Character Recognition using Perceptron Learning

Character Recognition using Perceptron Learning

David Leonard

Date: May 1, 2016

Contents

1 Introduction

In this problem, we are asked to use the Perceptron Learning rule to create a single-layer neural network which can recognize characters which are represented as 7 x 9 pixel inputs. We will discuss challenges with the algorithm and report on which characters were harder for the network to learn as additional letters were brought into the training set.

2 Architecture

The architecture is given for us within the supplied text - 63 neurons are used as inputs, represented as X_1, X_2, X_i, X_{63} where each input bit from our image is an input. These input neurons form our **input vector**, which consists of values ranging from -1 (pixel not located at that location), and +1 (pixel located at the corresponding location). For each output class (letter) which we would like to classify, we have an output neuron. In the base example we start with 7 output classes - A, B, C, D, E, J and K.

3 Results - Part A

The first task is to experiment with different threshold values and try to classify characters with varying degrees of noise (missing pixels, wrong data). All of the results discussed here from the simulations may be found in end of this report. Just from adding higher levels of noise, we can see that letters such as B and D are almost completely misclassified 100% of the time. In fact, with the initial training set of A, B, C, D, E, J and K, the neural net converges rather quickly. However, the addition of new letters Z, I, Y, S, Q and G cause the network to take roughly 100+ iterations to learn all the training patterns, which increases as the threshold is increased.

An interesting note is that classification of the letter S seems to do well compared to other letters when increasing the level of noise and the threshold, while letters such as Q consistently get misclassified (perhaps due to the other conflicting letters in the dataset). If the threshold is raised too high (10+), the network can no longer learn all of the training patterns and misclassifies everything 100% of the time. It is interesting to note that prior to adding new letters, this was not the case - the network would correctly learn all of the training patterns even when the threshold is 5, 10 or higher.

In order to gather these results, the network was trained a total of 100 times and then asked to classify the corresponding test data. Afterwards, the results were tabulated and statistics were taken to summarize the behavior and success of the network classifications.

4 Results - Part B

As stated in the previous section, certain letters such as Q are harder to classify - and often gets misclassified even with minimal amounts of noise. This is most likely due to how missing pixels on Q can create false positives for the letter C, O, and U. Another letter which is troublesome to learn is Z, which is most likely attributed to noise corrupting the middle of it. On average, the letters S and I tend to do well on average compared to the other characters when both the threshold and noise levels are high.

5 Source code

The entire source code can be found in it's entirely here.

```
from __future__ import division
from numpy.random import choice
from prettytable import PrettyTable
from numpy import array, dot, random, zeros, ones, all, any, array_equal, sum

def noise(arr, num):
```

```
  indices = choice(range(62), num, replace=False)
  for index in indices:
    # Toggle the value
    if arr[index] == -1:
      arr[index] = +1
    else:
      arr[index] = -1
  return arr

def report(trials, arr):
  percentages = []
  for index, val in enumerate(arr):
    percentages.append('{}%'.format(arr[index] / trials * 100))
  return percentages

# A, B, C, D, E, J, K, Z, I, Y, S, Q, G
training_data = [
  (array(
    [-1, -1, +1, +1, -1, -1, -1, -1,
     -1, -1, +1, -1, -1, -1, -1, -1,
     -1, +1, -1, -1, -1, -1, -1, +1,
     -1, +1, -1, -1, -1, -1, +1, -1,
     +1, -1, -1, -1, +1, +1, +1, +1,
     +1, -1, -1, +1, -1, -1, -1, +1,
     -1, -1, +1, -1, -1, -1, +1, -1,
     +1, +1, +1, -1, +1, +1, +1]),
  array([+1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1])),
  (array(
    [+1, +1, +1, +1, +1, +1, -1,
     -1, +1, -1, -1, -1, -1, +1,
     -1, +1, -1, -1, -1, -1, +1,
     -1, +1, -1, -1, -1, -1, +1,
     -1, +1, +1, +1, +1, +1, -1,
     -1, +1, -1, -1, -1, -1, +1,
     -1, +1, -1, -1, -1, -1, +1,
     -1, +1, -1, -1, -1, -1, +1,
     +1, +1, +1, +1, +1, +1, -1]),
  array([-1, +1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1])),
  (array(
    [-1, -1, +1, +1, +1, +1, +1,
     +1, +1, -1, -1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     -1, +1, -1, -1, -1, -1, +1,
     -1, -1, +1, +1, +1, +1, -1]),
  array([-1, -1, +1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, -1])),
  (array(
    [+1, +1, +1, +1, +1, -1, -1,
```

```
    -1, +1, -1, -1, -1, +1, -1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, +1, -1,
    +1, +1, +1, +1, +1, -1, -1]),
array([-1, -1, -1, +1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1])),
(array(
  [+1, +1, +1, +1, +1, +1, +1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, -1,
   -1, +1, -1, +1, -1, -1, -1,
   -1, +1, +1, +1, -1, -1, -1,
   -1, +1, -1, +1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, +1,
   +1, +1, +1, +1, +1, +1, +1]),
array([-1, -1, -1, -1, +1, -1, -1,
       -1, -1, -1, -1, -1, -1])),
(array(
  [-1, -1, -1, +1, +1, +1, +1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, +1, -1, -1, -1, +1, -1,
   -1, +1, -1, -1, -1, +1, -1,
   -1, -1, +1, +1, +1, -1, -1]),
array([-1, -1, -1, -1, -1, +1, -1,
       -1, -1, -1, -1, -1, -1])),
(array(
  [+1, +1, +1, -1, -1, +1, +1,
   -1, +1, -1, -1, +1, -1, -1,
   -1, +1, -1, +1, -1, -1, -1,
   -1, +1, +1, -1, -1, -1, -1,
   -1, +1, +1, -1, -1, -1, -1,
   -1, +1, -1, +1, -1, -1, -1,
   -1, +1, -1, -1, +1, -1, -1,
   -1, +1, -1, -1, -1, +1, -1,
   +1, +1, +1, -1, -1, +1, +1]),
array([-1, -1, -1, -1, -1, -1, +1,
       -1, -1, -1, -1, -1, -1])),
(array(
  [+1, +1, +1, +1, +1, +1, +1,
   +1, -1, -1, -1, -1, -1, +1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, +1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, +1, -1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, -1,
```

```
   +1, -1, -1, -1, -1, -1, +1,
   +1, +1, +1, +1, +1, +1, +1]),
array([-1, -1, -1, -1, -1, -1, -1,
       +1, -1, -1, -1, -1, -1])),
(array(
  [+1, +1, +1, +1, +1, +1, +1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   +1, +1, +1, +1, +1, +1, +1]),
array([-1, -1, -1, -1, -1, -1, -1,
       -1, +1, -1, -1, -1, -1])),
(array(
  [+1, -1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, +1, -1,
   -1, -1, +1, -1, +1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1,
   -1, -1, -1, +1, -1, -1, -1]),
array([-1, -1, -1, -1, -1, -1, -1,
       -1, -1, +1, -1, -1, -1])),
(array(
  [-1, +1, +1, +1, +1, +1, +1,
   +1, -1, -1, -1, -1, -1, -1,
   +1, -1, -1, -1, -1, -1, -1,
   +1, -1, -1, -1, -1, -1, -1,
   -1, +1, +1, +1, +1, +1, -1,
   -1, -1, -1, -1, -1, -1, +1,
   -1, -1, -1, -1, -1, -1, +1,
   -1, -1, -1, -1, -1, -1, +1,
   +1, +1, +1, +1, +1, +1, -1]),
array([-1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, +1, -1, -1])),
(array(
  [-1, +1, +1, +1, +1, +1, -1,
   +1, -1, -1, -1, -1, -1, +1,
   +1, -1, -1, -1, -1, -1, +1,
   +1, -1, -1, -1, -1, -1, +1,
   +1, -1, -1, -1, -1, -1, +1,
   +1, -1, -1, +1, -1, -1, +1,
   +1, -1, -1, -1, +1, +1, +1,
   +1, -1, -1, -1, +1, +1, +1,
   -1, +1, +1, +1, +1, +1, -1]),
array([-1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, +1, -1])),
(array(
  [-1, +1, +1, +1, +1, +1, -1,
```

```
      +1, -1, -1, -1, -1, -1, +1,
      +1, -1, -1, -1, -1, -1, +1,
      +1, -1, -1, -1, -1, -1, -1,
      +1, -1, -1, -1, -1, -1, -1,
      +1, -1, -1, +1, +1, +1, +1,
      +1, -1, -1, +1, -1, -1, +1,
      +1, -1, -1, -1, -1, -1, +1,
      -1, +1, +1, +1, +1, +1, -1]),
  array([-1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, +1])),
]

# Number of output classes
outputs = 13

# Number of times to train the network and test
trials = 100

# Noise level
noise_level = 15

# Results
classifications = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# Theshold for activation of neuron
threshold = 10

# Activation function
unit_step = lambda x: -1 if x < threshold else 1

# Initialize a matrix of weights (7 x 63)
w = zeros((outputs, 63))

# Initialize bias input vector for outputs
b = ones(outputs)

# Learning rate alpha
alpha = 0.2

# Iterations
iterations = 1000

# Iteration count
iteration_count = 0

for trial in xrange(trials):
  # Train over 1000 iterations
  for i in xrange(iterations):

    # Error count for classification
    error_count = 0

    # Iteration counter
    iteration_count += 1
```

```
  # Loop over training data
  for x, expected in training_data:

    # For each output neuron...
    for j in xrange(0, outputs):

      # Compute the total input for the jth output neuron
      result = dot(x, w[j]) + b[j]

      # Compute the output of the jth neuron
      output = unit_step(result)

      # Compute the error of the jth neuron
      error = expected[j] - output

      if error != 0:
        # Increment error count
        error_count += 1

        # Update the weight vector for the jth output neuron
        w[j] += alpha * x * error

        # Update the input bias for the jth output neuron
        b[j] += alpha * error

  if error_count == 0:
    print '{} {} {} : {}'.format('Converged in', str(iteration_count) +
        ' iterations', 'with weights', w)
    break

# Check if we can successfully classify our data
# for x, _ in training_data:
#   for j in xrange(0, 7):
#     result = dot(x, w[j]) + b[j]
#     print '{}: {} -> {}: {}'.format(j, result, unit_step(result), categories[j])

# Testing data with noise
testing_data = [
(noise(array(
  [-1, -1, +1, +1, -1, -1, -1, -1,
   -1, -1, +1, -1, -1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, +1, -1,
   +1, -1, -1, -1, +1, +1, +1, +1,
   +1, -1, -1, +1, -1, -1, -1, +1,
   -1, -1, +1, -1, -1, -1, +1, -1,
   +1, +1, +1, -1, +1, +1, +1]),
array([+1, -1, -1, -1, -1, -1, -1,
noise_level),
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [+1, +1, +1, +1, +1, +1, -1,
   -1, +1, -1, -1, -1, -1, +1,
```

```
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, +1, +1, +1, +1, -1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, -1, +1,
    +1, +1, +1, +1, +1, +1, -1]),
    noise_level),
array([-1, +1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [-1, -1, +1, +1, +1, +1, +1,
   +1, +1, -1, -1, -1, -1, +1,
   +1, -1, -1, -1, -1, -1, -1,
   +1, -1, -1, -1, -1, -1, -1,
   +1, -1, -1, -1, -1, -1, -1,
   +1, -1, -1, -1, -1, -1, -1,
   +1, -1, -1, -1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, -1, +1, +1, +1, +1, -1]),
    noise_level),
array([-1, -1, +1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [+1, +1, +1, +1, +1, -1, -1,
   -1, +1, -1, -1, -1, +1, -1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, +1, -1,
   +1, +1, +1, +1, +1, -1, -1]),
    noise_level),
array([-1, -1, -1, +1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [+1, +1, +1, +1, +1, +1, +1,
   -1, +1, -1, -1, -1, -1, +1,
   -1, +1, -1, -1, -1, -1, -1,
   -1, +1, -1, +1, -1, -1, -1,
   -1, +1, +1, +1, -1, -1, -1,
   -1, +1, -1, +1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, -1,
   -1, +1, -1, -1, -1, -1, +1,
   +1, +1, +1, +1, +1, +1, +1]),
    noise_level),
array([-1, -1, -1, -1, +1, -1, -1,
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [-1, -1, -1, +1, +1, +1, +1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, -1, +1, -1,
   -1, -1, -1, -1, -1, +1, -1,
```

```
    -1, -1, -1, -1, -1, +1, -1,
    -1, -1, -1, -1, -1, +1, -1,
    -1, +1, -1, -1, -1, +1, -1,
    -1, +1, -1, -1, -1, +1, -1,
    -1, -1, +1, +1, +1, -1, -1]),
    noise_level),
array([-1, -1, -1, -1, -1, +1, -1,
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [+1, +1, +1, -1, -1, +1, +1,
    -1, +1, -1, -1, +1, -1, -1,
    -1, +1, -1, +1, -1, -1, -1,
    -1, +1, +1, -1, -1, -1, -1,
    -1, +1, +1, -1, -1, -1, -1,
    -1, +1, -1, +1, -1, -1, -1,
    -1, +1, -1, -1, +1, -1, -1,
    -1, +1, -1, -1, -1, +1, -1,
    +1, +1, +1, -1, -1, +1, +1]),
    noise_level),
array([-1, -1, -1, -1, -1, -1, +1,
       -1, -1, -1, -1, -1, -1])),
(noise(array(
  [+1, +1, +1, +1, +1, +1, +1,
    +1, -1, -1, -1, -1, -1, +1,
    -1, -1, -1, -1, -1, +1, -1,
    -1, -1, -1, -1, +1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, +1, -1, -1, -1, -1,
    -1, +1, -1, -1, -1, -1, -1,
    +1, -1, -1, -1, -1, -1, +1,
    +1, +1, +1, +1, +1, +1, +1]),
    noise_level),
array([-1, -1, -1, -1, -1, -1, -1,
       +1, -1, -1, -1, -1, -1])),
(noise(array(
  [+1, +1, +1, +1, +1, +1, +1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    +1, +1, +1, +1, +1, +1, +1]),
    noise_level),
array([-1, -1, -1, -1, -1, -1, -1,
       -1, +1, -1, -1, -1, -1])),
(noise(array(
  [+1, -1, -1, -1, -1, -1, +1,
    -1, +1, -1, -1, -1, +1, -1,
    -1, -1, +1, -1, +1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
    -1, -1, -1, +1, -1, -1, -1,
```

```
      -1, -1, -1, +1, -1, -1, -1,
      -1, -1, -1, +1, -1, -1, -1,
      -1, -1, -1, +1, -1, -1, -1]),
      noise_level),
  array([-1, -1, -1, -1, -1, -1, -1,
         -1, -1, +1, -1, -1, -1])),
  (noise(array(
    [-1, +1, +1, +1, +1, +1, +1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     -1, +1, +1, +1, +1, +1, -1,
     -1, -1, -1, -1, -1, -1, +1,
     -1, -1, -1, -1, -1, -1, +1,
     -1, -1, -1, -1, -1, -1, +1,
     +1, +1, +1, +1, +1, +1, -1]),
     noise_level),
  array([-1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, +1, -1, -1])),
  (noise(array(
    [-1, +1, +1, +1, +1, +1, -1,
     +1, -1, -1, -1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, +1,
     +1, -1, -1, +1, -1, -1, +1,
     +1, -1, -1, -1, +1, +1, +1,
     +1, -1, -1, -1, +1, +1, +1,
     -1, +1, +1, +1, +1, +1, -1]),
     noise_level),
  array([-1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, +1, -1])),
  (noise(array(
    [-1, +1, +1, +1, +1, +1, -1,
     +1, -1, -1, -1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, -1, -1, -1, -1,
     +1, -1, -1, +1, +1, +1, +1,
     +1, -1, -1, +1, -1, -1, +1,
     +1, -1, -1, -1, -1, -1, +1,
     -1, +1, +1, +1, +1, +1, -1]),
     noise_level),
  array([-1, -1, -1, -1, -1, -1, -1,
         -1, -1, -1, -1, -1, +1])),
]

  # Training index
  index = 0

  # Attempt to classify noisy data
  for x, expected in testing_data:
    predicted = []
    for j in xrange(0, outputs):
```

```
        result = dot(x, w[j]) + b[j]
        predicted.append(unit_step(result))

      if not array_equal(array(expected), array(predicted)):
        classifications[index] += 1

      # Increment training data index
      index += 1

      # print 'Expected: {}, Predicted: {}, Match: {}'.
                format(expected, predicted, predicted == expected)

statistics = report(trials, classifications)

table = PrettyTable()
x = PrettyTable(['Total Trials', 'Theshold', 'Noise', 'Class',
     'Misclassification Count', 'Misclassification Percentage'])
x.padding_width = 1
x.add_row([trials, threshold, noise_level, 'A', classifications[0], statistics[0]])
x.add_row([trials, threshold, noise_level, 'B', classifications[1], statistics[1]])
x.add_row([trials, threshold, noise_level, 'C', classifications[2], statistics[2]])
x.add_row([trials, threshold, noise_level, 'D', classifications[3], statistics[3]])
x.add_row([trials, threshold, noise_level, 'E', classifications[4], statistics[4]])
x.add_row([trials, threshold, noise_level, 'J', classifications[5], statistics[5]])
x.add_row([trials, threshold, noise_level, 'K', classifications[6], statistics[6]])
x.add_row([trials, threshold, noise_level, 'Z', classifications[7], statistics[7]])
x.add_row([trials, threshold, noise_level, 'I', classifications[8], statistics[8]])
x.add_row([trials, threshold, noise_level, 'Y', classifications[9], statistics[9]])
x.add_row([trials, threshold, noise_level, 'S', classifications[10], statistics[10]])
x.add_row([trials, threshold, noise_level, 'Q', classifications[11], statistics[11]])
x.add_row([trials, threshold, noise_level, 'G', classifications[12], statistics[12]])
print x
```