**NNFL Mini Project**

**Title -** Predict type of Wheat from seven input parameters using back propagation network in Python.

**Team Members** – Yadnesh Dhanawade ( 7864 )

Sneha Samuel ( 8306 )

Alden Noronha ( 8292 )

**Problem Statement -**

Design a Neural network to classify three varieties of wheat kama, Canadian and rosa using 7 input parameters - area A, perimeter P, Compactness C = 4\*pi\*A/P^2, length of kernel, width of kernel, asymmetry coefficient and length of kernel groove. Then predict the type of wheat using seven input parameters using back propagation neural network.

|  |  |
| --- | --- |
| **Theory –**  The backpropagation algorithm is used in the classical feed-forward artificial neural network. It is the technique still used to train large [deep learning](http://machinelearningmastery.com/what-is-deep-learning/) networks. The learning algorithm is one of the most important developments in neural networks (Bryson and Ho, 1969; Werbos, 1974; Lecun, 1985; Parker, 1985; Rumelhan, 1986). This network has reawakened the scientific and engineering community to the modelling and processing of numerous phenomena using ne networks. This learning algorithm is applied to a multi layer feed-forward consisting of processing with continuous differentiable activation functions. Networks associated with back-propagation learning algorithm are also called as back propagation networks. (BPNs). For a given set of training input-output pair, this algorithm provides a procedure for changing the weights in a BPN to classify the given input patterns correctly. The basic concept for this weight update algorithm is simply the gradient descent method as used in the case of simple perceptron networks with differentiable units. This is a method where the error is propagated ack to the hidden unit    **Algorithm –**   1. Load the input dataset file ‘wheat\_dataset.csv’ till line no. 190 as input to the neural net. This forms as training dataset. Remaining lines will be used as testing dataset. 2. Put the value 1 for kama wheat type, 0.5 for canadian wheat type and 0 for Rosa wheat type. 3. Convert the string values of the file to float/int values. 4. Define the learning rate, number of hidden layers and number of epochs. 5. Initialize random weights and biases of the network in the range 0 to 1. 6. Train the network by repeat Steps 6 – 10 for each line entry to of training dataset. 7. Forward Propagate and calculate the net input to each neuron of hidden layer. 8. Apply binary sigmoidal activation function and calculate the error δ. 9. Back Propagate Error for each neuron. 10. Update the weights. 11. After training the neural net, print the final weights calculated. 12. Give the final weights to predict function. 13. For each line of testing dataset, calculate the output. 14. Deduct the predicted output from actual output (1, 0.5, 0). And calculate the score. 15. Print the final scores. 16. Take the mean of all scores and calculate the mean accuracy.   **Implementation Program –**  from random import seed  from random import randrange  from random import random  from csv import reader  from math import exp  def **load\_csv**(filename):  dataset = list()  with open(filename, *'r'*) as file:  csv\_reader = reader(file)  for row in csv\_reader:  if not row:  continue  dataset.append(row)  return dataset  def **str\_column\_to\_float**(dataset, column):  for row in dataset:  row[column] = float(row[column].strip())  def **str\_column\_to\_int**(dataset, column):  class\_values = [row[column] for row in dataset]  unique = set(class\_values)  lookup = dict()  for i, value in enumerate(unique):  lookup[value] = i  for row in dataset:  row[column] = lookup[row[column]]  return lookup  def **dataset\_minmax**(dataset):  minmax = list()  stats = [[min(column), max(column)] for column in zip(\*dataset)]  return stats  def **normalize\_dataset**(dataset, minmax):  for row in dataset:  for i in range(len(row)-1):  row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])  def **cross\_validation\_split**(dataset, n\_folds):  dataset\_split = list()  dataset\_copy = list(dataset)  fold\_size = int(len(dataset) / n\_folds)  for i in range(n\_folds):  fold = list()  while len(fold) < fold\_size:  index = randrange(len(dataset\_copy))  fold.append(dataset\_copy.pop(index))  dataset\_split.append(fold)  return dataset\_split  def **accuracy\_metric**(actual, predicted):  correct = 0  for i in range(len(actual)):  if actual[i] == predicted[i]:  correct += 1  return correct / float(len(actual)) \* 100.0  def **evaluate\_algorithm**(dataset, algorithm, n\_folds, \*args):  folds = cross\_validation\_split(dataset, n\_folds)  scores = list()  for fold in folds:  train\_set = list(folds)  train\_set.remove(fold)  train\_set = sum(train\_set, [])  test\_set = list()  for row in fold:  row\_copy = list(row)  test\_set.append(row\_copy)  row\_copy[-1] = None  predicted = algorithm(train\_set, test\_set, \*args)  actual = [row[-1] for row in fold]  accuracy = accuracy\_metric(actual, predicted)  scores.append(accuracy)  return scores  # Calculate neuron activation for an input  def **activate**(weights, inputs):  activation = weights[-1]  for i in range(len(weights)-1):  activation += weights[i] \* inputs[i]  return activation  def **transfer**(activation):  return 1.0 / (1.0 + exp(-activation))  def **forward\_propagate**(network, row):  inputs = row  for layer in network:  new\_inputs = []  for neuron in layer:  activation = activate(neuron[*'weights'*], inputs)  neuron[*'output'*] = transfer(activation)  new\_inputs.append(neuron[*'output'*])  inputs = new\_inputs  return inputs  def **transfer\_derivative**(output):  return output \* (1.0 - output)  def **backward\_propagate\_error**(network, expected):  for i in reversed(range(len(network))):  layer = network[i]  errors = list()  if i != len(network)-1:  for j in range(len(layer)):  error = 0.0  for neuron in network[i + 1]:  error += (neuron[*'weights'*][j] \* neuron[*'delta'*])  errors.append(error)  else:  for j in range(len(layer)):  neuron = layer[j]  errors.append(expected[j] - neuron[*'output'*])  for j in range(len(layer)):  neuron = layer[j]  neuron[*'delta'*] = errors[j] \* transfer\_derivative(neuron[*'output'*])  def **update\_weights**(network, row, l\_rate):  for i in range(len(network)):  inputs = row[:-1]  if i != 0:  inputs = [neuron[*'output'*] for neuron in network[i - 1]]  for neuron in network[i]:  for j in range(len(inputs)):  neuron[*'weights'*][j] += l\_rate \* neuron[*'delta'*] \* inputs[j]  neuron[*'weights'*][-1] += l\_rate \* neuron[*'delta'*]  def **train\_network**(network, train, l\_rate, n\_epoch, n\_outputs):  for epoch in range(n\_epoch):  for row in train:  outputs = forward\_propagate(network, row)  expected = [0 for i in range(n\_outputs)]  expected[row[-1]] = 1  backward\_propagate\_error(network, expected)  update\_weights(network, row, l\_rate)  def **initialize\_network**(n\_inputs, n\_hidden, n\_outputs):  network = list()  hidden\_layer = [{*'weights'*:[random() for i in range(n\_inputs + 1)]} for i in range(n\_hidden)]  network.append(hidden\_layer)  output\_layer = [{*'weights'*:[random() for i in range(n\_hidden + 1)]} for i in range(n\_outputs)]  network.append(output\_layer)  print(network)  return network  def **predict**(network, row):  outputs = forward\_propagate(network, row)  return outputs.index(max(outputs))  def **back\_propagation**(train, test, l\_rate, n\_epoch, n\_hidden):  n\_inputs = len(train[0]) - 1  n\_outputs = len(set([row[-1] for row in train]))  network = initialize\_network(n\_inputs, n\_hidden, n\_outputs)  train\_network(network, train, l\_rate, n\_epoch, n\_outputs)  predictions = list()  for row in test:  prediction = predict(network, row)  predictions.append(prediction)  return(predictions)  seed(1)  filename = *'wheat\_dataset.csv'*  dataset = load\_csv(filename)  for i in range(len(dataset[0])-1):  str\_column\_to\_float(dataset, i)  str\_column\_to\_int(dataset, len(dataset[0])-1)  minmax = dataset\_minmax(dataset)  normalize\_dataset(dataset, minmax)  n\_folds = 5  l\_rate = 0.3  n\_epoch = 500  n\_hidden = 5  scores = evaluate\_algorithm(dataset, back\_propagation, n\_folds, l\_rate, n\_epoch, n\_hidden)  print(*'Scores: %s'* % scores)  print(*'Mean Accuracy: %.3f%%'* % (sum(scores)/float(len(scores))))  **Result and Conclusion –**        Thus a neural net was successfully made which was able to predict the type of wheat from seven input parameters - area A, perimeter P, compactness C = 4\*pi\*A/P^2, length of kernel, width of kernel, asymmetry coefficient and length of kernel groove. The neural net was trained 70% of dataset and tested on 30% of the dataset. The different accuracy scores were calculated. A mean accuracy of 93% was achieved. |  |