|  |  |
| --- | --- |
| **Ex no : 4** | **Movie Recommender system using RBM** |
| **Date :** |

**Aim**

To implement multi-layer neural network for Movie Recommender system using Restricted Boltzmann Machines.

**Basic Theory of Restricted Boltzmann Machines.**

**Boltzmann Machines**

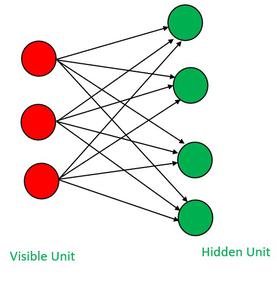
* + It is a network of neurons in which all the neurons are connected to each other. In this machine, there are two layers named visible layer or input layer and hidden layer. The visible layer is denoted as v and the hidden layer is denoted as the h.
  + In Boltzmann machine, there is no output layer. Boltzmann machines are random and generative neural networks capable of learning internal representations and are able to represent and (given enough time) solve tough combinatoric problems.
  + The Boltzmann distribution (also known as Gibbs Distribution) which is an integral part of Statistical Mechanics and also explain the impact of parameters like Entropy and Temperature on the Quantum States in Thermodynamics.

**Restricted Boltzmann Machines (RBM)**

* + A restricted term refers to that we are not allowed to connect the same type layer to each other.
  + In other words, the two neurons of the input layer or hidden layer can’t connect to each other. Although the hidden layer and visible layer can be connected to each other.
  + As in this machine, there is no output layer so the question arises how we are going to identify, adjust the weights and how to measure the that our prediction is accurate or not.
  + The RBM algorithm was proposed by Geoffrey Hinton (2007), which learns probability distribution over its sample training data inputs.

In RBM there are two phases through which the entire RBM works:

* + **1st Phase**: In this phase, we take the input layer and using the concept of weights and bias, we are going to activate the hidden layer. This process is said to be Feed Forward Pass. In Feed Forward Pass we are identifying the positive association and negative association.
  + Feed Forward Equation:
    - Positive Association — When the association between the visible unit and the hidden unit is positive.
    - Negative Association — When the association between the visible unit and the hidden unit is negative.
  + **2nd Phase:** As we don’t have any output layer. Instead of calculating the output layer, we are reconstructing the input layer through the activated hidden state. This process is said to be Feed Backward Pass.
  + We are just backtracking the input layer through the activated hidden neurons. After performing this we have reconstructed Input through the activated hidden state. So, we can calculate the error and adjust weight in this way.
  + Feed Backward Equation:
    - Error = Reconstructed Input Layer-Actual Input layer
    - Adjust Weight = Input\*error\*learning rate (0.1)
  + Let us consider an example in which we have some assumption that V1 visible unit activates the h1 and h2 hidden unit and V2 visible unit activates the h2 and h3 hidden.
  + Now when any new visible unit let V5 has come into the machine and it also activates the h1 and h2 unit. So, we can back trace the hidden units easily and also identify that the characteristics of the new V5 neuron is matching with that of V1. This is because V1 also activated the same hidden unit earlier.



**Code**

**Download the dataset from** [**https://github.com/srp98/Movie-Recommender-using-RBM/tree/a28ac757f07c1ca01042ac1cfbbe1d537a1bfabc/ml-1m**](https://github.com/srp98/Movie-Recommender-using-RBM/tree/a28ac757f07c1ca01042ac1cfbbe1d537a1bfabc/ml-1m)

import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Load the movies dataset and also pass header=None since files don't contain any headers

movies\_df = pd.read\_csv('/content/movies.dat', sep='::', header=None, engine='python')

print(movies\_df.head())

# Load the ratings dataset

ratings\_df = pd.read\_csv('ml-1m/ratings.dat', sep='::', header=None, engine='python')

print(ratings\_df.head())

# Lets rename our columns in these data frames so we can convey their data better

movies\_df.columns = ['MovieID', 'Title', 'Genres']

ratings\_df.columns = ['UserID', 'MovieID', 'Rating', 'Timestamp']

# Verify the changes done to the dataframes

print(movies\_df.head())

print(ratings\_df.head())

# Data Correction and Formatting

print('The Number of Movies in Dataset', len(movies\_df))

"""

- Our Movie ID's vary from 1 to 3952 while we have 3883 movies.

- Due to this, we won't be able to index movies through their ID since we would get memory indexing errors.

- To amend we can create a column that shows the spot in our list that particular movie is in:

"""

movies\_df['List Index'] = movies\_df.index

print(movies\_df.head())

# Merge movies\_df with ratings\_df by MovieID

merged\_df = movies\_df.merge(ratings\_df, on='MovieID')

# Drop unnecessary columns

merged\_df = merged\_df.drop('Timestamp', axis=1).drop('Title', axis=1).drop('Genres', axis=1)

# Display the result

print(merged\_df.head())

# Lets Group up the Users by their user ID's

user\_Group = merged\_df.groupby('UserID')

print(user\_Group.head())

"""

Formatting the data into input for the RBM.

Store the normalized users ratings into a list of lists called trX.

"""

# Amount of users used for training

amountOfUsedUsers = 1000

# Creating the training list

trX = []

# For each user in the group

for userID, curUser in user\_Group:

    # Create a temp that stores every movie's rating

    temp = [0]\*len(movies\_df)

    # For each movie in curUser's movie list

    for num, movie in curUser.iterrows():

        # Divide the rating by 5 and store it

        temp[movie['List Index']] = movie['Rating']/5.0

    # Add the list of ratings into the training list

    trX.append(temp)

    # Check to see if we finished adding in the amount of users for training

    if amountOfUsedUsers == 0:

        break

    amountOfUsedUsers -= 1

print(trX)

# Setting the models Parameters

hiddenUnits = 50

visibleUnits = len(movies\_df)

vb = tf.placeholder(tf.float32, [visibleUnits])  # Number of unique movies

hb = tf.placeholder(tf.float32, [hiddenUnits])  # Number of features were going to learn

W = tf.placeholder(tf.float32, [visibleUnits, hiddenUnits])  # Weight Matrix

# Phase 1: Input Processing

v0 = tf.placeholder("float", [None, visibleUnits])

\_h0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb)  # Visible layer activation

h0 = tf.nn.relu(tf.sign(\_h0 - tf.random\_uniform(tf.shape(\_h0))))  # Gibb's Sampling

# Phase 2: Reconstruction

\_v1 = tf.nn.sigmoid(tf.matmul(h0, tf.transpose(W)) + vb)  # Hidden layer activation

v1 = tf.nn.relu(tf.sign(\_v1 - tf.random\_uniform(tf.shape(\_v1))))

h1 = tf.nn.sigmoid(tf.matmul(v1, W) + hb)

""" Set RBM Training Parameters """

# Learning rate

alpha = 1.0

# Create the gradients

w\_pos\_grad = tf.matmul(tf.transpose(v0), h0)

w\_neg\_grad = tf.matmul(tf.transpose(v1), h1)

# Calculate the Contrastive Divergence to maximize

CD = (w\_pos\_grad - w\_neg\_grad) / tf.to\_float(tf.shape(v0)[0])

# Create methods to update the weights and biases

update\_w = W + alpha \* CD

update\_vb = vb + alpha \* tf.reduce\_mean(v0 - v1, 0)

update\_hb = hb + alpha \* tf.reduce\_mean(h0 - h1, 0)

# Set the error function, here we use Mean Absolute Error Function

err = v0 - v1

err\_sum = tf.reduce\_mean(err\*err)

""" Initialize our Variables with Zeroes using Numpy Library """

# Current weight

cur\_w = np.zeros([visibleUnits, hiddenUnits], np.float32)

# Current visible unit biases

cur\_vb = np.zeros([visibleUnits], np.float32)

# Current hidden unit biases

cur\_hb = np.zeros([hiddenUnits], np.float32)

# Previous weight

prv\_w = np.zeros([visibleUnits, hiddenUnits], np.float32)

# Previous visible unit biases

prv\_vb = np.zeros([visibleUnits], np.float32)

# Previous hidden unit biases

prv\_hb = np.zeros([hiddenUnits], np.float32)

sess = tf.Session()

sess.run(tf.global\_variables\_initializer())

# Train RBM with 15 Epochs, with Each Epoch using 10 batches with size 100, After training print out the error by epoch

epochs = 15

batchsize = 100

errors = []

for i in range(epochs):

    for start, end in zip(range(0, len(trX), batchsize), range(batchsize, len(trX), batchsize)):

        batch = trX[start:end]

        cur\_w = sess.run(update\_w, feed\_dict={v0: batch, W: prv\_w, vb: prv\_vb, hb: prv\_hb})

        cur\_vb = sess.run(update\_vb, feed\_dict={v0: batch, W: prv\_w, vb: prv\_vb, hb: prv\_hb})

        cur\_hb = sess.run(update\_hb, feed\_dict={v0: batch, W: prv\_w, vb: prv\_vb, hb: prv\_hb})

        prv\_w = cur\_w

        prv\_vb = cur\_vb

        prv\_hb = cur\_hb

    errors.append(sess.run(err\_sum, feed\_dict={v0: trX, W: cur\_w, vb: cur\_vb, hb: cur\_hb}))

    print(errors[-1])

plt.plot(errors)

plt.ylabel('Error')

plt.xlabel('Epoch')

plt.show()

"""

Recommendation System :-

- We can now predict movies that an arbitrarily selected user might like.

- This can be accomplished by feeding in the user's watched movie preferences into the RBM and then reconstructing the

  input.

- The values that the RBM gives us will attempt to estimate the user's preferences for movies that he hasn't watched

  based on the preferences of the users that the RBM was trained on.

"""

# Select the input User

inputUser = [trX[50]]

# Feeding in the User and Reconstructing the input

hh0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb)

vv1 = tf.nn.sigmoid(tf.matmul(hh0, tf.transpose(W)) + vb)

feed = sess.run(hh0, feed\_dict={v0: inputUser, W: prv\_w, hb: prv\_hb})

rec = sess.run(vv1, feed\_dict={hh0: feed, W: prv\_w, vb: prv\_vb})

# List the 20 most recommended movies for our mock user by sorting it by their scores given by our model.

scored\_movies\_df\_50 = movies\_df

scored\_movies\_df\_50["Recommendation Score"] = rec[0]

print(scored\_movies\_df\_50.sort\_values(["Recommendation Score"], ascending=False).head(20))

""" Recommend User what movies he has not watched yet """

# Find the mock user's UserID from the data

print(merged\_df.iloc[50])  # Result you get is UserID 150

# Find all movies the mock user has watched before

movies\_df\_50 = merged\_df[merged\_df['UserID'] == 150]

print(movies\_df\_50.head())

""" Merge all movies that our mock users has watched with predicted scores based on his historical data: """

# Merging movies\_df with ratings\_df by MovieID

merged\_df\_50 = scored\_movies\_df\_50.merge(movies\_df\_50, on='MovieID', how='outer')

# Dropping unnecessary columns

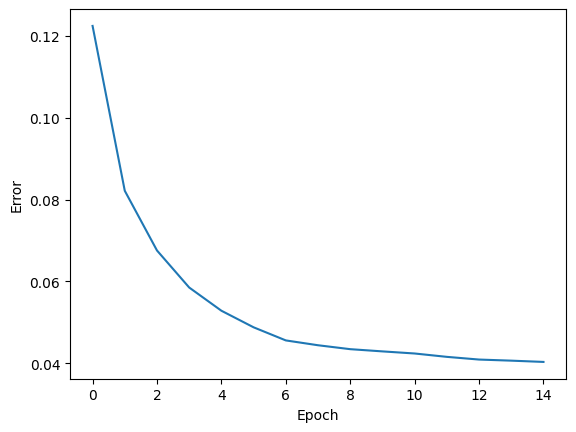
merged\_df\_50 = merged\_df\_50.drop('List Index\_y', axis=1).drop('UserID', axis=1)

# Sort and take a look at first 20 rows

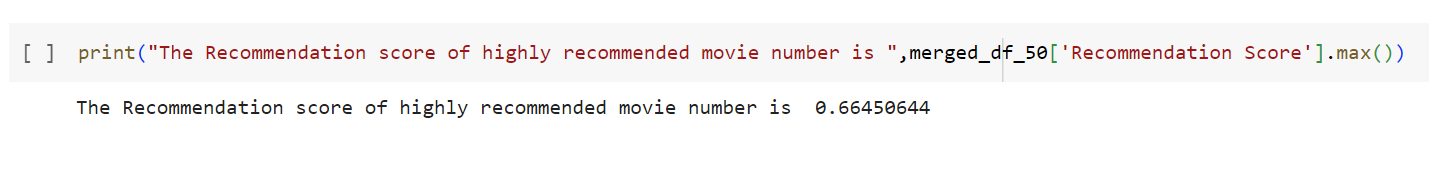
print(merged\_df\_50.sort\_values(['Recommendation Score'], ascending=False).head(20))

""" There are some movies the user has not watched and has high score based on our model. So, we can recommend them. """

**OUTPUT**



****

****

**RESULT**

Thus implementation of Movie Recommender system using Restricted Boltzmann Machines has been carried out successfully.