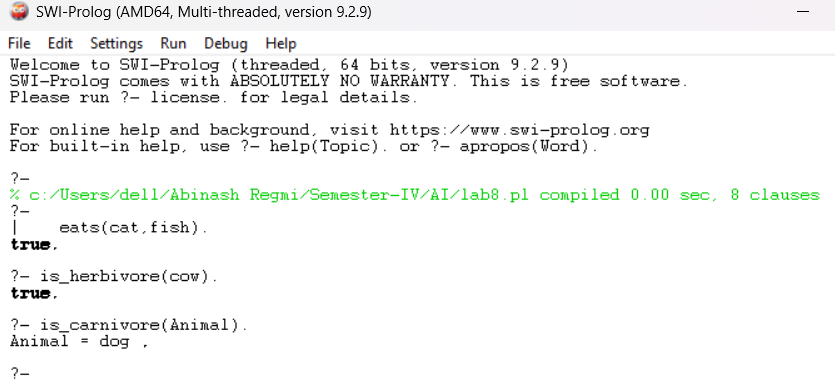
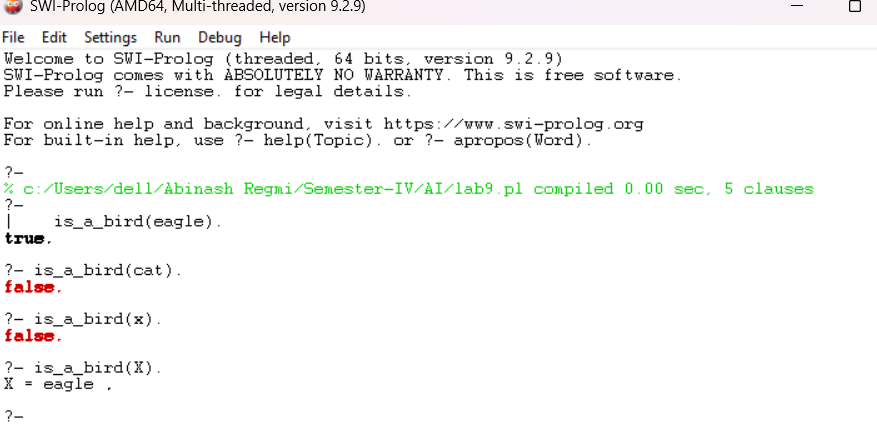
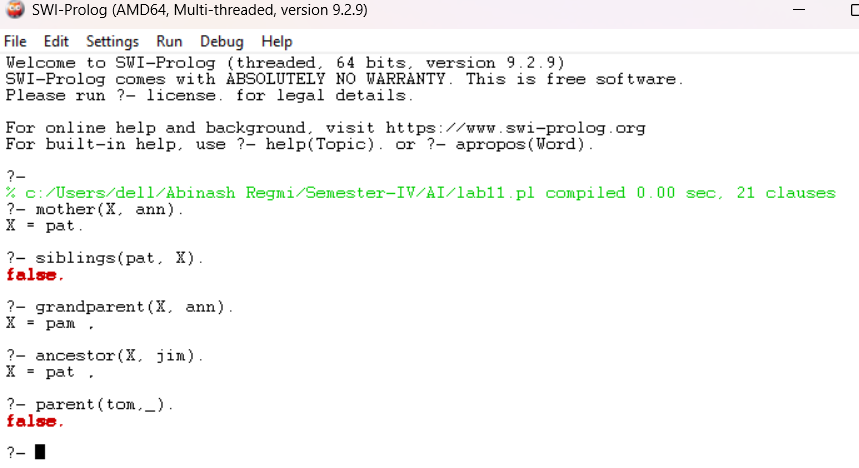
**OUTPUT:**

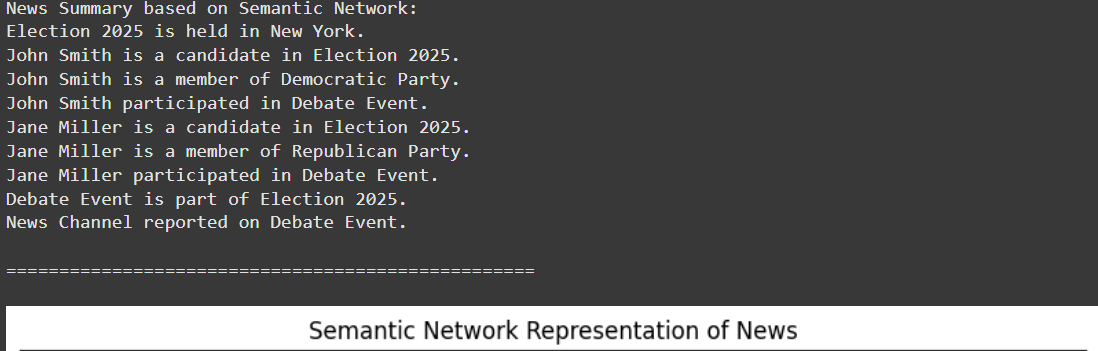
****

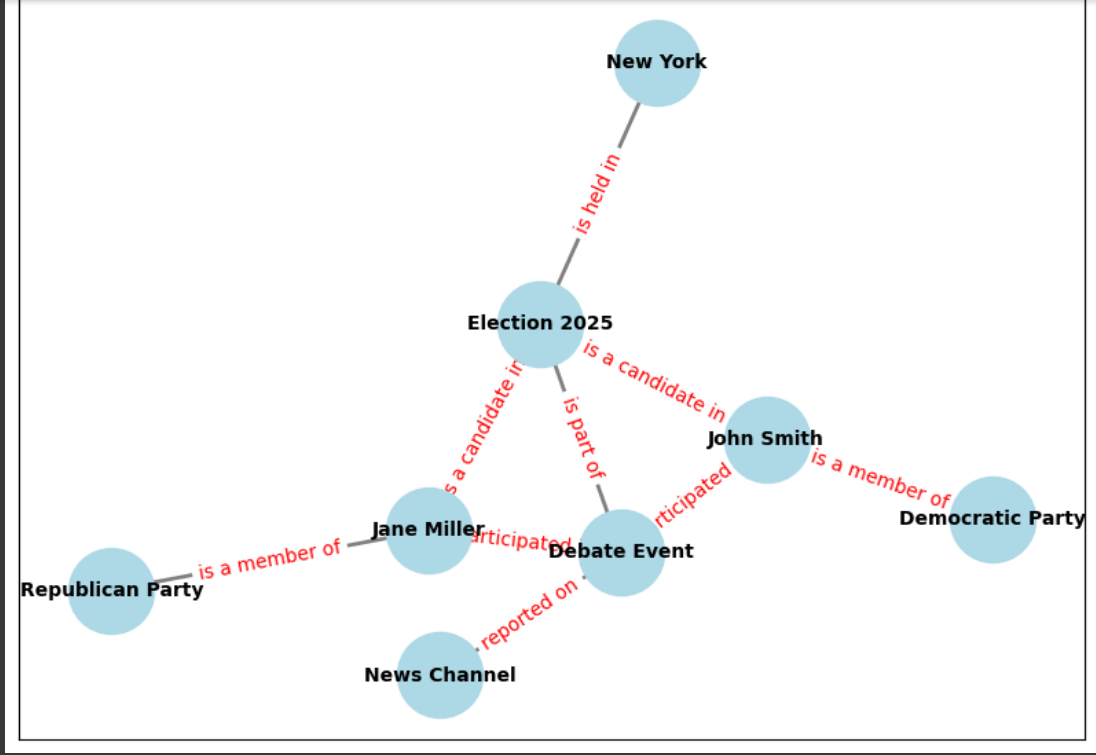
**OUTPUT:**

****

****

**OUTPUT:**

**OUTPUT:**



**SOURCE CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv('housing\_prices.csv')

X = df[['area\_sqft', 'num\_rooms']].values

y = df['price'].values

X = (X - X.mean(axis=0)) / X.std(axis=0)

y\_min, y\_max = y.min(), y.max()

y\_norm = (y - y\_min) / (y\_max - y\_min)

def sigmoid(z):

return 1 / (1 + np.exp(-z))

weights = np.random.randn(2)

bias = 0

lr = 0.1

epochs = 1000

for epoch in range(epochs):

z = np.dot(X, weights) + bias

y\_pred = sigmoid(z)

mse\_loss = np.mean((y\_pred - y\_norm) \*\* 2)

if((epoch+1)%50==0):

print(f"Epoch:{epoch+1}, loss= {mse\_loss}")

dE\_dy = (y\_pred - y\_norm)

dy\_dz = sigmoid(z) \* (1 - sigmoid(z))

dz\_dw = X

dz\_db = 1

grad\_w = np.dot(dz\_dw.T, dE\_dy \* dy\_dz) / len(X)

grad\_b = np.sum(dE\_dy \* dy\_dz \* dz\_db) / len(X)

weights -= lr \* grad\_w

bias -= lr \* grad\_b

weights

bias

test\_area = 2135

test\_rooms = 4

test\_X = np.array([test\_area, test\_rooms])

test\_X\_norm = (test\_X - df[['area\_sqft', 'num\_rooms']].mean().values) / df[['area\_sqft', 'num\_rooms']].std().values

z\_test = np.dot(test\_X\_norm, weights) + bias

pred\_norm = sigmoid(z\_test)

pred\_price = pred\_norm \* (y\_max - y\_min) + y\_min

print(f"Predicted Price for area={test\_area} sq ft and rooms={test\_rooms}: {pred\_price:.2f}")

import pandas as pd

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

import matplotlib.pyplot as plt

df = pd.read\_csv('housing\_prices.csv')

X = df[['area\_sqft', 'num\_rooms']].values

y = df['price'].values

X\_mean = X.mean(axis=0)

X\_std = X.std(axis=0)

X\_norm = (X - X\_mean) / X\_std

y\_min, y\_max = y.min(), y.max()

y\_norm = (y - y\_min) / (y\_max - y\_min)

X\_tensor = torch.tensor(X\_norm, dtype=torch.float32)

y\_tensor = torch.tensor(y\_norm.reshape(-1, 1), dtype=torch.float32)

class SingleLayerModel(nn.Module):

def \_\_init\_\_(self):

super(SingleLayerModel, self).\_\_init\_\_()

self.linear = nn.Linear(2, 1)

def forward(self, x):

z = self.linear(x)

return torch.sigmoid(z)

model = SingleLayerModel()

criterion = nn.MSELoss()

optimizer = optim.SGD(model.parameters(), lr=0.01)

epochs = 10000

for epoch in range(epochs):

y\_pred = model(X\_tensor)

loss = criterion(y\_pred, y\_tensor)

print(f"Epoch: {epoch+1}, loss:{loss}")

optimizer.zero\_grad()

loss.backward()

optimizer.step()

with torch.no\_grad():

final\_preds\_torch = model(X\_tensor).numpy().flatten()

final\_preds\_denorm = final\_preds\_torch \* (y\_max - y\_min) + y\_min

plt.scatter(df['area\_sqft'], df['price'], label="Data points")

plt.plot(df['area\_sqft'], final\_preds\_denorm, color='red', label="Best Fit Line (PyTorch)")

plt.xlabel("Area (sq ft)")

plt.ylabel("Price")

plt.title("Area vs Price (Custom nn.Module + SGD)")

plt.legend()

plt.show()

for name, param in model.named\_parameters():

print(f"{name}: {param.data.numpy()}")

print("Final MSE Loss:", loss.item())

test\_input = np.array([[2135, 4]], dtype=np.float32)

test\_input\_norm = (test\_input - X\_mean) / X\_std

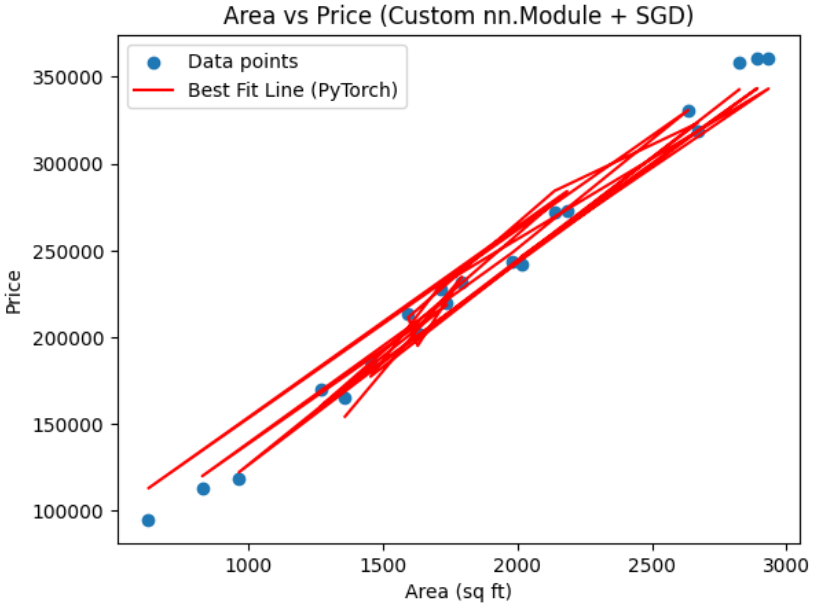
test\_tensor = torch.tensor(test\_input\_norm, dtype=torch.float32)

with torch.no\_grad():

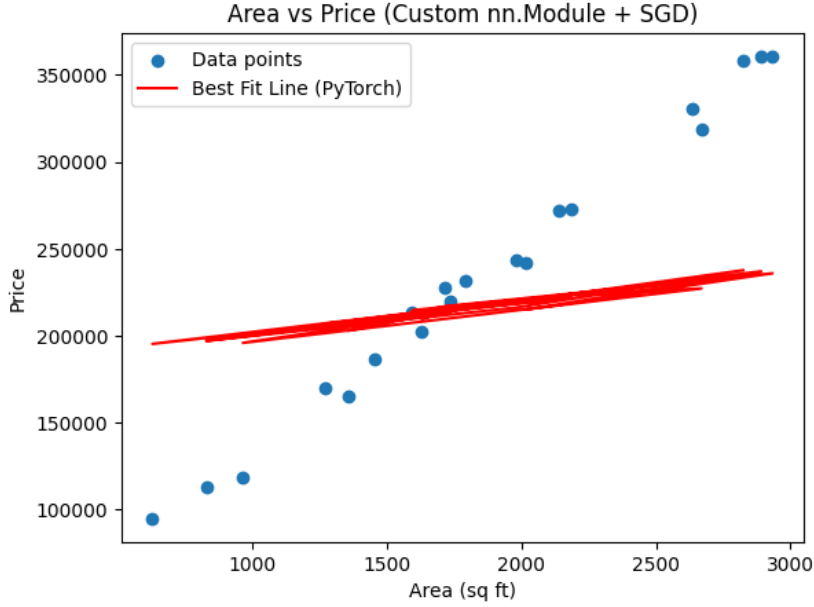
pred\_norm = model(test\_tensor).item()

pred\_price = pred\_norm \* (y\_max - y\_min) + y\_min

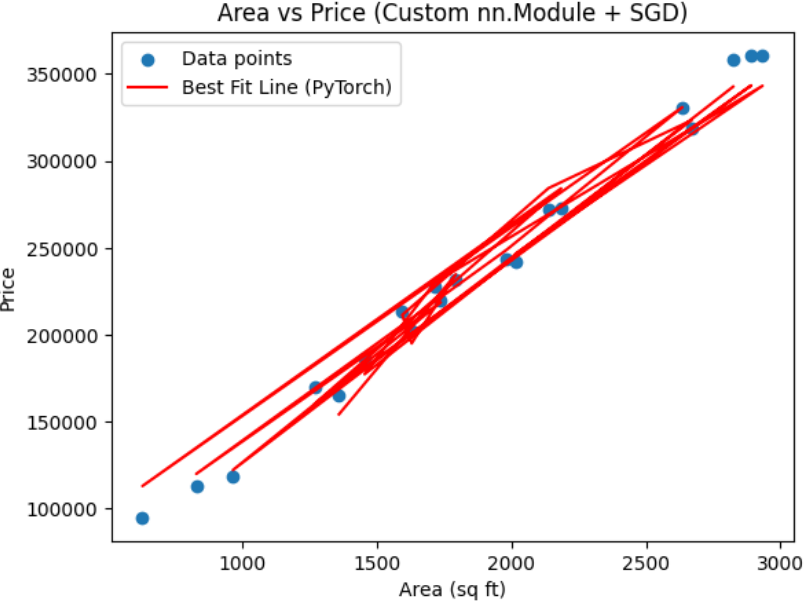
print(f"Predicted Price for area=2135 sq ft and rooms=4: {pred\_price:.2f}")

**OUTPUT (**lr=0.1**):**

**OUTPUT (lr=0.001):**

****

**OUTPUT (lr=0.4):**

****