REGULARIZATION ON IRIS DATASET USING PYTORCH

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load iris
from sklearn.model selection import train test split
from sklearn.preprocessing import StandardScaler


# Load and split
iris = load iris()
X, y = iris.data, iris.target
X train, X test, y train, y test = train test split(X, y,
test size=0.2, random state=42)


# Standardize
scaler = StandardScaler()
X train = torch.FloatTensor(scaler.fit transform(X train))
X test = torch.FloatTensor(scaler.transform(X test))
y train = torch.LongTensor(y train)
y test = torch.LongTensor(y test)
```

2. Neural Network Model with Dropout

Dropout randomly deactivates neurons during training to prevent over-reliance on specific features

```
class IrisNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(4, 16)
```

```
        self.dropout = nn.Dropout(0.2) # 20% dropout
        self.fc2 = nn.Linear(16, 3)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model = IrisNet()
criterion = nn.CrossEntropyLoss()
```

3. L1/L2 Regularization

L2 (Weight Decay) is directly supported in the optimizer, while L1 requires manual addition to the loss

```
# L2 Regularization (Weight Decay)
optimizer = optim.Adam(model.parameters(), lr=0.01,
weight_decay=1e-5)

# L1 Regularization (manual addition)
l1_lambda = 0.001
for param in model.parameters():
    l1_loss = l1_lambda * torch.sum(torch.abs(param))
    loss = criterion(output, y_train) + l1_loss
```

4. Early Stopping Implementation

Early stopping stops training when validation performance stops improving, preventing over-training

```
epochs = 100
```

```
best_loss = float('inf')
patience = 10
trigger_times = 0

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    # Optional L1 inclusion here

    loss.backward()
    optimizer.step()

    # Validation
    model.eval()
    with torch.no_grad():
        val_outputs = model(X_test)
        val_loss = criterion(val_outputs, y_test)

    # Early Stopping check
    if val_loss < best_loss:
        best_loss = val_loss
        trigger_times = 0
    else:
        trigger_times += 1
        if trigger_times >= patience:
```

```
            print(f"Early stopping at epoch {epoch}")
            break
```

5. Summary of Techniques

- **L1 (Lasso):** Shrinks less important weights to zero, encouraging sparsity.
- **L2 (Ridge/Weight Decay):** Penalizes large weights, improving model stability.
- **Dropout:** Randomly ignores neurons to prevent co-dependency.
- **Early Stopping:** Halts training when validation loss stops improving

****