```
#PROGRAM-2
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

def create_data():
  X = np.random.randn(1000, 10)
  y = np.random.randn(1000, 1)
  return X, y

def create_model():
  model = models.Sequential([
    layers.Dense(50, activation="relu", input_shape=(10,)),
    layers.Dense(20, activation="relu"),
    layers.Dense(1)
  ])
  model.summary()  # Display the model architecture
  return model

# Learning rate scheduler callback
def lr_schedule(epoch, lr):
  if epoch > 25:
    return lr * 0.5  # Reduce learning rate after 25 epochs
  return lr

# Early stopping callback
early_stop = callbacks.EarlyStopping(monitor="loss", patience=10, restore_best_weights=True)
```

```
def train_model_with_history(model, optimizer, X, y, batch_size, epochs, optimizer_name):
    model.compile(optimizer=optimizer, loss="mse")
    history = []

    lr_scheduler = callbacks.LearningRateScheduler(lr_schedule)
    for epoch in range(epochs):
        hist = model.fit(X, y, batch_size=batch_size, epochs=1, verbose=0, callbacks=[lr_scheduler,
early_stop])
        loss = hist.history['loss'][0]
        history.append(loss)
        print(f"Epoch {epoch+1}/{epochs} - {optimizer_name} Loss: {loss:.4f}")
        if early_stop.stopped_epoch > 0:
            print(f"{optimizer_name} Early Stopping triggered at epoch {epoch+1}")
            break  # Stop training if early stopping is triggered

    return history

# Generate synthetic data
X, y = create_data()

# Initialize models
model_sgd = create_model()
model_adam = create_model()

# Define optimizers
optimizer_sgd = optimizers.SGD(learning_rate=0.01)
optimizer_adam = optimizers.Adam(learning_rate=0.01)

# Set training parameters
epochs = 50
batch_size = 32
```

```
# Train with SGD optimizer

print("\nTraining with SGD Optimizer:")

sgd_loss = train_model_with_history(model_sgd, optimizer_sgd, X, y, batch_size, epochs,
"SGD")


# Train with Adam optimizer

print("\nTraining with Adam Optimizer:")

adam_loss = train_model_with_history(model_adam, optimizer_adam, X, y, batch_size, epochs,
"Adam")


# Plot the training loss

plt.plot(range(1, len(sgd_loss) + 1), sgd_loss, label='SGD', color='blue')

plt.plot(range(1, len(adam_loss) + 1), adam_loss, label="Adam", color='orange')

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.title("SGD vs Adam Optimizer: Loss Comparison")

plt.legend()

plt.grid(True)

plt.show()
```

To improve and optimize this deep neural network code further, we can make a few modifications:

1. **Learning Rate Schedulers**: We can add a learning rate scheduler to both optimizers to gradually reduce the learning rate during training, which can help the network converge better.
2. **Callback for Early Stopping**: Adding an early stopping callback to stop training when the loss plateaus or starts to increase can prevent overfitting and speed up training.
3. **Model Summary and Random Seed**: Setting a random seed ensures reproducibility, and displaying the model summary helps understand the architecture.

Explanation of the Modifications:

1. **Learning Rate Scheduler**: A scheduler reduces the learning rate by half after 25 epochs to fine-tune the learning process and help achieve better convergence in later epochs.
2. **Early Stopping**: This stops training if the model doesn't improve over 10 consecutive epochs. The model will automatically restore the best weights before stopping.

3. **Random Seed**: Setting a random seed ensures that the results are reproducible.
4. **Model Summary**: Printing the model summary provides a quick view of the architecture and parameter count.

Expected Output:

This modified code will provide:

- Training progress for each epoch, showing loss values and indicating when early stopping is triggered.
- A plot comparing the loss for SGD and Adam optimizers over epochs, showing how they perform differently over time. Generally, Adam might converge faster than SGD, especially with the learning rate adjustments.