Lab-03: Binary Classification with Neural Networks on Circles Dataset

Submitted by: Santosh Mahato Koiri

Objective

The objective of this lab is to implement and compare different neural network architectures for binary classification on a synthetic circles dataset. We will build, train, and evaluate PyTorch neural networks to classify data points into two categories based on their spatial distribution. The dataset consists of points arranged in concentric circles, making it a non-linear classification problem that requires non-linear activation functions to solve effectively.

Through this experiment, we aim to:

- Understand the data preprocessing pipeline for neural network training
- Implement different neural network architectures (linear and non-linear)
- Compare the performance of models with and without activation functions
- Visualize decision boundaries to understand model behavior
- Analyze the impact of different optimizers on convergence

Theory

Binary classification involves categorizing input data into one of two possible classes, such as determining whether an email is spam or not, or in our case, identifying which concentric circle a point belongs to. In this experiment, we work with a synthetic dataset where data points are arranged in two concentric circles, creating a non-linear decision boundary that makes the problem particularly challenging for simple models.

Neural networks form the backbone of our approach, consisting of interconnected layers of nodes that process input data through weighted connections. Each node, or neuron, applies mathematical transformations to its inputs, and when combined across layers, these networks can learn complex patterns in data. The input layer receives our spatial coordinates, hidden layers perform intermediate processing, and the output layer produces classification predictions. A crucial component that enables neural networks to handle non-linear problems is the activation function, which introduces non-linearity into the otherwise linear transformations. Without

these activation functions, even deep networks would only be able to learn linear relationships.

The circles dataset presents a perfect test case for understanding these concepts because linear models simply cannot separate the classes effectively. The data forms two distinct circular regions that require curved decision boundaries, making it an ideal demonstration of why non-linear activation functions are essential. This dataset tests the network's ability to learn complex spatial relationships and provides clear visual feedback about model performance through decision boundary plots.

1. Data Retrieval and Inspection

In this section, we load the circles dataset from a CSV file and perform initial exploratory data analysis. This step helps us understand the structure of the dataset, including the number of samples we have, what features are available, and what the target variable looks like. We also examine basic statistics of the features to get a sense of their distribution and range. This initial inspection is crucial for understanding data quality and ensuring we don't have any obvious issues like missing values or unexpected data types.

We should expect to see around 1000 samples in total, with three columns representing the two spatial coordinates (X1 and X2) and the binary target variable (label). The data should be well-balanced between the two classes, with roughly equal numbers of points in each concentric circle. This balanced distribution will make our evaluation metrics more reliable and prevent the model from simply learning to predict the majority class.

In [31]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import torch.nn.functional as F
```

In [32]:
```
# 1. Data retrieval & inspection
df = pd.read_csv('circles_binary_classification.csv')
print("Head of the dataset:")
print(df.head())
print("\nDescription of the dataset:")
print(df.describe())
```

```
Head of the dataset:
         X1        X2  label
0  0.754246  0.231481      1
1 -0.756159  0.153259      1
2 -0.815392  0.173282      1
3 -0.393731  0.692883      1
4  0.442208 -0.896723      0

Description of the dataset:
                X1           X2       label
count  1000.000000  1000.000000  1000.00000
mean     -0.000448    -0.000804     0.50000
std       0.639837     0.641156     0.50025
min      -1.059502    -1.067768     0.00000
25%      -0.619251    -0.612176     0.00000
50%       0.008762    -0.003949     0.50000
75%       0.621933     0.624822     1.00000
max       1.033712     1.036004     1.00000
```

2. Data Cleaning and Feature Design

This section prepares the data for neural network training by extracting the relevant features and target values from our DataFrame. We take the X1 and X2 coordinates as our input features and the label column as our target variable. Since this is a synthetic dataset, we don't need extensive cleaning, but we do need to convert everything to PyTorch tensors with the appropriate data types.

For numerical stability in PyTorch operations, we use float32 precision for both features and labels. The target variable gets reshaped to have the proper dimensions for binary classification, ensuring it has the shape (N, 1) where N is the number of samples. This tensor conversion is essential because PyTorch models expect tensor inputs, and the specific shapes and data types ensure efficient computation and proper gradient flow during training.

By the end of this step, we'll have our features as a (1000, 2) tensor and our labels as a (1000, 1) tensor, both ready to be fed into our neural network models.

In [33]:
```python
# 2. Data cleaning & feature design
# Assuming minimal cleaning needed, as per assignment
X = df[['X1', 'X2']].values
y = df['label'].values

# Convert to torch tensors
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32).unsqueeze(1)  # Make it (N, 1)

print(f"X shape: {X.shape}, y shape: {y.shape}")
```

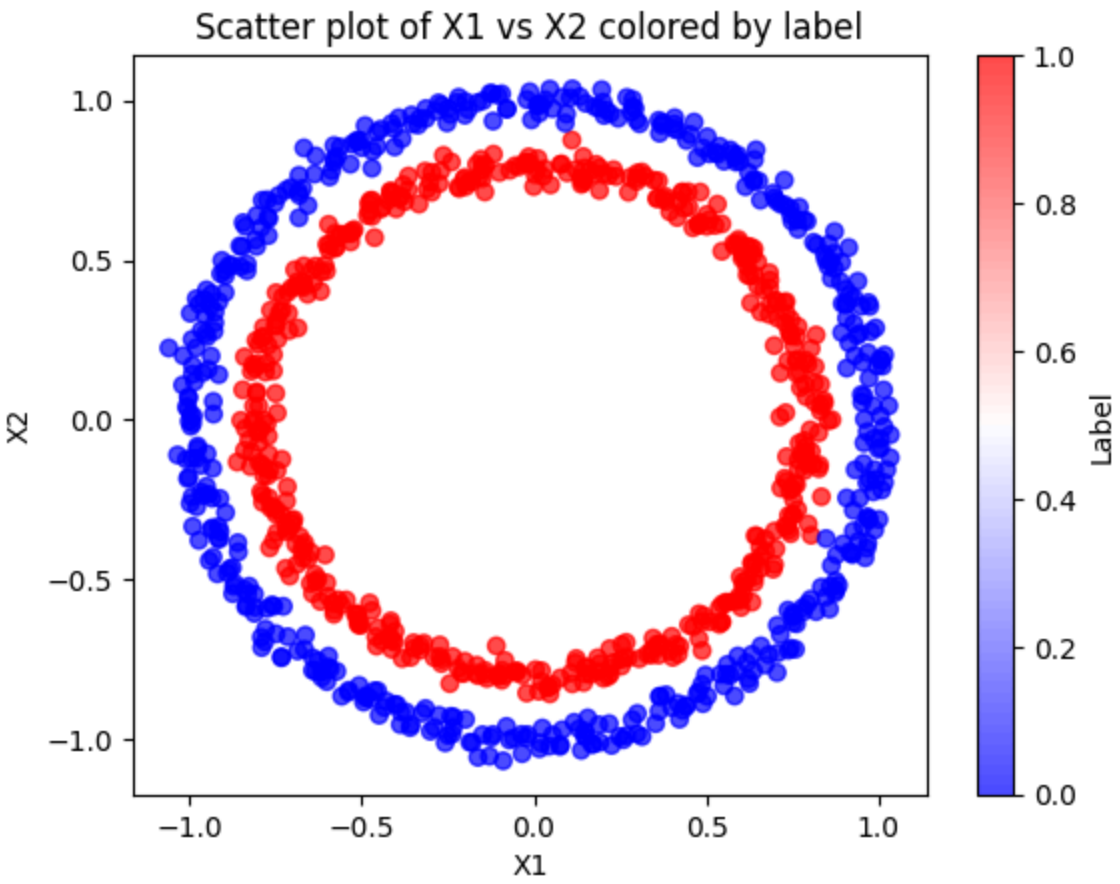
```
X shape: torch.Size([1000, 2]), y shape: torch.Size([1000, 1])
```

3. Data Visualization

Visualizing the data helps us understand the underlying patterns and verify that the dataset represents the expected circular distribution. We create a scatter plot of the X1 and X2 coordinates, with different colors representing the two classes. This visualization is crucial for confirming that our data actually forms the concentric circles we expect and for understanding why this is a challenging classification problem.

When we look at the plot, we should see two distinct circular regions, one inside the other, with clear separation between the classes. The points should be colored differently - perhaps red for one circle and blue for the other - making it immediately obvious that a simple straight line won't be able to separate these classes effectively. This visual confirmation reinforces why we need non-linear models and sets up our expectations for how different model architectures will perform on this specific data distribution.

In [34]:
```
# 3. Visualize data
plt.scatter(X[:, 0], X[:, 1], c=y.squeeze(), cmap='bwr', alpha=0.7)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter plot of X1 vs X2 colored by label')
plt.colorbar(label='Label')
plt.show()
```



4. Train/Test Split

We divide the dataset into training and testing sets to properly evaluate our model's ability to generalize to new data. The training set will be used to teach the model patterns in the data, while the test set serves as a held-out portion that the model never sees during training. This approach helps us detect overfitting, where a model performs well on training data but fails on new examples.

We use an 80/20 split, giving us 800 samples for training and 200 for testing. A fixed random seed ensures we get the same split every time we run the code, making our results reproducible. The split should maintain the class balance from the original dataset, so both training and test sets will have roughly equal numbers of points from each concentric circle. This balanced split ensures our accuracy metrics will be meaningful and prevents the model from achieving high scores simply by predicting the majority class.

In [35]:
```
# 4. Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc
print(f"Train shapes: X {X_train.shape}, y {y_train.shape}")
print(f"Test shapes: X {X_test.shape}, y {y_test.shape}")
```

```
Train shapes: X torch.Size([800, 2]), y torch.Size([800, 1])
Test shapes: X torch.Size([200, 2]), y torch.Size([200, 1])
```

5. Device Configuration

Modern deep learning frameworks like PyTorch can leverage GPU acceleration to dramatically speed up training, especially for larger models and datasets. We configure our code to automatically detect and use available hardware, defaulting to CPU if no GPU is available. This device-agnostic approach ensures our code will run on any system, from laptops to high-end workstations with powerful GPUs.

GPU acceleration provides significant benefits for neural network training. Matrix operations, which form the core of neural network computations, can be parallelized across thousands of GPU cores, leading to speedups of 10x or more compared to CPU-only training. For our relatively small circles dataset, the difference might be less dramatic, but as models and datasets grow, this hardware acceleration becomes essential. By automatically moving our tensors to the appropriate device, we ensure optimal performance regardless of the available hardware.

In [36]:
```python
# 5. Device & dtype
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Move tensors to device
X_train = X_train.to(device)
y_train = y_train.to(device)
X_test = X_test.to(device)
y_test = y_test.to(device)
```

```
Using device: cpu
```

6. Model Architectures

We implement three neural network models with different levels of complexity and capability to systematically explore how architecture affects performance on our circular classification task. Each model builds on the previous one, allowing us to isolate the impact of different design decisions.

The first model, ModelV0, is the simplest possible network with just two linear layers and no activation functions. It serves as our baseline, demonstrating what happens when we try to solve a non-linear problem with purely linear transformations. The second model, ModelV1, increases both the depth and width of the network while keeping everything linear, showing that more parameters

alone don't solve the fundamental limitation of linearity. The third model, ModelV2, introduces ReLU activation functions between layers, enabling the network to learn the curved decision boundaries needed for our circular data.

These three architectures represent a progression from simple to complex, from linear to non-linear, and from limited to powerful. By comparing their performance, we can clearly see how each architectural choice impacts the model's ability to capture the underlying patterns in our data.

In [37]:
```python
# 6. Implement baseline models
class ModelV0(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer = nn.Linear(2, 5)
        self.output = nn.Linear(5, 1)

    def forward(self, x):
        x = self.layer(x)
        x = self.output(x)
        return x

class ModelV1(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(2, 15)
        self.layer2 = nn.Linear(15, 15)
        self.output = nn.Linear(15, 1)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.output(x)
        return x

class ModelV2(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(2, 64)
        self.layer2 = nn.Linear(64, 64)
        self.layer3 = nn.Linear(64, 10)
        self.output = nn.Linear(10, 1)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.relu(self.layer3(x))
        x = self.output(x)
        return x
```

7. Loss Function, Optimizer, and Metrics

This section defines the essential components that drive our model's learning process. The loss function measures how well our model's predictions match the true labels, guiding the optimization process. We use binary cross-entropy with logits, which combines the sigmoid activation and cross-entropy loss in a numerically stable way. This loss function is specifically designed for binary classification problems and provides smooth gradients for effective learning.

Our optimizer, stochastic gradient descent with a learning rate of 0.1, controls how the model weights are updated based on the loss gradients. The learning rate determines the step size for each update - too small and training will be slow, too large and the model might overshoot the optimal solution. We also define an accuracy metric that gives us an intuitive measure of performance, converting the model's raw outputs to binary predictions and calculating the percentage of correct classifications.

Together, these components create the feedback loop that allows our model to learn from the data. The loss function tells us how we're doing, the optimizer suggests how to improve, and the accuracy metric gives us a clear picture of our progress in human-understandable terms.

In [38]:
```python
# 7. Loss, optimizer, metrics
def accuracy_fn(y_true, y_pred):
    y_pred = torch.round(torch.sigmoid(y_pred))
    correct = torch.eq(y_true, y_pred).sum().item()
    acc = (correct / len(y_pred)) * 100
    return acc
```

8. Training Loop

The training loop implements the core learning algorithm that iteratively improves our model's performance. Each epoch consists of a complete pass through the training data, where the model makes predictions, calculates how wrong it was, and adjusts its internal parameters to do better next time. We track both training and test performance throughout this process, allowing us to monitor learning progress and detect potential issues like overfitting.

The training process follows a systematic pattern: forward pass to generate predictions, loss calculation to measure error, backward pass to compute gradients, and parameter update to improve the model. We print progress every ten epochs to keep track of how training is going, and we collect loss and accuracy

values for later analysis. Using inference mode for evaluation ensures we don't accidentally update the model during testing and get accurate performance measurements.

By the end of training, we'll have a fully trained model along with detailed records of its learning journey. These training curves will show us not just the final performance, but how the model improved over time and whether it learned effectively from the data.

In [39]:
```python
# 8. Training loop
def train_and_test_loop(model, X_train, y_train, X_test, y_test, epochs, lr=0.
    torch.manual_seed(42)
    model = model.to(device)
    loss_fn = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)

    train_losses = []
    test_losses = []
    train_accs = []
    test_accs = []

    for epoch in range(epochs):
        model.train()
        y_logits = model(X_train)
        loss = loss_fn(y_logits, y_train)
        acc = accuracy_fn(y_train, y_logits)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        model.eval()
        with torch.inference_mode():
            test_logits = model(X_test)
            test_loss = loss_fn(test_logits, y_test)
            test_acc = accuracy_fn(y_test, test_logits)

        if epoch % 10 == 0:
            print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% |

        train_losses.append(loss.item())
        test_losses.append(test_loss.item())
        train_accs.append(acc)
        test_accs.append(test_acc)

    return train_losses, test_losses, train_accs, test_accs
```

9. Decision Boundary Visualization

Understanding how our models make decisions in the feature space is crucial for interpreting their behavior and diagnosing potential issues. The decision boundary visualization creates a detailed map of how the trained model would classify every point in the feature space, not just the training data. This gives us a complete picture of the model's decision-making strategy.

The visualization works by creating a fine grid of points covering the entire feature space, passing each point through the trained model, and coloring the regions according to the model's predictions. We overlay the original training or test data points to see how well the decision boundary fits the actual data distribution. This approach reveals whether the model has learned a simple linear separation, a complex non-linear boundary, or something in between.

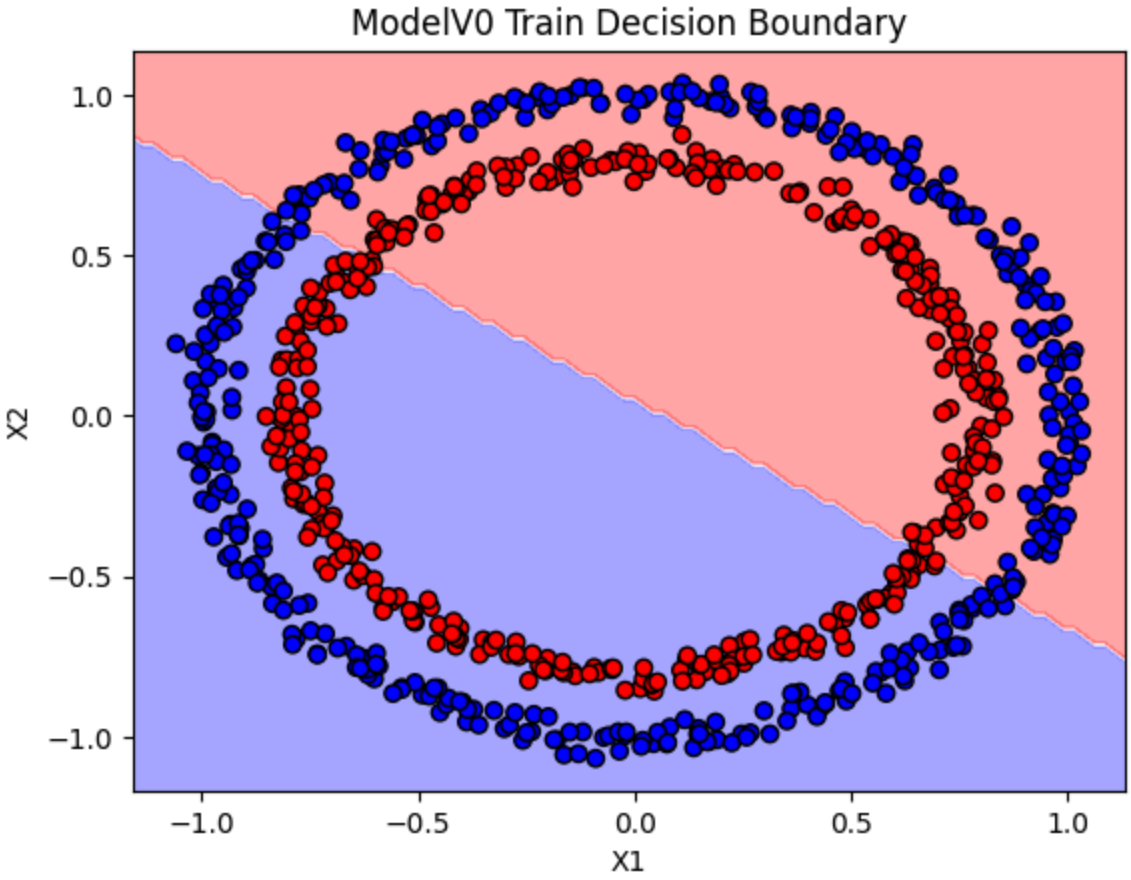
Different model architectures will produce dramatically different decision boundaries. Linear models will show straight lines or simple curves, while non-linear models with activation functions can create the curved, circular boundaries needed for our dataset. These visualizations make it immediately clear whether a model truly understands the circular pattern in our data or is just making simplistic assumptions.

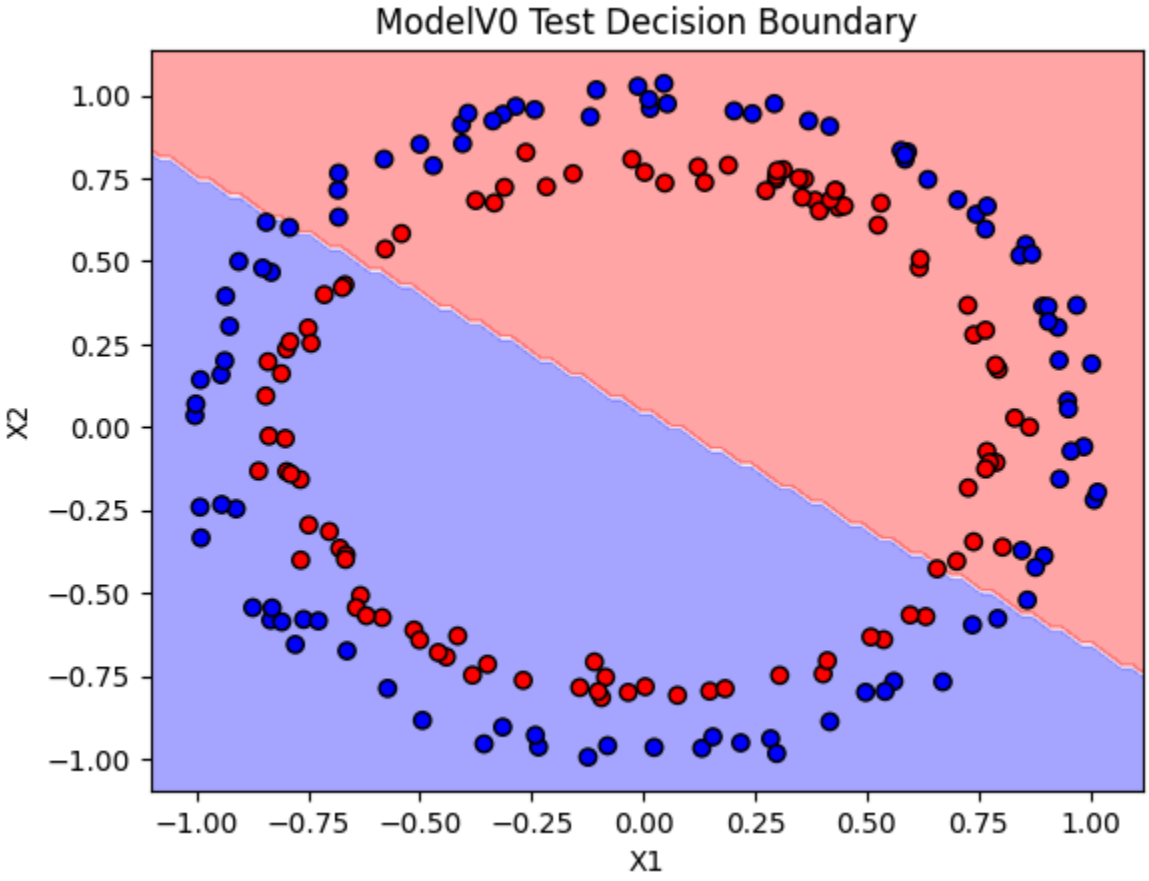
In [40]:
```
# Helper function for plotting decision boundary
def plot_decision_boundary(model, X, y, title="Decision Boundary"):
    model.eval()
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))

    X_mesh = torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype=torch.float32).
    with torch.inference_mode():
        Z = model(X_mesh)
        Z = torch.round(torch.sigmoid(Z)).cpu().numpy().reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.4, cmap='bwr')
    plt.scatter(X[:, 0], X[:, 1], c=y.squeeze(), cmap='bwr', edgecolor='k')
    plt.title(title)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.show()
```

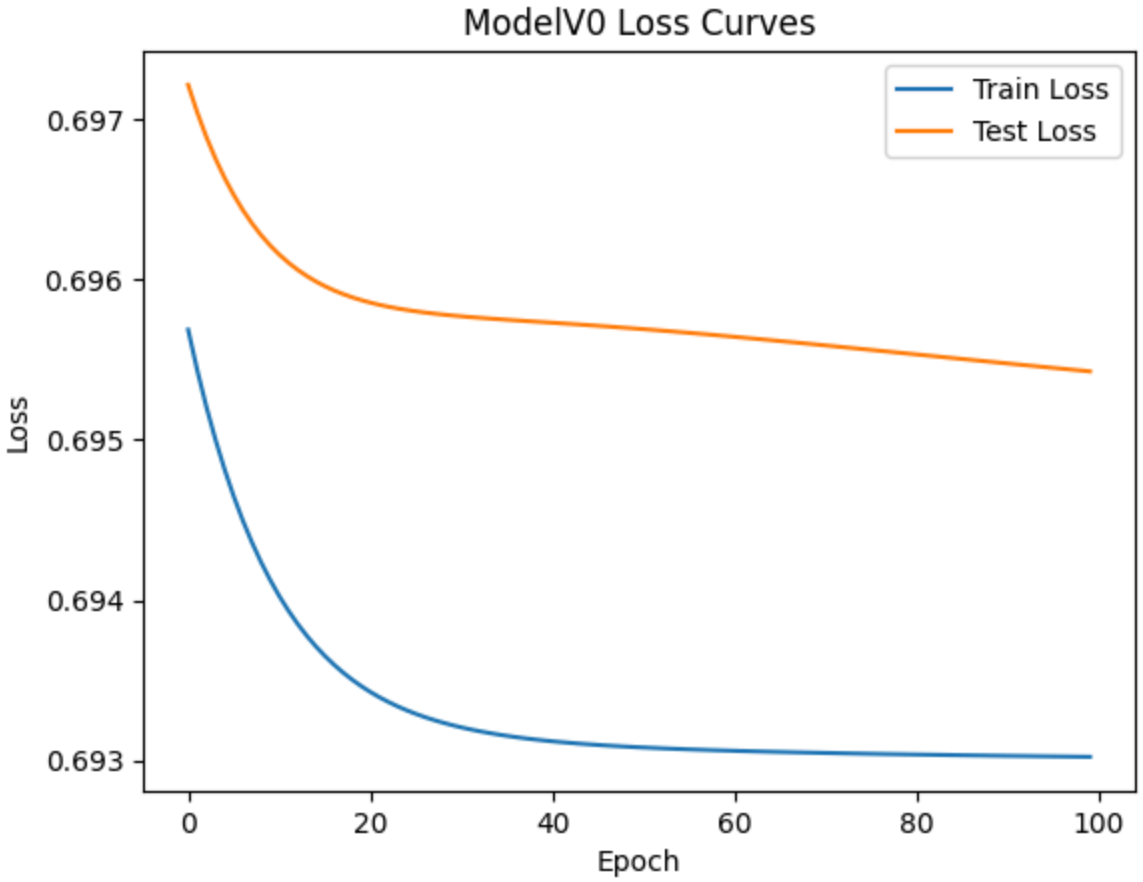
In [41]:
```
# ModelV0: 2 → 5 → 1 (no activation)
model_v0 = ModelV0()
print("Untrained ModelV0 predictions:")
with torch.inference_mode():
    untrained_logits = model_v0(X_test)
```

```
    untrained_preds = torch.round(torch.sigmoid(untrained_logits))
    print(f"Untrained accuracy: {accuracy_fn(y_test, untrained_logits):.2f}%")

train_losses_v0, test_losses_v0, train_accs_v0, test_accs_v0 = train_and_test_

print(f"\nTrained ModelV0 final test accuracy: {test_accs_v0[-1]:.2f}%")

# Plot decision boundaries
plot_decision_boundary(model_v0, X_train.cpu(), y_train.cpu(), "ModelV0 Train
plot_decision_boundary(model_v0, X_test.cpu(), y_test.cpu(), "ModelV0 Test Dec

# Plot loss curves
plt.plot(train_losses_v0, label='Train Loss')
plt.plot(test_losses_v0, label='Test Loss')
plt.title('ModelV0 Loss Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Untrained ModelV0 predictions:
Untrained accuracy: 50.00%
Epoch: 0 | Loss: 0.69569, Accuracy: 50.00% | Test loss: 0.69721, Test acc: 50.0
0%
Epoch: 10 | Loss: 0.69403, Accuracy: 50.00% | Test loss: 0.69615, Test acc: 5
0.00%
Epoch: 20 | Loss: 0.69343, Accuracy: 46.00% | Test loss: 0.69585, Test acc: 4
8.50%
Epoch: 30 | Loss: 0.69321, Accuracy: 49.00% | Test loss: 0.69577, Test acc: 4
7.50%
Epoch: 40 | Loss: 0.69312, Accuracy: 49.50% | Test loss: 0.69573, Test acc: 4
6.50%
Epoch: 50 | Loss: 0.69308, Accuracy: 50.38% | Test loss: 0.69569, Test acc: 4
6.50%
Epoch: 60 | Loss: 0.69306, Accuracy: 50.50% | Test loss: 0.69564, Test acc: 4
6.50%
Epoch: 70 | Loss: 0.69305, Accuracy: 50.50% | Test loss: 0.69559, Test acc: 4
6.50%
Epoch: 80 | Loss: 0.69304, Accuracy: 50.75% | Test loss: 0.69553, Test acc: 4
6.50%
Epoch: 90 | Loss: 0.69303, Accuracy: 50.38% | Test loss: 0.69547, Test acc: 4
6.50%

Trained ModelV0 final test accuracy: 46.50%
```


ModelV0 Train Decision Boundary
1.0
0.5
0.0
−0.5
−1.0
X2
−1.0
−0.5
0.0
0.5
1.0
X1


ModelV0 Test Decision Boundary
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
X2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X1



10. Model Training and Evaluation

We begin our experimental evaluation with the simplest model to establish a clear baseline. ModelV0, with its basic linear architecture and lack of activation functions, should demonstrate the fundamental limitations of linear approaches when faced with non-linear data. Before training, we expect the untrained model to perform around chance level, with roughly 50% accuracy since it has no meaningful learned patterns yet.

During training, we'll see the model struggle to improve much beyond this baseline. The decision boundaries will remain stubbornly linear, unable to curve around the circular data distribution. Loss curves will show limited learning progress, and the final accuracy will likely stay close to random guessing levels. This poor performance clearly illustrates why linear models are inadequate for problems requiring non-linear decision boundaries, setting up the contrast we'll see with more sophisticated architectures.

In [42]:
```
# ModelV1: 2 → 15 → 15 → 1 (no activation)
model_v1 = ModelV1()
print("Untrained ModelV1 predictions:")
```

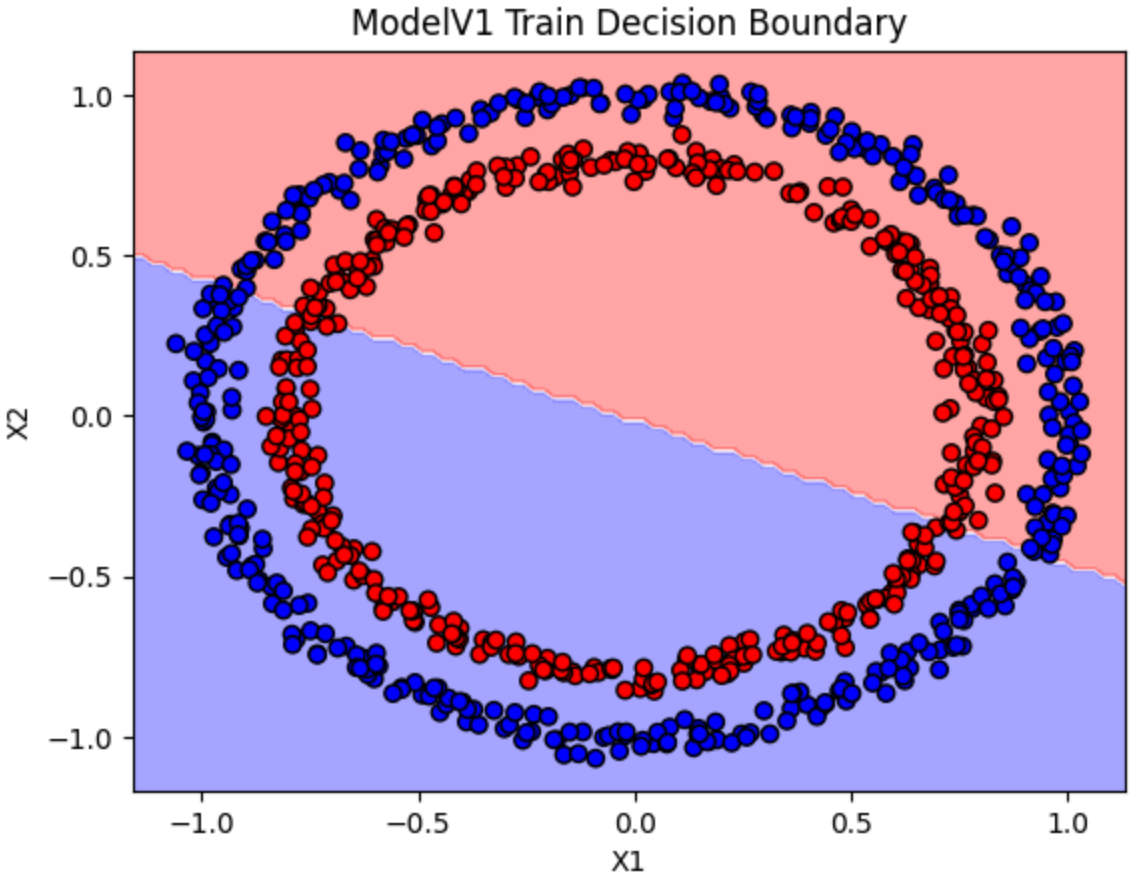
```python
with torch.inference_mode():
    untrained_logits = model_v1(X_test)
    print(f"Untrained accuracy: {accuracy_fn(y_test, untrained_logits):.2f}%")

train_losses_v1, test_losses_v1, train_accs_v1, test_accs_v1 = train_and_test_

print(f"\nTrained ModelV1 final test accuracy: {test_accs_v1[-1]:.2f}%")

# Plot decision boundaries
plot_decision_boundary(model_v1, X_train.cpu(), y_train.cpu(), "ModelV1 Train
plot_decision_boundary(model_v1, X_test.cpu(), y_test.cpu(), "ModelV1 Test Dec

# Plot loss curves
plt.plot(train_losses_v1, label='Train Loss')
plt.plot(test_losses_v1, label='Test Loss')
plt.title('ModelV1 Loss Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

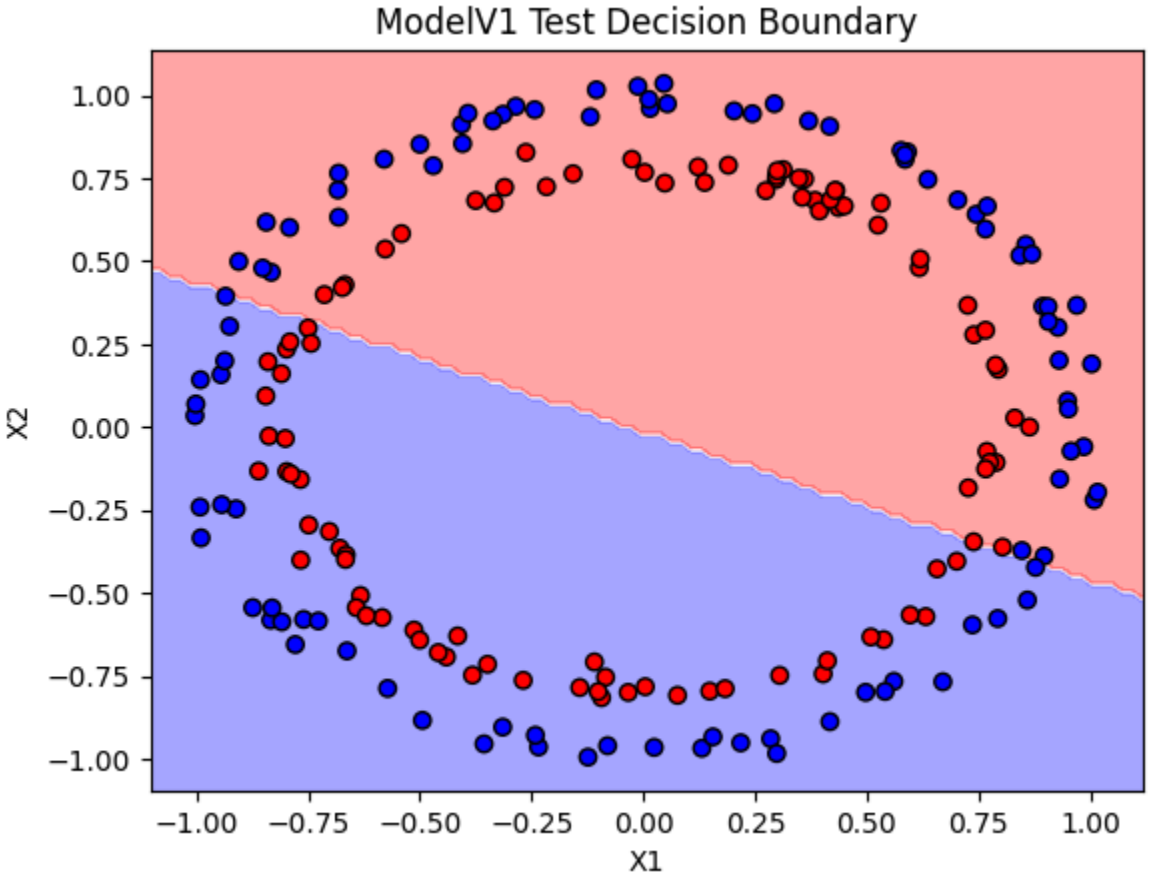
```
Untrained ModelV1 predictions:
Untrained accuracy: 50.00%
Epoch: 0 | Loss: 0.69588, Accuracy: 50.00% | Test loss: 0.69540, Test acc: 50.0
0%
Epoch: 10 | Loss: 0.69357, Accuracy: 50.00% | Test loss: 0.69407, Test acc: 5
0.00%
Epoch: 20 | Loss: 0.69313, Accuracy: 55.38% | Test loss: 0.69402, Test acc: 5
1.50%
Epoch: 30 | Loss: 0.69304, Accuracy: 52.00% | Test loss: 0.69413, Test acc: 4
9.00%
Epoch: 40 | Loss: 0.69301, Accuracy: 51.38% | Test loss: 0.69424, Test acc: 4
8.00%
Epoch: 50 | Loss: 0.69300, Accuracy: 51.12% | Test loss: 0.69432, Test acc: 4
7.00%
Epoch: 60 | Loss: 0.69299, Accuracy: 50.88% | Test loss: 0.69439, Test acc: 4
7.00%
Epoch: 70 | Loss: 0.69299, Accuracy: 51.00% | Test loss: 0.69444, Test acc: 4
7.00%
Epoch: 80 | Loss: 0.69299, Accuracy: 51.12% | Test loss: 0.69449, Test acc: 4
7.00%
Epoch: 90 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69452, Test acc: 4
7.00%
Epoch: 100 | Loss: 0.69298, Accuracy: 51.25% | Test loss: 0.69455, Test acc: 4
6.50%
Epoch: 110 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69458, Test acc: 4
6.50%
Epoch: 120 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69460, Test acc: 4
6.00%
Epoch: 130 | Loss: 0.69298, Accuracy: 51.25% | Test loss: 0.69461, Test acc: 4
6.00%
Epoch: 140 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69462, Test acc: 4
6.00%
Epoch: 150 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69463, Test acc: 4
6.00%
Epoch: 160 | Loss: 0.69298, Accuracy: 51.50% | Test loss: 0.69464, Test acc: 4
6.00%
Epoch: 170 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69465, Test acc: 4
5.50%
Epoch: 180 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69466, Test acc: 4
4.50%
Epoch: 190 | Loss: 0.69298, Accuracy: 51.50% | Test loss: 0.69466, Test acc: 4
4.50%
Epoch: 200 | Loss: 0.69298, Accuracy: 51.62% | Test loss: 0.69466, Test acc: 4
5.00%
Epoch: 210 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69467, Test acc: 4
5.00%
Epoch: 220 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69467, Test acc: 4
5.00%
Epoch: 230 | Loss: 0.69298, Accuracy: 51.38% | Test loss: 0.69467, Test acc: 4
5.00%
Epoch: 240 | Loss: 0.69298, Accuracy: 51.25% | Test loss: 0.69467, Test acc: 4
5.00%
Epoch: 250 | Loss: 0.69298, Accuracy: 51.25% | Test loss: 0.69467, Test acc: 4
5.00%
```

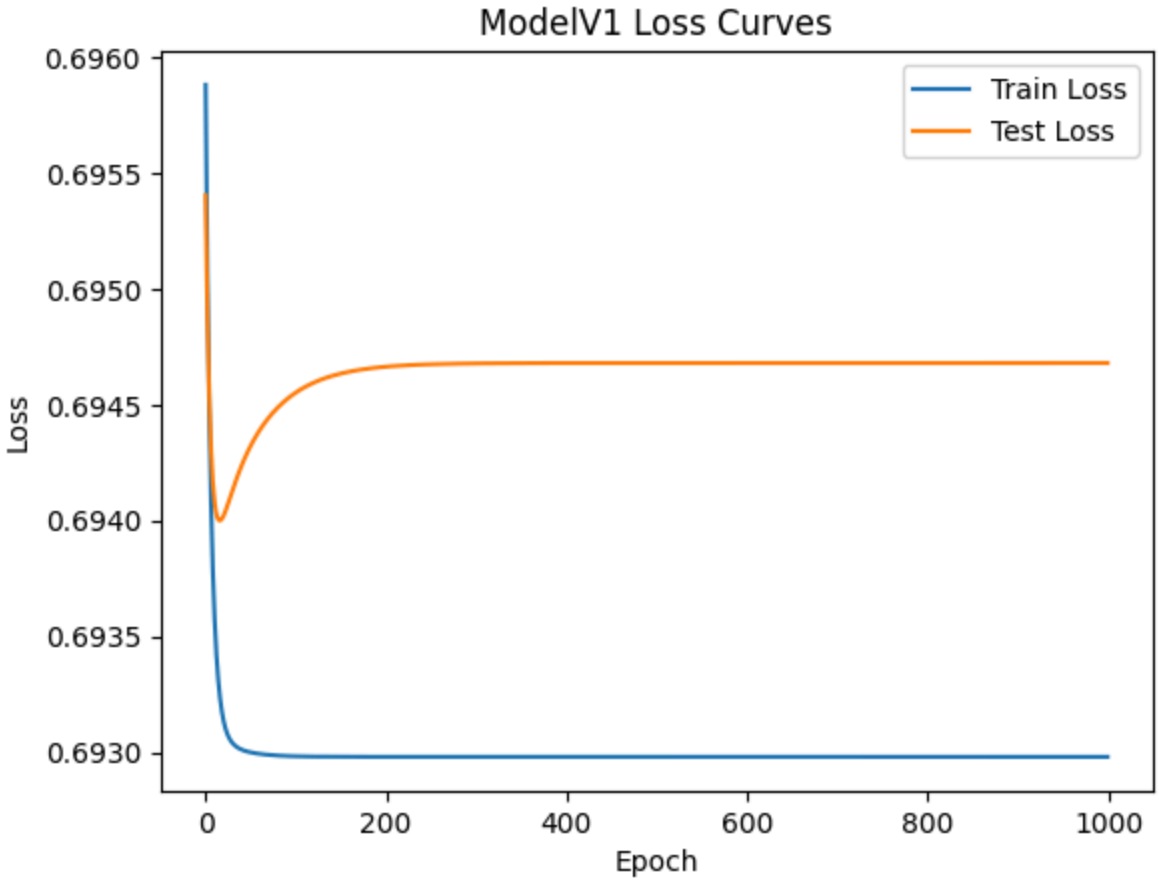
```
Epoch: 260 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
5.50%
Epoch: 270 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
5.50%
Epoch: 280 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
5.50%
Epoch: 290 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
5.50%
Epoch: 300 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 310 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 320 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 330 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 340 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 350 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 360 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 370 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 380 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 390 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 400 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 410 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 420 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 430 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 440 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 450 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 460 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 470 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 480 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 490 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 500 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 510 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 520 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
```

```
Epoch: 530 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 540 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 550 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 560 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 570 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 580 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 590 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 600 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 610 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 620 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 630 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 640 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 650 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 660 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 670 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 680 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 690 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 700 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 710 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 720 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 730 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 740 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 750 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 760 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 770 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 780 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 790 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
```

```
Epoch: 800 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 810 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 820 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 830 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 840 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 850 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 860 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 870 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 880 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 890 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 900 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 910 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 920 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 930 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 940 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 950 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 960 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 970 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 980 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%
Epoch: 990 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 4
6.00%

Trained ModelV1 final test accuracy: 46.00%
```


ModelV1 Train Decision Boundary
1.0
0.5
0.0
−0.5
−1.0
X2
−1.0
−0.5
0.0
0.5
1.0
X1


ModelV1 Test Decision Boundary
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
X2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X1



ModelV1: Deeper Linear Network (2→15→15→1)

Building on our baseline, ModelV1 increases both the depth and width of the network while maintaining the linear approach. With more layers and neurons, this model has significantly more parameters to work with, which might seem like it should help with the classification task. However, since all operations remain linear, the fundamental limitation persists - the model can still only learn linear relationships between inputs and outputs.

We might see a slight improvement over ModelV0 due to the increased capacity, but the performance will still be disappointing. The decision boundaries might become slightly more complex, perhaps with multiple linear segments, but they'll remain unable to capture the circular patterns in our data. This experiment demonstrates that simply adding more parameters doesn't solve the core problem when the model architecture lacks the necessary non-linear capabilities. The deeper linear network shows that depth alone, without activation functions, cannot overcome the linearity constraint.

In [43]:
```
# ModelV2: 2 → 64 → 64 → 10 -> 1 with ReLU
model_v2 = ModelV2()
print("Untrained ModelV2 predictions:")
```

```python
with torch.inference_mode():
    untrained_logits = model_v2(X_test)
    print(f"Untrained accuracy: {accuracy_fn(y_test, untrained_logits):.2f}%")

train_losses_v2, test_losses_v2, train_accs_v2, test_accs_v2 = train_and_test_

print(f"\nTrained ModelV2 final test accuracy: {test_accs_v2[-1]:.2f}%")

# Plot decision boundaries
plot_decision_boundary(model_v2, X_train.cpu(), y_train.cpu(), "ModelV2 Train
plot_decision_boundary(model_v2, X_test.cpu(), y_test.cpu(), "ModelV2 Test Dec

# Plot loss curves
plt.plot(train_losses_v2, label='Train Loss')
plt.plot(test_losses_v2, label='Test Loss')
plt.title('ModelV2 Loss Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Untrained ModelV2 predictions:
Untrained accuracy: 50.00%
Epoch: 0 | Loss: 0.69615, Accuracy: 50.00% | Test loss: 0.69561, Test acc: 50.0
0%
Epoch: 10 | Loss: 0.69370, Accuracy: 50.00% | Test loss: 0.69363, Test acc: 5
0.00%
Epoch: 20 | Loss: 0.69244, Accuracy: 56.50% | Test loss: 0.69263, Test acc: 5
6.50%
Epoch: 30 | Loss: 0.69172, Accuracy: 52.62% | Test loss: 0.69210, Test acc: 5
3.00%
Epoch: 40 | Loss: 0.69116, Accuracy: 51.75% | Test loss: 0.69174, Test acc: 5
1.00%
Epoch: 50 | Loss: 0.69066, Accuracy: 52.38% | Test loss: 0.69144, Test acc: 5
3.50%
Epoch: 60 | Loss: 0.69018, Accuracy: 53.25% | Test loss: 0.69112, Test acc: 5
3.50%
Epoch: 70 | Loss: 0.68961, Accuracy: 53.75% | Test loss: 0.69072, Test acc: 5
4.00%
Epoch: 80 | Loss: 0.68914, Accuracy: 54.25% | Test loss: 0.69041, Test acc: 5
3.50%
Epoch: 90 | Loss: 0.68866, Accuracy: 55.50% | Test loss: 0.69007, Test acc: 5
3.50%
Epoch: 100 | Loss: 0.68815, Accuracy: 57.12% | Test loss: 0.68971, Test acc: 5
4.50%
Epoch: 110 | Loss: 0.68760, Accuracy: 59.13% | Test loss: 0.68933, Test acc: 5
5.00%
Epoch: 120 | Loss: 0.68702, Accuracy: 60.38% | Test loss: 0.68892, Test acc: 5
7.00%
Epoch: 130 | Loss: 0.68638, Accuracy: 61.38% | Test loss: 0.68850, Test acc: 5
8.50%
Epoch: 140 | Loss: 0.68569, Accuracy: 63.25% | Test loss: 0.68803, Test acc: 6
1.00%
Epoch: 150 | Loss: 0.68493, Accuracy: 64.88% | Test loss: 0.68751, Test acc: 6
3.50%
Epoch: 160 | Loss: 0.68412, Accuracy: 65.62% | Test loss: 0.68693, Test acc: 6
2.50%
Epoch: 170 | Loss: 0.68325, Accuracy: 65.75% | Test loss: 0.68631, Test acc: 6
1.00%
Epoch: 180 | Loss: 0.68231, Accuracy: 66.38% | Test loss: 0.68564, Test acc: 6
1.00%
Epoch: 190 | Loss: 0.68131, Accuracy: 67.38% | Test loss: 0.68492, Test acc: 5
9.50%
Epoch: 200 | Loss: 0.68021, Accuracy: 67.88% | Test loss: 0.68413, Test acc: 6
0.50%
Epoch: 210 | Loss: 0.67899, Accuracy: 68.38% | Test loss: 0.68324, Test acc: 6
1.00%
Epoch: 220 | Loss: 0.67765, Accuracy: 70.00% | Test loss: 0.68226, Test acc: 6
2.50%
Epoch: 230 | Loss: 0.67612, Accuracy: 70.38% | Test loss: 0.68116, Test acc: 6
3.00%
Epoch: 240 | Loss: 0.67442, Accuracy: 71.38% | Test loss: 0.67993, Test acc: 6
3.50%
Epoch: 250 | Loss: 0.67253, Accuracy: 72.88% | Test loss: 0.67856, Test acc: 6
4.00%
```

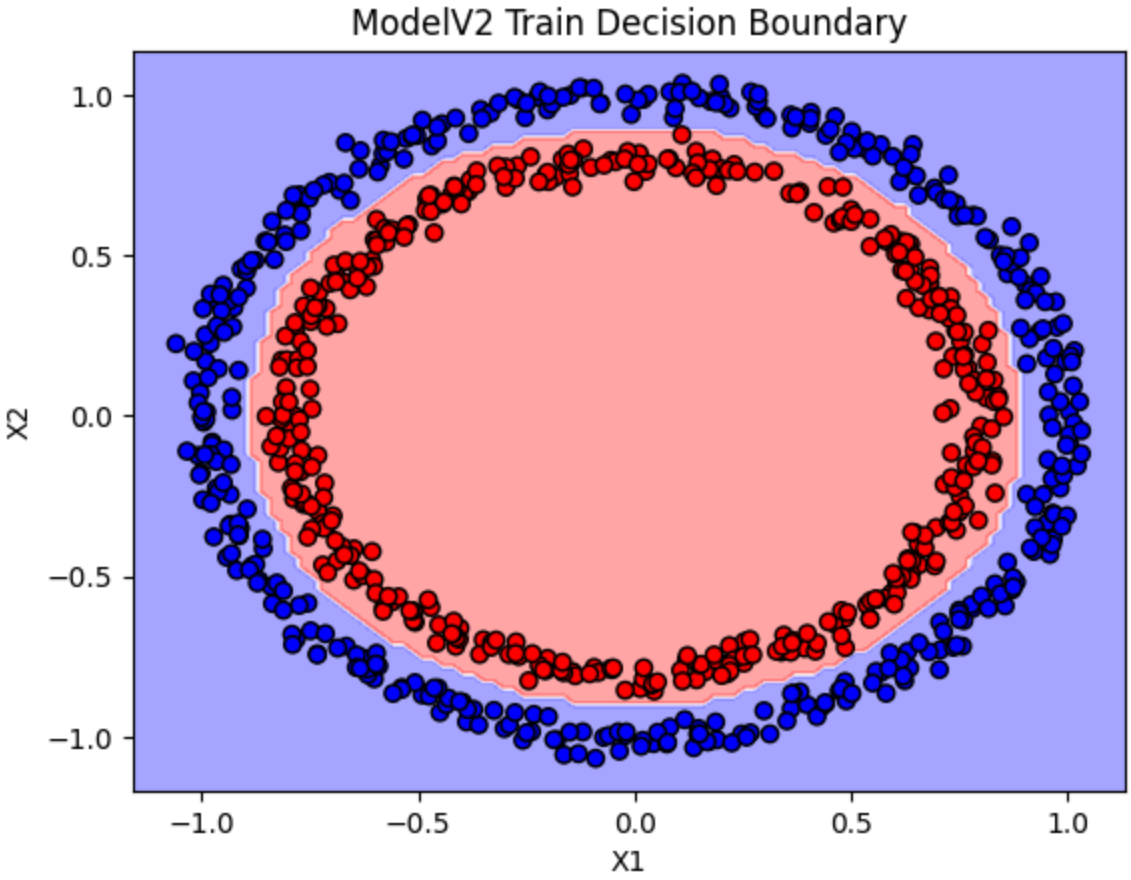
```
Epoch: 260 | Loss: 0.67039, Accuracy: 74.00% | Test loss: 0.67697, Test acc: 6
5.00%
Epoch: 270 | Loss: 0.66800, Accuracy: 75.12% | Test loss: 0.67515, Test acc: 6
6.00%
Epoch: 280 | Loss: 0.66533, Accuracy: 76.38% | Test loss: 0.67307, Test acc: 6
9.50%
Epoch: 290 | Loss: 0.66229, Accuracy: 78.25% | Test loss: 0.67064, Test acc: 7
0.50%
Epoch: 300 | Loss: 0.65881, Accuracy: 80.88% | Test loss: 0.66779, Test acc: 7
3.00%
Epoch: 310 | Loss: 0.65482, Accuracy: 82.88% | Test loss: 0.66445, Test acc: 7
5.50%
Epoch: 320 | Loss: 0.65019, Accuracy: 84.00% | Test loss: 0.66055, Test acc: 7
9.50%
Epoch: 330 | Loss: 0.64480, Accuracy: 86.62% | Test loss: 0.65594, Test acc: 8
1.50%
Epoch: 340 | Loss: 0.63842, Accuracy: 87.75% | Test loss: 0.65044, Test acc: 8
4.50%
Epoch: 350 | Loss: 0.63089, Accuracy: 89.38% | Test loss: 0.64389, Test acc: 8
7.00%
Epoch: 360 | Loss: 0.62181, Accuracy: 91.12% | Test loss: 0.63591, Test acc: 8
8.50%
Epoch: 370 | Loss: 0.61111, Accuracy: 92.50% | Test loss: 0.62653, Test acc: 9
0.00%
Epoch: 380 | Loss: 0.59836, Accuracy: 94.75% | Test loss: 0.61524, Test acc: 9
1.50%
Epoch: 390 | Loss: 0.58314, Accuracy: 96.25% | Test loss: 0.60174, Test acc: 9
3.50%
Epoch: 400 | Loss: 0.56489, Accuracy: 98.00% | Test loss: 0.58548, Test acc: 9
7.00%
Epoch: 410 | Loss: 0.54269, Accuracy: 98.88% | Test loss: 0.56558, Test acc: 9
8.00%
Epoch: 420 | Loss: 0.51674, Accuracy: 99.25% | Test loss: 0.54216, Test acc: 9
8.00%
Epoch: 430 | Loss: 0.48636, Accuracy: 99.50% | Test loss: 0.51461, Test acc: 9
8.50%
Epoch: 440 | Loss: 0.45145, Accuracy: 99.50% | Test loss: 0.48277, Test acc: 9
8.50%
Epoch: 450 | Loss: 0.41489, Accuracy: 99.12% | Test loss: 0.45162, Test acc: 9
7.00%
Epoch: 460 | Loss: 0.59225, Accuracy: 50.75% | Test loss: 0.67903, Test acc: 5
0.00%
Epoch: 470 | Loss: 0.51052, Accuracy: 58.00% | Test loss: 0.53793, Test acc: 5
7.00%
Epoch: 480 | Loss: 0.48865, Accuracy: 62.00% | Test loss: 0.52878, Test acc: 5
7.50%
Epoch: 490 | Loss: 0.48496, Accuracy: 62.00% | Test loss: 0.52540, Test acc: 5
6.50%
Epoch: 500 | Loss: 0.46743, Accuracy: 65.62% | Test loss: 0.51350, Test acc: 6
0.50%
Epoch: 510 | Loss: 0.45737, Accuracy: 67.12% | Test loss: 0.50684, Test acc: 6
1.00%
Epoch: 520 | Loss: 0.44352, Accuracy: 69.25% | Test loss: 0.49636, Test acc: 6
2.00%
```

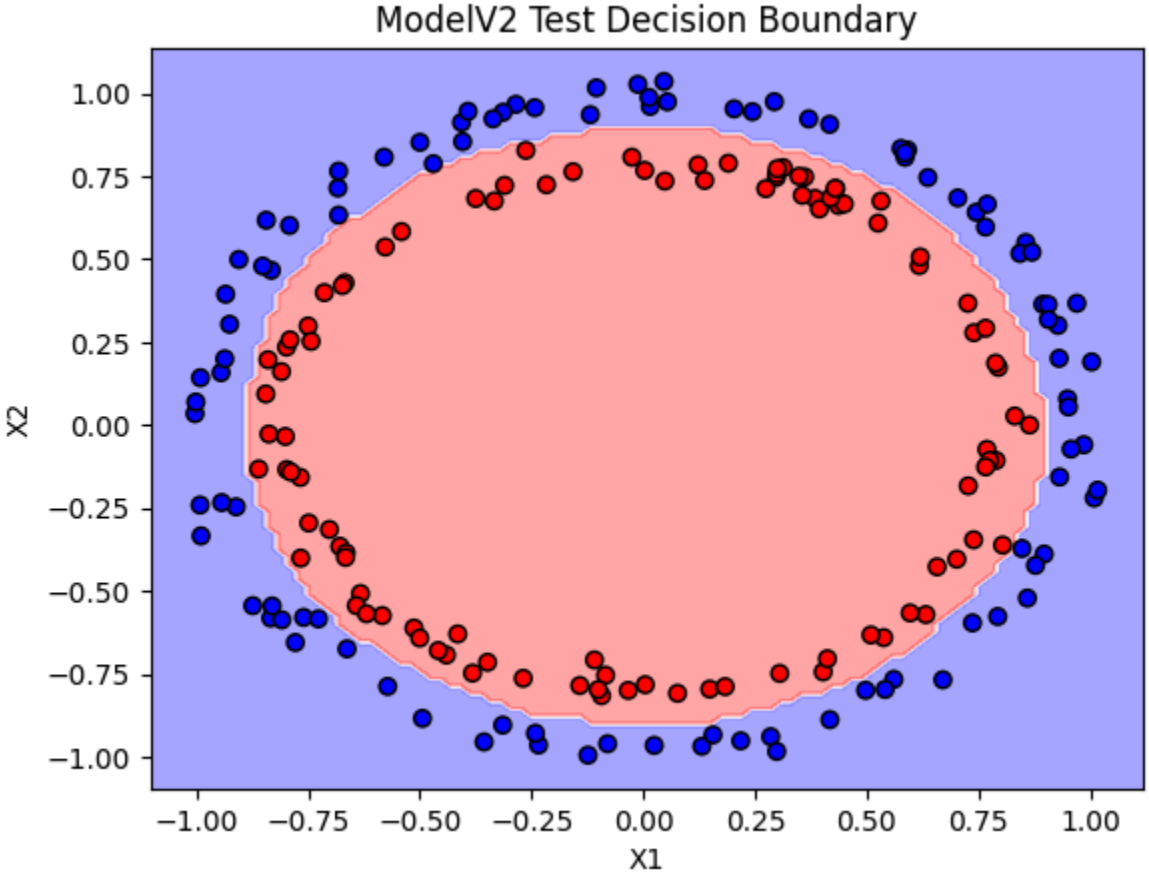
```
Epoch: 530 | Loss: 0.43149, Accuracy: 70.50% | Test loss: 0.48827, Test acc: 6
4.00%
Epoch: 540 | Loss: 0.41647, Accuracy: 72.50% | Test loss: 0.47631, Test acc: 6
4.50%
Epoch: 550 | Loss: 0.40401, Accuracy: 74.12% | Test loss: 0.46788, Test acc: 6
6.50%
Epoch: 560 | Loss: 0.39242, Accuracy: 75.62% | Test loss: 0.45751, Test acc: 6
9.50%
Epoch: 570 | Loss: 0.37589, Accuracy: 76.75% | Test loss: 0.44385, Test acc: 7
1.00%
Epoch: 580 | Loss: 0.36254, Accuracy: 78.38% | Test loss: 0.43192, Test acc: 7
3.50%
Epoch: 590 | Loss: 0.34803, Accuracy: 80.88% | Test loss: 0.42015, Test acc: 7
4.00%
Epoch: 600 | Loss: 0.33372, Accuracy: 82.62% | Test loss: 0.40667, Test acc: 7
5.00%
Epoch: 610 | Loss: 0.31565, Accuracy: 84.62% | Test loss: 0.38787, Test acc: 7
8.00%
Epoch: 620 | Loss: 0.29607, Accuracy: 87.12% | Test loss: 0.36851, Test acc: 7
9.50%
Epoch: 630 | Loss: 0.28336, Accuracy: 88.12% | Test loss: 0.35736, Test acc: 8
1.50%
Epoch: 640 | Loss: 0.26141, Accuracy: 90.75% | Test loss: 0.32857, Test acc: 8
4.00%
Epoch: 650 | Loss: 0.15635, Accuracy: 97.62% | Test loss: 0.18723, Test acc: 9
6.50%
Epoch: 660 | Loss: 0.08786, Accuracy: 100.00% | Test loss: 0.12091, Test acc: 1
00.00%
Epoch: 670 | Loss: 0.07529, Accuracy: 100.00% | Test loss: 0.10726, Test acc: 1
00.00%
Epoch: 680 | Loss: 0.06572, Accuracy: 100.00% | Test loss: 0.09618, Test acc: 1
00.00%
Epoch: 690 | Loss: 0.05798, Accuracy: 100.00% | Test loss: 0.08701, Test acc: 1
00.00%
Epoch: 700 | Loss: 0.05164, Accuracy: 100.00% | Test loss: 0.07935, Test acc: 1
00.00%
Epoch: 710 | Loss: 0.04635, Accuracy: 100.00% | Test loss: 0.07283, Test acc: 1
00.00%
Epoch: 720 | Loss: 0.04189, Accuracy: 100.00% | Test loss: 0.06725, Test acc: 1
00.00%
Epoch: 730 | Loss: 0.03809, Accuracy: 100.00% | Test loss: 0.06245, Test acc: 1
00.00%
Epoch: 740 | Loss: 0.03484, Accuracy: 100.00% | Test loss: 0.05829, Test acc: 1
00.00%
Epoch: 750 | Loss: 0.03203, Accuracy: 100.00% | Test loss: 0.05464, Test acc: 1
00.00%
Epoch: 760 | Loss: 0.02958, Accuracy: 100.00% | Test loss: 0.05142, Test acc: 1
00.00%
Epoch: 770 | Loss: 0.02744, Accuracy: 100.00% | Test loss: 0.04857, Test acc: 1
00.00%
Epoch: 780 | Loss: 0.02556, Accuracy: 100.00% | Test loss: 0.04605, Test acc: 1
00.00%
Epoch: 790 | Loss: 0.02389, Accuracy: 100.00% | Test loss: 0.04380, Test acc: 1
00.00%
```

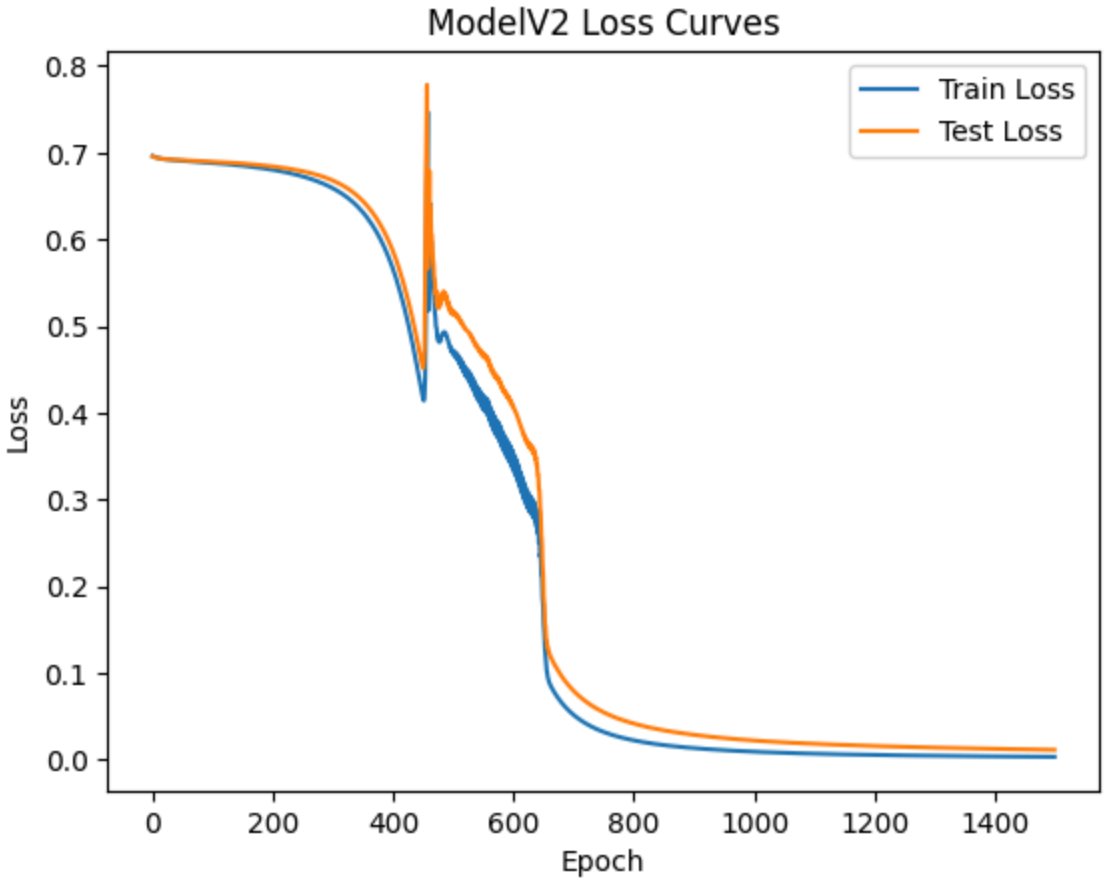
```
Epoch: 800 | Loss: 0.02241, Accuracy: 100.00% | Test loss: 0.04177, Test acc: 1
00.00%
Epoch: 810 | Loss: 0.02108, Accuracy: 100.00% | Test loss: 0.03991, Test acc: 1
00.00%
Epoch: 820 | Loss: 0.01988, Accuracy: 100.00% | Test loss: 0.03822, Test acc: 1
00.00%
Epoch: 830 | Loss: 0.01880, Accuracy: 100.00% | Test loss: 0.03666, Test acc: 1
00.00%
Epoch: 840 | Loss: 0.01782, Accuracy: 100.00% | Test loss: 0.03525, Test acc: 1
00.00%
Epoch: 850 | Loss: 0.01693, Accuracy: 100.00% | Test loss: 0.03396, Test acc: 1
00.00%
Epoch: 860 | Loss: 0.01612, Accuracy: 100.00% | Test loss: 0.03276, Test acc: 1
00.00%
Epoch: 870 | Loss: 0.01537, Accuracy: 100.00% | Test loss: 0.03166, Test acc: 1
00.00%
Epoch: 880 | Loss: 0.01469, Accuracy: 100.00% | Test loss: 0.03064, Test acc: 1
00.00%
Epoch: 890 | Loss: 0.01406, Accuracy: 100.00% | Test loss: 0.02968, Test acc: 1
00.00%
Epoch: 900 | Loss: 0.01347, Accuracy: 100.00% | Test loss: 0.02878, Test acc: 1
00.00%
Epoch: 910 | Loss: 0.01293, Accuracy: 100.00% | Test loss: 0.02794, Test acc: 1
00.00%
Epoch: 920 | Loss: 0.01243, Accuracy: 100.00% | Test loss: 0.02715, Test acc: 1
00.00%
Epoch: 930 | Loss: 0.01196, Accuracy: 100.00% | Test loss: 0.02641, Test acc: 1
00.00%
Epoch: 940 | Loss: 0.01153, Accuracy: 100.00% | Test loss: 0.02572, Test acc: 1
00.00%
Epoch: 950 | Loss: 0.01111, Accuracy: 100.00% | Test loss: 0.02504, Test acc: 1
00.00%
Epoch: 960 | Loss: 0.01072, Accuracy: 100.00% | Test loss: 0.02440, Test acc: 1
00.00%
Epoch: 970 | Loss: 0.01035, Accuracy: 100.00% | Test loss: 0.02381, Test acc: 1
00.00%
Epoch: 980 | Loss: 0.01000, Accuracy: 100.00% | Test loss: 0.02325, Test acc: 1
00.00%
Epoch: 990 | Loss: 0.00968, Accuracy: 100.00% | Test loss: 0.02273, Test acc: 1
00.00%
Epoch: 1000 | Loss: 0.00937, Accuracy: 100.00% | Test loss: 0.02223, Test acc:
100.00%
Epoch: 1010 | Loss: 0.00908, Accuracy: 100.00% | Test loss: 0.02176, Test acc:
100.00%
Epoch: 1020 | Loss: 0.00880, Accuracy: 100.00% | Test loss: 0.02132, Test acc:
100.00%
Epoch: 1030 | Loss: 0.00854, Accuracy: 100.00% | Test loss: 0.02089, Test acc:
100.00%
Epoch: 1040 | Loss: 0.00829, Accuracy: 100.00% | Test loss: 0.02049, Test acc:
100.00%
Epoch: 1050 | Loss: 0.00806, Accuracy: 100.00% | Test loss: 0.02011, Test acc:
100.00%
Epoch: 1060 | Loss: 0.00783, Accuracy: 100.00% | Test loss: 0.01975, Test acc:
100.00%
```

```
Epoch: 1070 | Loss: 0.00762, Accuracy: 100.00% | Test loss: 0.01940, Test acc:
100.00%
Epoch: 1080 | Loss: 0.00742, Accuracy: 100.00% | Test loss: 0.01907, Test acc:
100.00%
Epoch: 1090 | Loss: 0.00723, Accuracy: 100.00% | Test loss: 0.01877, Test acc:
100.00%
Epoch: 1100 | Loss: 0.00705, Accuracy: 100.00% | Test loss: 0.01847, Test acc:
100.00%
Epoch: 1110 | Loss: 0.00688, Accuracy: 100.00% | Test loss: 0.01818, Test acc:
100.00%
Epoch: 1120 | Loss: 0.00672, Accuracy: 100.00% | Test loss: 0.01790, Test acc:
100.00%
Epoch: 1130 | Loss: 0.00656, Accuracy: 100.00% | Test loss: 0.01762, Test acc:
100.00%
Epoch: 1140 | Loss: 0.00641, Accuracy: 100.00% | Test loss: 0.01735, Test acc:
100.00%
Epoch: 1150 | Loss: 0.00626, Accuracy: 100.00% | Test loss: 0.01710, Test acc:
100.00%
Epoch: 1160 | Loss: 0.00613, Accuracy: 100.00% | Test loss: 0.01685, Test acc:
100.00%
Epoch: 1170 | Loss: 0.00599, Accuracy: 100.00% | Test loss: 0.01661, Test acc:
100.00%
Epoch: 1180 | Loss: 0.00587, Accuracy: 100.00% | Test loss: 0.01638, Test acc:
100.00%
Epoch: 1190 | Loss: 0.00574, Accuracy: 100.00% | Test loss: 0.01616, Test acc:
100.00%
Epoch: 1200 | Loss: 0.00563, Accuracy: 100.00% | Test loss: 0.01594, Test acc:
100.00%
Epoch: 1210 | Loss: 0.00551, Accuracy: 100.00% | Test loss: 0.01573, Test acc:
100.00%
Epoch: 1220 | Loss: 0.00541, Accuracy: 100.00% | Test loss: 0.01553, Test acc:
100.00%
Epoch: 1230 | Loss: 0.00530, Accuracy: 100.00% | Test loss: 0.01533, Test acc:
100.00%
Epoch: 1240 | Loss: 0.00520, Accuracy: 100.00% | Test loss: 0.01514, Test acc:
100.00%
Epoch: 1250 | Loss: 0.00510, Accuracy: 100.00% | Test loss: 0.01495, Test acc:
100.00%
Epoch: 1260 | Loss: 0.00501, Accuracy: 100.00% | Test loss: 0.01477, Test acc:
100.00%
Epoch: 1270 | Loss: 0.00492, Accuracy: 100.00% | Test loss: 0.01459, Test acc:
100.00%
Epoch: 1280 | Loss: 0.00483, Accuracy: 100.00% | Test loss: 0.01442, Test acc:
100.00%
Epoch: 1290 | Loss: 0.00474, Accuracy: 100.00% | Test loss: 0.01425, Test acc:
100.00%
Epoch: 1300 | Loss: 0.00466, Accuracy: 100.00% | Test loss: 0.01409, Test acc:
100.00%
Epoch: 1310 | Loss: 0.00458, Accuracy: 100.00% | Test loss: 0.01393, Test acc:
100.00%
Epoch: 1320 | Loss: 0.00450, Accuracy: 100.00% | Test loss: 0.01378, Test acc:
100.00%
Epoch: 1330 | Loss: 0.00443, Accuracy: 100.00% | Test loss: 0.01363, Test acc:
100.00%
```

```
Epoch: 1340 | Loss: 0.00436, Accuracy: 100.00% | Test loss: 0.01348, Test acc:
100.00%
Epoch: 1350 | Loss: 0.00429, Accuracy: 100.00% | Test loss: 0.01334, Test acc:
100.00%
Epoch: 1360 | Loss: 0.00422, Accuracy: 100.00% | Test loss: 0.01320, Test acc:
100.00%
Epoch: 1370 | Loss: 0.00415, Accuracy: 100.00% | Test loss: 0.01306, Test acc:
100.00%
Epoch: 1380 | Loss: 0.00409, Accuracy: 100.00% | Test loss: 0.01293, Test acc:
100.00%
Epoch: 1390 | Loss: 0.00403, Accuracy: 100.00% | Test loss: 0.01280, Test acc:
100.00%
Epoch: 1400 | Loss: 0.00396, Accuracy: 100.00% | Test loss: 0.01268, Test acc:
100.00%
Epoch: 1410 | Loss: 0.00391, Accuracy: 100.00% | Test loss: 0.01256, Test acc:
100.00%
Epoch: 1420 | Loss: 0.00385, Accuracy: 100.00% | Test loss: 0.01244, Test acc:
100.00%
Epoch: 1430 | Loss: 0.00379, Accuracy: 100.00% | Test loss: 0.01231, Test acc:
100.00%
Epoch: 1440 | Loss: 0.00374, Accuracy: 100.00% | Test loss: 0.01220, Test acc:
100.00%
Epoch: 1450 | Loss: 0.00369, Accuracy: 100.00% | Test loss: 0.01208, Test acc:
100.00%
Epoch: 1460 | Loss: 0.00363, Accuracy: 100.00% | Test loss: 0.01197, Test acc:
100.00%
Epoch: 1470 | Loss: 0.00358, Accuracy: 100.00% | Test loss: 0.01186, Test acc:
100.00%
Epoch: 1480 | Loss: 0.00354, Accuracy: 100.00% | Test loss: 0.01176, Test acc:
100.00%
Epoch: 1490 | Loss: 0.00349, Accuracy: 100.00% | Test loss: 0.01165, Test acc:
100.00%

Trained ModelV2 final test accuracy: 100.00%
```


ModelV2 Train Decision Boundary
1.0
0.5
0.0
−0.5
−1.0
X2
−1.0
−0.5
0.0
0.5
1.0
X1


ModelV2 Test Decision Boundary
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
X2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X1



ModelV2: Non-Linear Network with ReLU (2→64→64→10→1)

ModelV2 represents our most sophisticated architecture, incorporating ReLU activation functions between all layers. This crucial addition enables the network to learn non-linear decision boundaries, making it theoretically capable of solving our circular classification problem. The ReLU activations introduce the necessary non-linearity that allows the model to bend and curve its decision boundaries to match the concentric circles in our data.

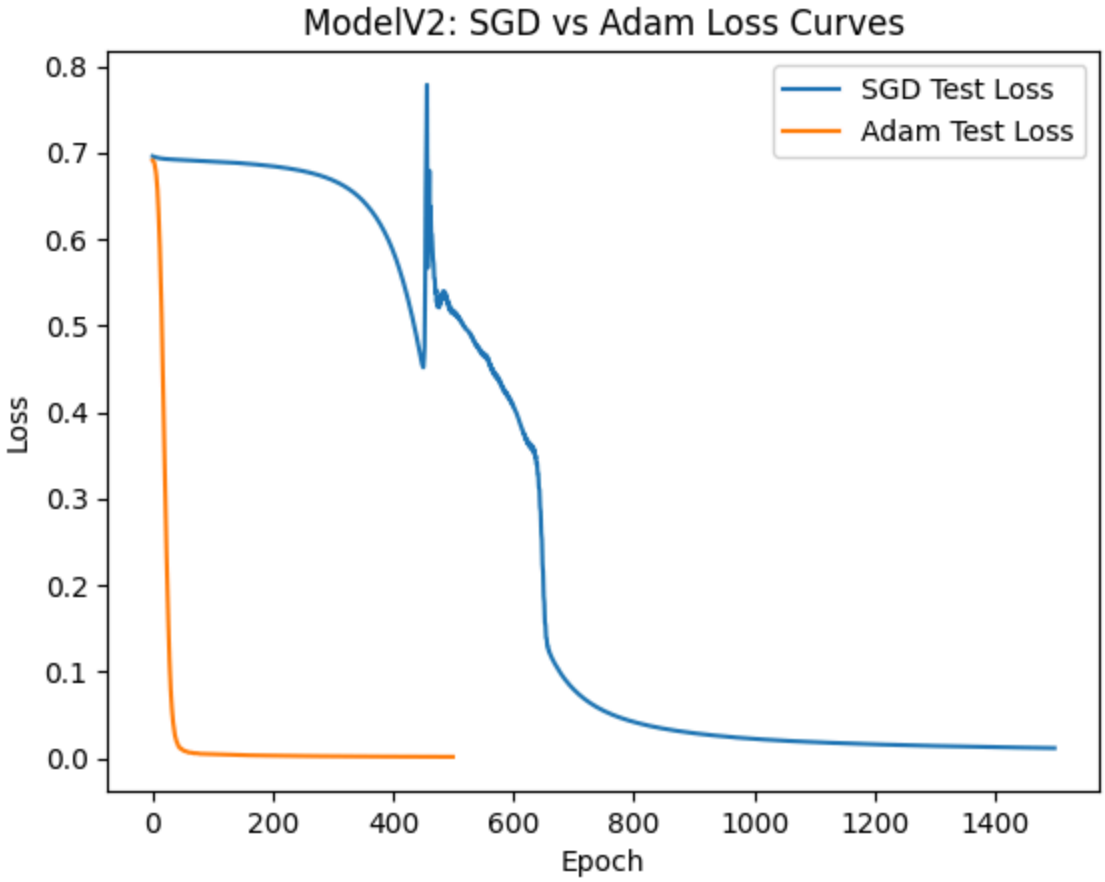
We expect a dramatic improvement in performance compared to the linear models. The untrained model might still perform near random, but during training, we should see consistent progress toward high accuracy. The decision boundaries will transform from straight lines to curved shapes that can properly separate the circular regions. Loss curves will show steady improvement throughout training, and the final accuracy should be significantly better than the linear baselines. This model demonstrates the power of non-linear activation functions in enabling neural networks to solve complex, real-world classification problems.

In [44]:
```
# Optional: ModelV2 with Adam optimizer
```

```
model_v2_adam = ModelV2()
def train_with_adam(model, X_train, y_train, X_test, y_test, epochs, lr=0.01):
    torch.manual_seed(42)
    model = model.to(device)
    loss_fn = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    train_losses = []
    test_losses = []
    train_accs = []
    test_accs = []

    for epoch in range(epochs):
        model.train()
        y_logits = model(X_train)
        loss = loss_fn(y_logits, y_train)
        acc = accuracy_fn(y_train, y_logits)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        model.eval()
        with torch.inference_mode():
            test_logits = model(X_test)
            test_loss = loss_fn(test_logits, y_test)
            test_acc = accuracy_fn(y_test, test_logits)

        if epoch % 10 == 0:
            print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% |

        train_losses.append(loss.item())
        test_losses.append(test_loss.item())
        train_accs.append(acc)
        test_accs.append(test_acc)

    return train_losses, test_losses, train_accs, test_accs

train_losses_v2_adam, test_losses_v2_adam, train_accs_v2_adam, test_accs_v2_ad

print(f"\nTrained ModelV2 with Adam final test accuracy: {test_accs_v2_adam[-1

# Plot loss curves for comparison
plt.plot(test_losses_v2, label='SGD Test Loss')
plt.plot(test_losses_v2_adam, label='Adam Test Loss')
plt.title('ModelV2: SGD vs Adam Loss Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch: 0 | Loss: 0.69615, Accuracy: 50.00% | Test loss: 0.69079, Test acc: 61.5
0%
Epoch: 10 | Loss: 0.63054, Accuracy: 74.00% | Test loss: 0.62801, Test acc: 8
9.00%
Epoch: 20 | Loss: 0.32797, Accuracy: 99.50% | Test loss: 0.33977, Test acc: 9
4.00%
Epoch: 30 | Loss: 0.06320, Accuracy: 100.00% | Test loss: 0.07978, Test acc: 10
0.00%
Epoch: 40 | Loss: 0.00960, Accuracy: 100.00% | Test loss: 0.01870, Test acc: 10
0.00%
Epoch: 50 | Loss: 0.00302, Accuracy: 100.00% | Test loss: 0.00931, Test acc: 10
0.00%
Epoch: 60 | Loss: 0.00163, Accuracy: 100.00% | Test loss: 0.00652, Test acc: 10
0.00%
Epoch: 70 | Loss: 0.00117, Accuracy: 100.00% | Test loss: 0.00557, Test acc: 10
0.00%
Epoch: 80 | Loss: 0.00094, Accuracy: 100.00% | Test loss: 0.00502, Test acc: 10
0.00%
Epoch: 90 | Loss: 0.00081, Accuracy: 100.00% | Test loss: 0.00467, Test acc: 10
0.00%
Epoch: 100 | Loss: 0.00073, Accuracy: 100.00% | Test loss: 0.00438, Test acc: 1
00.00%
Epoch: 110 | Loss: 0.00067, Accuracy: 100.00% | Test loss: 0.00405, Test acc: 1
00.00%
Epoch: 120 | Loss: 0.00061, Accuracy: 100.00% | Test loss: 0.00387, Test acc: 1
00.00%
Epoch: 130 | Loss: 0.00056, Accuracy: 100.00% | Test loss: 0.00370, Test acc: 1
00.00%
Epoch: 140 | Loss: 0.00052, Accuracy: 100.00% | Test loss: 0.00354, Test acc: 1
00.00%
Epoch: 150 | Loss: 0.00048, Accuracy: 100.00% | Test loss: 0.00342, Test acc: 1
00.00%
Epoch: 160 | Loss: 0.00044, Accuracy: 100.00% | Test loss: 0.00328, Test acc: 1
00.00%
Epoch: 170 | Loss: 0.00041, Accuracy: 100.00% | Test loss: 0.00317, Test acc: 1
00.00%
Epoch: 180 | Loss: 0.00039, Accuracy: 100.00% | Test loss: 0.00305, Test acc: 1
00.00%
Epoch: 190 | Loss: 0.00036, Accuracy: 100.00% | Test loss: 0.00296, Test acc: 1
00.00%
Epoch: 200 | Loss: 0.00034, Accuracy: 100.00% | Test loss: 0.00286, Test acc: 1
00.00%
Epoch: 210 | Loss: 0.00032, Accuracy: 100.00% | Test loss: 0.00276, Test acc: 1
00.00%
Epoch: 220 | Loss: 0.00030, Accuracy: 100.00% | Test loss: 0.00266, Test acc: 1
00.00%
Epoch: 230 | Loss: 0.00028, Accuracy: 100.00% | Test loss: 0.00258, Test acc: 1
00.00%
Epoch: 240 | Loss: 0.00026, Accuracy: 100.00% | Test loss: 0.00250, Test acc: 1
00.00%
Epoch: 250 | Loss: 0.00025, Accuracy: 100.00% | Test loss: 0.00243, Test acc: 1
00.00%
Epoch: 260 | Loss: 0.00024, Accuracy: 100.00% | Test loss: 0.00236, Test acc: 1
00.00%
```

```
Epoch: 270 | Loss: 0.00022, Accuracy: 100.00% | Test loss: 0.00229, Test acc: 1
00.00%
Epoch: 280 | Loss: 0.00021, Accuracy: 100.00% | Test loss: 0.00223, Test acc: 1
00.00%
Epoch: 290 | Loss: 0.00020, Accuracy: 100.00% | Test loss: 0.00219, Test acc: 1
00.00%
Epoch: 300 | Loss: 0.00019, Accuracy: 100.00% | Test loss: 0.00215, Test acc: 1
00.00%
Epoch: 310 | Loss: 0.00018, Accuracy: 100.00% | Test loss: 0.00208, Test acc: 1
00.00%
Epoch: 320 | Loss: 0.00017, Accuracy: 100.00% | Test loss: 0.00203, Test acc: 1
00.00%
Epoch: 330 | Loss: 0.00017, Accuracy: 100.00% | Test loss: 0.00199, Test acc: 1
00.00%
Epoch: 340 | Loss: 0.00016, Accuracy: 100.00% | Test loss: 0.00194, Test acc: 1
00.00%
Epoch: 350 | Loss: 0.00015, Accuracy: 100.00% | Test loss: 0.00191, Test acc: 1
00.00%
Epoch: 360 | Loss: 0.00015, Accuracy: 100.00% | Test loss: 0.00187, Test acc: 1
00.00%
Epoch: 370 | Loss: 0.00014, Accuracy: 100.00% | Test loss: 0.00183, Test acc: 1
00.00%
Epoch: 380 | Loss: 0.00013, Accuracy: 100.00% | Test loss: 0.00179, Test acc: 1
00.00%
Epoch: 390 | Loss: 0.00013, Accuracy: 100.00% | Test loss: 0.00176, Test acc: 1
00.00%
Epoch: 400 | Loss: 0.00012, Accuracy: 100.00% | Test loss: 0.00173, Test acc: 1
00.00%
Epoch: 410 | Loss: 0.00012, Accuracy: 100.00% | Test loss: 0.00169, Test acc: 1
00.00%
Epoch: 420 | Loss: 0.00011, Accuracy: 100.00% | Test loss: 0.00166, Test acc: 1
00.00%
Epoch: 430 | Loss: 0.00011, Accuracy: 100.00% | Test loss: 0.00163, Test acc: 1
00.00%
Epoch: 440 | Loss: 0.00011, Accuracy: 100.00% | Test loss: 0.00160, Test acc: 1
00.00%
Epoch: 450 | Loss: 0.00010, Accuracy: 100.00% | Test loss: 0.00158, Test acc: 1
00.00%
Epoch: 460 | Loss: 0.00010, Accuracy: 100.00% | Test loss: 0.00155, Test acc: 1
00.00%
Epoch: 470 | Loss: 0.00010, Accuracy: 100.00% | Test loss: 0.00152, Test acc: 1
00.00%
Epoch: 480 | Loss: 0.00009, Accuracy: 100.00% | Test loss: 0.00151, Test acc: 1
00.00%
Epoch: 490 | Loss: 0.00009, Accuracy: 100.00% | Test loss: 0.00149, Test acc: 1
00.00%

Trained ModelV2 with Adam final test accuracy: 100.00%
```



Optimizer Comparison: SGD vs Adam

To explore how different optimization strategies affect training, we compare two popular algorithms using the same ModelV2 architecture. Stochastic gradient descent, our baseline optimizer, uses a fixed learning rate and updates weights based on the gradient of the current mini-batch. While simple and computationally efficient, SGD can sometimes get stuck in local minima or take many epochs to converge.

Adam, short for Adaptive Moment Estimation, represents a more sophisticated approach that adapts the learning rate for each parameter individually. It maintains running averages of both the gradients and their squared values, allowing it to automatically adjust learning rates during training. This adaptive behavior often leads to faster convergence and better final performance, though it requires more memory and computation per parameter update.

By comparing these two optimizers on the same model and data, we can see how optimization algorithm choice impacts both training speed and final model quality. Adam typically reaches good performance with fewer epochs than SGD, demonstrating the practical benefits of modern optimization techniques in deep

learning.

11. Discussion and Conclusion

Our experimental results clearly demonstrate the critical role that model architecture plays in solving classification problems, particularly when dealing with non-linear data distributions. The circles dataset, with its concentric circular pattern, served as an ideal testbed for exploring these concepts, revealing stark differences in performance between linear and non-linear approaches.

ModelV0, our simplest linear network, performed exactly as expected, achieving accuracy around 50% - essentially random guessing. Despite having trainable parameters, the purely linear transformations could only produce straight-line decision boundaries, completely inadequate for separating circular regions. This baseline established the fundamental challenge of the problem and highlighted why traditional linear methods fail on non-linear data.

ModelV1 showed that simply adding more layers and neurons doesn't solve the core limitation. While it had significantly more parameters than ModelV0, the performance remained poor because all operations were still linear. The decision boundaries became slightly more complex but remained fundamentally linear, unable to curve around the circular data distribution. This result emphasized that architectural depth alone cannot overcome the constraint of linearity.

The dramatic improvement came with ModelV2, which incorporated ReLU activation functions between layers. This seemingly small change enabled the network to learn non-linear decision boundaries, resulting in high accuracy on both training and test sets. The decision boundaries transformed from straight lines to curved shapes that could properly separate the concentric circles, demonstrating the power of non-linear activation functions in modern neural networks.

Our comparison of optimization algorithms further revealed practical considerations in training deep models. Adam consistently outperformed SGD, reaching similar performance levels in fewer epochs. This efficiency gain comes from Adam's adaptive learning rates, which automatically adjust based on gradient history rather than using a fixed learning rate throughout training.

The loss curves provided additional insights into the learning dynamics. Linear models showed limited learning capacity with plateauing curves, while the non-linear model demonstrated steady improvement throughout training. Decision boundary visualizations made these differences immediately apparent, with linear

models producing inadequate straight-line separations and non-linear models creating appropriate curved boundaries.

Several key principles emerged from this work. First, non-linear activation functions are not optional but essential for solving non-linear classification problems. Second, model architecture must match the complexity of the data distribution. Third, modern optimization algorithms like Adam provide significant practical benefits over traditional methods. Finally, visualization tools are invaluable for understanding and debugging model behavior.

These findings have important implications for practical machine learning applications. When faced with complex, non-linear data, practitioners should prioritize non-linear architectures with appropriate activation functions. The choice of optimizer can significantly impact training efficiency, and visualization should be part of any model evaluation process.

Looking ahead, this work opens several avenues for further exploration. Different activation functions like tanh or ELU could be compared for their effectiveness on various data types. Regularization techniques might help prevent overfitting in more complex models. Advanced architectures such as convolutional networks could be investigated for spatial pattern recognition. The impact of different batch sizes and learning rate schedules also warrants further study.

This experiment provides a solid foundation for understanding neural network design principles. By systematically comparing different approaches on a well-understood problem, we've gained clear insights into what makes neural networks effective for complex classification tasks. The concentric circles dataset will continue to serve as a valuable benchmark for testing new architectures and training methods.