```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split

import pandas as pd
import matplotlib.pyplot as plt
```

```
class Model(nn.Module):
    def __init__(self, in_features=4, h1=11, h2=11, out_features=3):
        super().__init__()
        self.fc1 = nn.Linear(in_features,h1)    # input layer
        self.fc2 = nn.Linear(h1, h2)            # hidden layer
        self.out = nn.Linear(h2, out_features)  # output layer

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.out(x)
        return x
```

```
torch.manual_seed(32)
model = Model()
```

```
df = pd.read_csv('iris.csv')
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

Start coding or generate with AI.

```
X = df.drop('target',axis=1).values
y = df['target'].values

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=33)

X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
# y_train = F.one_hot(torch.LongTensor(y_train))  # not needed with Cross Entropy Loss
# y_test = F.one_hot(torch.LongTensor(y_test))
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

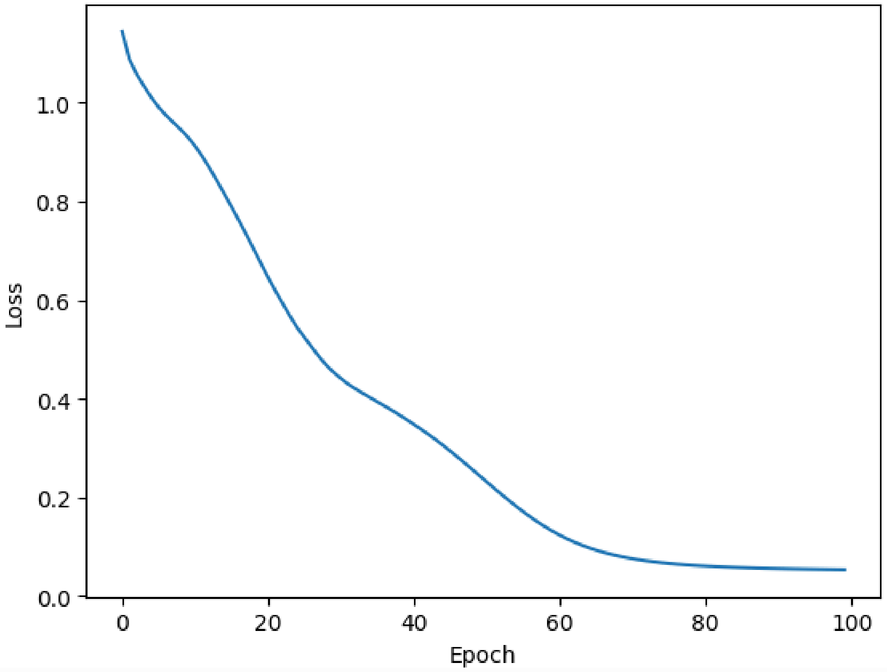
```
trainloader = DataLoader(X_train, batch_size=60, shuffle=True)

testloader = DataLoader(X_test, batch_size=60, shuffle=False)
```

```
torch.manual_seed(4)
model = Model()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)


epochs = 100
losses = []

for i in range(epochs):
    i+=1
    y_pred = model.forward(X_train)
    loss = criterion(y_pred, y_train)
    losses.append(loss)

    # a neat trick to save screen space:
    if i%10 == 1:
        print(f'epoch: {i:2}  loss: {loss.item():10.8f}')

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

```
epoch:  1  loss: 1.14436662
epoch: 11  loss: 0.91206443
epoch: 21  loss: 0.64608634
epoch: 31  loss: 0.44114026
epoch: 41  loss: 0.34795168
epoch: 51  loss: 0.23137699
epoch: 61  loss: 0.12352300
epoch: 71  loss: 0.07574140
epoch: 81  loss: 0.06095325
epoch: 91  loss: 0.05576581
```

```
import numpy as np
import matplotlib.pyplot as plt


losses_np = np.array([loss.detach().cpu().numpy() if hasattr(loss, "detach") else loss for loss in losses])

plt.plot(range(epochs), losses_np)
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



Start coding or generate with AI.

```
with torch.no_grad():
    y_val = model.forward(X_test)
    loss = criterion(y_val, y_test)
print(f'{loss:.8f}')
```

```
0.05263474
```

```
correct = 0
with torch.no_grad():
    for i,data in enumerate(X_test):
        y_val = model.forward(data)
        print(f'{i+1:2}. {str(y_val):38}  {y_test[i]}')
        if y_val.argmax().item() == y_test[i]:
            correct += 1
print(f'\n{correct} out of {len(y_test)} = {100*correct/len(y_test):.2f}% correct')
```

```
 1. tensor([-3.7714,  5.2608, -1.4803])     1
 2. tensor([-3.7917,  5.9327, -2.2263])     1
 3. tensor([ 6.5779, -0.6325, -9.8107])     0
 4. tensor([-5.1013,  4.9140,  0.0534])     1
 5. tensor([-8.4585,  1.8756,  5.9377])     2
 6. tensor([-11.8704,  -0.7724,  11.4140])  2
 7. tensor([ 6.4367, -0.2859, -9.9269])     0
 8. tensor([  7.2504,  -0.9941, -10.5402])  0
 9. tensor([-8.4212,  2.2811,  5.4861])     2
10. tensor([-10.0072,   1.3507,   7.7682])  2
11. tensor([-10.6990,   0.7184,   8.9832])  2
12. tensor([ 6.5320, -0.8837, -9.4954])     0
13. tensor([-10.2032,   0.6214,   8.6037])  2
14. tensor([-5.2836,  4.4757,  0.6222])     1
15. tensor([-8.3126,  2.5742,  5.0969])     2
16. tensor([-3.6743,  5.7227, -2.0963])     1
17. tensor([-6.7409,  3.0659,  3.2318])     2
18. tensor([  7.3239,  -0.9691, -10.6842])  0
19. tensor([-5.1669,  4.9011,  0.0932])     1
20. tensor([-9.1697,  2.6318,  5.8424])     2
21. tensor([  6.8763,  -0.8617, -10.0639])  0
22. tensor([  7.5326,  -0.7552, -11.2263])  0
23. tensor([-10.6115,   0.4025,   9.2094])  2
24. tensor([ 6.8009, -0.8969, -9.9106])     0
25. tensor([-7.5690,  2.1266,  4.9112])     2
26. tensor([-6.8622,  3.0615,  3.3581])     2
27. tensor([-4.8371,  4.8778, -0.1623])     1
28. tensor([-3.1381,  5.3976, -2.4138])     1
29. tensor([-8.1703,  2.3858,  5.1577])     2
30. tensor([-7.7386,  2.2014,  4.9405])     2

30 out of 30 = 100.00% correct
```

```
torch.save(model.state_dict(), 'IrisDatasetModel.pt')
```

```
new_model = Model()
new_model.load_state_dict(torch.load('IrisDatasetModel.pt'))
new_model.eval()
```

```
Model(
  (fc1): Linear(in_features=4, out_features=11, bias=True)
  (fc2): Linear(in_features=11, out_features=11, bias=True)
  (out): Linear(in_features=11, out_features=3, bias=True)
)
```

```
with torch.no_grad():
    y_val = new_model.forward(X_test)
    loss = criterion(y_val, y_test)
print(f'{loss:.8f}')
```

```
0.05263474
```

```
mystery_iris = torch.tensor([5.6,3.7,2.2,0.5])
```

```
with torch.no_grad():
    print(new_model(mystery_iris))
    print()
    print(labels[new_model(mystery_iris).argmax()])
```

```
tensor([  6.4299,   0.2522, -10.4254])

 Iris setosa
```

```
#print(levaku lakshmi mounika)
212223100026
```

```
212223100026
```