```
print('Sai Vishal D')
212223230180
```

```
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
%matplotlib inline
```

```
class Model(nn.Module):
    def __init__(self, in_features=4, h1=10, 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('/content/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

```
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()
```

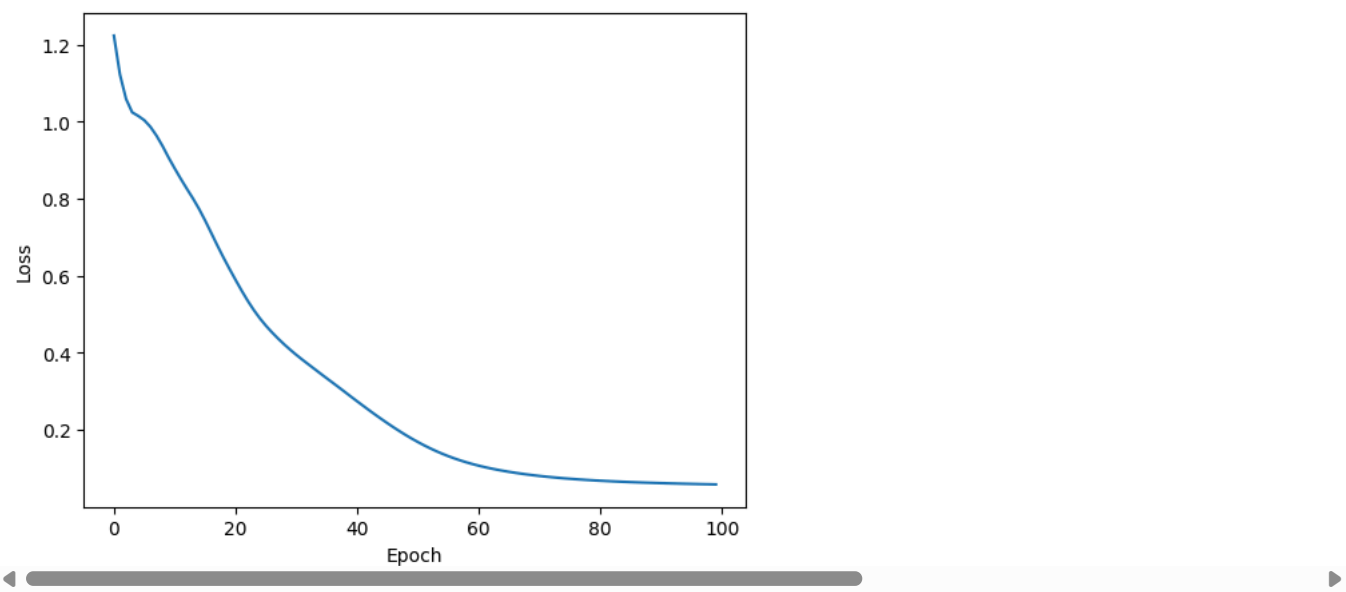
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
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: 1.22303259
epoch: 11  loss: 0.87833655
epoch: 21  loss: 0.58939141
epoch: 31  loss: 0.39461419
epoch: 41  loss: 0.27418667
epoch: 51  loss: 0.16842622
epoch: 61  loss: 0.10710016
epoch: 71  loss: 0.08045476
epoch: 81  loss: 0.06811187
epoch: 91  loss: 0.06185398
```

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

# Convert each tensor in the list to a NumPy array
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()
```




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

```
0.06064259
```

```
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([-2.0868,  5.4851, -0.1823])     1
 2. tensor([-1.3220,  6.1087, -1.5964])     1
 3. tensor([  9.5932,   4.2332, -17.2719])  0
 4. tensor([-3.6740,  5.9091,  1.9384])     1
 5. tensor([-8.4603,  5.7072,  8.9899])     2
 6. tensor([-13.1001,   6.0068,  15.5544])  2
 7. tensor([  9.5459,   4.4276, -17.2599])  0
 8. tensor([ 10.5079,   4.4073, -18.7977])  0
 9. tensor([-8.3007,  5.9583,  8.6161])     2
10. tensor([-10.5082,   6.2144,  11.7113])  2
```

```
11. tensor([-11.4983,   6.1714,  13.1680])  2
12. tensor([  9.3643,   3.9591, -16.8128])  0
13. tensor([-11.0567,   5.9112,  12.6369])  2
14. tensor([-4.0058,  5.7319,  2.4906])     1
15. tensor([-8.2574,  6.1449,  8.4863])     2
16. tensor([-1.3825,  5.8885, -1.3931])     1
17. tensor([-6.2744,  5.6553,  5.8262])     2
18. tensor([ 10.6742,   4.4654, -19.0832])  0
19. tensor([-3.6942,  5.9322,  1.9201])     1
20. tensor([-8.8572,  6.4742,  9.1737])     2
21. tensor([  9.9655,   4.2527, -17.8747])  0
22. tensor([ 11.1947,   4.8331, -20.0420])  0
23. tensor([-11.5588,   5.9512,  13.3713])  2
24. tensor([  9.8141,   4.1626, -17.6011])  0
25. tensor([-7.5715,  5.4811,  7.8430])     2
26. tensor([-6.4338,  5.7112,  6.0476])     2
27. tensor([-3.3512,  5.7538,  1.5223])     1
28. tensor([-0.8566,  5.6694, -2.0454])     1
29. tensor([-8.1018,  5.9380,  8.3533])     2
30. tensor([-7.6796,  5.6191,  7.8819])     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=10, bias=True)
  (fc2): Linear(in_features=10, 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}')
```

```
0.06064259
```

```
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([  9.6885,   5.0318, -17.6774])

Iris setosa
```

Start coding or generate with AI.