Computational Intelligence **Due: 27/05/2022**

Submission Assignment 5

Instructor: JakubM. Tomczak Name: Vanshita Sharma Kumar, NetId[2687732]

1 Introduction

In the field of AI, the branch of evolutionary computing aims to create algorithms that emulate evolutionary processes. “The idea is to let the computer evolve solutions to problems rather than trying to “calculate” them”[leiden]. Evolutionary algorithms evolve these solutions by applying nature inspired processes such as selection, mutation and recombination among a population of individuals [1]. The respective sequence of steps correspond with one another to converge from a randomly generated population to a population which produces the best individuals in each generation. With the application of evolutionary algorithms we are able to diversify and apply these algorithms to train neural networks.

Neural networks, which are also known as artificial neural networks (ANN or CNN) are a subset of machine learning and are extensively used in deep learning algorithms. The name and structure of neural networks is designed and inspired by the procedures of the brain. Neural networks mimic the biological process of neurons sending signals to one another in order to perform an action. ANNs are composed of node layers of input data, weights, a bias (or threshold), and an output. ANNs rely on training data to learn and improve their accuracy [2]. For example, if we apply ANNs on our respective sklearn digits library, we can continuously train the network and create more accurate predictions as the network is constantly learning, hence making it a powerful tool for classification.

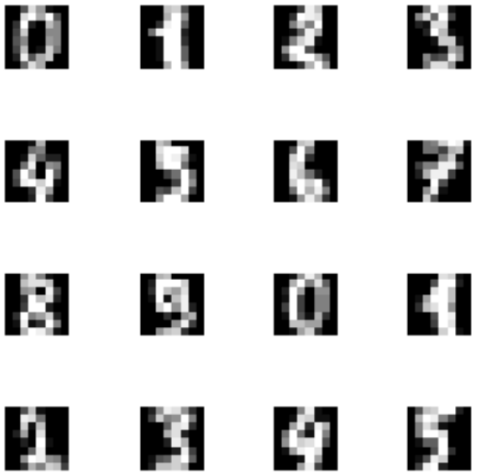

Figure 1. Visual Representation of Our Digits Dataset

Overtime, the development in AI research gives the opportunity to work with neural networks and EA algorithms together, therefore, in the respective report, I aim to apply evolutionary algorithms for selecting the best structure of a neural network.

2 Problem statement

In the respective assignment, I aim to tackle the problem of creating a neural network which can be optimized with an evolutionary algorithm. Because this problem involves two completely unique applications, I will first start by explaining the role of the neural network.

Initially the neural network is trained on a dataset of small natural images with multiple classes, in this case we are applying the proposed CNN structure to the sklearn digits library, where we classify small images (8px x 8px) of handwritten digits to one of 10 classes. In order to create these classifications, the network is trained using a negative log likelihood function. The negative log likelihood function is a cost function applied to our machine learning model, giving insight and predictions on the performance of the model. These predictions are calculated as such [4],

$$l(\theta) = -\sum_{i=1}^{n}(y_i \cdot log \cdot \hat{y}_{\theta,i} + (1 - y_i) \cdot log \cdot (1 - \hat{y}_{\theta,i}))$$

Where $\hat{y}_{\theta,i}$ is the predicted probability of the ith data, and 1 - $\hat{y}_{\theta,i}$ is the negative probability of the ith data point, and because this is an application of binary classification, y either takes values 0 or 1, hence for each index i we are either adding the log of $\hat{y}_{\theta,i}$ or subtracting 1 - $\hat{y}_{\theta,i}$ [4]. Over time the loss is minimized, where the distinction between the predicted label and actual class have become as small as possible.

Since the EA algorithm implements a recombination operator, a mutation operator, and selection mechanisms, and analyzes behavior, the algorithm does not depend on the networks' classification confidence but rather its classification accuracy, which is our classification errors created as output from our network after training. In EA algorithms, as mentioned before apply natural processes, a mix of potential solutions to a problem is populated randomly at first. Then the population tests for a fitness value, which later chooses the fittest individuals for reproduction. The cycle begins again as the fitness of the population is evaluated and the least fit individuals are eliminated. In the second part of our problem, our EA algorithm takes the fitness values from the network. However, in order to pass our fittest value we have to adapt our objective function. Because the respective evolutionary process favors individuals with a better fitness value, the EA algorithm finds the best architecture to optimize the neural network. As aforementioned, we are interested in adapting our objective function, the objective function f of the EA, is the fitness from our network (CE) which is produced by the accurate classification of images from the validation dataset.

$$objective(x) = ce + \lambda\frac{N_p}{N_{max}}$$

$$where\ ce\ is\ the\ classification\ \ error\ produced\ by\ our\ CNN$$

With the respective objective function, will guide the EA algorithm in finding the best CNN structure that is well adjusted to the objective function with the application of classifying with low errors and maintaining the minimal number of weights.

3 Methodology

To approach our problem, it is easy to break it up into steps. Our first step is to first create our new fitness function which will help us extract the classification error. In the respective assignment we are advised to follow this structure for our network.

Conv2d → f(.) → Pooling → Flatten → Linear 1 → f(.) → Linear 2 → Softmax

Within this structure, there are a number of hyperparameters which represent a variety of choices that can be passed through the EA. Starting with our Conv2d, apply a 2D convolutional "over an input signal composed of several input planes"[5]. In our Conv2d we pass the arguments stride, padding and kernel. Stride controls the stride for the cross-correlation, often having a value of 1. Padding controls the amount of padding applied to the input." It can be either a string {'valid', 'same'} or a tuple of ints giving the amount of implicit padding applied on both sides" [5]. And finally the kernel size which is the size of the convolving kernel, which can be passed the value of tuple (3,3) or (5,5).

Next we have our activation function, denoted by f(.), defines how the weighted sum of the respective convulsions input is transformed into an output in a layer of the network, there are various activation functions applied provided as values, such as nn.ReLu, nn.Tanh, nn.ELU, etc.

We then have our pooling layer, the pooling layer of the network aims to progressively reduce the spatial size of the representation, which in result reduces the amount of parameters and computations in the network, as it operates on each feature map independently. In our respective code we are given the choice between Average and MaxPool2D. Where Max pooling takes the best features in a m*m matrix, and average pooling takes the average value of size m*m, taking only the average features.

We next have our linear layer, the linear layer is used in the last stages of the network, bringing it all together. The linear layer helps in changing the dimensionality of the previous layer so that the model is then able to define the relationship of the values presented by the data on what the model is working on.

And finally we use our Softmax layer, The negative log-likelihood in practice is applied in tandem to the softmax activation function. The log softmax activation function is applied to multiclass learning problems where a set of features can be related to one-of-K classes.[3] The output of log softmax describes the probability (confidence level) of our neural network that a certain sample belongs to a certain class, giving us an understanding of how accurate our predictions are. It is commonly calculated as such,

$$LogSoftMax(x_i) = log(\frac{exp(x_i)}{\sum_j exp(x_j)})$$

All these respective layers can be represented as a table, illustrating all the various possible combinations which can be applied by the EA in order to find the optimized architecture. This table can be visualized below, to give an idea of what our network can look like.

Nr of filters, p(1)	$0<p(0)<2$, $p(1) \in \mathbb{N}$
Kernel size, padding and stride, p(2)	$0<p(1)<1$, $p(2) \in \mathbb{N}$
The activation functions, denoted as f(.), p(3)	$0<p(2)<4$, $p(3) \in \mathbb{N}$
Max/avg pooling, p(4)	$0<p(3)<1$, $p(4) \in \mathbb{N}$
Number of nodes/neurons for linear, p(5)	$0<p(5)<9$, $p(5) \in \mathbb{N}$

Table 1, The Various Choices Which can be Passed in our Network

The table above illustrates all possible combinations for each respective layer, from the number of filters in the Conv2D layer, to the number of nodes/neurons in our linear layer. The second part of the methodology explores the EA class. The EA class is where we take our classification error as a fitness value and finds the optimized architecture for our network.

For mutation, I adapted mine from assignment 3. It is still a single point mutation. It loops through x_children and in the for loop generates a random value from 0 to 4, if the value is less than the max value, for example 0, then we increase the value if i[j] by 1, otherwise if we randomly select the last value, we subtract i[j] by 1. This mutation procedure, based on the biological point mutation, is called single point mutation. Mutation introduces randomness to the sample of x_children in EA, making it beneficial in most circumstances to converge to a better solution faster.

For my selection mechanism I applied the same one from assignment 3, which was the roulette wheel selection. Because it weights the objects individually according to the fitness, it illustrates that the instances of the fitter x_parents being selected, have a higher likelihood. Roulette is a non deterministic way of selecting in the long term, since no trait that might help in a few generations get eliminated early on.

Finally I use survivor selection with respect to each individual which is the product of x_parents1 and x_parents2, and is evaluated based on the objective function (objective) as a phenotype. These individuals are then ranked based on the phenotype and the better half is passed down to the next generation. The approach compares both parents and takes the better performing individuals.

4 Plots and Discussion

When we run our EA overtime the algorithm will find an optimal CNN structure. Once the computations of the number of generations is done, the best genotype vector is returned. This vector illustrates the best possible combination for the respective CNN architecture as it gives us the lowest fitness value, which means the best fit individual in every generation. So the final value produced, which for example could be Generation: 9, best fitness: 0.02878 corresponds to the best architecture (represented as a vector), that is [1 0 0 0 0]. Below I can visually see the various architectures presented as vectors and their respective fitness values.

```
Generation: 0, best fitness: 0.040274927395934176
Generation: 1, best fitness: 0.03736357350297331
Generation: 2, best fitness: 0.03168026552344074
Generation: 3, best fitness: 0.0316492877886876
Generation: 4, best fitness: 0.0316492877886876
Generation: 5, best fitness: 0.028784400497856452
Generation: 6, best fitness: 0.028784400497856452
Generation: 7, best fitness: 0.028784400497856452
Generation: 8, best fitness: 0.028784400497856452
Generation: 9, best fitness: 0.028784400497856452
FINISHED!
[[1 0 0 0 0]
 [1 0 0 1 0]
 [1 0 0 0 0]
 [1 0 0 1 0]
 [1 0 0 1 1]
 [1 0 0 0 1]] [0.0287844  0.0287844  0.03164154 0.03164154 0.03164929 0.03164929]
```

Figure 2, The Optimal Architecture Structure Proposed from EA

References

[1] https://studiegids.universiteitleiden.nl/courses/98776/evolutionary-algorithms

[2] https://www.ibm.com/cloud/learn/neural-networks

[3] https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

[4] https://towardsdatascience.com/cross-entropy-negative-log-likelihood-and-all-that-jazz-47a95bd2e81

[5] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html