Artificial Neural Data Iris Datasets

[568]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

[569]:
```
# create a model class that inherits nn.module
class Model(nn.Module): # A class(nn.Module) that inherits a subclass(Model)
    def __init__(self, in_features=4, h1=8, h2=9, out_features=3): # constructor␣
 ↪method of the model
        super().__init__() # constructor of the parent class
        self.fc1 = nn.Linear(in_features, h1) # instance of the nn.Linear class␣
 ↪(Fully Connected Linear Layer)
        self.fc2 = nn.Linear(h1, h2) # Another instance (Fully connected Linear␣
 ↪Layer)
        #self.fc3 = nn.Linear(h2, out_features) # Another instance (Fully␣
 ↪Connected Linear Layer)
        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
```

[570]:
```
# pick a randsom seed for randomization
torch.manual_seed(41)
# create an instance of model
model = Model()
```

[571]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

[572]:
```
url = 'https://gist.githubusercontent.com/netj/8836201/raw/
 ↪6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv'
my_df = pd.read_csv(url)
#from sklearn import datasets
#import pandas as pd
#iris = datasets.load_iris()
#my_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
```

[573]:
```
# Change Last Column From Strng to int
my_df['variety'] = my_df['variety'].replace('Setosa', 0.0)
my_df['variety'] = my_df['variety'].replace('Virginica', 1.0)
my_df['variety'] = my_df['variety'].replace('Versicolor', 2.0)
```

[574]:
```
# Train test and split
X = my_df.drop('variety', axis=1)
y = my_df['variety']
```

[575]:
```
# convert this into numpy arrays
X = X.values
y = y.values
```

[576]:
```
from sklearn.model_selection import train_test_split
```

[577]:
```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=26)
```

[578]:
```
import torch
```

[579]:
```
# convert X labels to FloatTensor
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
```

[580]:
```
# Convert y labels to tensor long
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
```

[581]:
```
import torch.nn as nn
```

[582]:
```
# Set the criterion of model to measure the error, how far off the predictions␣
↪are from:
criterion = nn.CrossEntropyLoss()
# choose the optimizer (Adam Optimizer), lr = learning rate (if error does not␣
↪go down after a bunch of iterations (epochs), lower our learning rate)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

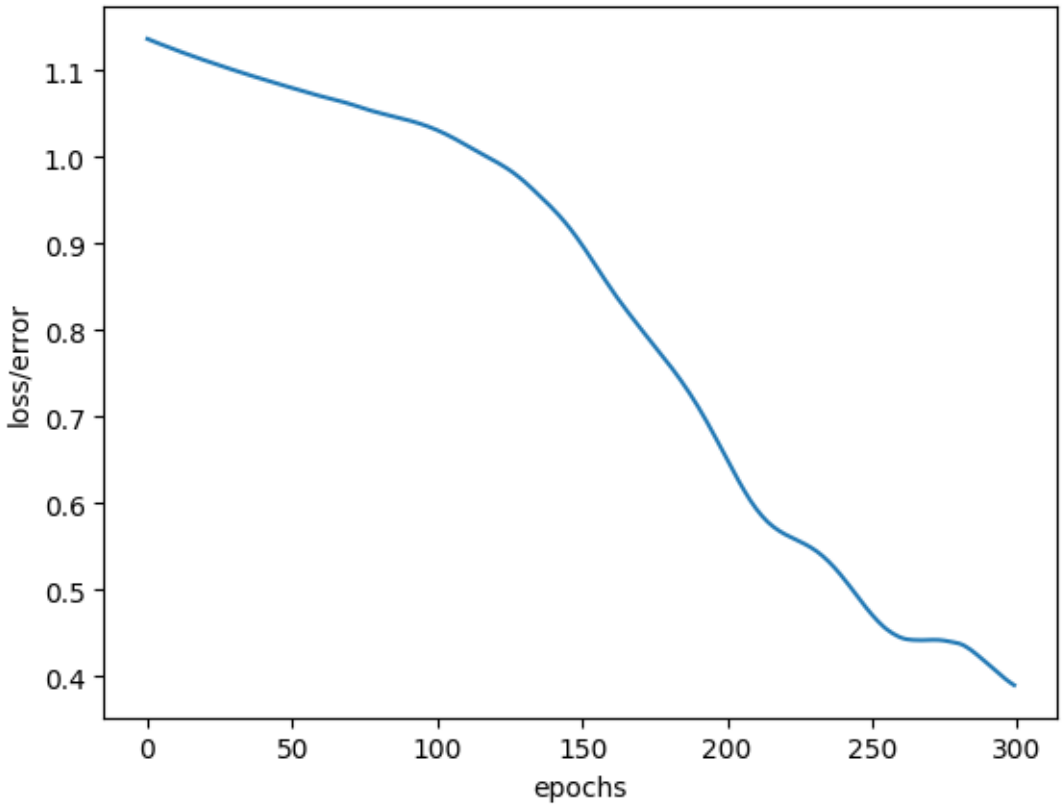
[583]:
```
# Train our Model
epoch = 300 # Epochs (one run through all the training data in our network)
losses = []
for i in range(epoch):
    y_pred = model.forward(X_train) # Go forward and get a prediction
    loss = criterion(y_pred, y_train) # Measure the error
    losses.append(loss.detach().numpy()) # keep track of our loss
    if i%10 == 0:
        print(f"Epoch: {i} and loss: {loss}")

    # Backpropagation: take the error rate data from the forward propagation and␣
↪feed it back through
    # the network to fine tune the weights
    optimizer.zero_grad
```

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

```
Epoch: 0 and loss: 1.1367099285125732
Epoch: 10 and loss: 1.1236135959625244
Epoch: 20 and loss: 1.1116485595703125
Epoch: 30 and loss: 1.1004196405410767
Epoch: 40 and loss: 1.0898969173431396
Epoch: 50 and loss: 1.0798994302749634
Epoch: 60 and loss: 1.0702873468399048
Epoch: 70 and loss: 1.0613794326782227
Epoch: 80 and loss: 1.0511637926101685
Epoch: 90 and loss: 1.0427014827728271
Epoch: 100 and loss: 1.0311806201934814
Epoch: 110 and loss: 1.0138578414916992
Epoch: 120 and loss: 0.995255172252655
Epoch: 130 and loss: 0.9719210267066956
Epoch: 140 and loss: 0.9399265050888062
Epoch: 150 and loss: 0.8983412981033325
Epoch: 160 and loss: 0.8476247191429138
Epoch: 170 and loss: 0.8026411533355713
Epoch: 180 and loss: 0.760327935218811
Epoch: 190 and loss: 0.7114966511726379
Epoch: 200 and loss: 0.6505463123321533
Epoch: 210 and loss: 0.593819260597229
Epoch: 220 and loss: 0.5642219185829163
Epoch: 230 and loss: 0.5466979146003723
Epoch: 240 and loss: 0.5140414834022522
Epoch: 250 and loss: 0.47119370102882385
Epoch: 260 and loss: 0.4448593556880951
Epoch: 270 and loss: 0.4422735571861267
Epoch: 280 and loss: 0.4381070137023926
Epoch: 290 and loss: 0.414358913898468
```

[584]:
```
plt.plot(range(epoch), losses)
plt.ylabel("loss/error")
plt.xlabel("epochs")
```

[584]:
```
Text(0.5, 0, 'epochs')
```



[585]:
```
# Evaluate our model on test data set
with torch.no_grad(): # Basically turn off back propagation
    y_eval = model.forward(X_test) # X-test are features are features from our␣
 ↪test set, y_eaval is
    loss = criterion(y_eval, y_test) # find the loss or error
```

[586]:
```
loss
```

[586]:
```
tensor(0.3521)
```

[587]:
```
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)} \t {y_test[i]} \t {y_val.argmax().item()}")
        # will tell us what type of flower class our network th

        # correct or not
        if y_val.argmax().item() == y_test[i]:
            correct+=1
print(f"we got {correct} correct")
```

```
1.) tensor([-0.8530,  1.1762,  1.3145])          2       2
2.) tensor([-0.6187,  0.9430,  1.1603])          2       2
3.) tensor([ 4.1957, -3.1212, -0.9854])          0       0
4.) tensor([ 4.2012, -3.1356, -1.0329])          0       0
5.) tensor([-2.4050,  2.4922,  1.9698])          1       1
6.) tensor([-0.1667,  0.5432,  0.9382])          2       2
7.) tensor([ 4.1948, -3.1198, -1.0054])          0       0
8.) tensor([-2.6978,  2.7833,  2.1595])          1       1
9.) tensor([-3.6842,  3.6236,  2.6223])          1       1
10.) tensor([-2.7580,  2.7909,  2.1188])         1       1
11.) tensor([ 3.8635, -2.8582, -0.8536])         0       0
12.) tensor([-0.4528,  0.8320,  1.1383])         2       2
13.) tensor([-0.1499,  0.5424,  0.9505])         2       2
14.) tensor([ 3.8615, -2.8622, -0.8984])         0       0
15.) tensor([0.0341, 0.3923, 0.8666])    2       2
16.) tensor([ 4.0664, -3.0358, -1.0105])         0       0
17.) tensor([-2.9237,  2.9182,  2.1943])         1       1
18.) tensor([0.3815, 0.0750, 0.6991])    2       2
19.) tensor([-2.6887,  2.7563,  2.1243])         1       1
20.) tensor([ 3.9949, -2.9696, -0.9463])         0       0
21.) tensor([ 4.4425, -3.3164, -1.0979])         0       0
22.) tensor([-3.4176,  3.3683,  2.4661])         1       1
23.) tensor([-2.4777,  2.5574,  2.0095])         1       1
24.) tensor([-3.2938,  3.2604,  2.3782])         1       1
25.) tensor([-3.4952,  3.4274,  2.4708])         1       1
26.) tensor([-2.0512,  2.1495,  1.7612])         1       1
27.) tensor([-0.3826,  0.7603,  1.0822])         2       2
28.) tensor([ 4.9836, -3.7627, -1.3628])         0       0
29.) tensor([ 3.9522, -2.9316, -0.9123])         0       0
30.) tensor([-2.8488,  2.8674,  2.1817])         1       1
we got 30 correct
```

[588]:
```python
# Find information from new data
new_iris = torch.tensor([4.7, 3.2, 1.3, 0.2])
```

[595]:
```python
with torch.no_grad():
    y_eval = model(new_iris)

    if y_test[i] == 0:
        x = "Setosa"
    elif y_test[i] == 1:
        x = "Versicolor"
    else:
        x = "Viginica"

    print(f'{str(y_val)} \t {x}')
```

```
tensor([-2.8488,  2.8674,  2.1817])      Versicolor
```

[590]:
```
newer_iris = torch.tensor([5.9, 3.0, 5.1, 1.8])
```

[597]:
```
with torch.no_grad():
    y_val = model(newer_iris)

    if y_test[i] == 0:
        x = "Setosa"
    elif y_test[i] == 1:
        x = "Versicolor"
    else:
        x = "Viginica"

    print(f'{str(y_val)} \t {x}')
```

```
tensor([-2.1585,  2.2683,  1.8464])      Versicolor
```