|  |
| --- |
| import numpy as np |
|  | import matplotlib.pyplot as plt |
|  | import tensorflow as tf |
|  | from tensorflow.keras.initializers import Initializer |
|  | from tensorflow.keras.layers import Layer |
|  | from tensorflow.keras.initializers import RandomUniform, Initializer, Constant |
|  |  |
|  | def gaussian\_rbf(x, landmark, gamma=1): |
|  | return np.exp(-gamma \* np.linalg.norm(x - landmark)\*\*2) |
|  | # solving problem using matrices form |
|  | # AW = Y |
|  | def end\_to\_end(X1, X2, ys, mu1, mu2): |
|  | from\_1 = [gaussian\_rbf(i, mu1) for i in zip(X1, X2)] |
|  | from\_2 = [gaussian\_rbf(i, mu2) for i in zip(X1, X2)] |
|  | # plot |
|  |  |
|  | plt.figure(figsize=(13, 5)) |
|  | plt.subplot(1, 2, 1) |
|  | plt.scatter((x1[0], x1[3]), (x2[0], x2[3]), label="Class\_0") |
|  | plt.scatter((x1[1], x1[2]), (x2[1], x2[2]), label="Class\_1") |
|  | plt.xlabel("$X1$", fontsize=15) |
|  | plt.ylabel("$X2$", fontsize=15) |
|  | plt.title("Xor: Linearly Inseparable", fontsize=15) |
|  | plt.legend() |
|  |  |
|  | plt.subplot(1, 2, 2) |
|  | plt.scatter(from\_1[0], from\_2[0], label="Class\_0") |
|  | plt.scatter(from\_1[1], from\_2[1], label="Class\_1") |
|  | plt.scatter(from\_1[2], from\_2[2], label="Class\_1") |
|  | plt.scatter(from\_1[3], from\_2[3], label="Class\_0") |
|  | plt.plot([0, 0.95], [0.95, 0], "k--") |
|  | plt.annotate("Seperating hyperplane", xy=(0.4, 0.55), xytext=(0.55, 0.66), |
|  | arrowprops=dict(facecolor='black', shrink=0.05)) |
|  | plt.xlabel(f"$mu1$: {(mu1)}", fontsize=15) |
|  | plt.ylabel(f"$mu2$: {(mu2)}", fontsize=15) |
|  | plt.title("Transformed Inputs: Linearly Seperable", fontsize=15) |
|  | plt.legend() |
|  |  |
|  | # solving problem using matrices form |
|  | # AW = Y |
|  | A = [] |
|  |  |
|  | for i, j in zip(from\_1, from\_2): |
|  | temp = [] |
|  | temp.append(i) |
|  | temp.append(j) |
|  | temp.append(1) |
|  | A.append(temp) |
|  |  |
|  | A = np.array(A) |
|  | W = np.linalg.inv(A.T.dot(A)).dot(A.T).dot(ys) |
|  | print(np.round(A.dot(W))) |
|  | print(ys) |
|  | print(f"Weights: {W}") |
|  | return W |
|  |  |
|  |  |
|  | def predict\_matrix(point, weights): |
|  | gaussian\_rbf\_0 = gaussian\_rbf(np.array(point), mu1) |
|  | gaussian\_rbf\_1 = gaussian\_rbf(np.array(point), mu2) |
|  | A = np.array([gaussian\_rbf\_0, gaussian\_rbf\_1, 1]) |
|  | return np.round(A.dot(weights)) |
|  |  |
|  | # points |
|  | x1 = np.array([0, 0, 1, 1]) |
|  | x2 = np.array([0, 1, 0, 1]) |
|  | ys = np.array([0, 1, 1, 0]) |
|  |  |
|  | # centers |
|  | mu1 = np.array([0, 1]) |
|  | mu2 = np.array([1, 0]) |
|  |  |
|  | w = end\_to\_end(x1, x2, ys, mu1, mu2) |
|  |  |
|  | # testing |
|  |  |
|  | print(f"Input:{np.array([0, 0])}, Predicted: {predict\_matrix(np.array([0, 0]), w)}") |
|  | print(f"Input:{np.array([0, 1])}, Predicted: {predict\_matrix(np.array([0, 1]), w)}") |
|  | print(f"Input:{np.array([1, 0])}, Predicted: {predict\_matrix(np.array([1, 0]), w)}") |
|  | print(f"Input:{np.array([1, 1])}, Predicted: { |