

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F



1 # Create a Model Class that inherits nn.Module
2 class Model(nn.Module):
3     # Input layer (4 features of the flower)
4     # --> Hidden Layer1 H1 (number of neurone)
5     # --> Hidden Layer H2 (n)
6     # --> output (which 3 classes of iris flower)
7
8     def __init__(self, in_features=4, h1=8, h2=9, out_features=3):
9
10         super().__init__() # Instantiate nn.Module (parent class)
11
12         self.fc1 = nn.Linear(in_features, h1)
13         self.fc2 = nn.Linear(h1, h2)
14         self.out = nn.Linear(h2, out_features)
15
16
17     def forward(self, x):
18         x = F.relu(self.fc1(x))
19         x = F.relu(self.fc2(x))
20         x = self.out(x)
21
22         return x
23
24

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

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4

1 url = "https://gist.githubusercontent.com/netj/" \
2     "8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"
3 my_df = pd.read_csv(url)
4

1 my_df
2
```

	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	

<b>3</b>	4.6	3.1	1.5	0.2	Setosa
<b>4</b>	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
<b>145</b>	6.7	3.0	5.2	2.3	Virginica
<b>146</b>	6.3	2.5	5.0	1.9	Virginica
<b>147</b>	6.5	3.0	5.2	2.0	Virginica
<b>148</b>	6.2	3.4	5.4	2.3	Virginica
<b>149</b>	5.9	3.0	5.1	1.8	Virginica

150 rows × 5 columns



```
1 my_df.head()
```

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

```
1 my_df.tail()
```

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

```
1 # Change last column from string to numbers (use as integers afterwards)
2 my_df["variety"] = my_df.variety.replace('Setosa', 0.0)
3 my_df["variety"] = my_df.variety.replace('Versicolor', 1.0)
4 my_df["variety"] = my_df.variety.replace('Virginica', 2.0)
5 my_df
6 # my_df['variety'] = my_df['variety'].replace('Setosa', 0.0)
7 # my_df['variety'] = my_df['variety'].replace('Versicolor', 1.0)
8 # my_df['variety'] = my_df['variety'].replace('Virginica', 2.0)
9 # my_df
10
```

	sepal.length	sepal.width	petal.length	petal.width	variety	
<b>0</b>	5.1	3.5	1.4	0.2	0.0	
<b>1</b>	4.9	3.0	1.4	0.2	0.0	
<b>2</b>	4.7	3.2	1.3	0.2	0.0	
<b>3</b>	4.6	3.1	1.5	0.2	0.0	
<b>4</b>	5.0	3.6	1.4	0.2	0.0	
...	...	...	...	...	...	
<b>145</b>	6.7	3.0	5.2	2.3	2.0	
<b>146</b>	6.3	2.5	5.0	1.9	2.0	
<b>147</b>	6.5	3.0	5.2	2.0	2.0	
<b>148</b>	6.2	3.4	5.4	2.3	2.0	
<b>149</b>	5.9	3.0	5.1	1.8	2.0	

150 rows × 5 columns

```
1 # Train Test Split: Set X, y
2 X = my_df.drop('variety', axis=1)
3 y = my_df['variety']
4
```

```
1 # Convert to numpy arrays
2 X = X.values
3 y = y.values
```

```
1 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],
       [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. ],
```

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

```
1 # Train Test Split
```

```
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=41)
```

```
1 # Convert X features to float tensors
```

```
2 X_train = torch.FloatTensor(X_train)
```

```
3 X_test = torch.FloatTensor(X_test)
```

```
1 # Convert y labels to tensors long
```

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

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

```
1 # Set the criterion of model to measure the error, how far off the predictions are from the data
```

```
2 criterion = nn.CrossEntropyLoss()
```

```
3 # Choose Adam Optimizer, lr = learning rate (if error does not go down after
```

```
4 # a bunch of iterations (epochs), lower the learning rate)
```

```
5 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
1 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)
)>
```

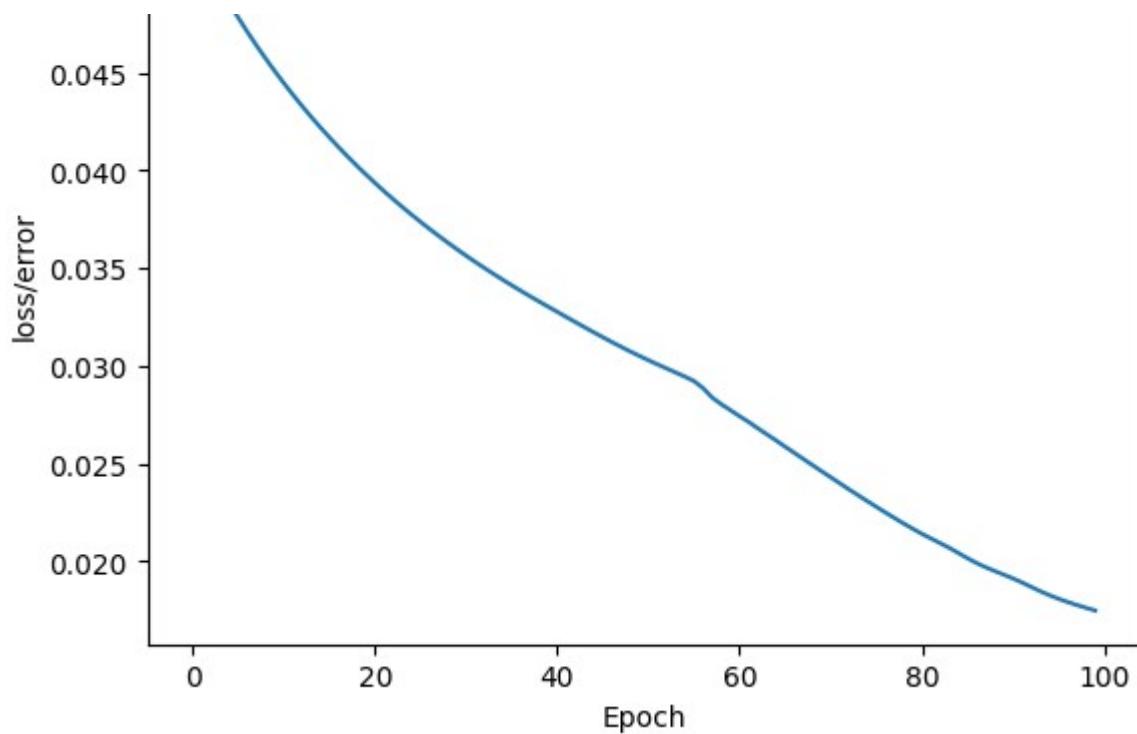
```
1 # Train our model!
2 # Epochs? (one run all the training data in our network)
3 epochs = 100 # How many times
4 losses = []
5 for i in range(epochs):
6   # Go forward and get a prediction
7   y_pred = model.forward(X_train) # Get predicted results
8
9   # Measure the loss/error, will be high at first
10  loss = criterion(y_pred, y_train) # Predicted values vs. the y_train values
11
12  # Keep track of the losses
13  losses.append(loss.detach().numpy())
14
15  # Print every 10 epochs
16  if i % 10 == 0:
17    print(f'Epoch: {i} and loss: {loss}')
18
19  # Do some back propagation: take the error rate of forward propagation
20  #   and feed it back through the network to fine tune the weights
21  optimizer.zero_grad()
22  loss.backward()
23  optimizer.step()
24
25
26
27
```

```
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: 0.02430540882050991
Epoch: 80 and loss: 0.021412163972854614
Epoch: 90 and loss: 0.019128229469060898
```

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

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





```

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

```

```

1 loss

```

```

    tensor(0.1784)

```

```

1 correct = 0
2
3 with torch.no_grad():
4     for i, data in enumerate(X_test):
5         y_val = model.forward(data)
6
7         if y_test[i] == 0:
8             x = 'Setosa'
9         elif y_test[i] == 1:
10            x = 'Versicolor'
11        else:
12            x = 'Virginica'
13
14        # Will print what type of flower class the network thinks it is
15        print(f'{i+1}. {str(y_val)} \t {y_test[i]}:{x} \t {y_val.argmax().item()}')
16
17        # Correct or not
18        if y_val.argmax().item() == y_test[i]:
19            correct += 1
20
21 print(f'We get {correct} correct!')
22

```

```

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!