# Plan of Attack

02 December 2024 18:41

Lec = 4

3 videos → [ Intro, Tensors, Computation + Autograd ]

1. We will build a simple neural network ✓
2. Train it on a real world dataset ✓
3. Will mimic the PyTorch workflow ✓
4. Will have a lot of manual elements ✓
5. End result is not important ✓

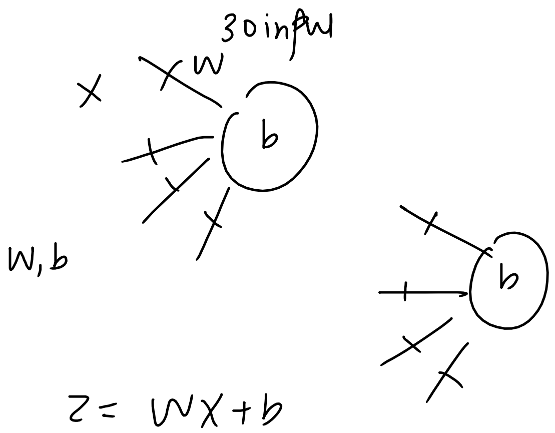
Breast cancer

$$\begin{tikzcd}
\text{Dataset} \arrow[r] & \text{NN} \arrow[d] & \\
& \underline{\text{training pipeline}} \arrow[r] & \text{foundation}
\end{tikzcd}$$

# Code flow

02 December 2024 18:41

1. Load the dataset ✓
2. Basic preprocessing ✓
3. Training Process
   a. Create the model ✓
   b. Forward pass ✓
   c. Loss calculation ✓
   d. Backprop ✓
   e. Parameters update ✓
4. Model evaluation → accuracy



$z = wx + b$

$\sigma(z)$

$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b}$

$W_{new} = w_{old} - lr \frac{\partial L}{\partial w}$

PyTorch-training-pipeline-code-explanation

### Pura Process in 5 Steps:

1. **Data Ki Tayyari (Data Preparation)**: Kabaadi se sona nikalna.
2. **NumPy se PyTorch me Badalna**: Kaam karne ke liye sahi auzaar (tools) uthana.
3. **Model ka Blueprint Banana (Model Definition)**: Machine ka design taiyar karna.
4. **Model ko Train Karna (Training Pipeline)**: Machine ko kaam sikhana.
5. **Model ko Test Karna (Evaluation)**: Yeh dekhna ki machine ne kitna a_cha seekha.

---

### Step 1: Data Ki Tayyari (Data Preparation)

Kisi bhi AI model ko banane se pehle, humein data ko saaf-suthra aur istemal karne layak banana padta hai. Ise "Data Preprocessing" kehte hain.

#### a) Data Load karna aur Faltu Cheezein Hatana

codePython

```
import pandas as pd

# Data ko internet se utha kar ek table (DataFrame) me daalna
df = pd.read_csv('https://raw.githubusercontent.com/gscdit/Breast-Cancer-De

# Faltu columns ko hatana jinki model ko zaroorat nahi
df.drop(columns=['id','Unnamed: 32'], inplace= True)
```

- **Kyun kiya?**: Hamare data me id column har patient ka unique number hai, isse model kuch seekh nahi sakta. Unnamed: 32 ek extra, khaali column hai. Yeh dono model ke liye "kachra" hain, isliye humne drop() function se inko hata diya.

#### b) Data ko Training aur Testing me Baantna

codePython

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

```
# Data ko X (features) aur y (target) me alag karna
X = df.iloc[:, 1:] # Saari rows, aur column no. 1 se aakhir tak
y = df.iloc[:, 0]  # Saari rows, aur sirf column no. 0 ('diagnosis')

# Data ko 80% training aur 20% testing hisso me baantna
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

- **Kyun kiya?**: Yeh sabse zaroori steps me se ek hai. Hum model ko 80% data par train karte hain (sikhate hain) aur bache hue 20% data par test karte hain. Isse humein yeh pata chalta hai ki hamara model naye, anjaan data par kaisa perform karega.
- Analogy: Jaise aap exam ki taiyari sample papers (training data) se karte hain, aur final exam ek naye, anjaan paper (testing data) par dete hain.

### c) Data ko Scale Karna (Feature Scaling)

codePython

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Scaler ko training data se 'seekhne' ko kehna aur data ko transform karna
x_train = scaler.fit_transform(x_train)

# Test data par WAHI 'seekhi hui' scaling apply karna
x_test = scaler.transform(x_test)
```

- **Kyun kiya?**: Hamare data me alag-alag features ki values bahut alag-alag range me hain. Jaise area_mean ki value 1000 me ho sakti hai, aur smoothness_mean ki value 0.1 me. Isse model badi value wale feature ko zyada importance de sakta hai. StandardScaler sabhi features ko ek jaisi scale (mean=0, std=1) par le aata hai, taaki sabko barabar ka mauka mile.
- **Important**: Hum fit_transform sirf training data par karte hain, aur usi “fit” (seekhe hue mean/std) ko transform ke zariye test data par apply karte hain. Aisa isliye taaki hamara model test data ke baare me pehle se kuch na jaan le (ise data leakage kehte hain).

### d) Labels ko Numbers me Badalna (Label Encoding)

codePython

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

# 'M' (Malignant) aur 'B' (Benign) ko 1 aur 0 me badalna
y_train = encoder.fit_transform(y_train)
y_test = encoder.transform(y_test)
```

- **Kyun kiya?**: Machine Learning models “M” ya “B” jaise text ko nahi samajhte. Woh sirf numbers samajhte hain. LabelEncoder is text ko numbers me badal deta hai, jaise M=1 aur B=0.

---

## Step 2: NumPy se PyTorch Tensor me Badalna

Ab tak hamara saara data NumPy arrays me hai. PyTorch me kaam karne ke liye, humein ise PyTorch ki special data type, yaani **Tensor**, me badalna padega.

codePython

```
import torch

# NumPy arrays ko PyTorch Tensors me convert karna
x_train_tensor = torch.from_numpy(x_train)
x_test_tensor = torch.from_numpy(x_test)
y_train_tensor = torch.from_numpy(y_train)
y_test_tensor = torch.from_numpy(y_test)
```

- **Kyun kiya?**: PyTorch ke saare powerful features jaise Automatic Differentiation (Autograd) aur GPU acceleration sirf Tensors par kaam karte hain. Isliye yeh conversion zaroori hai.

---

## Step 3: Model ka Blueprint Banana (Model Definition)

Ab hum apni machine (Neural Network) ka design banayenge. Humne yeh kaam ek Python class ke andar kiya hai.

codePython

```
class MySimpleNN():
    def __init__(self, x):
        # Model ke parameters (jinhe model seekhega)
        self.weights = torch.rand(x.shape[1], 1, dtype=torch.float64, requi
        self.bias = torch.zeros(1, dtype=torch.float64, requires_grad=True)

    def forward(self, x):
        # Step 1: Input ko weights se multiply karke bias add karna
        z = torch.matmul(x, self.weights) + self.bias
        # Step 2: Result ko 0 aur 1 ke beech ki probability me badalna
        y_pred = torch.sigmoid(z)
        return y_pred

    def loss_function(self, y_pred, y_true):
        # Yeh naapna ki model ki prediction kitni galat hai
        epsilon = 1e-7
        y_pred = torch.clamp(y_pred, epsilon, 1 - epsilon)
        loss = -(y_true * torch.log(y_pred) + (1 - y_true) * torch.log(1 -
        return loss
```

- **init**: Yeh function model ke “learnable parameters” banata hai - weights aur bias.
  - weights: Har feature (hamare case me 30 features) ke liye ek weight. Yeh batata hai ki kaun sa feature kitna important hai.
  - bias: Ek extra parameter jo model ko thoda flexible banata hai.
  - requires_grad=True: Yeh PyTorch ke Autograd engine ko batata hai ki, “Hey! In tensors ka dhyaan rakhna, mujhe inke respect me aage chalkar derivatives (gradients) nikalne hain.”
- **forward**: Yeh function “Forward Pass” ko define karta hai. Yeh batata hai ki jab model ko input (x) milega, to woh prediction (y_pred) kaise banayega. Yeh logistic regression ka formula hai.
- **loss_function**: Yeh function model ki galti ko naapta hai. Hamara goal is “loss” ko kam se kam karna hai. Humne yahan **Binary Cross-Entropy Loss** use kiya hai, jo binary classification problems ke liye standard hai.

## Step 4: Model ko Train Karna (The Training Pipeline)

Yeh woh hissa hai jahan model asal me seekhta hai. Yeh ek loop me chalta hai. Har loop ko ek **epoch** kehte hain.

codePython

```
learning_rate = 0.1
epochs = 25
model = MySimpleNN(x_train_tensor)

for epoch in range(epochs):
    # 1. FORWARD PASS: Model se predictions lena
    y_pred = model.forward(x_train_tensor)

    # 2. LOSS CALCULATE KARNA: Galti ko naapna
    loss = model.loss_function(y_pred, y_train_tensor)
    print(f'Epoch: {epoch+1}, Loss: {loss.item()}')

    # 3. BACKWARD PASS: Galti ke liye kisko kitna blame karein?
    loss.backward()

    # 4. PARAMETERS UPDATE KARNA: Galtiyon se seekhna
    with torch.no_grad():
        model.weights -= learning_rate * model.weights.grad
        model.bias -= learning_rate * model.bias.grad

    # 5. GRADIENTS KO ZERO KARNA: Agle round ke liye slate saaf karna
    model.weights.grad.zero_()
    model.bias.grad.zero_()
```

1. **Forward Pass**: Hum training data ko model me daalte hain aur dekhte hain ki woh kya predict karta hai.

2. **Calculate Loss**: Hum us prediction ko asli answer se compare karke loss nikalte hain.

3. **Backward Pass (loss.backward())**: Yahan PyTorch ka jaadu (Autograd) chalta hai. Yeh automatically calculate karta hai ki loss ko kam karne ke liye har weight aur bias ko kis direction me (badhana hai ya ghatana hai) aur kitna change karna chahiye. Yeh information .grad attribute me store ho jaati hai.

4. **Update Parameters**: Hum weights aur bias ko unke gradient ke opposite direction me thoda sa move karte hain. learning_rate control karta hai ki yeh "thoda sa" kitna hoga. Yeh step with torch.no_grad(): ke andar hai kyunki hum nahi chahte ki PyTorch is update ke action ko track kare.

5. **Zero Gradients**: Hum gradients ko zero kar dete hain taaki agle epoch ki calculation fresh shuru ho, purane gradients usme na judein.

---

## Step 5: Model ko Test Karna (Evaluation)

Training ke baad, ab final exam ka time hai. Hum test data (jo model ne pehle kabhi nahi dekha) par model ki performance check karenge.

codePython

```python
with torch.no_grad():
    # Test data par predictions nikalna
    y_pred = model.forward(x_test_tensor)

    # Probabilities ko final decision (0 ya 1) me badalna
    y_pred = (y_pred > 0.5).float()

    # Accuracy calculate karna
    accuracy = (y_pred.reshape(-1) == y_test_tensor).float().mean()
    print(f'Accuracy: {accuracy.item()}')
```

- **with torch.no_grad()**: Hum PyTorch ko batate hain ki yahan gradients calculate karne ki zaroorat nahi hai, kyunki hum sirf test kar rahe hain, seekh nahi rahe. Isse calculation fast ho jaati hai.

- **(y_pred > 0.5).float()**: Model 0 se 1 ke beech ki probability deta hai. Hum ek rule banate hain: agar probability 0.5 se zyada hai to use 1 (Malignant) maano, warna 0 (Benign). .float() True/False ko 1.0/0.0 me badal deta hai.

- **Accuracy**: Hum check karte hain ki model ki predictions kitni baar asli answers se match karti hain. Iska average nikalne se humein accuracy mil jaati hai.