Letter Reconstruction

Evolutionary Computing A412

Henry Thomas - htthomas@alaska.edu

Dakota Crowder - dcrowder2@alaska.edu

April 30, 2018

Proposal

What is the problem? Why is it an interesting and/or important problem to solve? Who would be interested in its solution?

Our problem is to create handwritten letters using an Artificial Neural Network. Neural networks can be adjusted to create letters that provide the expected output, however the results often look nothing like numbers to the human eye. The objective here is to create letters that could be recognized by a human as well as by a neural network using the randomness of the Genetic Algorithms. Programmers and security experts that use neural networks to identify things like cars, people and faces could use this information to help better their efforts.

What representation of candidate solutions is most suitable, and why? What type of fitness function can be used to evaluate the quality of a candidate solution? What values for control parameters do you intend to use, and why?

The most suitable representation of candidate solutions is GA, we can use it to create a bit string representation of handwritten letters on a pixel by pixel level. We can also use crossover and mutation to modify the strings so that they converge on an output which will be recognizable by both humans and neural networks.

Our fitness function will be fed through a neural network that uses 20,000 examples of handwritten letters to adjust its weights and biases. The network will have 26 output neurons that representing each letter of the alphabet as a result. The objective is to have the ‘brightest’ output neuron correspond to the letter that was passed in to the neural network, and all other output neurons to be ‘dim’. This state would represent that the network is confident that the input is a specific letter, and not any other letter.

Control parameters will be fairly standard, with some possible modifications later on if needed. Expected values are a Generation Size of 100 individuals, a Generation Count of 100-1000, probability of crossover of 95%, and probability of mutation of 1-5%.

Source Code Listing

format.py - Preprocessing, reformats the original dataset to a usable one

bitstrings.py - Preprocessing, generates and stores bitstrings for each letter image in the dataset (stores in ‘bitstrings.txt’)

average_letterdata.py - Parses the stored feature vectors and classifications in ‘bitstrings.txt’

letter_reconstruction.py - The main file, implements the genetic algorithm with recombination and mutation

ann.py - Implementation of the artificial neural network, stores weights and biases to propagate inputs through the network to classify letters from a bitmap

weights.txt - The stored model of the trained neural network

Summary of Outcome

We were able to generate images using our genetic algorithm, however they ended up not looking like handwritten letters, instead they looked more like random static. The genetic algorithm outputs a 2500 character bitstring with 0’s and 1’s, each representing a black or white pixel respectively, going row by row. The resulting image is then generated from this bitstring.

We classified the initial dataset using our artificial neural network from our machine learning course, training the network to take in a handwritten letter image and classify it as one of the 26 possible letters. After running the network on our formatted dataset overnight, it arrived at 77% classification accuracy, which was a little less than we hoped for. It is still acceptable, however the accuracy being in the 90% range would have yielded us better results overall.

We think that the generated images are seemingly random due to the fitness function not valuing ‘chunks’ of black pixels highly enough. The input image bit-strings have more compacted ‘sets’ of ones and zeros, which you would expect for human readability.

Interestingly though, the neural network is confident that these ‘random’ images are in fact their associated letters.

Improvements and Extensions

One potential improvement would be to reduce the ‘noise’ that is seen on the output images. The algorithm would need to be more selective to generate images that actually look like handwritten letters. Also, our fitness function produced very small fitness values (10^-15) which may have confused the neural network. It’s possible there were underflow issues because of the small values as well.

Pattern recognition could be implemented as part of the fitness function, to look for chunks of sequences of zeros and ones and assigning those a high fitness value, so the bit-strings would be structured similarly to the input images. We implemented a simple function called ‘outliers’ to look for 1’s in the bitstring (black pixels) that had no neighboring black pixels, but this didn’t seem to change the output images much when added to the fitness function.

Another thing to consider was that it took a decent amount of time to calculate each generation, which would be made even worse trying to implement a more advanced fitness function. We think it has to do with the fact that the bit-strings are 2500 characters long, it takes a while to iterate through the entire string and look for patterns, or whatever else we decided to add to the fitness function.