04_PyTorch-training -pipeline

June 30, 2025

1 Training Pipeline in PyTorch

2 Importing libraries

[1]:
```
import numpy as np
import pandas as pd
import torch
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler as ss, LabelEncoder as le
```

3 Importing Data

[2]:
```
df = pd.read_csv('https://raw.githubusercontent.com/gscdit/
↪Breast-Cancer-Detection/refs/heads/master/data.csv')
print(df.shape)
df.head()
```

```
(569, 33)
```

[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	…	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	…	17.33	184.60	2019.0	0.1622	
1	…	23.41	158.80	1956.0	0.1238	
2	…	25.53	152.50	1709.0	0.1444	

```
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

[3]:
```
# we have to remove id and unamed column
df = df.drop(['id', 'Unnamed: 32'], axis = 1)
```

[4]:
```
print(df.shape)
df.head()
```

```
(569, 31)
```

[4]:
```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  …         25.38          17.33           184.60
1         0.1812  …         24.99          23.41           158.80
2         0.2069  …         23.57          25.53           152.50
3         0.2597  …         14.91          26.50            98.87
4         0.1809  …         22.54          16.67           152.20
```

```
   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 31 columns]
```

4 Split data into train and test

[5]:
```
# lets split the data into train and test
X = df.drop(['diagnosis'], axis = 1)
y = df['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 0)
```

5 Scaling

[6]:
```
# lets scale the data
ss = ss()
X_train = ss.fit_transform(X_train)
X_test  = ss.fit_transform(X_test)
```

[7]:
```
X_train
```

[7]:
```
array([[-1.15036482, -0.39064196, -1.12855021, …, -0.75798367,
        -0.01614761, -0.38503402],
       [-0.93798972,  0.68051405, -0.94820146, …, -0.60687023,
         0.09669004, -0.38615797],
       [ 0.574121  , -1.03333557,  0.51394098, …, -0.02371948,
        -0.20050207, -0.75144254],
       …,
       [-1.32422924, -0.20048168, -1.31754581, …, -0.97974953,
        -0.71542314, -0.11978123],
       [-1.24380987, -0.2245526 , -1.28007609, …, -1.75401433,
        -1.58157125, -1.00601779],
       [-0.73694129,  1.14989702, -0.71226578, …, -0.27460457,
```

```
        -1.25895095,  0.21515662]])
```

[8]:
```
y_train
```

[8]:
```
338    B
427    B
406    B
96     B
490    B
      ..
277    M
9      M
359    B
192    B
559    B
Name: diagnosis, Length: 455, dtype: object
```

6 Encoding

[9]:
```
# lets encode y
le = le()
y_train = le.fit_transform(y_train)
y_test  = le.fit_transform(y_test)
```

[10]:
```
y_train
```

[10]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
```

```
1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0])
```

7 Numpy arrays to PyTorch tensors

[11]:
```
X_train_tensor = torch.from_numpy(X_train)
y_train_tensor = torch.from_numpy(y_train)
X_test_tensor  = torch.from_numpy(X_test)
y_test_tensor  = torch.from_numpy(y_test)
```

[12]:
```
X_train_tensor.shape
```

[12]:
```
torch.Size([455, 30])
```

[13]:
```
y_train_tensor.shape
```

[13]:
```
torch.Size([455])
```

8 Definning the model

[14]:
```
# lets build our first NN-model using PyTorch
class MySimpleNN():
    def __init__(self, X):

        self.weights = torch.rand(X.shape[1],  1, dtype = torch.float64,␣
 ↪requires_grad=True)
        self.bias = torch.zeros(1, dtype=torch.float64, requires_grad=True)

    def forward(self, X):
        z = torch.matmul(X, self.weights) + self.bias
        y_pred = torch.sigmoid(z)
        return y_pred

    def loss_function(self, y_pred, y):
        # Clamp predictions to avoid log(0)
        epsilon = 1e-7
        y_pred = torch.clamp(y_pred, epsilon, 1 - epsilon)

        # Calculate loss
        loss = -(y_train_tensor * torch.log(y_pred) + (1 - y_train_tensor) *␣
 ↪torch.log(1 - y_pred)).mean()
        return loss
```

8.1 Important Parameters

[15]:
```
learning_rate = 0.1
epochs = 25
```

8.2 Training Pipeline

[16]:
```
# Create mdoel
model = MySimpleNN(X_train_tensor)

# define loop
for epoch in range(epochs):
    # forward pass
    y_pred = model.forward(X_train_tensor)
    # loss calculate
    loss = model.loss_function(y_pred, y_train_tensor)
    # backward pass
    loss.backward()
    # parameters update
    with torch.no_grad():
        model.weights -= learning_rate * model.weights.grad
        model.bias -= learning_rate * model.bias.grad
    # zero the gradients
    model.weights.grad.zero_()
    model.bias.grad.zero_()

    # print loss
    print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}')
```

```
Epoch 1/25, Loss: 2.981145427891622
Epoch 2/25, Loss: 2.8457649043606423
Epoch 3/25, Loss: 2.706927470598495
Epoch 4/25, Loss: 2.5682703320135
Epoch 5/25, Loss: 2.428032484756076
Epoch 6/25, Loss: 2.2844306557257084
Epoch 7/25, Loss: 2.144872388455418
Epoch 8/25, Loss: 2.010004067330586
Epoch 9/25, Loss: 1.8808960362284857
Epoch 10/25, Loss: 1.758196296407852
Epoch 11/25, Loss: 1.642110511076718
Epoch 12/25, Loss: 1.530099369313076
Epoch 13/25, Loss: 1.428184553972619
Epoch 14/25, Loss: 1.3359190072557725
Epoch 15/25, Loss: 1.255372835750116
Epoch 16/25, Loss: 1.185873345657141
Epoch 17/25, Loss: 1.1263143375575786
Epoch 18/25, Loss: 1.075403172532063
Epoch 19/25, Loss: 1.0318348920335714
```

```
Epoch 20/25, Loss: 0.9943987278199385
Epoch 21/25, Loss: 0.962036359831233
Epoch 22/25, Loss: 0.9338599857616446
Epoch 23/25, Loss: 0.9091433766024155
Epoch 24/25, Loss: 0.8873009891429741
Epoch 25/25, Loss: 0.8678653993505461
```

[17]:
```
model.bias
```

[17]:
```
tensor([-0.2172], dtype=torch.float64, requires_grad=True)
```

9 Model Evaluation

[32]:
```
# model evaluation
with torch.no_grad():
    y_pred = model.forward(X_test_tensor)
    y_pred = (y_pred > 0.5).float()
    accuracy = (y_pred == y_test_tensor).float().mean()
    print(f'Accuracy: {accuracy.item()}')
```

```
Accuracy: 0.5184672474861145
```