**AI人工智慧 神經網路**

**AI人工智慧 神經網路概述**

2020-09-24 11:17 更新

神經網路是平行計算設備，它們試圖構建大腦的電腦模型。 背後的主要目標是開發一個系統來執行各種計算任務比傳統系統更快。 這些任務包括模式識別和分類，近似，優化和資料聚類。

**什麼是人工神經網路(ANN)**

人工神經網路(ANN)是一個高效的計算系統，其核心主題是借用生物神經網路的類比。人工神經網路也被稱為人工神經系統，並行分散式處理系統和連接系統。 ANN獲取了大量以某種模式相互連接的單元，以允許它們之間的通信。這些單元也稱為節點或神經元，是平行作業的簡單一處理器。

每個神經元通過連接連結與其他神經元連接。每個連接鏈路與具有關於輸入信號的資訊的權重相關聯。這是神經元解決特定問題最有用的資訊，因為體重通常會激發或抑制正在傳遞的信號。每個神經元都有其內部狀態，稱為啟動信號。在組合輸入信號和啟動規則之後產生的輸出信號可以被發送到其他單元。

如果想詳細研究神經網路，那麼可以參考以下示例 - [人工神經網路](https://www.yiibai.com/artificial_neural_network/index.html)。

**安裝有用的包**

在 Python 中創建神經網路，可以使用一個強大的 NeuroLab 神經網路包。它是一個基本的神經網路演算法庫，具有靈活的網路配置和 Python 學習演算法。可以在命令提示符下使用以下命令來安裝此套裝軟體 -

1. pip install NeuroLab

如果使用的是 Anaconda 環境，請使用以下命令安裝 NeuroLab -

1. conda install -c labfabulous neurolab

**構建神經網路**

在本節中，讓我們使用 NeuroLab 套裝軟體在 Python 中構建一些神經網路。

**AI人工智慧 基於感知器的分類器**

2020-09-24 11:00 更新

感知器是 ANN 的基石。 如果您想瞭解更多關於 Perceptron 的資訊，可以點選連結 - [artificial\_neural\_network](https://www.yiibai.com/artificial_neural_network/artificial_neural_network_supervised_learning.html" \t "_blank)

以下是逐步執行 Python 代碼，用於構建基於感知器的簡單神經網路分類器 -

如下所示導入必要的套裝軟體 -

1. import matplotlib.pyplot as plt
2. import neurolab as nl

請注意，這是一個監督學習的例子，因此您也必須提供目標值。

1. input = [[0, 0], [0, 1], [1, 0], [1, 1]]
2. target = [[0], [0], [0], [1]]

用 2 個輸入和 1 個神經元創建網路 -

1. net = nl.net.newp([[0, 1],[0, 1]], 1)

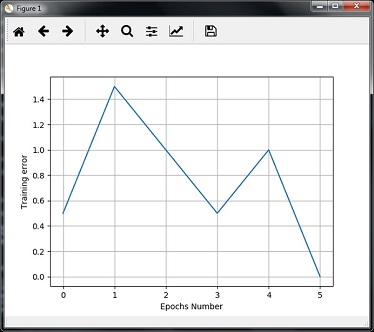
現在，訓練網路。 在這裡使用 Delta 規則進行訓練。

1. error\_progress = net.train(input, target, epochs=100, show=10, lr=0.1)

接下來，視覺化輸出並繪製圖表 -

1. plt.figure()
2. plt.plot(error\_progress)
3. plt.xlabel('Number of epochs')
4. plt.ylabel('Training error')
5. plt.grid()
6. plt.show()

可以看到下圖顯示了使用錯誤度量標準的訓練進度 -



**AI人工智慧 單層神經網路**

2020-09-24 10:57 更新

在這個例子中，我們來創建一個單層神經網路，它由獨立的神經元組成，這些神經元在輸入資料上起作用以產生輸出。 請注意，這裡使用 neural\_simple.txt 檔作為輸入。

如下所示導入所需的套裝軟體 -

1. import numpy as np
2. import matplotlib.pyplot as plt
3. import neurolab as nl

載入資料集如下代碼 -

1. input\_data = np.loadtxt(“/Users/admin/neural\_simple.txt')

以下是我們要使用的資料。 請注意，在此資料中，前兩列是特徵，最後兩列是標籤。

1. array([[2. , 4. , 0. , 0. ],
2. [1.5, 3.9, 0. , 0. ],
3. [2.2, 4.1, 0. , 0. ],
4. [1.9, 4.7, 0. , 0. ],
5. [5.4, 2.2, 0. , 1. ],
6. [4.3, 7.1, 0. , 1. ],
7. [5.8, 4.9, 0. , 1. ],
8. [6.5, 3.2, 0. , 1. ],
9. [3. , 2. , 1. , 0. ],
10. [2.5, 0.5, 1. , 0. ],
11. [3.5, 2.1, 1. , 0. ],
12. [2.9, 0.3, 1. , 0. ],
13. [6.5, 8.3, 1. , 1. ],
14. [3.2, 6.2, 1. , 1. ],
15. [4.9, 7.8, 1. , 1. ],
16. [2.1, 4.8, 1. , 1. ]])

現在，將這四列分成2個資料列和2個標籤 -

1. data = input\_data[:, 0:2]
2. labels = input\_data[:, 2:]

使用以下命令繪製輸入資料 -

1. plt.figure()
2. plt.scatter(data[:,0], data[:,1])
3. plt.xlabel('Dimension 1')
4. plt.ylabel('Dimension 2')
5. plt.title('Input data')

現在，為每個維度定義最小值和最大值，如下所示 -

1. dim1\_min, dim1\_max = data[:,0].min(), data[:,0].max()
2. dim2\_min, dim2\_max = data[:,1].min(), data[:,1].max()

接下來，如下定義輸出層中神經元的數量 -

1. nn\_output\_layer = labels.shape[1]

現在，定義一個單層神經網路 -

1. dim1 = [dim1\_min, dim1\_max]
2. dim2 = [dim2\_min, dim2\_max]
3. neural\_net = nl.net.newp([dim1, dim2], nn\_output\_layer)

訓練神經網路的時代數和學習率如下所示 -

1. error = neural\_net.train(data, labels, epochs = 200, show = 20, lr = 0.01)

現在，使用以下命令視覺化並繪製訓練進度 -

1. plt.figure()
2. plt.plot(error)
3. plt.xlabel('Number of epochs')
4. plt.ylabel('Training error')
5. plt.title('Training error progress')
6. plt.grid()
7. plt.show()

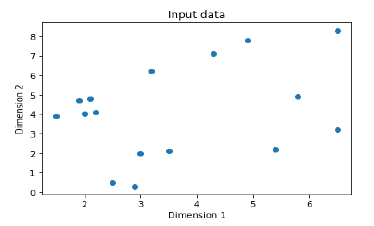
現在，使用上述分類器中的測試資料點 -

1. print('\nTest Results:')
2. data\_test = [[1.5, 3.2], [3.6, 1.7], [3.6, 5.7],[1.6, 3.9]] for item in data\_test:
3. print(item, '-->', neural\_net.sim([item])[0])

下面是測試結果 -

1. [1.5, 3.2] --> [1. 0.]
2. [3.6, 1.7] --> [1. 0.]
3. [3.6, 5.7] --> [1. 1.]
4. [1.6, 3.9] --> [1. 0.]

您可以看到迄今為止討論的代碼的輸出圖表 -





**AI人工智慧 多層神經網路**

2020-09-24 11:03 更新

在這個例子中，我們創建了一個由多個層組成的多層神經網路，以提取訓練資料中的基礎模式。 這個多層神經網路將像一個回歸器一樣工作。 我們將根據下面等式生成一些資料點: y = 2x2 + 8。

如下所示導入必要的套裝軟體 -

1. import numpy as np
2. import matplotlib.pyplot as plt
3. import neurolab as nl

根據上述公式生成一些資料點 -

1. min\_val = -30
2. max\_val = 30
3. num\_points = 160
4. x = np.linspace(min\_val, max\_val, num\_points)
5. y = 2 \* np.square(x) + 8
6. y /= np.linalg.norm(y)

現在，重塑這個資料集如下 -

1. data = x.reshape(num\_points, 1)
2. labels = y.reshape(num\_points, 1)

使用以下命令視覺化並繪製輸入資料集 -

1. plt.figure()
2. plt.scatter(data, labels)
3. plt.xlabel('Dimension 1')
4. plt.ylabel('Dimension 2')
5. plt.title('Data-points')

現在，構建神經網路，其具有兩個隱藏層，第一隱藏層中具有十個神經元的神經元，第二隱藏層中六個，輸出層中一個神經元。

1. neural\_net = nl.net.newff([[min\_val, max\_val]], [10, 6, 1])

現在使用梯度訓練演算法 -

1. neural\_net.trainf = nl.train.train\_gd

現在訓練網路的目標是學習上面生成的資料 -

1. error = neural\_net.train(data, labels, epochs = 1000, show = 100, goal = 0.01)

訓練資料點上運行神經網路 -

1. output = neural\_net.sim(data)
2. y\_pred = output.reshape(num\_points)

現在繪圖並視覺化任務 -

1. plt.figure()
2. plt.plot(error)
3. plt.xlabel('Number of epochs')
4. plt.ylabel('Error')
5. plt.title('Training error progress')

現在將繪製實際與預測輸出關係圖 -

1. x\_dense = np.linspace(min\_val, max\_val, num\_points \* 2)
2. y\_dense\_pred = neural\_net.sim(x\_dense.reshape(x\_dense.size,1)).reshape(x\_dense.size)
3. plt.figure()
4. plt.plot(x\_dense, y\_dense\_pred, '-', x, y, '.', x, y\_pred, 'p')
5. plt.title('Actual vs predicted')
6. plt.show()

執行上述代碼，您可以觀察如下所示的圖形 -

