1. create a multi-layer deep learning model
2. load data
3. train and validate the model
4. save and load a trained model

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

# CREATE MODEL CLASS

```
#input layer features=4,hidden layer1=8 neurons, hidden layer2=9 neurons, output layer features=3
#fc1 --> fully connected layer 1
#we'll use the rectified linear unit (ReLU) function, f(x)=max(0,x)
#here x basically means the activation function
class Model(nn.Module):
  def __init__(self,in_features=4,h1=8,h2=9,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
```

```
# Instantiate the Model class using parameter defaults:
torch.manual_seed(32)
model = Model()
```

```
print(model)
```

```
Model(
  (fc1): Linear(in_features=4, out_features=8, bias=True)
  (fc2): Linear(in_features=8, out_features=9, bias=True)
  (out): Linear(in_features=9, out_features=3, bias=True)
)
```

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

# LOAD DATASET

```
df = pd.read_csv('/content/iris.csv') # to load iris.csv copy its path here
```

```
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 |

```
df.tail()
```

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | 2.0 |

| | | | | | |
|---|---|---|---|---|---|
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | 2.0 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | 2.0 |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | 2.0 |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | 2.0 |

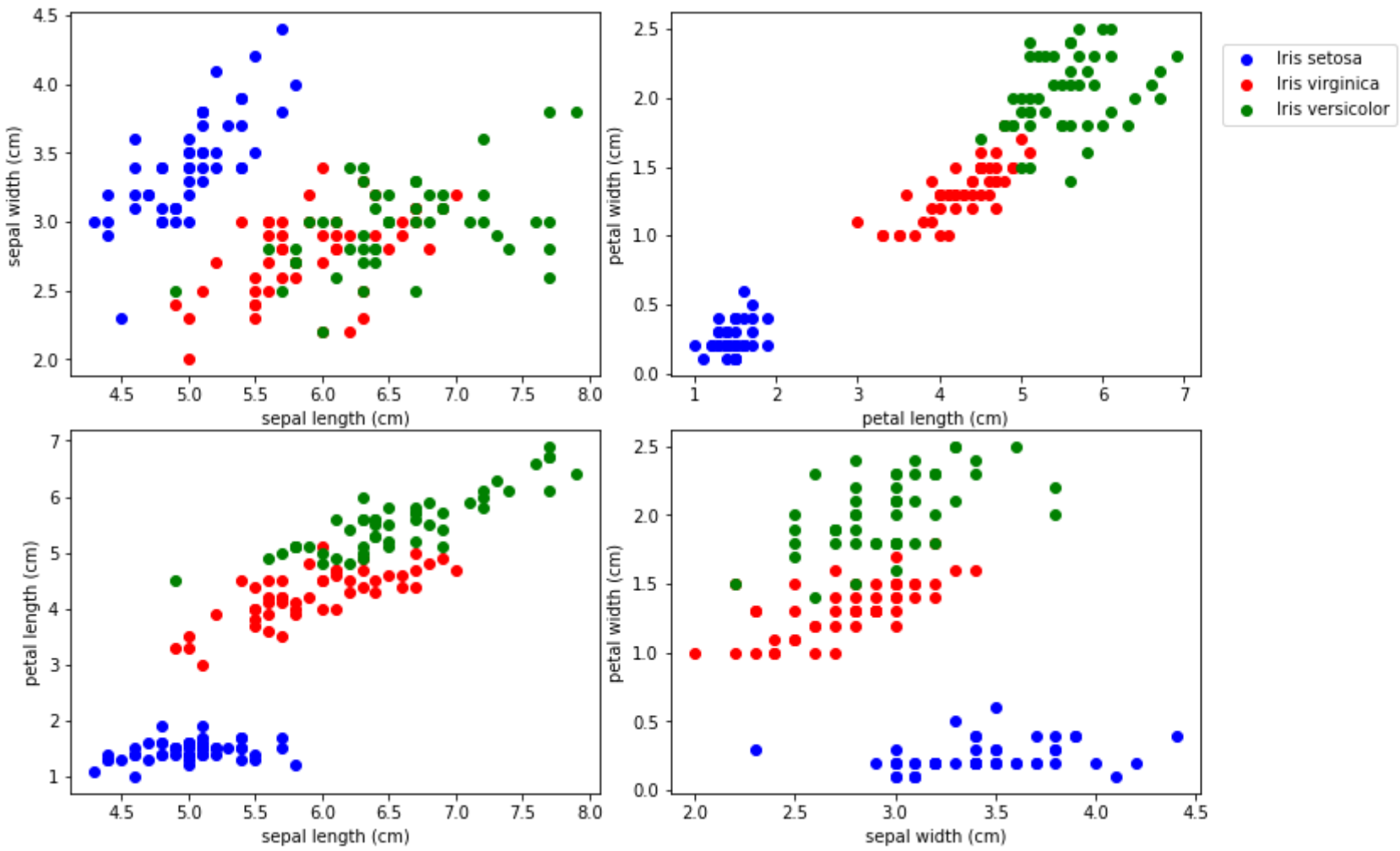
# PLOT DATASET

```
#Here (0,1) sets "sepal length (cm)" as x and "sepal width (cm)" as y
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,7))
fig.tight_layout()

plots = [(0,1),(2,3),(0,2),(1,3)]
colors = ['b', 'r', 'g']
labels = ['Iris setosa','Iris virginica','Iris versicolor']

for i, ax in enumerate(axes.flat):
    for j in range(3):
        x = df.columns[plots[i][0]]
        y = df.columns[plots[i][1]]
        ax.scatter(df[df['target']==j][x], df[df['target']==j][y], color=colors[j])
        ax.set(xlabel=x, ylabel=y)

fig.legend(labels=labels, loc=3, bbox_to_anchor=(1.0,0.85))
plt.show()
```




# PERFORM TRAIN/TEST SPLIT

```
from sklearn.model_selection import train_test_split
```

```
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 = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

# PREPARE DATALOADER

```
1 from torch.utils.data import Dataset, DataLoader
```

```
1 trainloader = DataLoader(X_train, batch_size=60, shuffle=True)
2 testloader = DataLoader(X_test, batch_size=60, shuffle=False)
```

# DEFINE LOSS EQUATIONS AND OPTIMIZATIONS

```
1 # for redo
2 torch.manual_seed(4)
3 model = Model()
```

```
1 # we'll use a variation of Stochastic Gradient Descent called Adam (short for Adaptive Moment Estimati
2 loss_fn = nn.CrossEntropyLoss()
3 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

# TRAIN MODEL

```
 1 epochs = 100
 2 losses = []
 3
 4 for i in range(epochs):
 5     i+=1
 6     y_pred = model.forward(X_train)
 7     loss = loss_fn(y_pred, y_train)
 8     losses.append(loss)
 9
10
11     if i%10 == 0:
12         print(f'epoch: {i}  loss: {loss}')
13
14     optimizer.zero_grad()
15     loss.backward()
16     optimizer.step()
```

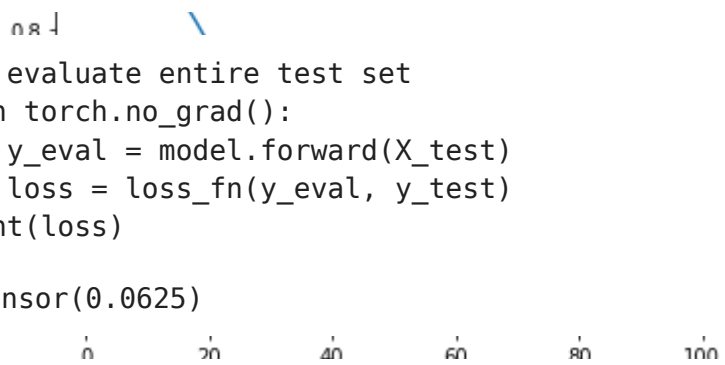
```
epoch: 10  loss: 0.995311975479126
epoch: 20  loss: 0.7865303754806519
epoch: 30  loss: 0.5158727765083313
epoch: 40  loss: 0.3626713156700134
epoch: 50  loss: 0.24050042033195496
epoch: 60  loss: 0.1422368437051773
epoch: 70  loss: 0.09449019283056259
epoch: 80  loss: 0.07497227936983109
epoch: 90  loss: 0.06607460975646973
epoch: 100  loss: 0.061247698962688446
```

# PLOT LOSS FUNC

```
1 plt.plot(range(epochs), losses)
2 plt.ylabel('Loss')
3 plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```

# VALIDATE MODEL



```
#to evaluate entire test set
with torch.no_grad():
    y_eval = model.forward(X_test)
    loss = loss_fn(y_eval, y_test)
print(loss)
```

```
tensor(0.0625)
```

```
correct = 0
with torch.no_grad():
    for i,data in enumerate(X_test):
        y_val = model.forward(data)
        print(f'{i+1})   {str(y_val)}        {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([-0.3360,  7.3629,  1.3780])        1
2)   tensor([0.2770, 8.1552, 0.4267])        1
3)   tensor([ 11.9968,   6.1842, -19.1980])        0
4)   tensor([-2.0192,  7.9662,  4.2445])        1
5)   tensor([-6.1353,  7.9516, 11.0908])        2
6)   tensor([-10.2640,   8.3102,  17.9992])        2
7)   tensor([ 12.0541,   6.4316, -19.2913])        0
8)   tensor([ 12.9496,   6.4815, -20.7530])        0
9)   tensor([-5.7727,  8.2435, 10.5079])        2
10)   tensor([-7.8872,  8.6126, 14.0726])        2
11)   tensor([-8.7060,  8.6074, 15.4331])        2
12)   tensor([ 11.6348,   5.8164, -18.6210])        0
13)   tensor([-8.1013,  8.2331, 14.3883])        2
14)   tensor([-2.0796,  7.7751,  4.3185])        1
15)   tensor([-6.0833,  8.3916, 11.0582])        2
16)   tensor([0.1354, 7.8658, 0.6407])        1
17)   tensor([-4.0880,  7.7216,  7.6638])        2
18)   tensor([ 13.1511,   6.5907, -21.0787])        0
19)   tensor([-1.5649,  8.0220,  3.4751])        1
20)   tensor([-6.2865,  8.9727, 11.4244])        2
21)   tensor([ 12.3848,   6.2568, -19.8265])        0
22)   tensor([ 13.8199,   7.0854, -22.1532])        0
23)   tensor([-8.8475,  8.3181, 15.6471])        2
24)   tensor([ 12.1968,   6.1261, -19.5250])        0
25)   tensor([-5.8089,  7.5468, 10.5336])        2
26)   tensor([-4.4530,  7.7875,  8.2861])        2
27)   tensor([-1.4289,  7.7785,  3.2325])        1
28)   tensor([ 0.5351,  7.5358, -0.0494])        1
29)   tensor([-5.8235,  8.1573, 10.5971])        2
30)   tensor([-5.2573,  7.7475,  9.6101])        2

29 out of 30 = 96.67% correct
```

## SAVE TRAINED MODEL

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

## LOAD A NEW MODEL

```
#weight and bias unknown to model
new_model = Model()
new_model.load_state_dict(torch.load('IrisDatasetModel.pt'))
new_model.eval()
```

```
Model(
  (fc1): Linear(in_features=4, out_features=8, bias=True)
  (fc2): Linear(in_features=8, out_features=9, bias=True)
  (out): Linear(in_features=9, out_features=3, bias=True)
)
```

```
with torch.no_grad():
    y_eval = new_model.forward(X_test)
    loss = loss_fn(y_eval, y_test)
print(loss)
```

```
tensor(0.0625)
```

# APPLY MODEL TO CLASSIFY NEW UNSEEN DATA

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

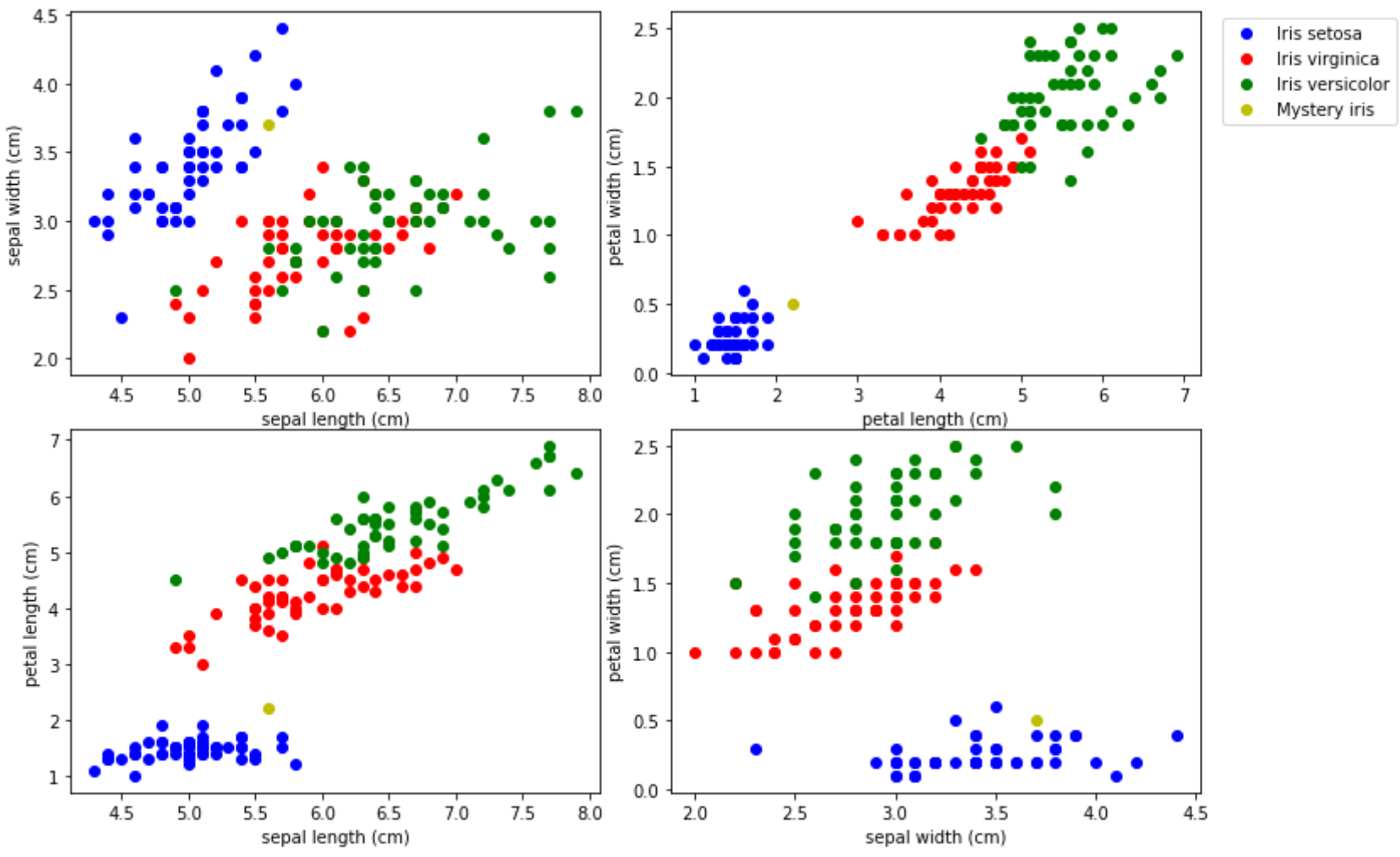
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,7))
fig.tight_layout()

plots = [(0,1),(2,3),(0,2),(1,3)]
colors = ['b', 'r', 'g']
labels = ['Iris setosa','Iris virginica','Iris versicolor','Mystery iris']

for i, ax in enumerate(axes.flat):
    for j in range(3):
        x = df.columns[plots[i][0]]
        y = df.columns[plots[i][1]]
        ax.scatter(df[df['target']==j][x], df[df['target']==j][y], color=colors[j])
        ax.set(xlabel=x, ylabel=y)

    # Add a plot for our mystery iris:
    ax.scatter(mystery_iris[plots[i][0]],mystery_iris[plots[i][1]], color='y')

fig.legend(labels=labels, loc=3, bbox_to_anchor=(1.0,0.85))
plt.show()
```




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

```
tensor([ 12.2112,   7.1279, -19.5248])
Iris setosa
```