AISC

Class: B.E COMP

Experiment 10: Solve given problem using Un-Supervised Neural Network.

.

Learning Objective:

Basic Experiments

Solve given problem using Un-Supervised Neural Network.
Tools: Python

Theory:

1. Determine the type of training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In the case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
2. Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.
4. Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or decision trees.
5. Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a *validation* set) of the training set, or via cross-validation.
6. Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.
7. **Artificial neural networks** (**ANNs**), usually simply called **neural networks** (**NNs**), are computing systems vaguely inspired by the biological neural networks that constitute animal brains.

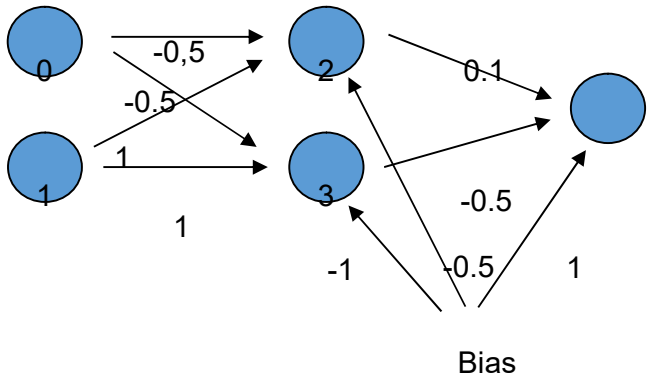
An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called *edges*. Neurons and edges typically have a *weight* that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the

aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Neural networks learn (or are trained) by processing examples, each of which contains a known "input" and "result," forming probability-weighted associations between the two, which are stored within the data structure of the net itself. The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and a target output. This is the error. The network then adjusts its weighted associations according to a learning rule and using this error value. Successive adjustments will cause the neural network to produce output which is increasingly similar to the target output. After a sufficient number of these adjustments the training can be terminated based upon certain criteria. This is known as supervised learning.

Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.

Supervised learning uses a set of paired inputs and desired outputs. The learning task is to produce the desired output for each input. In this case the cost function is related to eliminating incorrect deductions.[52] A commonly used cost is the mean-squared error, which tries to minimize the average squared error between the network's output and the desired output. Tasks suited for supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). Supervised learning is also applicable to sequential data (e.g., for hand writing, speech and gesture recognition). This can be thought of as learning with a "teacher", in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.



Implementation:

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt

from scipy.ndimage import convolve
from sklearn import linear_model, datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.neural_network import BernoulliRBM
from sklearn.pipeline import Pipeline
from sklearn.base import clone

def nudge_dataset(X, Y):
    """
    This produces a dataset 5 times bigger than the original one,
    by moving the 8x8 images in X around by 1px to left, right, down, up
    """
    direction_vectors = [
        [[0, 1, 0],
         [0, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [1, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 1],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 0],
         [0, 1, 0]]]

    def shift(x, w):
        return convolve(x.reshape((8, 8)), mode='constant', weights=w).ravel()

    X = np.concatenate([X] +
                       [np.apply_along_axis(shift, 1, X, vector)
                        for vector in direction_vectors])
    Y = np.concatenate([Y for _ in range(5)], axis=0)
    return X, Y
X, y = datasets.load_digits(return_X_y=True)
X = np.asarray(X, 'float32')
X, Y = nudge_dataset(X, y)
X = (X - np.min(X, 0)) / (np.max(X, 0) + 0.0001)  # 0-1 scaling

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=0)
logistic = linear_model.LogisticRegression(solver='newton-cg', tol=1)
```

```
rbm = BernoulliRBM(random_state=0, verbose=True)

rbm_features_classifier = Pipeline(
    steps=[('rbm', rbm), ('logistic', logistic)])

rbm.learning_rate = 0.06
rbm.n_iter = 10

rbm.n_components = 100
logistic.C = 6000

rbm_features_classifier.fit(X_train, Y_train)
raw_pixel_classifier = clone(logistic)
raw_pixel_classifier.C = 100.
raw_pixel_classifier.fit(X_train, Y_train)

Y_pred = rbm_features_classifier.predict(X_test)
print("Logistic regression using RBM features:\n%s\n" % (
    metrics.classification_report(Y_test, Y_pred)))

Y_pred = raw_pixel_classifier.predict(X_test)
print("Logistic regression using raw pixel features:\n%s\n" % (
    metrics.classification_report(Y_test, Y_pred)))
plt.figure(figsize=(4.2, 4))
for i, comp in enumerate(rbm.components_):
    plt.subplot(10, 10, i + 1)
    plt.imshow(comp.reshape((8, 8)), cmap=plt.cm.gray_r,
               interpolation='nearest')
    plt.xticks(())
    plt.yticks(())
plt.suptitle('100 components extracted by RBM', fontsize=16)
plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)

plt.show()
```

Output:

```
[BernoulliRBM] Iteration 1, pseudo-likelihood = -25.39, time = 0.31s
[BernoulliRBM] Iteration 2, pseudo-likelihood = -23.77, time = 0.44s
[BernoulliRBM] Iteration 3, pseudo-likelihood = -22.94, time = 0.43s
[BernoulliRBM] Iteration 4, pseudo-likelihood = -21.91, time = 0.43s
[BernoulliRBM] Iteration 5, pseudo-likelihood = -21.69, time = 0.75s
[BernoulliRBM] Iteration 6, pseudo-likelihood = -21.06, time = 0.44s
[BernoulliRBM] Iteration 7, pseudo-likelihood = -20.89, time = 0.53s
[BernoulliRBM] Iteration 8, pseudo-likelihood = -20.64, time = 0.68s
[BernoulliRBM] Iteration 9, pseudo-likelihood = -20.36, time = 0.63s
[BernoulliRBM] Iteration 10, pseudo-likelihood = -20.09, time = 0.64s
```

Logistic regression using RBM features:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	174
1	0.92	0.93	0.92	184
2	0.95	0.95	0.95	166
3	0.96	0.88	0.91	194
4	0.97	0.94	0.96	186
5	0.91	0.92	0.92	181
6	0.98	0.97	0.97	207
7	0.93	0.98	0.96	154
8	0.88	0.90	0.89	182
9	0.87	0.92	0.90	169
accuracy			0.94	1797
macro avg	0.94	0.94	0.94	1797
weighted avg	0.94	0.94	0.94	1797

Logistic regression using raw pixel features:

	precision	recall	f1-score	support
0	0.90	0.93	0.91	174
1	0.60	0.58	0.59	184
2	0.75	0.85	0.80	166
3	0.78	0.78	0.78	194
4	0.81	0.84	0.82	186
5	0.77	0.77	0.77	181
6	0.91	0.87	0.89	207
7	0.86	0.88	0.87	154
8	0.67	0.58	0.62	182
9	0.75	0.76	0.75	169
accuracy			0.78	1797
macro avg	0.78	0.78	0.78	1797
weighted avg	0.78	0.78	0.78	1797

Learning Outcomes: Students should have the ability to ANN

LO1: Understand the problem formulation

Course Outcomes: Upon completion of the course students will be able to understand ANN

Conclusion: Thus, the aim to study and implement ANN

Viva Questions:

Ques.1 What do you understand by ANN?

Ques. 2. Explain the basic steps in ANN?

Ques. 3. Write types of problem discussed in detail.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				