Optimizing Digit Recognition with DEAP and SKLEARN

Asutosh Dhakal, worked with Kalpana Vaidya

May 1, 2017

Abstract

In this experiment, we explored neural networks and genetic algorithms to be able to recognize handwritten digits. The data set was acquired from *Kaggle*, with 28 x 28 sized images of handwritten digits. We optimized various hyper parameters for the *MLPClassifier* from the *sklearn* module in Python. We used *DEAP* to optimize the number of hidden layers to optimize the performance of the neural network and the accuracy of digit recognition. After careful analysis, we concluded that a hidden layer of size 52-54 produced the most optimal results.

1 Introduction

Our project deals with a subfield of Computer Vision– handwriting recognition. We broke this problem into a smaller subset by exploring only digits, 0 - 9. Using a two data sets, one for training and one for testing, we are analyzing pixels and classifying images as a certain digit, 0 through 9.

There are several ways to classify a handwritten digit as a number from 0 - 9. However, they all involve key components which include the implementation of a neural network, a training period, and tweaking several hyper parameters. Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradient based learning technique.[2] In any sort of machine learning or artificial intelligence, the key to being successful is having the right kind of data.[1] In our case, we have a large dataset of pixel values for thousands of images. As such, we transformed that data into very large matrices for analysis and used a neural network library (*sklearn*) and an evolutionary algorithm to find optimum accuracy. This report will include the methodology we followed, as well as show our results with various trials and tweaking.

2 Methodology

We used *DEAP*, an evolutionary algorithm library for python, to optimize the number of hidden layers the *MLPClassifier* should take when computing the values for the neural network. DEAP uses an evolutionary algorithm approach to find optimal solutions to problems.

As such, we altered various hyper parameters and created classes.

Here is the implementation and set up we had for *DEAP*:

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
population_size = 20
toolbox = base.Toolbox()
toolbox.register("bit", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.
toolbox.register("population", tools.initRepeat, list, toolbox.individual, n=
toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=.5)
CXPB, MUTPB, n_gen = .5, .2, 20
pop = toolbox.population()
toolbox.register("select", tools.selTournament, tournsize=3)
```

Additionally, we created our own algorithm for looping through the individuals for every generation. We used a *cxTwoPoint* cross over technique. To keep track of the best fitness values and to graph the results, we saved the *bestInd* for every generation like so:

```
for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit
        sumFit = sumFit + fit[0]
        if(fit[0] > bestInd):
            bestInd = fit[0]

    averageFit[g] = float(sumFit)/len(offspring)
    bestFit[g] = bestInd
```

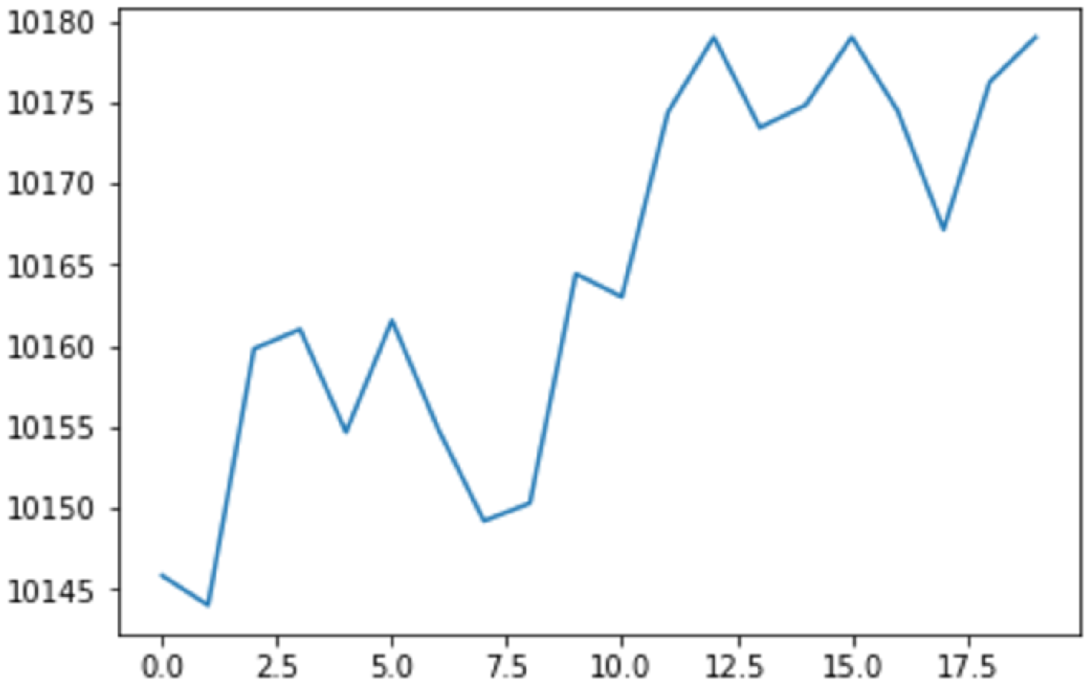
For every iteration through the evaluation function, we used the *sklearn* neural network library that essentially created a confusion matrix of size 10 x 10 for the ten digits, 0 - 9. The matrix would be represented as something like this:

$$ConfusionM = \begin{bmatrix} 1035 & 0 & 4 & 2 & 3 & 2 & 6 & 3 & 7 & 1 \\ 1 & 1163 & 4 & 1 & 3 & 2 & 0 & 4 & 3 & 1 \\ 3 & 3 & 956 & 8 & 5 & 2 & 1 & 13 & 11 & 3 \\ 1 & 1 & 14 & 1036 & 2 & 15 & 2 & 12 & 19 & 6 \\ 4 & 4 & 0 & 0 & 946 & 0 & 8 & 2 & 3 & 18 \\ 10 & 2 & 4 & 25 & 4 & 881 & 1 & 2 & 5 & 6 \\ 9 & 0 & 11 & 1 & 9 & 8 & 998 & 0 & 6 & 0 \\ 0 & 3 & 14 & 11 & 4 & 1 & 0 & 1059 & 7 & 12 \\ 2 & 15 & 6 & 13 & 1 & 8 & 5 & 2 & 962 & 7 \\ 5 & 4 & 1 & 12 & 25 & 7 & 0 & 15 & 9 & 965 \end{bmatrix}$$

The way we optimized the results was to reward individuals (hidden layer amount) who created the least confusion, i.e the sum of the diagonal of the matrix was maximized.

3 Results and Discussions

As mentioned above, our *DEAP* algorithm used a population size of 20 and iterated through 20 generations. The graph below displays the best average fitness value and the generation number.



As can be seen in the graph, our evolutionary algorithm improved the sum of the diagonals of the confusion matrix. While there were some strange dips and disparities in the graph, there is a general upward trend from generation to generation. In the end, our highest sum resulted in 10,179 when the hidden layer size was 54. The various discrepancies in the graph resulted from the random integers that the *DEAP* module got for the individuals, so some outliers were present.

4 Conclusions

There were many ways we could have approached this problem of optimizing the accuracy of a digit recognition neural network. The size of the hidden layer in a neural network is something that has been debated since it's conception. But usually there are several factors when deciding the number and size of a hidden layer including the number of input and output nodes, amount of training data available, complexity of the function that is trying to be learned, as well as the training algorithm. We optimized the size of the parameter for the size of the hidden layer in the *MLPClassifier* method from *sklearn* by using an evolutionary algorithm provided by DEAP and tweaked by us, as we used our own algorithm to go through an optimize each generation with crossover and mutation functions. In the end, we found that the most optimal hidden layer size is 54. There are many ways we could have changed our experiment or altered it. We could have used stochastic gradient descent, or other logistic regression models to optimize the solution.

References

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2002.

[2] S. Benzoubeir, A. Hmamed, H. Qjidaa, *"Hypergeometric Laguerre moment for handwritten digit recognition"*, Multimedia Computing and Systems 2009. ICMCS '09. International Conference on, pp. 449-453, 2009.