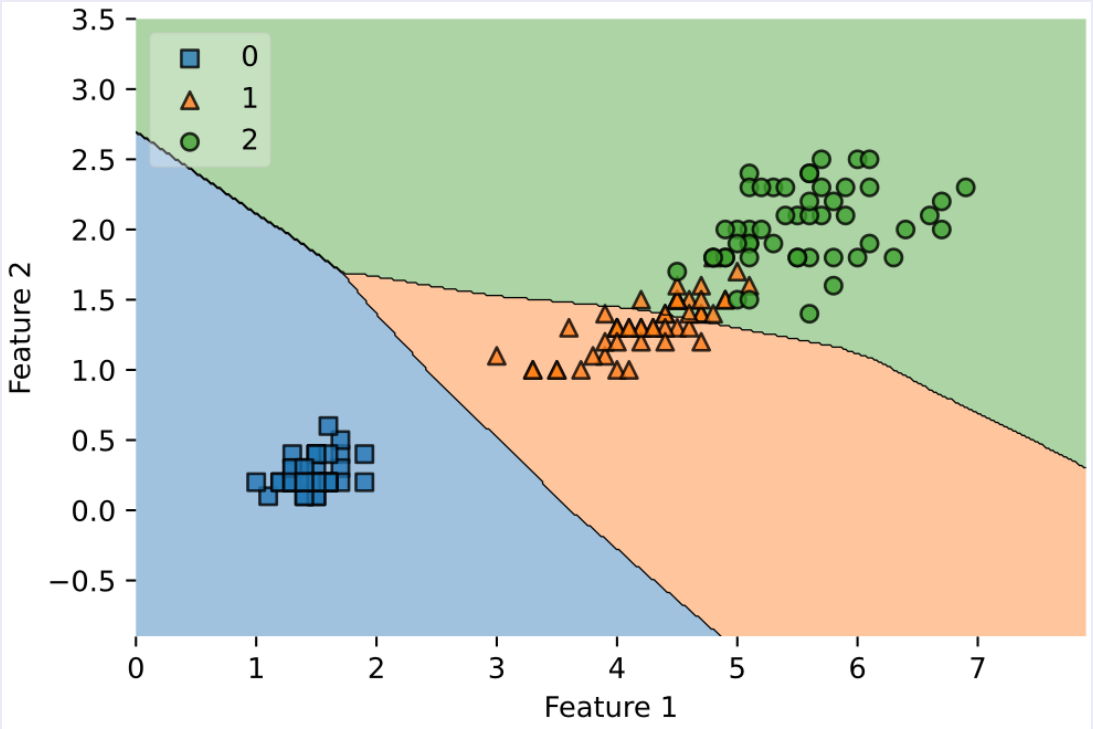


Để giải quyết vấn đề *có multiple classes*, chúng ta sử dụng Softmax regression bằng cách *thêm vào multiple output nodes và Cross Entropy Loss*



Tuy nhiên, Softmax regression chỉ có thể tạo ra Linear decision boundary.

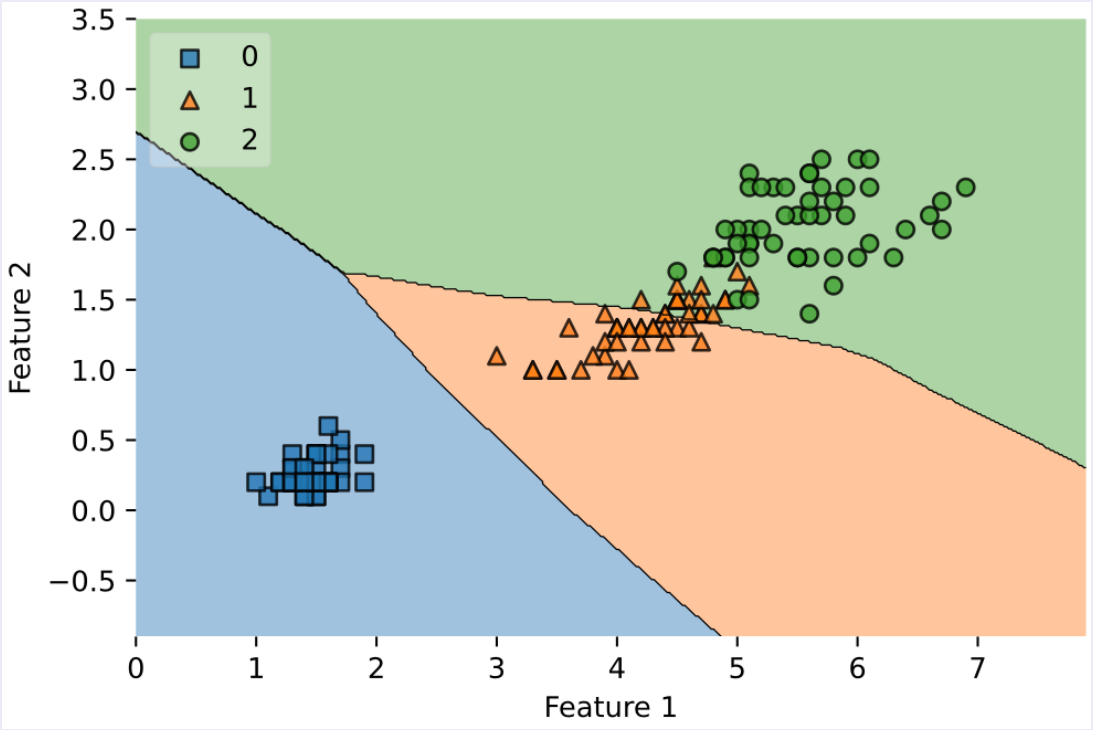
Để khắc phục nhược điểm của *softmax regression*, chúng ta sử dụng *multiple hidden layers* để tạo thành Multilayer perceptron model.

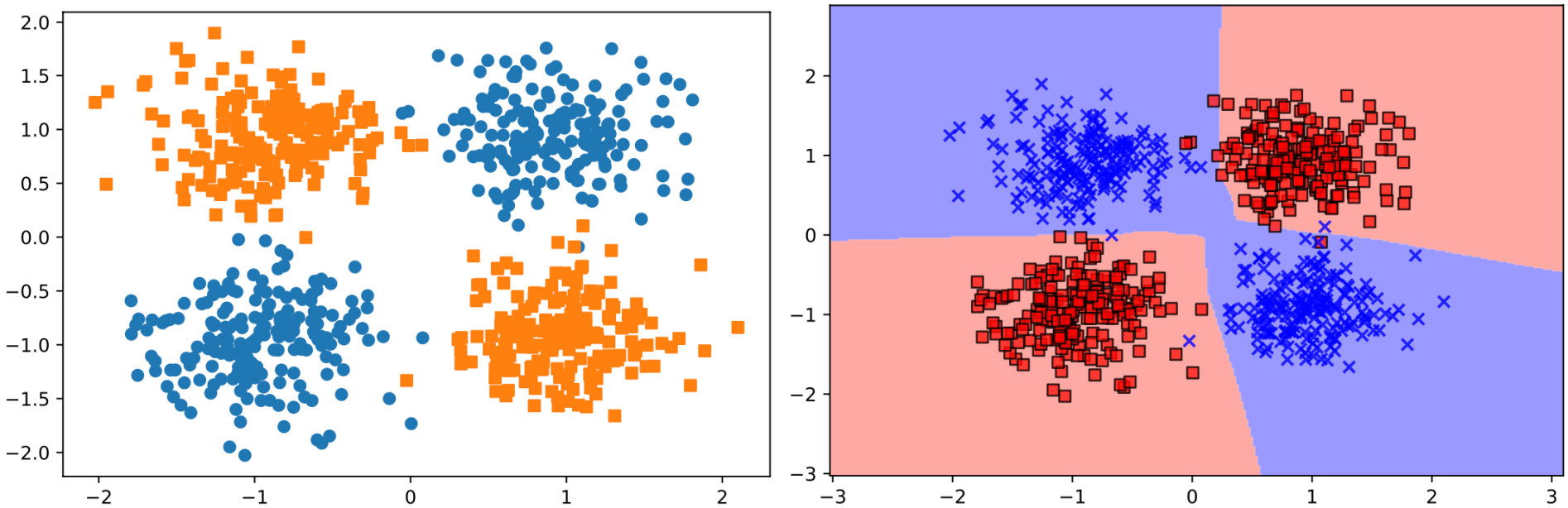


Ưu điểm của Multilayer perceptron model:

- can be deal with multiple classes
- can generate complex decision boundaries

Tuy nhiên *multilayer perceptron* yêu cầu cần nhiều data hơn nhằm hiệu chỉnh và huấn luyện để mô hình hoạt động tốt.





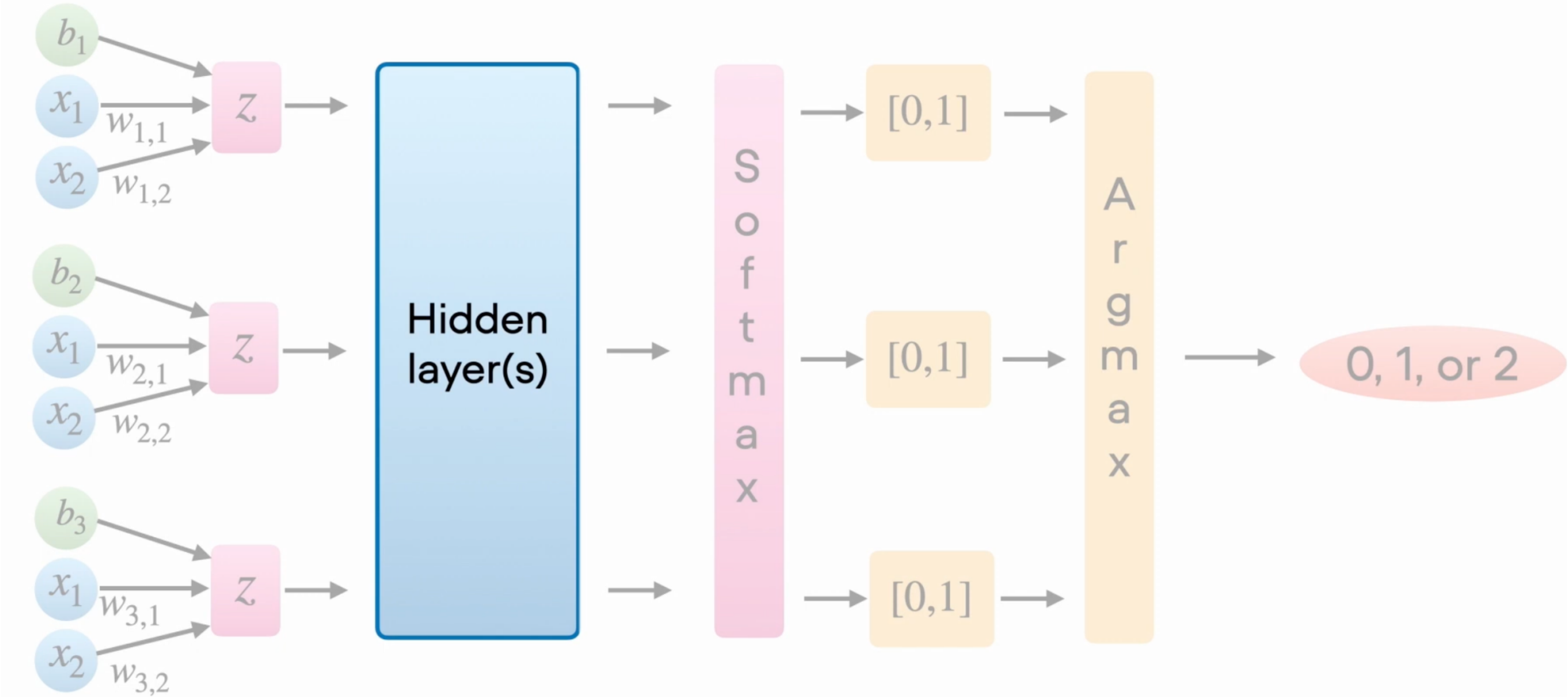
(a) A simple problem logistic regression cannot solve

(b) A problem MLPs can solve

Hình 1: Ví dụ XOR problem

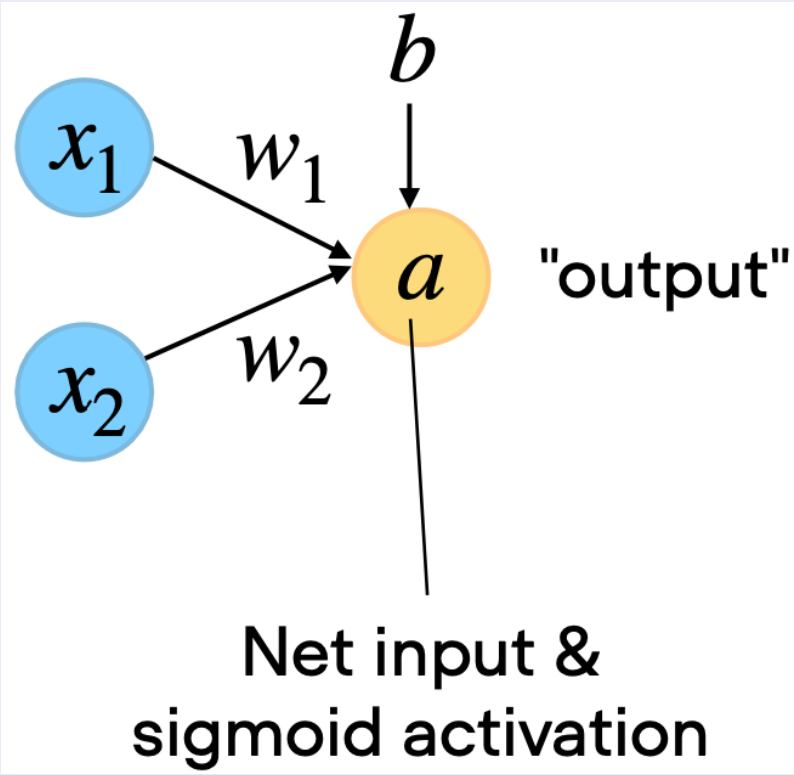
The Multilayer Perceptron Architecture

Kiến trúc của Multilayer Perceptron (MLP) tương tự như Softmax Regression Model nhưng bổ sung thêm các hidden layers.

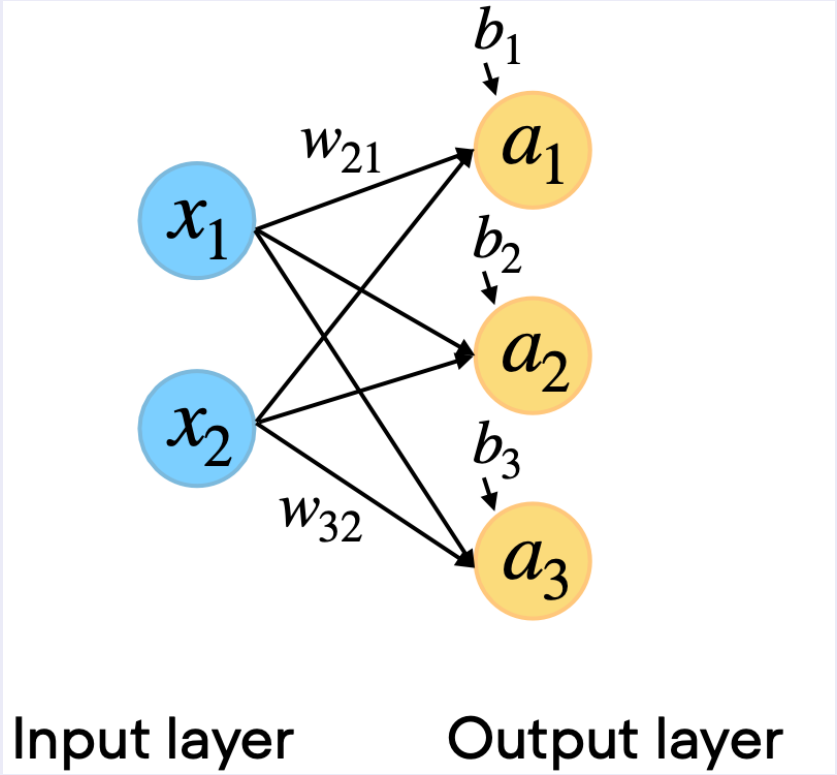


The Multilayer Perceptron Architecture

Để hiểu hơn về kiến trúc của Multilayer Perceptron, ta bắt đầu từ điều chỉnh Logistic regression và softmax regression như sau



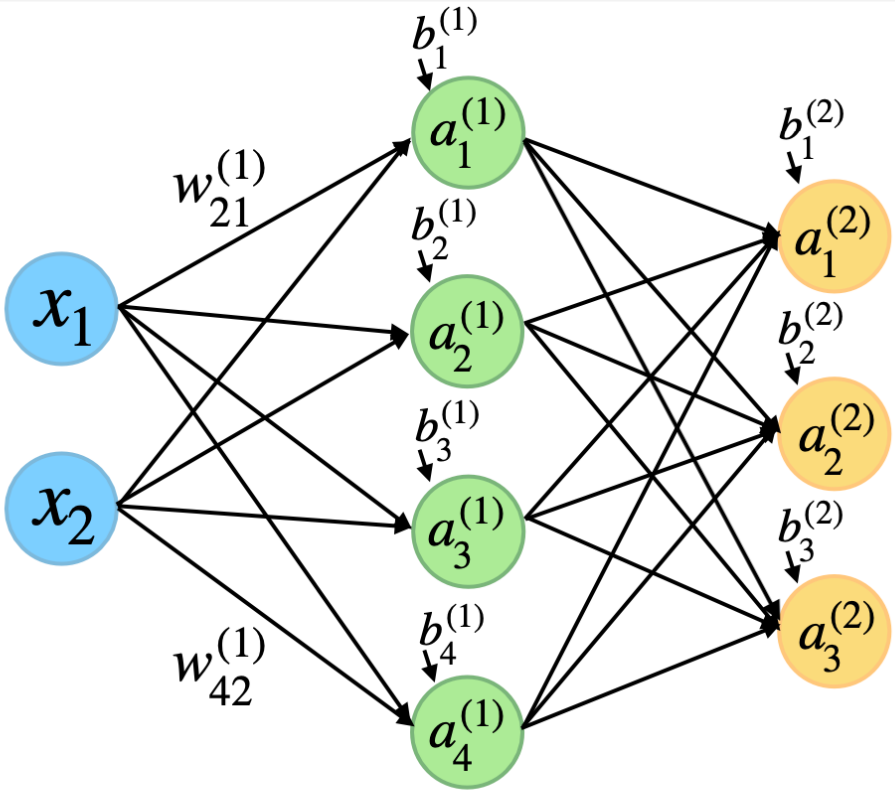
(a) *Logistic regression* với 2 input features, 1 output.



(b) Điều chỉnh mô hình ở (a) thành *softmax* với 3 nodes output.

The Multilayer Perceptron Architecture

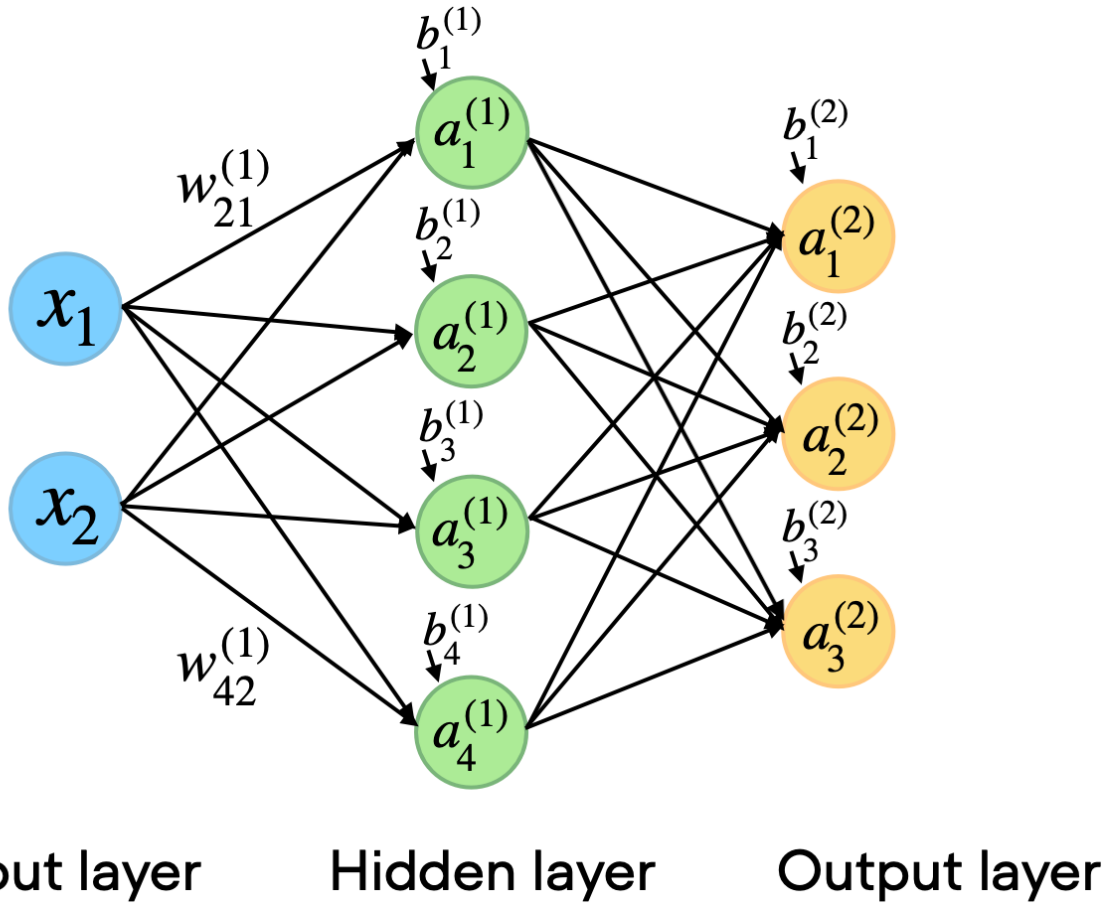
Điều chỉnh mô hình *softmax regression* thành *multilayer perceptron* bằng cách thêm hidden layer giữa input layer và output layer.



Input layer Hidden layer Output layer

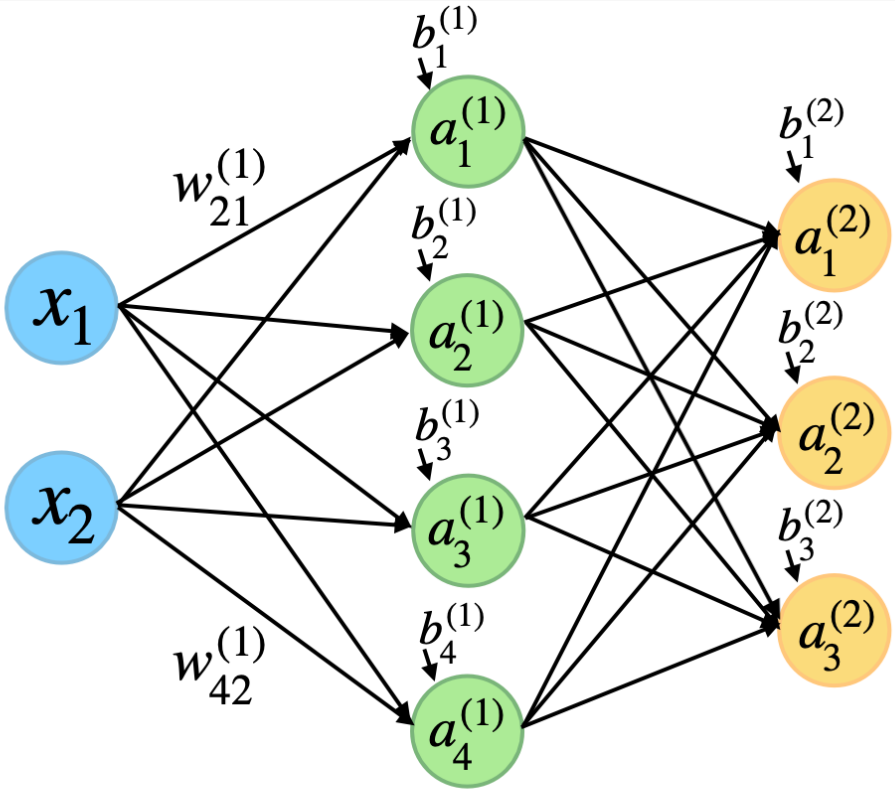
The Multilayer Perceptron Architecture

Trong trường hợp này *chỉ có một hidden layer*, bạn có thể chèn nhiều hidden layer với số lượng bất kì.



Nhận xét: a multilayer perceptron is a **fully-connected** **feedforward** neural networks. Nghĩa là:

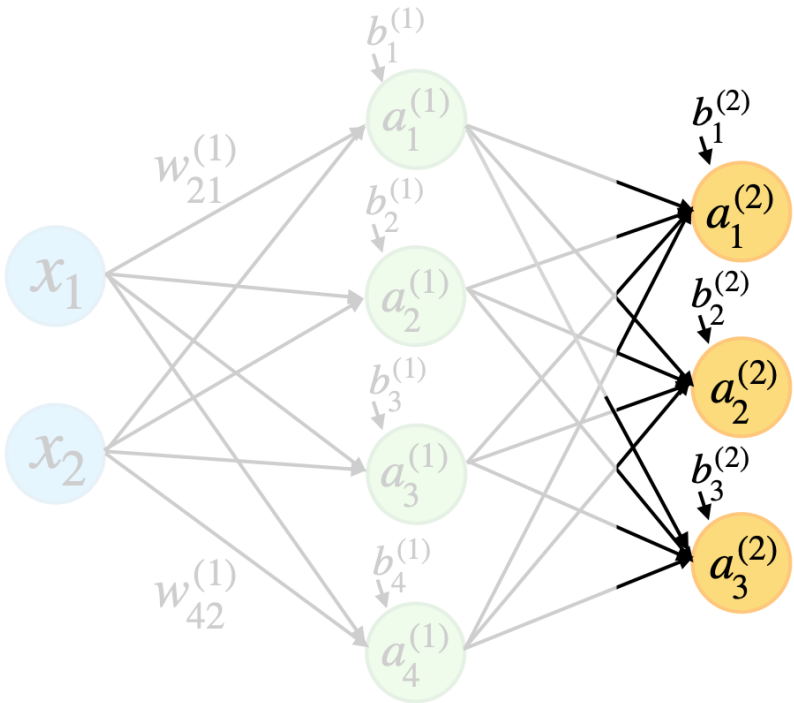
- each node in one layer is connected to each node in the next layer.
- the information flows from one side to the other, from left to right.



Activation Function

Lớp output

Hàm kích hoạt được *sử dụng trong lớp output* của MLP cho bài toán phân loại nhiều lớp vẫn là **Softmax activation**.

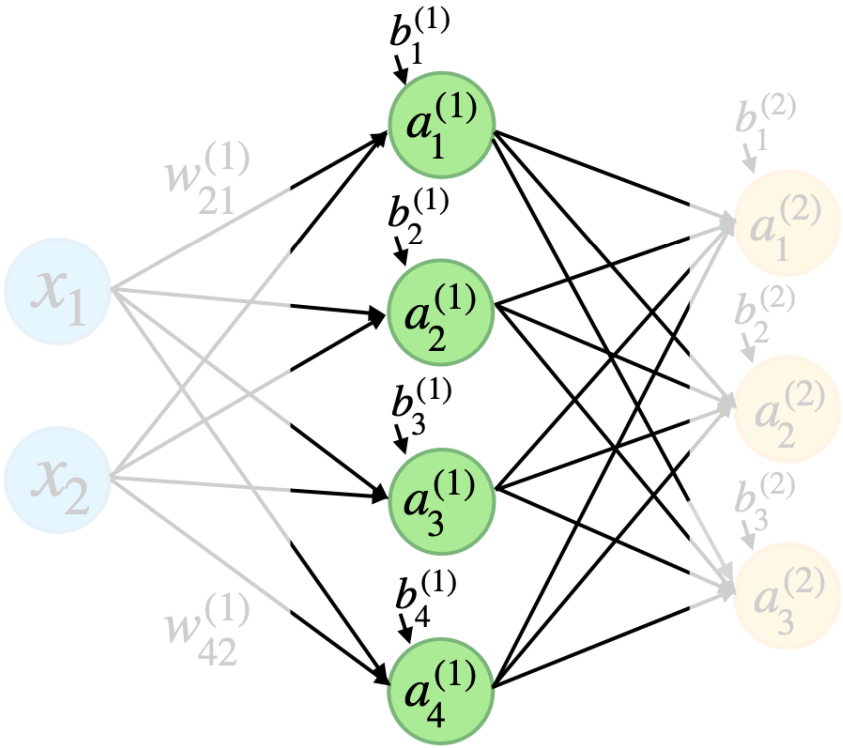


Hình 2: Softmax activation and multiple nodesfor multi-class settings. Về mặt tổng quát, bài toán phân loại nhị phân có thể dùng *Softmax activation + 2 ouput nodes* thay vì dùng *Sigmoid activation + 1 ouput node.*

Activation Function

Hidden layer

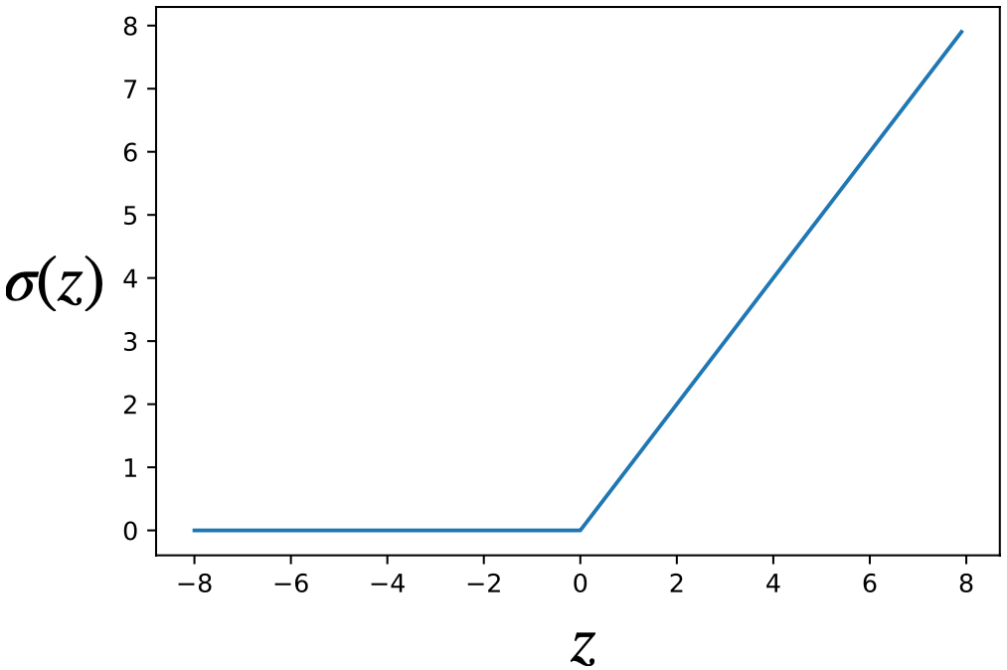
Các hàm kích hoạt được *sử dụng trong hidden layer* của MLP có thể là Sigmoid activation giống như logistic regression.



Activation Function

Hidden layer

Nhưng ngày nay, hàm kích hoạt Rectified Linear Unit–**RELU** được sử dụng phổ biến hơn cho lớp ẩn .



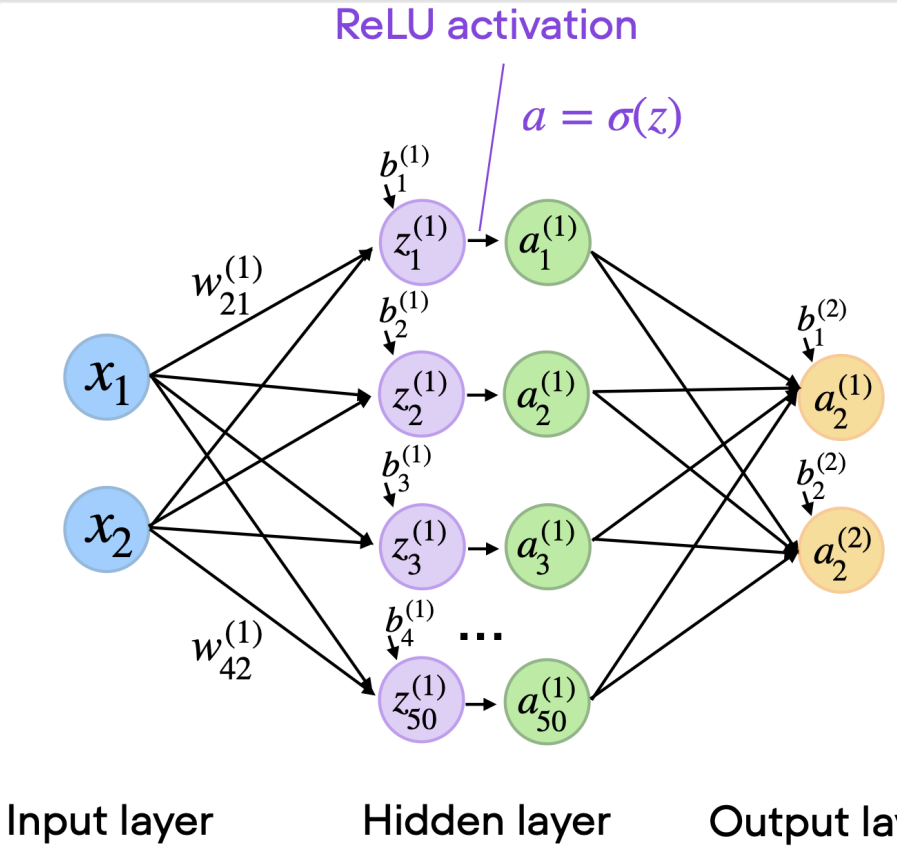
Hình 3:

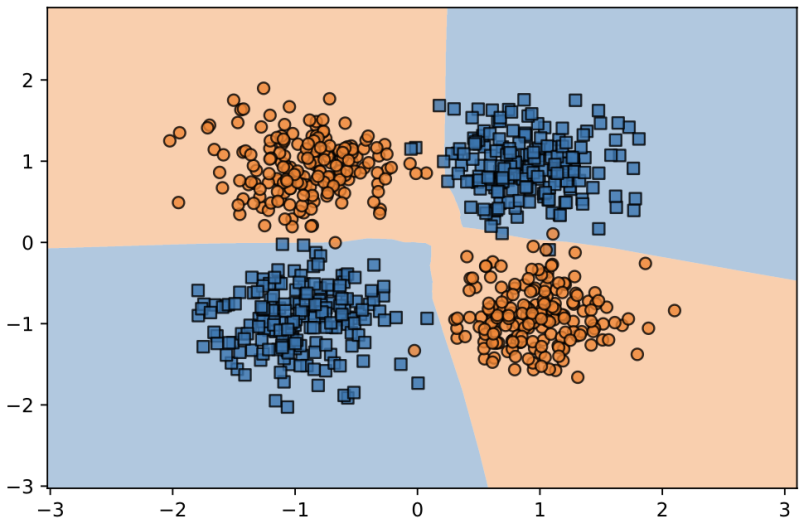
$$ReLU(z) = \begin{cases} z, & \text{if} z \geq 0 \\ 0, & otherwise \end{cases}$$

hoặc $ReLU(z) = max(0, z)$

Lưu ý Nonlinear activation

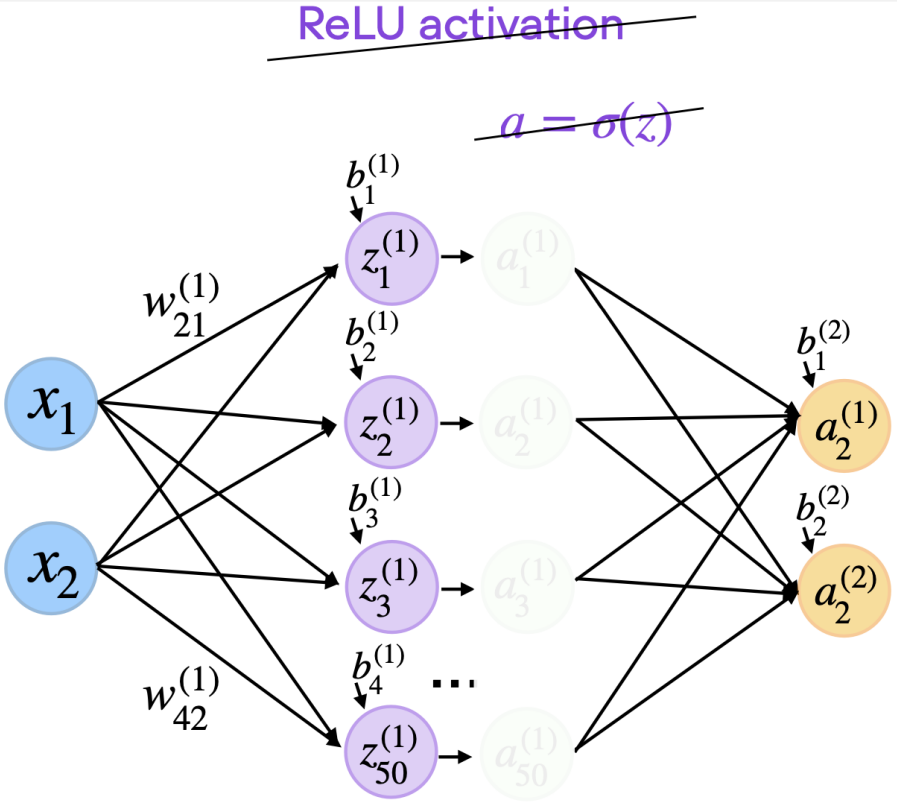
Tại sao chúng ta cần phải sử dụng các hàm kích hoạt là *nonlinear activation*? So suppose we have this architecture of the multilayer perceptron with one hidden layer for XOR dataset.

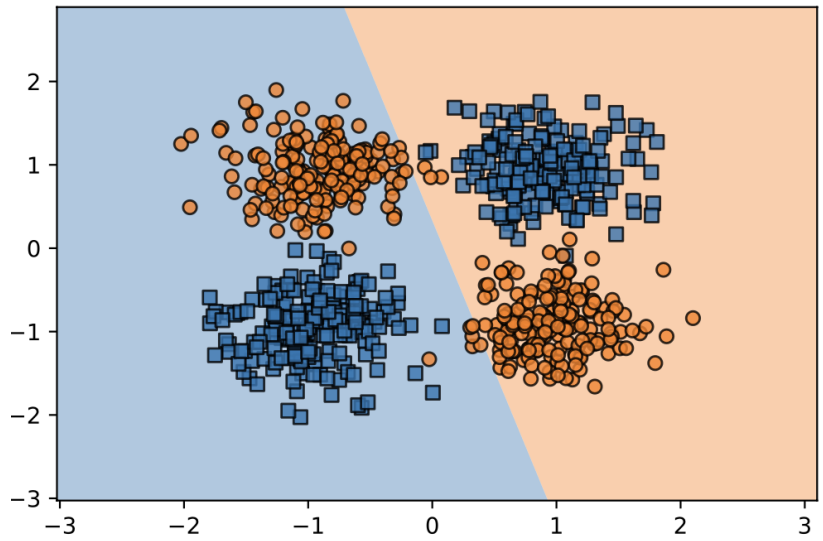




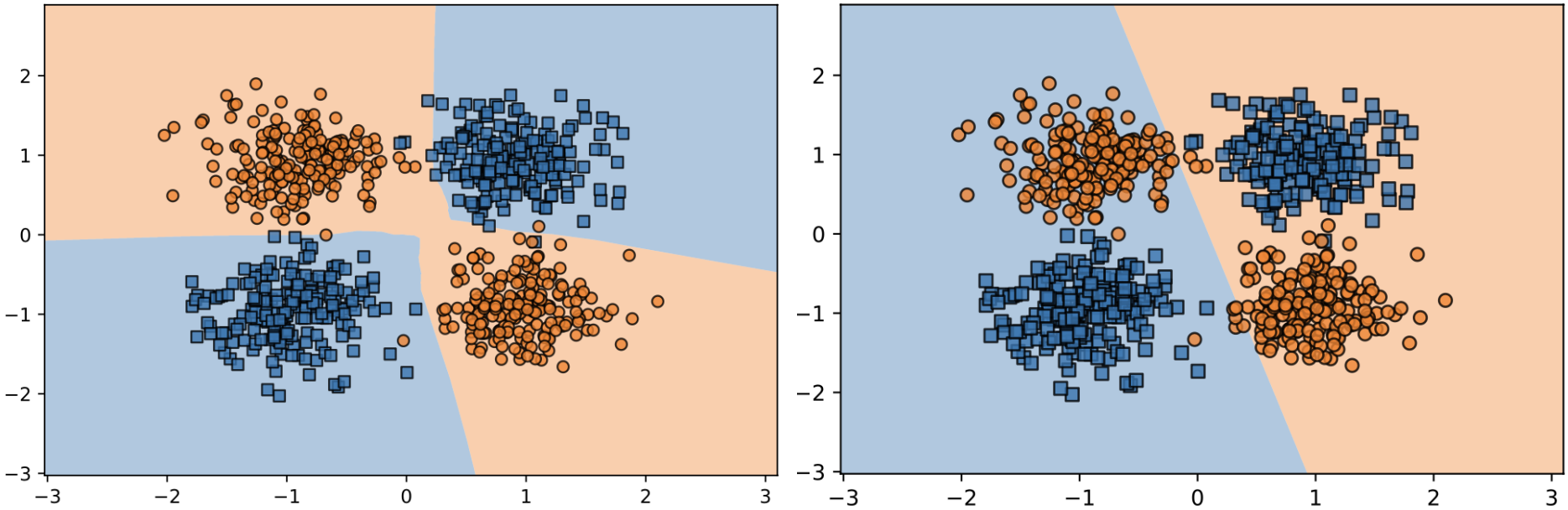
Lưu ý Nonlinear activation

Nếu chúng ta bỏ qua Nonlinear activation RELU trong kiến trúc mạng thì bài toán từ *Nonlinear decision boundaries* trở về thành *Linear decision boundary*.





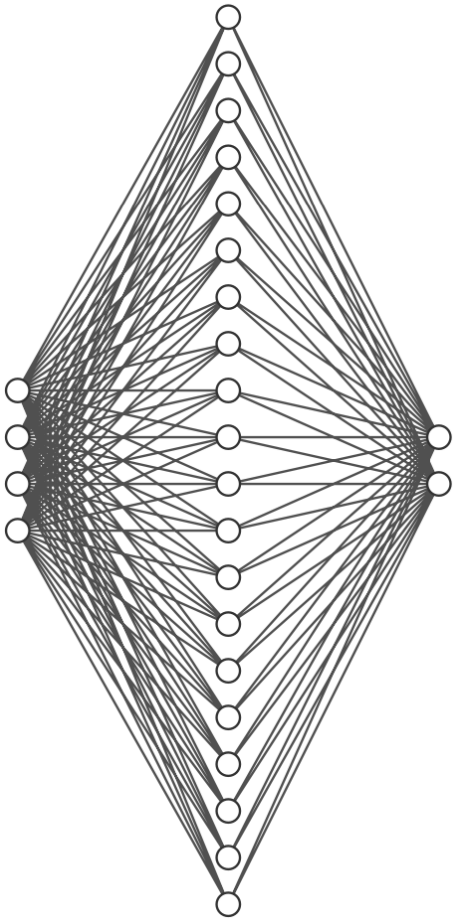
Nếu không có hàm kích hoạt phi tuyến thì sự kết hợp của các hàm tuyến tính là là một hàm tuyến tính trong Perceptron. Điều đó biến MLP trở thành Logistic regression/Softmax regression.



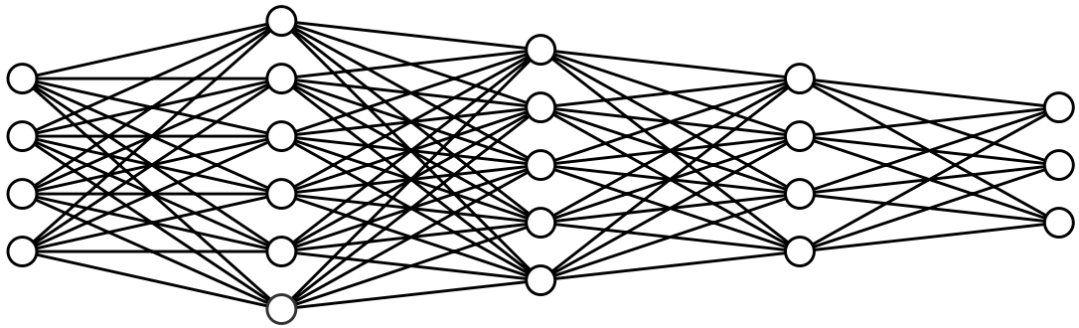
(e) Nolinear classifier

(f) Linear classifier

Lưu ý: Wide Vs Deep Networks



(g) Wide Networks



(h) Deep Networks

Lưu ý: Wide Vs Deep Networks

- A multilayer perceptron with only one hidden layer that is shallow multilayer perceptron.
- A multilayer perceptron with three hidden layers, but each of the hidden layers is pretty narrow, this is a narrow and deep architecture

In theory, every multilayer perceptron with at least one hidden layer and a nonlinear activation function can approximate any function.

Lưu ý: Wide Vs Deep Networks

Wide neural networks

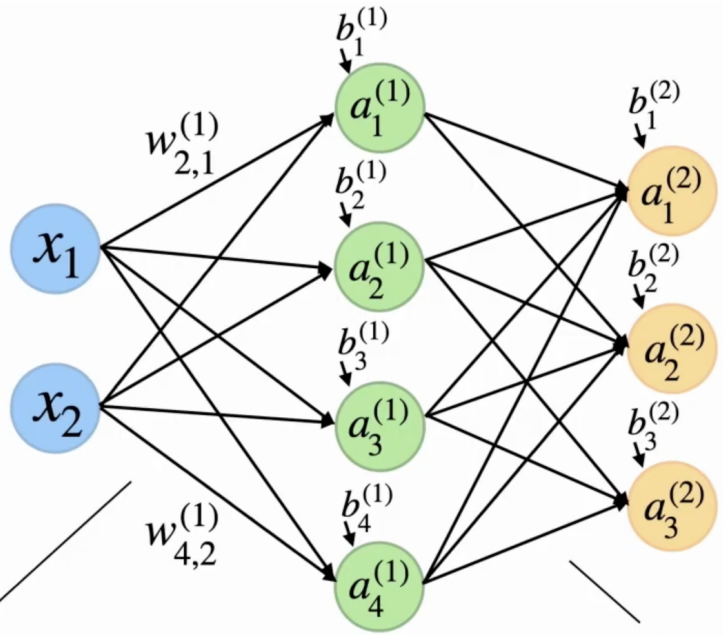
- Advantages: A sufficiently wide MLP with 1 hidden layer can approximate an arbitrary function (given enough training data)
- Downsides: Needs a lot of units in that one hidden layer, this multilayer perceptron will become very prone to overfitting.

Narrow and Deep neural networks

- Advantages: Needs fewer units (nodes); And needing fewer units will help the network to generalize better to new data (abstract features; see CNN)
- Downsides: Harder to train because of the vanishing and exploding gradient problems;

Khởi tạo các trọng số của mô hình

Trong MLPs chúng ta có thể khởi tạo ban đầu các trọng số **W** **bằng 0** như Logistic hay Softmax regression được không?

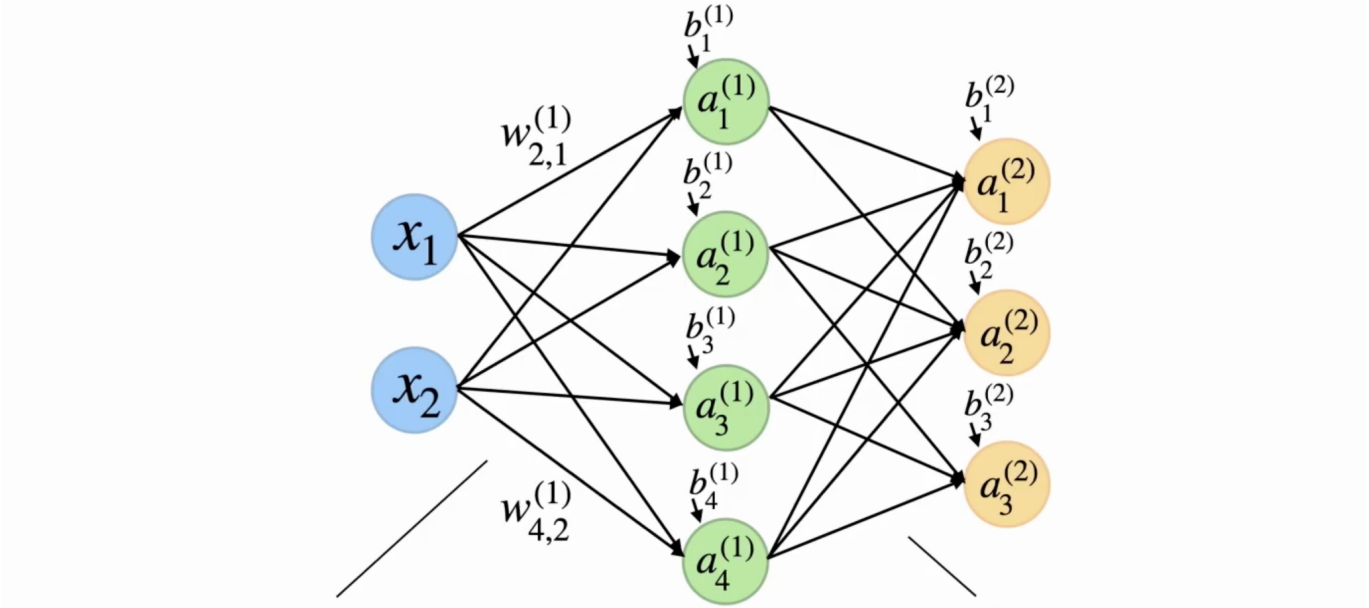


$$W^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h,1} & w_{n,2} & \cdots & w_{h,m} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,h} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,h} \\ \vdots & \vdots & \ddots & \vdots \\ w_{o,1} & w_{n,2} & \cdots & w_{o,h} \end{bmatrix}$$

Khởi tạo các trọng số của mô hình

Đối với MLPs chúng ta cần nhiều trọng số hơn, những trọng số này đóng vai trò kết nối với các lớp với nhau. Ví dụ từ lớp *input* -> *hidden* là ma trận $\mathbf{W}^1$.

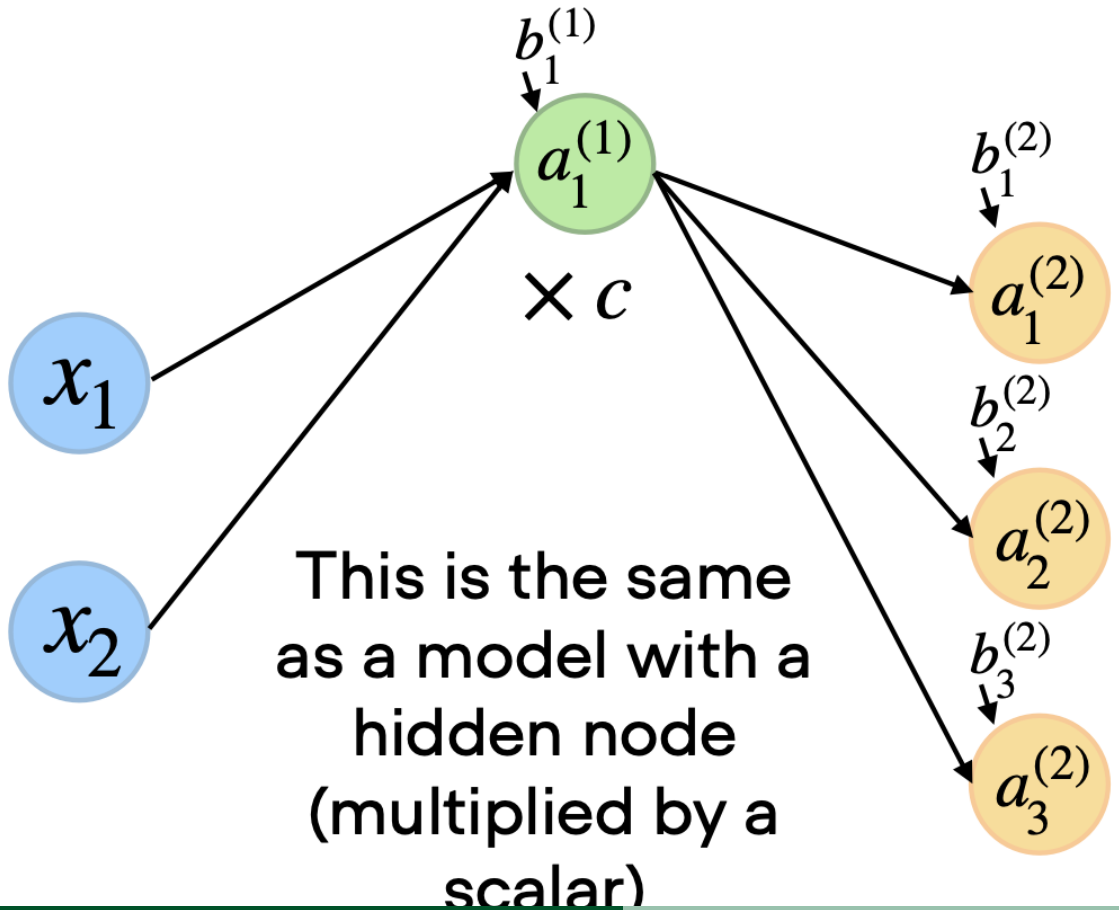


$$W^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h,1} & w_{n,2} & \cdots & w_{h,m} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,h} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,h} \\ \vdots & \vdots & \ddots & \vdots \\ w_{o,1} & w_{n,2} & \cdots & w_{o,h} \end{bmatrix}$$

Khởi tạo các trọng số của mô hình

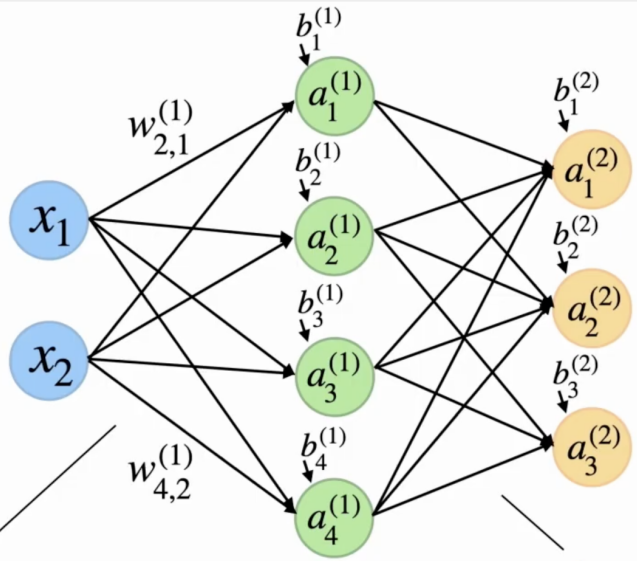
Nếu các trọng số trong $\mathbf{W}^1$ bằng 0, thì tất cả các node trong hidden layer sẽ có cùng một giá trị.



Hình 4: Ví dụ cho 1 node ở hidden layer khi trọng số khởi tạo bằng 0. Dẫn đến việc hidden layer có giá trị bằng giá trị của 1 hidden unit nhân cho hằng số c lần, với c là số lượng node ở hidden layer. Một mạng MLPs chỉ có chung 1 giá trị ở lớp ẩn thì không có ưu điểm gì khác so với softmax regression.

Khởi tạo các trọng số của mô hình

Vậy nên, chúng ta bắt buộc khởi tạo trọng số bắt đầu bằng các số ngẫu nhiên nhỏ để các hidden unit có giá trị khác nhau để đảm bảo quá trình cập nhật trọng số được tối ưu trong quá trình huấn luyện.



$$W^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h,1} & w_{n,2} & \cdots & w_{h,m} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,h} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,h} \\ \vdots & \vdots & \ddots & \vdots \\ w_{o,1} & w_{n,2} & \cdots & w_{o,h} \end{bmatrix}$$

Multilayer Perceptron trong PyTorch I

Định nghĩa lớp MLPs trong PyTorch

```
1  import torch
2
3  class PyTorchMLP(torch.nn.Module):
4    def __init__(self, num_features, num_classes):
5      super().__init__()
6
7      self.all_layers = torch.nn.Sequential(
8
9        # 1st hidden layer
10       torch.nn.Linear(num_features, 25),
11       torch.nn.ReLU(),
12
13       # 2nd hidden layer
14       torch.nn.Linear(25, 15),
15       torch.nn.ReLU(),
16
17       # output layer
18       torch.nn.Linear(15, num_classes),
```

Multilayer Perceptron trong PyTorch II

```
19      )
20
21    def forward(self, x):
22      logits = self.all_layers(x)
23      return logits
```

Định nghĩa quá trình huấn luyện

```
1   import torch.nn.functional as F
2
3   torch.manual_seed(1)
4   model = PyTorchMLP(num_features=2, num_classes=2)
5   optimizer = torch.optim.SGD(model.parameters(), lr=0.05) # Stochastic
        gradient descent
6
7   num_epochs = 10
8
9   for epoch in range(num_epochs):
10
```

Multilayer Perceptron trong PyTorch III

```
11    model = model.train()
12    for batch_idx, (features, labels) in enumerate(train_loader):
13
14      logits = model(features)
15
16      loss = F.cross_entropy(logits, labels) # Loss function
17
18      optimizer.zero_grad()
19      loss.backward()
20      optimizer.step()
21
22      ### LOGGING
23      print(f"Epoch: {epoch+1:03d}/{num_epochs:03d}"
24          f" | Batch {batch_idx:03d}/{len(train_loader):03d}"
25          f" | Train/Val Loss: {loss:.2f}")
26
27    train_acc = compute_accuracy(model, train_loader)
28    val_acc = compute_accuracy(model, val_loader)
29    print(f"Train Acc {train_acc*100:.2f}% | Val Acc {val_acc*100:.2f}%")
```

Multilayer Perceptron trong PyTorch IV

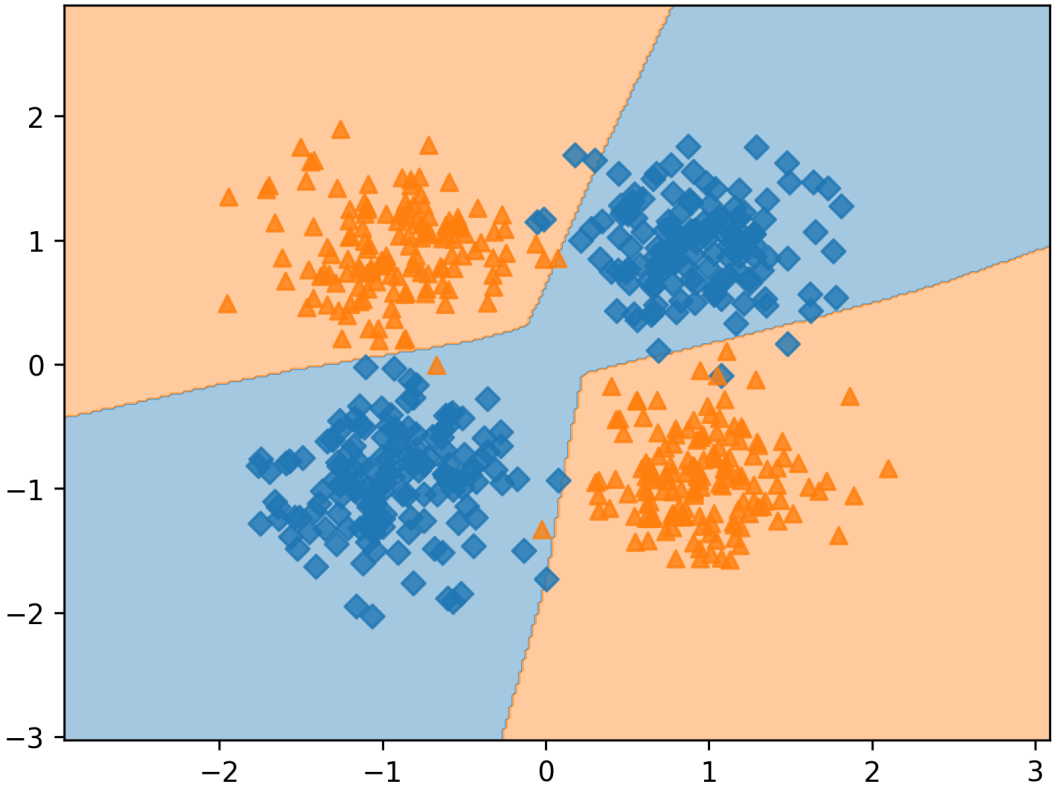
Định nghĩa dữ liệu Dataloader

```
1  from torch.utils.data import Dataset, DataLoader
2
3  class MyDataset(Dataset):
4    def __init__(self, X, y):
5
6      self.features = torch.tensor(X, dtype=torch.float32)
7      self.labels = torch.tensor(y, dtype=torch.int64)
8
9    def __getitem__(self, index):
10     x = self.features[index]
11     y = self.labels[index]
12     return x, y
13
14   def __len__(self):
15     return self.labels.shape[0]
16
17
18 train_ds = MyDataset(X_train, y_train)
```

Multilayer Perceptron trong PyTorch V

```
19  val_ds = MyDataset(X_val, y_val)
20  test_ds = MyDataset(X_test, y_test)
21
22  train_loader = DataLoader(
23    dataset=train_ds,
24    batch_size=32,
25    shuffle=True,
26  )
27
28  val_loader = DataLoader(
29    dataset=val_ds,
30    batch_size=32,
31    shuffle=False,
32  )
33
34  test_loader = DataLoader(
35    dataset=test_ds,
36    batch_size=32,
37    shuffle=False,
38  )
```

Multilayer Perceptron trong PyTorch VI



Hình 5: Kết quả phân loại sử dụng MLPs trên tập dữ liệu XOR

Chi tiết code xem ở file
"*Chuong_4.1.MultilayerPerceptron_XOR.pdf*"

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) .