Neural Network Report

**Basic Information**

Language: python

Programming Files (4): interface.py, NN.py, generate\_NN.py, process.py

* Interface.py: creates GUI that uses NN.py and generate\_NN.py to allow the user to generate a Neural Network (NN) of customized structure, train a NN, and test a NN
* NN.py: contains the NN class, which has the essential functions of a NN such as train, test, activation/gradience functions
* generate\_NN.py: contains function that generates a NN text file of pseudo-randomly generated weights (found random float values between 0 and 1) in the desired form based on the provided number of layers and number of nodes per each layer from the user interface entries
* process.py: resizes, gray-scales, and outputs individual pixel values of the images to desired output (training & testing) files. To use, the path, which has been manually entered to match my machine, must be changed.

Custom Dataset:

* Background: The VIP group that I am currently working with is working on designing an open-sourced and low cost bionic hand (<https://michellekatz.wixsite.com/ai-rock-paper-scisso>). One of the ultimate goals (in a simplified phrase) is to implement a neural network that can detect objects and have the hand react against it (using already mapped out motor positions). As a proof of concept, our advisors (Professor Shah and Weiser) suggested that we first create a physical model that can play rock, paper, scissor against real opponents. To train previous NN’s, we successfully underwent a full IRB and received permission to collect images of the Cooper Union community’s hands forming rock, paper, and scissor. Thus far, we have been able to collect 43 total images (that are unidentifiable) from various students and faculty who have given permission to utilize and publish their hand images as part of the dataset. So, the images that I used for the custom dataset consists of the Cooper Union members forming the shapes rock, paper, and scissor.
* To process the images of the hands, process.py first resizes each image to 200x200, then gray-scales them. After, each image is outputted to the desired file as pixel values, which represent input nodes, on a single line. At the end of each line, there are 3 values (two 0’s and one 1) that represent the possible outputs of rock, paper, scissor in the respective order. Ex) 1 0 0 = rock

Ex)

A close-up of a hand

Description automatically generated

A close up of a rose

Description automatically generated with low confidence

A close-up of a hand

Description automatically generated with medium confidence

**User Interface**

\*images of ex 1 & 2 were taken prior to adding a activation function switch. Priorly, all of the activation and descent was based on sigmoid function.

Ex 1)

Graphical user interface, application

Description automatically generated

* Pressing generate would generate a NN with 2 hidden layers (with 5 & 6 nodes) as well as input and output nodes with 10 nodes each as weights that account for a bias term as the first term of each layer

Ex 2)

Graphical user interface, application

Description automatically generated

Ex 3)Graphical user interface, application

Description automatically generated

The train and test entries are same as the stated requirements (except that I have a choice of either sigmoid or ReLu available). The generate entries take in the number of layers and the number of nodes per layer with a single space as the delimiter. As the \* shows, the number of layers and the number of values entered into the number of nodes entry must match. To apply any of the functions, the buttons on the bottom and can be pressed. It is also important to note that the activation function applies to both train and test functions.

Custom Dataset

* Although 200x200 dimension and 43 total images would be quick to process for any modern NN (especially running on GPU), the “manual” method of computing values on an index basis proved to be a long task for my machine. For example, I had to abandon a training instance of a single hidden layer model of a 100 epochs because it was taking too long with an unforeseeable future. However, shorter epoch studies with various NN structures were accomplished. With that being said, shorter epoch iterations still took a lengthy number of minutes to compute.
* For all cases, the testing set consisted of 9 examples (3 of each) and the training set consisted of 34 examples.
* Case Study: Single Hidden Layer (16 nodes unless stated otherwise)
  + Learn rate = 0.08, Epochs = 10, Activation = sigmoid

Text

Description automatically generated

* + Learn rate = 0.14, Epochs = 15, Activation = sigmoid

Text

Description automatically generated

* + - same as example above
  + Learn rate = 0.25, Epochs = 40, Activation = sigmoid

Text

Description automatically generated

* + Learn rate = 0.25, Epochs = 10, Activation = ReLU

Text

Description automatically generated

* Case Study: Double Hidden Layers (16 nodes each unless stated otherwise)
  + Learn rate = 0.2, Epochs = 10, Activation = sigmoidText

    Description automatically generated

Double hidden layer didn’t make a big improvement.

* + Learn rate = 0.2, Epochs = 10, Activation = ReLU

Text

Description automatically generated

* + Overall, with regard to the requirement for the course, a single hidden layer with 16 nodes and the following traits: Learn rate = 0.25, Epochs = 40, Activation = sigmoid led to the best result.

Improvements

* A CNN would be better suited for object detection
* The structure could be changed to run on GPU which would decrease the time complexity by a very large amount
* The number of input nodes could be reduced by resizing to a smaller dimension, but then details of the image are sacrificed as result (need to find a balance)
* Increase number of examples

For more documents/files, visit https://github.com/chrislee8684/Neural\_Network