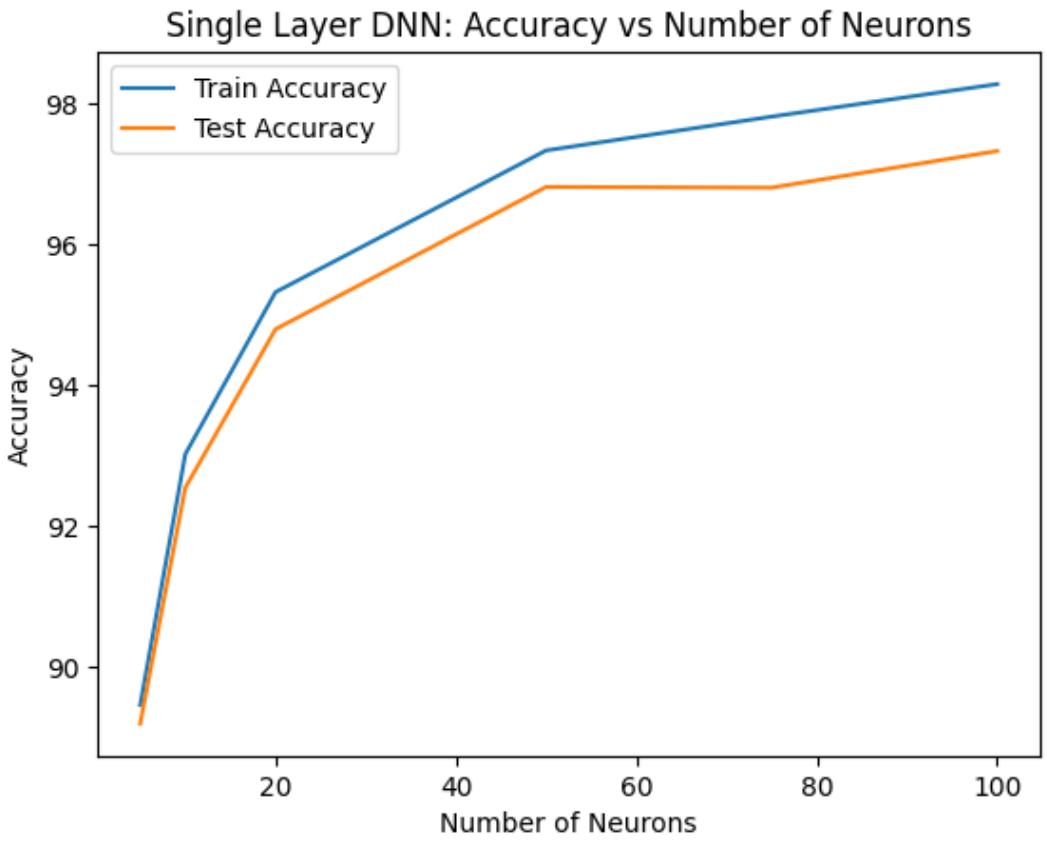
Part1

(a)單層 DNN

圖表趨勢:



Accuracy:

Hidden Size	Train Accuracy:	Test Accuracy
5	89.45333333333333	89.19
10	93.01666666666667	92.54
20	95.31666666666666	94.79
50	97.32833333333333	96.81
75	97.80833333333334	96.8
100	98.27	96.8

數學原理:

1:線性變換：

在隱藏層中，每個神經元對輸入數據進行線性變換，表示為：

$$z=Wx+bz$$

其中 x是輸入向量，W 是權重矩陣，b 是偏置向量，z 是線性變換的輸出。

2:激活函數：

線性變換後，會應用一個激活函數來引入非線性，這有助於網絡學習更複雜的模式。在你的模型中，使用了 ReLU（修正線性單元）

函數：

$$a=\max(0,z)$$

其中 a 是激活後的輸出。選擇 ReLU 是因為它能減少梯度消失問題的可能性，並且計算效率高。

3:輸出層：

隱藏層的輸出接著被傳遞到輸出層，輸出層同樣進行一個線性變換。這一層將輸出用於分類任務的預測結果，通常透過 softmax 函數將其轉化為概率，以決定每個類別的預測概率。在二元或多類別分類中，這可以透過交叉熵損失函數來進行優化。

分析圖表:

從圖表中可以觀察到，隨著隱藏層神經元數量的增加，訓練和測試準確率都有所提升。這表明增加更多的神經元可以幫助模型更好地學習訓練數據中的特徵和模式。不過，兩條曲線的趨勢有所不同：

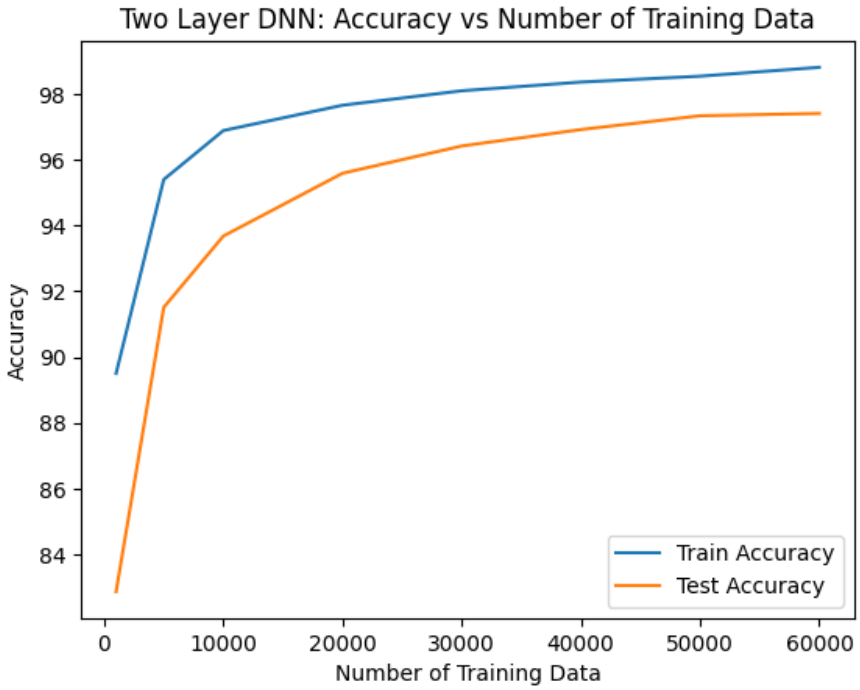
1.訓練準確率：隨著神經元數量的增加，訓練準確率穩步提升，這可能是因為模型的容量增加，能夠更徹底地學習訓練數據。

2.測試準確率：雖然測試準確率也隨著神經元的增加而提高，但增速有所放緩。這可能表明隨著模型複雜度的增加，過擬合的風險也隨之增加。模型在訓練數據上的表現越來越好，但對於未見過的數據（測試數據），提升有限。

這些觀察結果表示在設計神經網絡時需要權衡的重要性：模型容量、過擬合的風險以及如何選擇合適的模型複雜度以優化泛化能力。

(b)雙層 DNN

圖表趨勢:



Accuracy:

Training Size	Train Accuracy:	Test Accuracy
1000	89.5	82.86
5000	95.4	91.51
10000	96.89	93.68
20000	97.66	95.59
30000	98.1	96.42
40000	98.365	96.92
50000	98.544	97.34
60000	98.813333	97.41

雙層深度神經網絡（DNN）包含兩個隱藏層，其運作原理可以分解為以下幾個步驟：

1. **第一層線性變換**：數據首先通過第一個隱藏層的線性變換：

$$z1=W1*x+b1*z1=W1*x+b1$$

其中 x 是輸入數據，$W1$ 是第一層的權重矩陣，$b1$ 是偏置向量，$z1$ 是第一層的輸出。

2. **第一層激活函數**：對第一層的輸出應用 ReLU 激活函數：

$$a1=\max(0,z1)$$

這有助於引入非線性，使得模型能夠捕捉較為複雜的數據結構。

3. **第二層線性變換**：經過激活函數處理後的數據傳遞到第二個隱藏層，再次進行線性變換：

$$z2=W2*a1+b2*z2$$

其中 $W2W$ 和 $b2$ 分別是第二層的權重矩陣和偏置向量。

4. **第二層激活函數**：對第二層的輸出再次應用 ReLU 激活函數：

$$a2=\max(0,z2)$$

5. **輸出層**：最後，過第二層激活的數據被送入輸出層，進行最終的線性變換以獲得分類結果：

$$y=W3*a2+b3*y$$

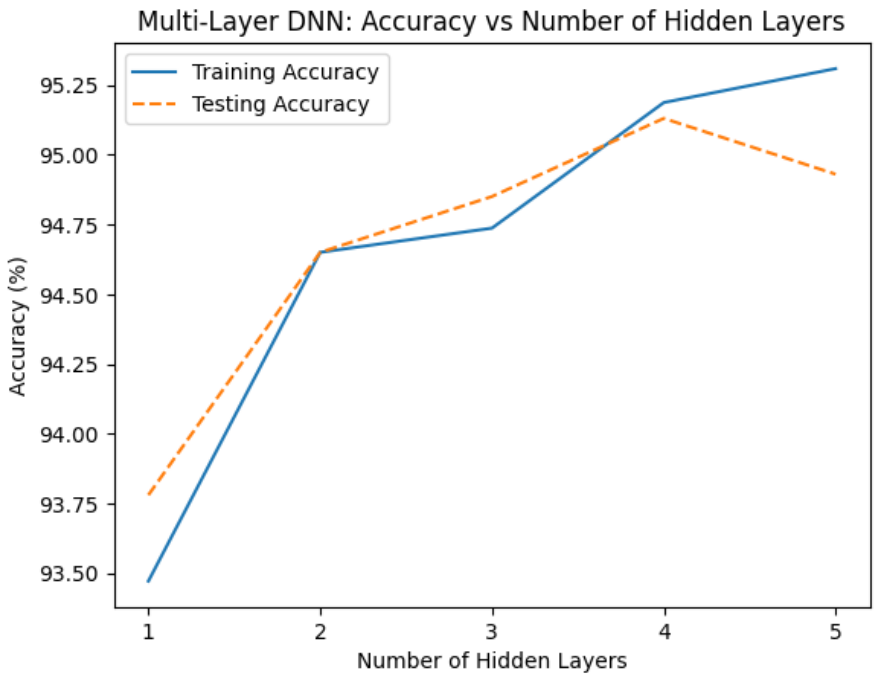
在多類分類任務中，通常會通過 softmax 函數將輸出轉換為概率分佈。

圖表分析

從圖表中可以觀察到以下趨勢：

- **隨著訓練數據的增加，準確率提高**：隨著可用於訓練的數據量增加，訓練準確率和測試準確率都有顯著提升。這表明更多的數據可以幫助模型更好地學習並泛化到未見過的數據。
- **訓練與測試準確率的差距**：在訓練初期，訓練準確率與測試準確率之間的差距較大，這可能指示過擬合的現象。隨著訓練數據的增加，這一差距逐漸縮小，顯示模型的泛化能力有所提高。
- **性能趨於穩定**：當訓練數據量達到一定水平後，準確率的提升趨於平緩。這表明進一步增加數據量對模型性能的提升作用有限，模型可能接近其學習能力的上限。

(c)多層 DNN 圖表趨勢:



Accuracy:

Number of Layers	Training Accuracy	Testing Accuracy
1	93.471666	93.78
2	94.65	94.65
3	94.7366	94.85
4	98.045	95.13
5	95.308333	94.93

數學原理:

多層深度神經網絡（DNN）的主要組件包括多個線性層和非線性激活函數。這些層按順序堆疊，每一層的輸出都是下一層的輸入。

1. **線性層（全連接層）**：每一層計算的是 **z = Wx + b**，其中 **x** 是輸入向量，**W** 是權重矩陣，**b** 是偏差向量。
2. **激活函數**: 本例中使用的是 ReLU 函數，公式為 **a = max(0, z)**。這一步的目的是增加網絡的非線性能力，使得模型能夠學習更複雜的數據模式。
3. **輸出層**: 最後的輸出層通常也是一個線性層，用於將最後一個隱藏層的輸出轉化為最終的分類結果。在多類別分類中，這一層的輸出通常會通過 softmax 函數處理，將其轉換為概率分布。

圖表分析與比較:

從提供的圖表來看，隨著隱藏層數量的增加，訓練準確率和測試準確率呈現不同的變化趨勢：

- **訓練準確率**：隨著層數增加，訓練準確率維持在較高水平，一開始略有上升後趨於平穩，這表明模型能夠越來越好地擬合訓練數據。
- **測試準確率**：隨著層數的增加，測試準確率先是上升，然後在四層時達到峰值，之後出現明顯的下降，出現 trainning data 和 testing data 的 accuracyvu 的差值增加。這種現象可能表明當隱藏層數量超過四時後，模型開始過擬合，即在訓練數據上表現良好，但對未見數據的泛化能力下降。

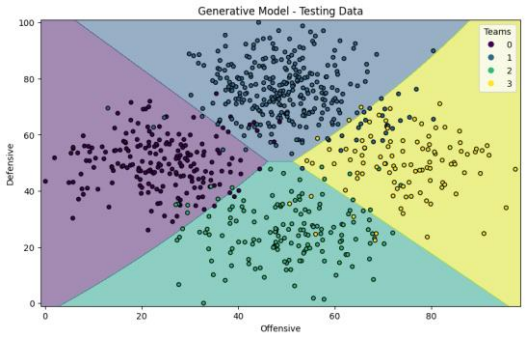
因此選擇 hidden layers=4 在這裡較為適合，既不會過擬合**保有**泛化能力**。**

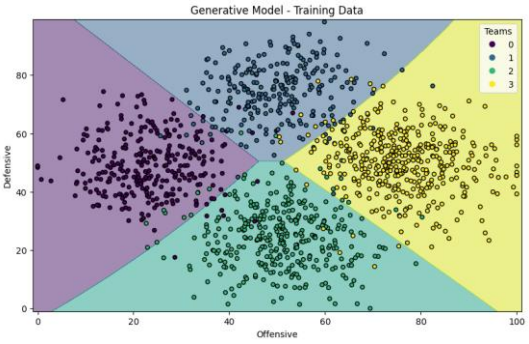
- **與之前的單層 DNN 比較:**

與之前分析的單層 DNN 相比，多層 DNN 在初期能夠達到更高的準確率，這表明增加層數有助於模型學習更複雜的特徵，從而改善性能。然而，多層模型也更容易過擬合，尤其是在層數較多時，這需要通過技術如正則化、dropout 或者早停（early stopping）等方法來緩解。

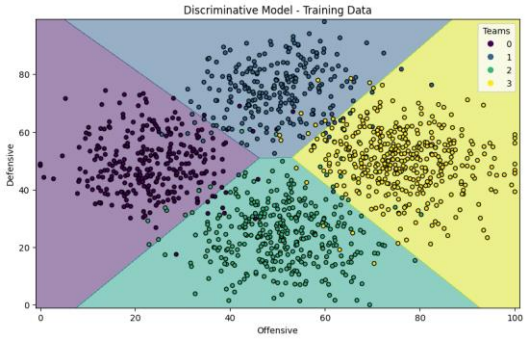
因此，選擇最適合的隱藏層數量需要權衡模型的復雜性和泛化能力，避免過擬合的同時保持足夠的模型容量來捕捉數據中的關鍵特徵。

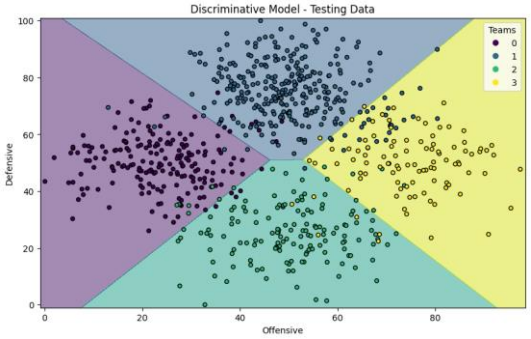
PART2:生成模型與判別模型圖:





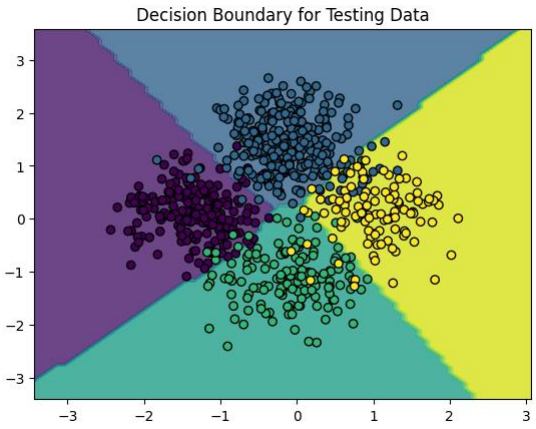
Testing Data Accuracy: 0.906667　　Training Data Accuracy: 0.943077





Training Data Accuracy: 0.942308　　Testing Data Accuracy: 0.909333

DNN 圖形:



Training Accuracy: 0.947692307　　Testing Accuracy: 0.913333334

模型比較與分析

模型類型:

1. **深度神經網絡 (DNN)**: 使用多個層次的非線性映射來學習數據的複雜結構。
2. **生成模型 (GaussianNB)**: 基於特征的聯合概率分佈來建模，通常用於生成新的樣本數據。
3. **判別模型 (Logistic Regression)**: 直接對條件概率 P(y|x)建模，優化決策邊界。

決策邊界分析

- **DNN 的決策邊界**: 能看出非常複雜且彈性的決策邊界(鋸齒狀邊界)，這是因為深層網絡能學習數據的高階特徵。此類模型在區分重疊或非線性分隔的數據集時表現較好。
- **生成模型的決策邊界**: 決策邊界較為直觀簡單，這是因為它假設特征間獨立且通常呈現線性分割，對於某些複雜的或特征相依的數據集來說可能不夠精確。
- **判別模型的決策邊界**: 通常是線性或分段線性的，適用於實際特征間分隔明確的情況。這類模型專注於最大化不同類別間的區分度。

訓練數據準確率排序

1. **DNN 模型**: 準確率為 0.9477
2. **生成模型 (Gaussian Naive Bayes)**: 準確率為 0.9431
3. **判別模型 (Logistic Regression)**: 準確率為 0.9423

測試數據準確率排序

1. **DNN 模型**: 準確率為 0.9133
2. **判別模型 (Logistic Regression)**: 準確率為 0.9093
3. **生成模型 (Gaussian Naive Bayes)**: 準確率為 0.9067

分析原因:

DNN 模型 在訓練和測試數據上均表現最好，這說明深度學習模型具有強大的特徵學習能力，並且由於其複雜性，當訓練得當時，能夠在多種數據集上表現出良好的泛化能力。DNN 的高維度和非線性模型特性使其能夠捕捉更複雜的數據分佈，但也更容易過擬合。

- **生成模型** 和 **判別模型** 在訓練數據上的準確率較接近，但在測試數據上表現略有不同。生成模型通常在假設特徵獨立的情況下效果較好，但如果這些假設不成立，其性能可能會受到影響。判別模型則直接學習從特徵到標籤的映射，這使得它在具體問題上可能更直接有效。
- **判別模型** 的表現在訓練和測試上都相對穩定，這可能是因為這類模型在設計時就考慮到了直接的決策邊界優化，使其在未見數據上的表現通常較為穩健。

額外討論:

1:DNN 與傳統方法（生成模型和判別模型）的區別

DNN（深度神經網絡）:

DNN 利用多層結構和非線性激活函數來學習數據的高階抽象特徵。在處理複雜和高維度的數據集時，DNN 特別有效，因為它可以捕捉到數據的複雜模式和關係。DNN 能夠透過多層神經元結構進行特徵提取，從而自動學習數據的潛在表示。這使得 DNN 在圖像識別、語音識別等需要處理大量非結構化數據的任務中具有優越的表現。

生成模型：

生成模型通過學習輸入數據的聯合概率分佈來進行預測。這類模型適用於需要根據現有數據生成或模擬新數據點的場景。例如，高斯貝葉斯分類器是一種典型的生成模型，它假設數據的特徵是相互獨立並且服從高斯分佈。生成模型可以有效地處理缺失數據和不平衡數據，但其性能高度依賴於數據的分佈假設是否準確。

判別模型：

判別模型直接對條件概率 P(ylx)進行建模，旨在找到不同類別之間的最佳決策邊界。這些模型通常對於預測任務來說更為直接和高效。例如，邏輯回歸和支持向量機（SVM）就是常見的判別模型。邏輯回歸通過線性決策邊界來分類數據，而 SVM 則尋找能最大化類別間隔的超平面。這些模型在特徵提取和數據維度較低時表現良好，但在處理高維度和複雜數據時可能表現不如 DNN。

2:utilize the toolbox of Pytorch:

```
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
import numpy as np
```

1:torch 和 **torch.nn**：Pytorch 的核心庫，用於構建神經網絡模型、定義層次和激活函數。

2:torch.optim：Pytorch 中的優化器模塊，用於優化模型參數。

3:torch.utils.data 的 DataLoader 和 TensorDataset：用於將數據加載到 Pytorch 中，進行批處理。

```
# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

# Create dataloaders
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

4: 使用 **torch.tensor** 將數據轉換為 Pytorch 張量。

5: 使用 **TensorDataset** 和 **DataLoader** 將數據組織成批次，方便後續的訓練和測試。

```
# Define the neural network
class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        self.layer1 = nn.Linear(2, 128)
        self.relu = nn.ReLU(
        self.layer2 = nn.Lin
        self.output_layer = nn.Linear(64, 4)

    def forward(self, x):
        x = self.relu(self.layer1(x))
        x = self.relu(self.layer2(x))
        x = self.output_layer(x)
        return x
```

(module) nn

6:繼承 **nn.Module**，定義了一個具有兩個隱藏層和一個輸出層的簡單 DNN。每個隱藏層之間使用 ReLU 激活函數。

Pytorc 使用優點: **靈活的動態計算圖**

> **動態計算圖**：Pytorch 使用動態計算圖（Dynamic Computational Graph），這意味著圖是在運行時構建的，而不是靜態地預先定義。這使得調試和修改模型變得更加靈活和直觀。
>
> 自動微分功能
>
> Pytorch 的 autograd 模塊提供了強大的自動微分功能，能夠自動計算任意張量操作的梯度。這對於實現反向傳播和梯度下降等優化算法至關重要
>
> 與硬件加速的集成

GPU 加速：Pytorch 無縫支持 GPU 加速，通過簡單的 .cuda() 方法即可將張量和模型轉移到 GPU 上，大大提高計算效率。

- 分布式訓練：Pytorch 提供了分布式訓練支持，使得在多 GPU 和多節點環境中訓練大型模型變得更加高效

其他觀察現象:

過擬合

- **過擬合**：過擬合是指模型在訓練數據上表現良好，但在測試數據上表現不佳。這通常是因為模型過於複雜，能夠記住訓練數據的噪音和細節

深度學習的可解釋性

可解釋性：DNN 通常被認為是“黑盒”模型，因為其內部結構和決策過程很難理解和解釋。因此個人認為這就像人的大腦類似，無法理解其中過程

訓練效率和計算資源

- **GPU 加速**：使用 GPU 進行訓練可以顯著提高 DNN 的訓練速度。Pytorch 對 GPU 的支持非常好，通過簡單的 **.cuda()** 方法即可將張量和模型轉移到 GPU 上，而當時沒注意到用 cpu 運算算了 2 多小時，而且 cpu 吃滿讓我無法進行其他操作，一直當機直到跑完 trainning，才瞭解為啥 NVIDIA 在 AI 浪潮吃到如此多紅利。
- **分布式訓練**：對於特別大的數據集和模型，可以使用多 GPU 和多節點進行分布式訓練。Pytorch 提供了分布式訓練的支持，使得在大規模計算環境中進行高效訓練成為可能，但也要朱義分類的計算輛，如我的筆電在跑的時候就顯示我的記憶體數量不足，因而會硬體也會影響最終使用計算方式。

。