# Summer of Code' 25

Sreshtha Lahiri

Midterm Checkpoint -96 Intro into Deep Learning

## 1 Index- (This document contains my progress so far and an overview of what I have learned)

## 2 Python and Its Libraries

Now since we have to code the instructions to "train" the machine we can use several programming languages available on the net. The most widely used and at the same time easy to learn programming language is python.

Python is a programming language that is commonly used in data science, machine learning, and artificial intelligence. Its simplicity and easiness to handle make it a favorite among beginners and professionals alike.

To handle specific tasks like data analysis, visualization, and machine learning, Python offers a variety of libraries. Following are some of the most popular ones and a brief overview of each of them :

## Numpy

- It is short for "Numerical Pythom."It's used for working with large, multi-dimensional arrays and matrices of numerical data. We can think of it as a supercharged calculator for scientific computations. We can use it for performing mathematical operations like addition , multiplication or finding mean on large datasets
  - For example: a = np.array([1, 2, 3])
  - This code is used to define an array , where numpy is used as np

## Pandas

- It can be defined as a library for data manipulation and analysis.It makes it easy to work with data stored in tables (called DataFrames), allowing you to clean, organize, and analyze it.This can be done using several resources , one such open source platform is jupyter notebook.
  - For example pd.read()
  - This code is used to read files like csv files, and pandas is imported as pd

## Matplotlib

- Matplotlib is a library for creating static, interactive, and dynamic representation of data .It's great for plotting graphs like bar charts, histograms, and scatter plots.
  - For example: Visualizing how sales change over time using a line graph.This library has a variety of other functions

  .

## Seaborn

- Built on as an extension of Matplotlib, it's used for statistical data visualization. It provides attractive graphs like heatmaps, pair plots, and violin plots. For example: Comparing relationships between multiple variables in a dataset

## Scikit-learn

- This library makes it simple to implement algorithms like regression, classification, clustering, and more. It includes tools for splitting data into training and testing sets, evaluating models, and pre-processing data (all of this we will understand further). Example: Building a model to predict house prices based on features like size and location.

# 3 Gradient Descent

Gradient descent is a very common or in better words a generic optimization algorithm used in supervised machine learning . It is used to minimize a loss function ( a cost function , that we will study about in a bit ) . Imagine hiking in the mountains on a foggy day, and you don't know where your destination is. You can't see where to go, but you can feel the slope of the ground under your feet. A logical strategy to find the bottom of the valley is to move downhill in the steepest direction. This example might be rudimentary but this is exactly how Gradient Descent works in machine learning.

## Thought Process to easily understand this

- Begin with randomly assigned values for the parameters.
- Gradually update the parameters by moving in the direction of the steepest descent of the error function.
- The process continues until the slope becomes zero, indicating a minimum in the error function.

Now the steps taken in gradient descent depends on the learning rate . This learning rate on becoming too low can make the process slow or if its too large then, it can make the process too erratic , we may never reach the conclusion. Hence, an optimum learning rate is taken.

## Types of Gradient Descent

- Stochastic gradient descent

  This computes the gradient using one random data point at a time. This too produces faster results but the high variance produces here can lead to instability in predictions.

- Batch gradient descent

  This process uses the entire dataset to produce accurate calculations but may become slow for large datasets.

- Mini-batch gradient descent

This type of gradient descent as the name suggests uses mini sets of data points.But this process requires fine-tuning or scaling the batch size.

## 4 Linear Regression

Linear Regression is a supervised learning algorithm. It uses a linear equation that uses weights and biases. The weights decides the basically the importance of each input into the data and then the bias acts as an error term. the equation is as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon$$

$$\beta_p$$

= weights calculated in training

$$x_p$$

= inputs / features ; y = output.

$$epsilon$$

= error This line must fit the distribution of the dataset the best . Meaning : most efficiently and with least cost funtion and mean error.There are several types of losses :

- L1
- L2
- Mean Absolute Error
- Mean Squared Error

**Code implementation of linear regression using pandas library:**

```python
import pandas as pd
from sklearn.model_selection import
    train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
    r2_score

# Creating a sample DataFrame here
data = {
    'Feature1': [1, 2, 3, 4, 5],
    'Feature2': [2, 4, 6, 8, 10],
    'Target': [2.2, 2.8, 4.5, 3.7, 5.5]
}
df = pd.DataFrame(data)

# Splitting features and target for model training
    and then testing
X = df[['Feature1', 'Feature2']]  # Predictors (
    independent variables)
y = df['Target']  # Target (dependent variable)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size=0.2, random_state=42)

# Initialize and train the model - this is where we
     specify algorithm
model = LinearRegression()
model.fit(X_train, y_train)

# Extract coefficients and intercept
bias = model.intercept_
weights = model.coef_

print(f"Bias (Intercept): {bias}")
print(f"Weights (Coefficients): {weights}")

# Make predictions here
y_pred = model.predict(X_test)

# Evaluate the model , we can also find out the
    accuracy etc.
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
```

**Output block:**

```
Bias (Intercept): 0.2999999999999998
Weights (Coefficients): [0.1 0.2]
Mean Squared Error: 0.04000000000000004
R-squared: 0.96
```

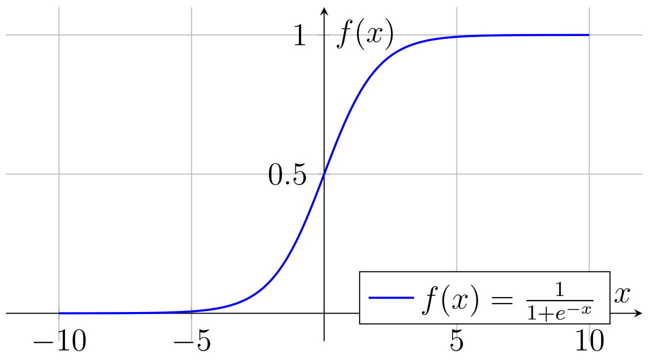
### 4.1 Hyperparameters

- Learning Rate: determines the step size with which the model updates its weights.
- Batch Size:the number of samples processed before the model uodates the weights.
- Epochs: one complete pass through the training dataset.

## 5 Logistic Regression

It is a supervised learning algorithm used to make binary (for example : true or false) or multilevel ( based on on several factors and into several categories ) classification.For example: it can find out the probability of an email being a spam or not and then classify it into a subsequent folder . The logistic formula is the sigmoid function.

Graph of the Sigmoid Function



# 6 Introduction into Deep Learning

## 6.1 Neural Network

This network works like the human brain. It consists of several interconnected nodes(layers).It consists of :

- Input layer
- Hidden layers that influences the process of reaching the output.
- Output layer.

**Example of neural network**

```python
# Import necessary libraries , I sometimes
    forget to include specific sets of a given
    library
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import
   train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the Iris dataset , its a very commonly
    used one , this, mnist etc.
iris = load_iris()
X = iris.data  # Features here
y = iris.target.reshape(-1, 1)  # Labels

# One-hot encode the labels - gpt suggested
    this for error , I'm not above asking for
    help when necessary
encoder = OneHotEncoder(sparse=False)
y_encoded = encoder.fit_transform(y)

# Split the dataset into training and testing
    sets
X_train, X_test, y_train, y_test =
   train_test_split(X, y_encoded, test_size
   =0.2, random_state=42)

# Build the neural network model , here the
    algorithm as we saw earlier
model = Sequential([
    Dense(10, input_dim=X_train.shape[1],
       activation='relu'),  # Input layer +
       Hidden layer
    Dense(10, activation='relu'),  # Another
       hidden layer
    Dense(y_train.shape[1], activation='softmax
       ')  # Output layer
])
# Compile the model
model.compile(optimizer='adam', loss='
   categorical_crossentropy', metrics=['
   accuracy'])
model.fit(X_train, y_train, epochs=50,
   batch_size=5, verbose=1)
loss, accuracy = model.evaluate(X_test, y_test,
```

## 6.2 Forward Propagation

Here the input layer takes the features and values of inputs. The hidden layers then calculate weighted sum of these inputs with a bias.

This results is then passed through an activation function for example ReLU or the sigmoid function.This is done to introduce non-linearity and then finally the output layes shows the final result.

## 6.3 Backward Propagation

First we calculate the difference between predicted output and actual output using a loss function .

Then we compute the gradient of the loss function with respect to weights and biases using the chain rule.

After that the weights and biases are adjusted in the direction of decreasing loss using an optimization algorithm like Gradient Descent

**example using epochs**

```python
import numpy as np

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input features
y = np.array([[0], [1], [1], [0]])  # XOR outputs

# Initialize weights and biases
np.random.seed(42)  # For reproducibility
input_size = 2
hidden_size = 2
output_size = 1

weights_input_hidden = np.random.rand(input_size, hidden_size)
weights_hidden_output = np.random.rand(hidden_size, output_size)

bias_hidden = np.random.rand(1, hidden_size)
bias_output = np.random.rand(1, output_size)

# Learning rate
learning_rate = 0.1

# Training the network
epochs = 10000
for epoch in range(epochs):
    # Forward propagation is being used here
    hidden_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)

    # Calculate error
    error = y - final_output
```

```python
# Backpropagation
    d_output = error * sigmoid_derivative(
       final_output)
    error_hidden_layer = d_output.dot(
       weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer *
       sigmoid_derivative(hidden_output)

    # Update weights and biases
    weights_hidden_output += hidden_output.T.dot(
       d_output) * learning_rate
    bias_output += np.sum(d_output, axis=0,
       keepdims=True) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer)
        * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0,
       keepdims=True) * learning_rate

    # Print loss every 1000 epochs
    if epoch % 1000 == 0:
        loss = np.mean(np.square(error))
        print(f"Epoch {epoch}, Loss: {loss}")

# Testing the network  not accuracy here tho
print("Final outputs after training:")
print(final_output)
```

## 6.4 Activation Layers

This consists of the activation function we mentioned earlier on

**An example of usage of activation layer**

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Defining a simple neural network model over here
model = Sequential([
    # Input layer with ReLU activation
    Dense(64, input_dim=10, activation='relu'),

    # Hidden layer with Sigmoid activation
    Dense(32, activation='sigmoid'),

    # Output layer with Softmax activation for
       multi-class classification
    Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='
   categorical_crossentropy', metrics=['accuracy'])

# Display the model
model.summary()
```

## 6.5 Overfitting and Underfitting

Overfitting data means that the model is too precise and that this trained model would not be suitable for any new data testing.

Underfitting means that the model is "too dumb". And that it has not been trained weel and would require more and more fine tuning.

The bias and weights are toggled to reach the maximum efficiency for both types of models and a varinace tradeoff is done.

## 6.6 Regularization in Deep Learning

here comes the functions, L1 , L2 that we talked about earlier.The regularization is a set of techniques that is implemented to avoid overfitting of datasets. Overfitting occurs when the models learns noise and unnecessary data more than underlying patterns and regularisation helps in preventing this from happening.

- L1 : adds absolute value of weights to loss
- L2 : adds square of weights to loss

## 6.7 Dropouts in Dep Learning

As the name suggests , this process drops a bunch of random neurons or input values form the training dataset. This is done so that the model does not become overly reliant on the dataset.

**Example of using Dropout**

```python
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Generating a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing our data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the model with Dropout ( here is where we first includeit)
model = Sequential([
    Dense(128, input_dim=20, activation='relu'),
    Dropout(0.5),  # Dropout applied here
    Dense(64, activation='relu'),
    Dropout(0.5),  # Dropout applied again
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

## 6.8 Optimizers

Optimizers are algorithms or methods used to update the weights and biases of a neural network during training, based on the computed gradients from backpropagation. They play a crucial role in minimizing the loss function and improving model performance. a few examples :

- Gradient Descent
- Momentum
- RMSProp (Root Mean Square Propagation)
- Adam (Adaptive Moment Estimation)
- Adagrad (Adaptive Gradient Algorithm

## 6.9 Multiclass Classification

Multiclass classification is the same as binary classification except that instead of 2 categories there are multiple.

```python
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Generating a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=3, n_informative=15, random_state=42)
y = to_categorical(y)  # Convert labels to one-hot encoding

# Split data for testing and training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the model
model = Sequential([
    Dense(64, input_dim=X.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(y.shape[1], activation='softmax')  # Softmax for multiclass classification
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

# 7 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a special type of neural network specifically made for processing structured data like images. It is really effective in capturing spatial and temporal dependencies in data, making it the backbone of modern computer vision systems. key components:

- convolution layers do operations on the input data by passing them through filters called kernels .The kernels extract features and the output of this layer is called a feature map.
- Pooling layer to decrease the spatial dimension of data and control overfitting.
- flatten layer converts 2D feature maps into 1D
- Activation functions and then dropout

Architecture of CNN layer :

- input layer
- convolutional layer
- pooling layer
- fully connected layers
- output layer

**CNN Implementation**

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess data here
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Building the CNN model here
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=64)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```