HOMEWORK #5
Neural Network

Group Members: Onzali Suba (9690-3228-04)
and Nitya Pydipati (3219-5181-98)

Comparison with Scikit Learn

- Machine Learning Library in Python

Class sklearn.neural_network.MLPClassifier

- The class has recently been added to scikit's library as part of the 0.18 version
- It provides various parameters as some mentioned below in order to fine tune the performance of the neural network.
- It provides us with four choices for activation functions like "relu" (default), "identity", "logistic" or also known as the sigmoid function and "tanh".
- It has over four choices in its solver functions out of which two - 'adam' (default) optimizes weight and works very well with large datasets in terms of training time and validation score and 'lbfgs' provides the same optimization for smaller datasets. Whereas 'sgd' is commonly used with cases pertaining to stochastic gradient descent.
- Class : MLPClassifier use parameter 'alpha' for regularization (L2 regularization) term which helps in avoiding overfitting by penalizing weights with large magnitudes.
- The max_iter parameter is used to determine how many times the solver has to iterate until convergence.
- The learning_rate_init is only used when the solver is 'sgd' and controls the step-size in updating our weights

Application of Neural Networks

Neural networks are used to find solution for problems where algorithmic solutions may not exist or are computationally very expensive.

- **Pattern/Character Recognition:**
  It pre-processes the input from the image uploaded, segment and perform feature extraction. After this classification and recognition are used to recognize handwritten characters.
- **Image Compression:**
  The ratio of the size of the input layer to the size of the intermediate layer is - of course - the compression ratio.
- **Financial: Stock Market Prediction, Credit Worthiness, Fraud Detection:**
  Dealing with uncertainty in finance primarily involves recognition of patterns in data and using these patterns to predict future events. Artificial Neural Networks offer qualitative methods for economic systems that traditional quantitative tools cannot quantify due to the complexity in translating the systems into precise mathematical functions.
- **The famous Travelling Salesman Problem:**
  The basic neural network approaches- elastic nets and self-organizing maps appear to be the best approaches for solving the TSP since it is a combinatorial approach problem.

- **Time Series Prediction:**
  Identifying patterns in the data that we collected and applying the neural network to train so as to predict correct outcomes for future data.
- **Learning to Control like in Self Driving cars:**
  The images from the camera of the phone are taken as inputs into the neural network which performs back propagation to train and determine patterns and take decisions on how to drive the car.
- **Medical Diagnosis:**
  Data which consists of symptoms is taken and fed into the neural network and trained. When a new test case is presented it determines the closest diagnosis.
- Further applications include processing and evaluating input data from several soft sensors, Anomaly or Intrusion Detection, Google Voice complemented by NLP, Employee Selection and Hiring, Retail Margin Forecasting.

Data structures used in the algorithms

1. **Numpy Arrays:**
   We use numpy arrays in many instances. For eg: to store,retrieve and manipulate values of data and weights.

   Examples:
   ```
   self.W1=np.random.uniform(low=-0.001,high=0.001,
   size=(self.inputLayerSize,self.hiddenLayerSize))
   self.W2=np.random.uniform(low=-0.001,high=0.001,
   size=(self.hiddenLayerSize,self.outputLayerSize))
   delta_input=np.multiply(np.dot(delta_hidden,self.W2.T),(1-np.square(self.z2)))
   ```

2. **List:**
   We are using lists in many instances. For eg: for storing the images, given labels after they are read.

   ```
   training_images = []
   training_labels = []

   training_images.append(load_pgm_image(training_image))
   training_labels.append(1)
   ```

Optimizations

- Avoiding dot operator:

for loop evaluates the function references each time and thus this leads to inefficiency. Thus we can initialize do this, so each time in the *for* loop when we calculate the deltas or do updation of weights we don't have to reference it repeatedly:

```
yH=self.yHat
learningRate=self.learningRate
sigmoid_prime=self.sigmoid_prime

error=y - yH
# Deltas
delta_output = np.multiply(error,sigmoid_prime(yH))
```

Challenges faced

- We faced the below issue while computing the activation function values and the delta values which resulted in nan values:

  RuntimeWarning: overflow encountered in square

- Another issue was that the predicted values for all inputs always turned out to be 1 after the second update to the weights.

Output

```
gestures/A/A_down_1.pgm: 1
gestures/A/A_down_2.pgm: 1
gestures/A/A_hold_1.pgm: 0
gestures/A/A_hold_10.pgm: 0
gestures/A/A_stop_1.pgm: 0
gestures/A/A_stop_4.pgm: 0
gestures/A/A_up_1.pgm: 0
gestures/A/A_up_10.pgm: 0
gestures/B/B_down_1.pgm: 1
gestures/B/B_down_2.pgm: 0
gestures/B/B_hold_1.pgm: 0
gestures/B/B_hold_2.pgm: 0
gestures/B/B_stop_1.pgm: 0
gestures/B/B_stop_2.pgm: 0
gestures/B/B_up_1.pgm: 0
gestures/B/B_up_4.pgm: 1
gestures/C/C_down_1.pgm: 1
gestures/C/C_down_2.pgm: 1
gestures/C/C_hold_1.pgm: 0
gestures/C/C_hold_2.pgm: 0
gestures/C/C_stop_2.pgm: 0
gestures/C/C_stop_3.pgm: 0
gestures/C/C_up_1.pgm: 0
gestures/D/D_down_1.pgm: 0
gestures/D/D_down_2.pgm: 0
gestures/D/D_hold_1.pgm: 0
```

```
gestures/D/D_hold_2.pgm: 0
gestures/D/D_hold_6.pgm: 0
gestures/D/D_stop_1.pgm: 0
gestures/D/D_stop_2.pgm: 0
gestures/D/D_up_1.pgm: 0
gestures/D/D_up_3.pgm: 0
gestures/E/E_down_1.pgm: 1
gestures/E/E_hold_1.pgm: 0
gestures/E/E_hold_5.pgm: 0
gestures/E/E_stop_1.pgm: 0
gestures/E/E_stop_2.pgm: 0
gestures/E/E_up_1.pgm: 0
gestures/E/E_up_2.pgm: 1
gestures/F/F_down_1.pgm: 1
gestures/F/F_down_4.pgm: 1
gestures/F/F_hold_1.pgm: 0
gestures/F/F_hold_2.pgm: 0
gestures/F/F_stop_2.pgm: 0
gestures/F/F_stop_5.pgm: 0
gestures/G/G_down_2.pgm: 1
gestures/G/G_down_3.pgm: 1
gestures/G/G_hold_4.pgm: 0
gestures/G/G_stop_2.pgm: 0
gestures/G/G_stop_5.pgm: 0
gestures/G/G_up_2.pgm: 1
gestures/G/G_up_5.pgm: 0
gestures/H/H_down_2.pgm: 0
gestures/H/H_hold_10.pgm: 0
gestures/H/H_hold_2.pgm: 0
gestures/H/H_hold_5.pgm: 0
gestures/H/H_stop_5.pgm: 0
gestures/H/H_stop_6.pgm: 0
gestures/H/H_up_5.pgm: 0
gestures/I/I_hold_2.pgm: 0
gestures/I/I_hold_5.pgm: 0
gestures/I/I_down_3.pgm: 1
gestures/I/I_stop_5.pgm: 0
gestures/I/I_stop_6.pgm: 0
gestures/I/I_up_2.pgm: 0
gestures/I/I_up_3.pgm: 0
gestures/J/J_down_5.pgm: 1
gestures/J/J_down_6.pgm: 0
gestures/J/J_hold_2.pgm: 0
gestures/J/J_hold_3.pgm: 0
gestures/J/J_stop_7.pgm: 0
gestures/J/J_stop_8.pgm: 0
gestures/J/J_up_1.pgm: 0
gestures/J/J_up_2.pgm: 0
gestures/K/K_down_2.pgm: 1
gestures/K/K_down_3.pgm: 1
```

```
gestures/K/K_hold_1.pgm: 0
gestures/K/K_hold_2.pgm: 0
gestures/K/K_hold_3.pgm: 0
gestures/K/K_stop_1.pgm: 0
gestures/K/K_stop_2.pgm: 0
gestures/K/K_stop_1.pgm: 0
gestures/K/K_stop_2.pgm: 0
correct rate: 75.0
```

CONTRIBUTIONS

Code: Collaborated
Report: Collaborated