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

```
# Create a Model Class that inherits nn.Module
class Model(nn.Module):
  # Input layer (4 features of the flower)
  #   --> Hidden Layer1 H1 (number of neurone)
  #   --> Hidden Layer H2 (n)
  #   --> output (which 3 classes of iris flower)

  def __init__(self, in_features=4, h1=8, h2=9, out_features=3):

    super().__init__() # Instantiate nn.Module (parent class)

    self.fc1 = nn.Linear(in_features, h1)
    self.fc2 = nn.Linear(h1, h2)
    self.out = nn.Linear(h2, out_features)


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

    return x


```

```
# Pick a manual seed for randomization
torch.manual_seed(41)
# Create an instance of the model Model
model = Model()
```

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

```

```
url = "https://gist.githubusercontent.com/netj/" \
  "8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"
my_df = pd.read_csv(url)

```

```
my_df

```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

150 rows × 5 columns

```
my_df.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
my_df.tail()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

```
# Change last column from string to numbers (use as integers afterwards)
my_df["variety"] = my_df.variety.replace('Setosa', 0.0)
my_df["variety"] = my_df.variety.replace('Versicolor', 1.0)
my_df["variety"] = my_df.variety.replace('Virginica', 2.0)
my_df
```

```
# my_df['variety'] = my_df['variety'].replace('Setosa', 0.0)
# my_df['variety'] = my_df['variety'].replace('Versicolor', 1.0)
# my_df['variety'] = my_df['variety'].replace('Virginica', 2.0)
# my_df

```

	sepal.length	sepal.width	petal.length	petal.width	variety
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
...	...	...	...	...	...
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

150 rows × 5 columns

```
# Train Test Split: Set X, y
X = my_df.drop('variety', axis=1)
y = my_df['variety']

```

```
# Convert to numpy arrays
X = X.values
y = y.values
```

```
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1]
```

```
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
```

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

```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=41)
```

```
# Convert X features to float tensors
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
```

```
# Convert y labels to tensors long
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

```
# Set the criterion of model to mesure the error, how far off the predictions are from the data
criterion = nn.CrossEntropyLoss()
# Choose Adam Optimizer, lr = learning rate (if error does not go down after
#   a bunch of iterations (epochs), lower the learning rete)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
model.parameters
```

```
<bound method Module.parameters of 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)
)>
```

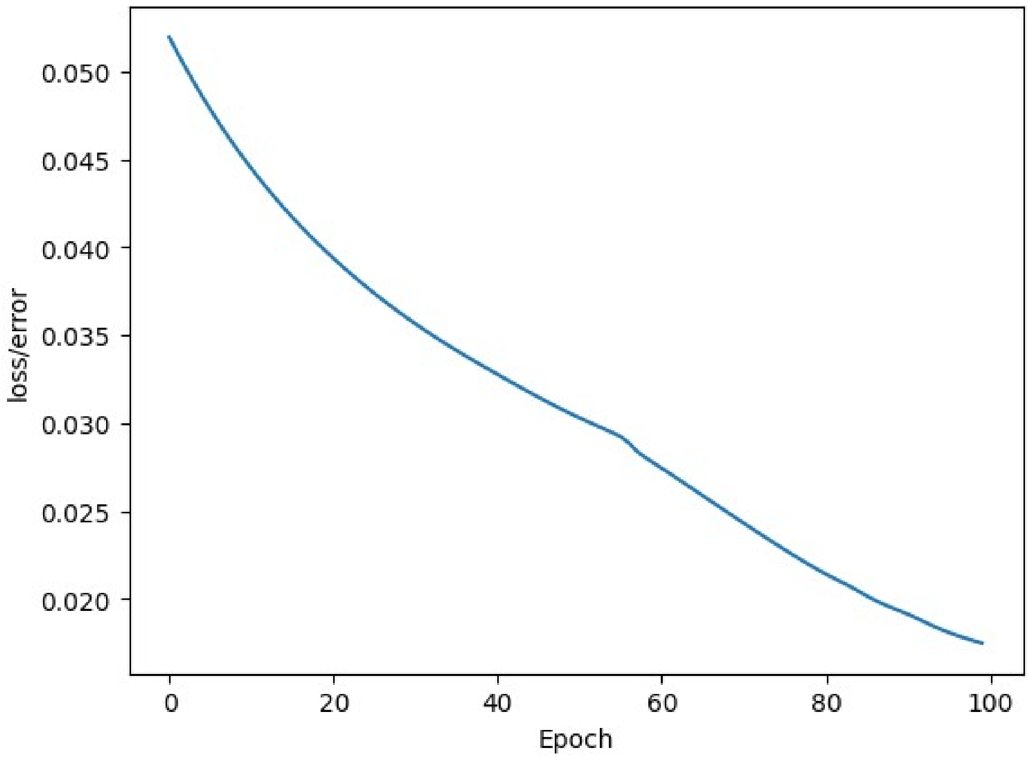
```
# Train our model!
# Epochs? (one run all the training data in our network)
epochs = 100 # How many times
losses = []
for i in range(epochs):
  # Go forward and get a prediction
  y_pred = model.forward(X_train) # Get predicted results

  # Mesure the loss/error, will be high at first
  loss = criterion(y_pred, y_train) # Predicted values vs. the y_train values

  # Keep track of the losses
  losses.append(loss.detach().numpy())

  # Print every 10 epochs
  if i % 10 == 0:
    print(f'Epoch: {i} and loss: {loss}')

  # Do some back propagration: take the error rate of forward propagation
  #   and feed it back through the netword to fine tune the weights
  optimizer.zero_grad()
  loss.backward()
  optimizer.step()



```

```
Epoch: 0 and loss: 0.05193043872714043
Epoch: 10 and loss: 0.04446204751729965
Epoch: 20 and loss: 0.03935651853680611
Epoch: 30 and loss: 0.03563718870282173
Epoch: 40 and loss: 0.032763414084911346
Epoch: 50 and loss: 0.030291257426142693
Epoch: 60 and loss: 0.02742672711610794
```

```
Epoch: 70 and loss: 0.024305408820509911
Epoch: 80 and loss: 0.021412163972854614
Epoch: 90 and loss: 0.019128229469060898
```

```
# Graph it out!
plt.plot(range(epochs), losses)
plt.ylabel('loss/error')
plt.xlabel('Epoch')
```

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




```
# Evaluate Model on Test Data Set (validate model on test set)
with torch.no_grad(): # Basically turn off propagration
  y_val = model.forward(X_test) # X_test are features from the test set. y_val will be predictions
  loss = criterion(y_val, y_test) # Find the loss or error
```

```
 loss
```

```
tensor(0.1784)
```

```
correct = 0

with torch.no_grad():
  for i, data in enumerate(X_test):
    y_val = model.forward(data)

    if y_test[i] == 0:
      x = 'Setosa'
    elif y_test[i] == 1:
      x = 'Versicolor'
    else:
      x = 'Virginica'

    # Will print what type of flower class the network thinks it is
    print(f'{i+1}.) {str(y_val)} \t  {y_test[i]}:{x} \t {y_val.argmax().item()}')

```

```
  # Correct or not
  if y_val.argmax().item()  == y_test[i]:
    correct += 1

print(f'We get {correct} correct!')

```

```
1.) tensor([-7.1455,  3.8413,  8.9871])           2:Virginica    2
2.) tensor([-9.9048,  1.6718, 16.1223])           2:Virginica    2
3.) tensor([-10.9332,   2.8676,  16.4577])        2:Virginica    2
4.) tensor([-3.9721,  7.8497, -1.0292])           1:Versicolor   1
5.) tensor([-9.0422,  3.4844, 12.5741])           2:Virginica    2
6.) tensor([-2.1100,  8.5816, -5.1552])           1:Versicolor   1
7.) tensor([-6.9497,  4.8942,  7.4118])           2:Virginica    2
8.) tensor([-3.8674,  8.0269, -1.4135])           1:Versicolor   1
9.) tensor([-7.9727,  4.1934,  9.9512])           2:Virginica    2
10.) tensor([-10.5891,   1.7318,  17.2016])       2:Virginica    2
11.) tensor([-6.5618,  5.0615,  6.5757])          2:Virginica    2
12.) tensor([ 11.6368,   1.5849, -20.5907])       0:Setosa       0
13.) tensor([ 10.6881,   1.4099, -18.7038])       0:Setosa       0
14.) tensor([-0.4877,  6.8020, -5.8184])          1:Versicolor   1
15.) tensor([  9.4158,   2.5891, -17.9623])       0:Setosa       0
16.) tensor([-6.0888,  5.6836,  5.0628])          2:Virginica    1
17.) tensor([ 10.4663,   1.7461, -18.7670])       0:Setosa       0
18.) tensor([-6.9453,  4.2227,  8.2130])          1:Versicolor   2
19.) tensor([ 12.4135,   1.1981, -21.4224])       0:Setosa       0
20.) tensor([  9.0034,   2.1077, -16.6591])       0:Setosa       0
21.) tensor([-1.1123,  7.4944, -5.5712])          1:Versicolor   1
22.) tensor([-10.0344,   2.6442,  15.2228])       2:Virginica    2
23.) tensor([  9.5920,   2.4860, -18.1442])       0:Setosa       0
24.) tensor([ 11.3869,   1.3970, -19.9024])       0:Setosa       0
25.) tensor([-1.0643,  7.7232, -5.9343])          1:Versicolor   1
26.) tensor([-2.0369,  8.2982, -4.9376])          1:Versicolor   1
27.) tensor([-4.3994,  7.8715, -0.3420])          1:Versicolor   1
28.) tensor([-1.5124,  7.9307, -5.4029])          1:Versicolor   1
29.) tensor([ 11.6238,   1.5221, -20.4831])       0:Setosa       0
30.) tensor([-4.4544,  7.2546,  0.4784])          1:Versicolor   1
We get 28 correct!
```

```
# New data point
new_iris = torch.tensor([4.7, 3.2, 1.3, 0.2])
```

```
#, Add point to iris
with torch.no_grad():
  print(model(new_iris))
```

```
tensor([ 11.3869,   1.3970, -19.9024])
```

```
# New data pint with a 2 in the veriety column (last row of the original iris dataset)
newer_iris = torch.tensor([5.9, 3.0, 5.1, 1.8])
```

```
#, Add point to iris
with torch.no_grad():
  print(model(newer_iris))
```

```
tensor([-7.3228,  3.8048,  9.3361])
```

```
# Save the NN Model
torch.save(model.state_dict(), 'my_really_awesome_iris_model.pt')
```

```
# Load the save Model
new_model = Model()
new_model.load_state_dict(torch.load('my_really_awesome_iris_model.pt'))
```

```
<All keys matched successfully>
```

```
# Make sur the model is loaded correctely
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)
)
```