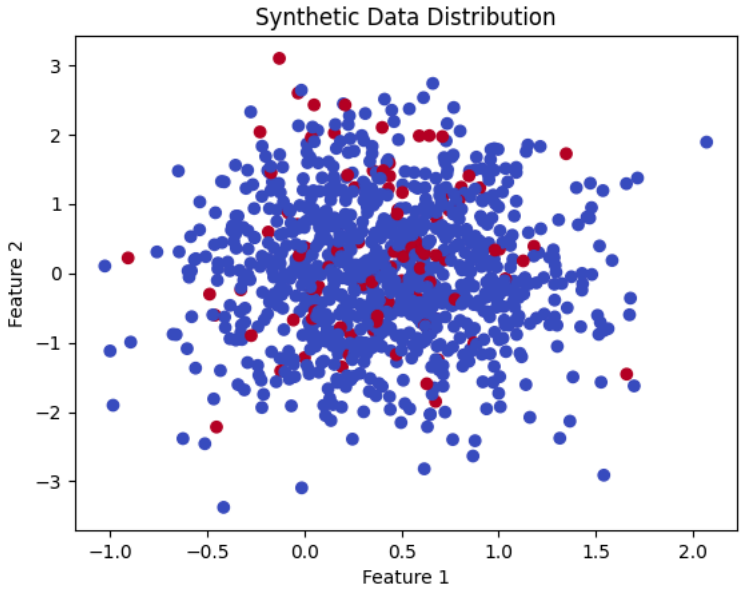
```
import numpy as np
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

# Генерація синтетичних даних
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
                           n_clusters_per_class=1, weights=[0.9, 0.1], flip_y=0, random_state=42)

# Перевірка розподілу класів
print(f"Кількість прикладів класу 0: {np.sum(y == 0)}")
print(f"Кількість прикладів класу 1: {np.sum(y == 1)}")

# Візуалізація перших двох ознак
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Synthetic Data Distribution')
plt.show()
```

```
Кількість прикладів класу 0: 900
Кількість прикладів класу 1: 100
```




```
from sklearn.model_selection import train_test_split

# Розділення даних
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Навчальна вибірка: {X_train.shape[0]} прикладів")
print(f"Валідаційна вибірка: {X_val.shape[0]} прикладів")
print(f"Тестова вибірка: {X_test.shape[0]} прикладів")
```

```
Навчальна вибірка: 600 прикладів
Валідаційна вибірка: 200 прикладів
Тестова вибірка: 200 прикладів
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Функція для створення моделі
def create_model(input_dim, hidden_units=64, layers=2, lambda_reg=0.01):
    model = Sequential()
    model.add(Dense(hidden_units, input_dim=input_dim, activation='relu', kernel_regularizer='l2'))

    for _ in range(layers - 1):
        model.add(Dense(hidden_units, activation='relu', kernel_regularizer='l2'))

    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

    return model

# Створення моделі з 2 шарами
```

```
model = create_model(X_train.shape[1], hidden_units=64, layers=2)
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 64)	1,344
dense_7 (Dense)	(None, 64)	4,160
dense_8 (Dense)	(None, 1)	65

```
Total params: 5,569 (21.75 KB)
Trainable params: 5,569 (21.75 KB)
Non-trainable params: 0 (0.00 B)
```

```
# Тренування моделі
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val), verbose=1)
```

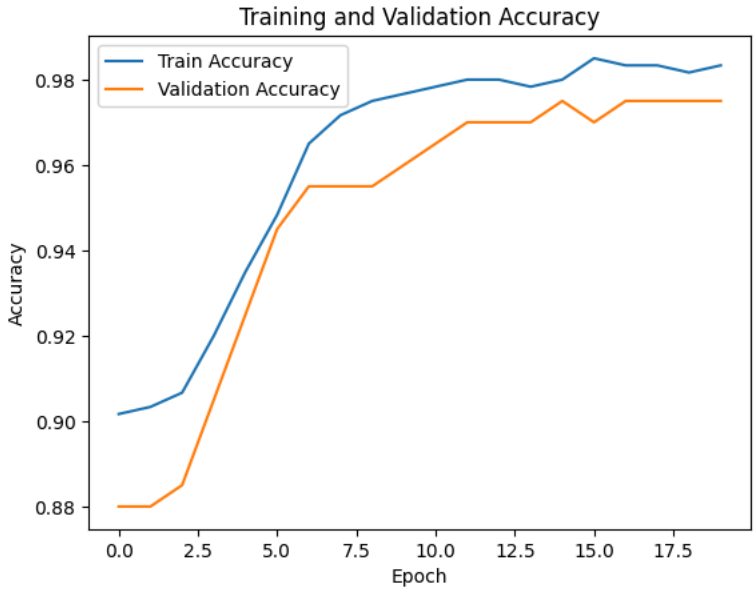
```
Epoch 1/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9127 - loss: 1.3193 - val_accuracy: 0.8800 - val_loss: 1.1305
Epoch 2/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8963 - loss: 1.0660 - val_accuracy: 0.8800 - val_loss: 0.9684
Epoch 3/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8900 - loss: 0.9289 - val_accuracy: 0.8850 - val_loss: 0.8264
Epoch 4/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9199 - loss: 0.7610 - val_accuracy: 0.9050 - val_loss: 0.7065
Epoch 5/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9236 - loss: 0.6733 - val_accuracy: 0.9250 - val_loss: 0.6058
Epoch 6/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9491 - loss: 0.5687 - val_accuracy: 0.9450 - val_loss: 0.5206
Epoch 7/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9661 - loss: 0.4871 - val_accuracy: 0.9550 - val_loss: 0.4525
Epoch 8/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9742 - loss: 0.4203 - val_accuracy: 0.9550 - val_loss: 0.3982
Epoch 9/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9701 - loss: 0.3860 - val_accuracy: 0.9550 - val_loss: 0.3522
Epoch 10/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9725 - loss: 0.3257 - val_accuracy: 0.9600 - val_loss: 0.3146
Epoch 11/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9820 - loss: 0.2877 - val_accuracy: 0.9650 - val_loss: 0.2835
Epoch 12/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9838 - loss: 0.2500 - val_accuracy: 0.9700 - val_loss: 0.2583
Epoch 13/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9810 - loss: 0.2367 - val_accuracy: 0.9700 - val_loss: 0.2373
Epoch 14/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9861 - loss: 0.2078 - val_accuracy: 0.9700 - val_loss: 0.2185
Epoch 15/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9848 - loss: 0.1950 - val_accuracy: 0.9750 - val_loss: 0.2041
Epoch 16/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9882 - loss: 0.1828 - val_accuracy: 0.9700 - val_loss: 0.1916
Epoch 17/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9719 - loss: 0.1957 - val_accuracy: 0.9750 - val_loss: 0.1804
Epoch 18/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9746 - loss: 0.1750 - val_accuracy: 0.9750 - val_loss: 0.1715
Epoch 19/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9846 - loss: 0.1493 - val_accuracy: 0.9750 - val_loss: 0.1630
Epoch 20/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9826 - loss: 0.1526 - val_accuracy: 0.9750 - val_loss: 0.1555
```

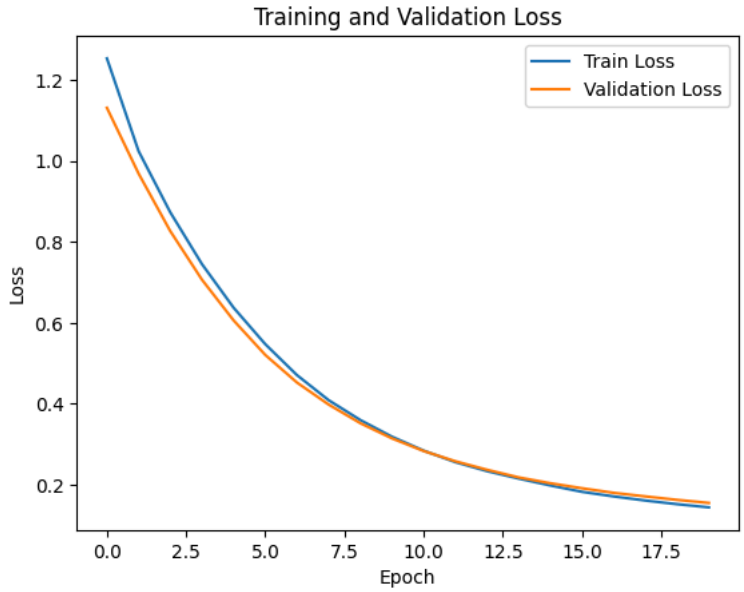
```
# Оцінка на тестових даних
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Тестова точність: {test_acc:.4f}")

# Побудова кривих навчання
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9720 - loss: 0.1776
Тестова точність: 0.9700
```






```
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
import numpy as np

# Створення функції для моделі з регуляризацією
def create_model(lambda_reg=0.01):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu', kernel_regularizer=l2(lambda_reg)))
    model.add(Dense(64, activation='relu', kernel_regularizer=l2(lambda_reg)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Функція для тренування моделі
def train_model(model, X_train, y_train, epochs=20, batch_size=32):
    return model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=0)

# Створення GridSearchCV вручну
def grid_search(X_train, y_train):
    param_grid = {'lambda_reg': [0.001, 0.01, 0.1, 1.0]}
    best_model = None
    best_score = -np.inf
    best_lambda = None

    for lambda_reg in param_grid['lambda_reg']:
        # Створення моделі з поточним значенням lambda_reg
        model = create_model(lambda_reg)
```

```
            history = train_model(model, X_train, y_train)

            # Оцінка моделі на валідаційних даних
            val_accuracy = history.history['accuracy'][-1]  # Останній етап навчання

            if val_accuracy > best_score:
                best_score = val_accuracy
                best_lambda = lambda_reg
                best_model = model

        return best_model, best_lambda, best_score

    # Виконання пошуку
    best_model, best_lambda, best_score = grid_search(X_train, y_train)

    print(f"Найкращий λ: {best_lambda}")
    print(f"Найкраща точність на валідації: {best_score}")
```

Показати прихований результат