8 Classifier Using a Single Perceptron

Roberto Cruz

I. Algorithm Implementation

Max iterations

When building the classifier, the best way to figure out the number of iterations needed to get the most accuracy in classification seemed to be to look for convergence. Therefore, the learning phase is treated as a fixed point algorithm, and the iterations stop when the fluctuation in number of correct classifications is below a certain threshold, here taken to be 1.

History & Convergence

To determine the convergence, a moving window was used to examine the previous 10 iterations at each current iteration. Using this stored history, the current amount of convergence can be calculated using the maximum number of correct classifications within that window and the minimum number of correct classification within that window. When the difference within that window between the maximum number of correct classifications and the minimum number of correct classifications is 1 or less (in other words, less than the threshold), and the maximum number of correct classifications is reached somewhere in the window, the training will finish.

Looping over training set

Within each iteration, the perceptron loops through and “relearns” the entire training set of 217 files, looking at both the positive and negative examples. The order of examination is made arbitrary using pseudorandom number generation. This is done so that the order of positive vs negative examples isn’t directly hardcode-able.

The overall algorithm is shown in pseudocode as follows:

```
# when using convergence approach, use the following
# continue iterating until the fluctuation decreases beyond a certain radius
while radius > 1 or min(history) > 0:
    epoch += 1

    # rewind file order to start from the first file
    reset()

    # counters for correct and incorrect classifications
    num_correct = 0
    num_incorrect = 0

    # loop through training set
```

```
    for i in range(TRAINING_SIZE):
          image_array = next_file()
          label = get_label()

          output = sum_of_weights(image_array)

          if label == "8":
              actual_output = 1
          else:
              actual_output = -1

          ERROR = actual_output - output

          if ERROR:
              num_incorrect += 1
          else:
              num_correct += 1

          update_weights(image_array, ERROR)

    # update radius based on new iteration
    history.append(num_incorrect)
    radius = (max(history) - min(history))
```

Learning Rate

The learning rate used is 0.1, which seems to be the best for this particular perceptron application. By looking at the average number of iterations needed until convergence, the best learning rate can be determined. The results found are shown below:

0.1	0.2	0.3	0.4
73.143	93.857	86.286	95.714

These are averages found over 7 trials of running the perceptron learning algorithm at each learning rate. 0.1 seems to be the best at 73.143 iterations for convergence on average. NOTE: these iteration counts were taken and averaged earlier when no minimum number of iterations was enforced. Currently, a minimum number of iterations of 100 is strictly enforced through a revised while loop condition:

```
while radius > threshold or min(history) > 0 or epoch < MAX_ITERATIONS:
```