Computational Intelligence 2021 **(Due: 27/5/2022)**

Submission Assignment 5

Instructor: Jakub M. Tomczak *Name:* Zahra Moradi, *Netid:* zmi450

1 Introduction

Deep learning has attracted the interest of many machine learning researchers in the recent years and has achieved notable results in various domains such as computer vision [3], speech recognition [8], video games [13] and many more artificial intelligence (AI) applications . The design of the architecture of neural networks (NNs), the optimization and hyperparameter settings that are used in training are key factors that immensely impact the performance of the neural network. The search for the best architecture is therefore of great interest of many AI projects [6]. Evolutionary methods are of interest in particular as they are inspired by the evolutionary biology theory of Charles Robert Darwin - an English naturalist, geologist and biologist - and simulate basic principles of life. This papers aims to explore evolutionary approaches in finding the best architecture for a neural network. To explain further, the evolutionary algorithm aims to minimize a differentiable function f(x) on the domain x where f(x) corresponds to the the error of the neural network regarding the task and x corresponds to input data.

This method is also refereed to as "Neuroevolution", seeking to evolve neural networks by means of evolutionary algorithms[5]. Artificial Neural Networks (ANNs) are inspired by the structure of the human brain, and they try to strengthen the connections and improve the performance of the neurons of the model by evolving the attributes (namely weights and biases) and the structure of connections which yields better results. In this paper, we specifically focus on convolutional neural networks (CNNs) with the task of classifying handwritten digits. We consider a specific function f(x) that corresponds to the number of wrongly classified images by the neural network and x corresponds to input images. There are several configurations of each building block of a convolutional neural network that need to be determined before the initialization of an experiment. These configurations include - but not limited to - the following: the size of the input layer (input neurons), output layer and hidden layers: the kernel size, pooling method (average pooling or max pooling), padding and stride settings, etc. The evolutionary algorithm intends to explore these options and determine the most efficient configuration where the performance of the CNN is maximized.

2 Problem statement

The objective of our CNN is to classify handwritten digits. In particular, the task of the model is to predict the probability of the most likely label (to see which classification label is the most probable). We can then denote the objective as $p(x_i) = log(\frac{exp(x_i)}{\sum_{j=1}^{k} exp(x_j)})$ where $p(x)$ is the most probable label, x_i is an element for a class i (the input we want to classify), and k is the number of all the classes. We can then define the negative log-likelihood for inputs x_i that we want to classify with target y_i (which is the index of the correct class of the input in range 1 to k and a number of k classes as $L = -\sum_i^k y_i * -p(x_i)$ and since $y_i = 1$ for the correct class and $y_i = 0$ for other classes for a specific class i, we can write the negative log-likelihood for each input as $nn.NLLLoss = -p(x_{i,k})$. If "reduction=sum", then $TotalLoss = -\sum_i^k -p(x_{i,k})$, and if "reduction=avg" then we have $TotalLoss = -\frac{1}{\sum_1^k w_k}(-p(x_{i,k}))$ where w_k is the vector of assigned weights to each of the classes [12].

The model is trained on a data set of images for a small number of epochs by means of supervised learning. To further clarify, the model is given an input x_i for classification. The model then calculates the probability $p(x_i)$ that this input belongs to each of the k classes. The model outputs the labels with the highest probability as the predicted label of the given input. this output is then compared with its true labels and the model is evaluated based on the its classification error rate (CE) over all of the inputs of the validation set. This value is used as a indication for the EA to determine the fittest individuals in the following steps.

The population is a collection of different CNN models that aim to classify the inputs. The EA seeks to detect the model with the lowest classification error. However, each model receives a penalty based on the number of weights it holds. To be precise, the EA calculates the fitness of each generation based on the

$Fitness = CE + \lambda * \frac{N_P}{N_{max}}$, where N_P is the number of weights of the current mode, N_{max} is the number of weights of the largest possible neural network in the current search space (population) and λ is hyper parameter (in this paper, we take $\lambda = 0.01$).

There are some difficulties that need to be considered when conducting such experiment. The evolutionary algorithm contains different operations of selecting the parents from the population based on a fitness function, the reproduction of children from these parents, the mutation of the children, and the survival selection for the population of the next generation. Since each individual in the population is a CNN, the fitness calculation may take some time depending on the number of the neurons and the complexity of the architecture of the CNN (e.g. the calculations of back propagation, activation of layers, etc.) [9]. This in turn leads to a long optimization process since each generation is evaluated based on the fitness function and processing images is time-consuming. Using a graphics processing unit (GPU) could help decrease this time, and is highly recommended for higher resolution inputs. We do not use a GPU in this case since the inputs are 8*8 pixels and do not need high processing power.

Another problem is that this optimization task is not convex, and there are no analytical forms available to compute the exact solution for the classification task. This means that there does not exist one solution with the best value (global minimum) for the objective function [2]. In addition, population based method offer good approximate solutions to problems that cannot be solved by analytical forms and it could be even more intensive computationally to find the exact solution for the classification task [15]. For this reason, the evolutionary algorithm (which is a population based method) is used as it offers near-optimal solutions and approximate the CNN architecture with better performance in each generation.

It is important to mention that even though careful considerations and prior research has been taken into account to overcome these obstacles, random factors such as noises in the data, time limitations and measurement errors (e.g. poor choice of hyperparameters) have an effect on the overall results. However, these misconceptions are common in a scientific research to a fault and it is important to be aware of them in the final conclusion.

3 Methodology

3.1 CNN General Design

We first start by determining the general design of the CNN. The CNN is constructed by the following layers of $Conv2D \rightarrow Activation \rightarrow Pooling \rightarrow Flatten \rightarrow Linear1 \rightarrow Activation \rightarrow Linear2 \rightarrow Softmax$. This layer design stays unchanged for all the individuals of the population. The difference between individuals are determined by difference in the settings of each layer. To explain in more detail:

- For the $Conv2D$ layer which is the input layer of the neural network, we have different choices of "8", "16" and "32" filters. In addition, for the kernel size, padding and stride of the $Conv2D$ we have two choices of
  - "kernel=(3,3), stride=1, padding=1"
  - "kernel=(5,5), stride=1, padding=2"
- For the *Activation* layers which are applied to the layer of the neural network, we have different choices of "ReLU", "sigmoid", "tanh", "softplus" and "ELU" activation function [10].
- For the *Pooling* layer, we have different choices of:
  - Size of "2*2" and "1*1" (identity) [1].
  - Method of average pooling and max pooling [14]
- For the *Linear* layers, we have different choices of number of neurons 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

3.2 Data Collection

The dataset consists of handwritten digit images from the Sklearn Datasets "$load_digits$". This dataset includes 10 classes, 1 for each digit and each digit has its corresponding label. The task of the CNN population is to classify an input from the testing dataset with as accurately as possible. Each CNN model is trained for a small number of epochs before it is evaluated by the EA algorithm. Since image processing is very time consuming, 5 epochs was determined before the start of the experiment as the hyperparameter for training. The dataset is

divided into three parts of 1000 training, 350 validation, and 447 testing images. The images are then batched by using the Dataloader with 64 images in each batch. The validation set is further used for the evaluation of individuals in the EA algorithm.

3.3 Population Generation

We generate the population of CNNs by choosing from the offered options randomly. We then propose the calculations for each operation of the evolutionary algorithm. There are 4500 possible configurations for a CNN model based on the proposed general design. To represent these configurations, we store the attributes of each individual in a list with a predefined order to better access these settings in future adjustments. The order of stored attributes are as follows: The filter size of the CNN, The kernel+padding size of the CNN, the pooling size of the pooling layer, pooling type of the pooling layer, the method of the first activation layer, the method of the second activation layer and the number of neurons of the output of first linear layer. The numerical configurations (e.g. number of neurons, sized, etc.) are stored as an integer number, the descriptive configurations (e.g. pooling methods, etc.) are represented by strings and the module configurations (e.g. activation methods, etc) are stored as a "torch.nn.module" in the representative list.

3.4 Evolutionary Algorithm

The evolutionary algorithm then takes the population of CNNs, their number of weights and their fitness values as input. It also requires the training function, the training Dataloader (batches) and the validation Dataloader to evaluate the next generation of each iteration. The structure of the EA and the choices are explained in more detail in the comments of the submitted code.

3.4.1 Fitness Function

First, we formalize an approach to determine the fitness function in order to evaluate the individuals in the population. For this purpose, we the classification error of each model (the rate of the wrongly classified input in a dataset) in addition to a penalty for the number of weights in the model (the size of the model structure). The fitness is the calculated as $\gamma = CE + \lambda * \frac{N_P}{N_{max}}$, where $\frac{N_P}{N_{max}}$ is proportion of the model's weights to the largest possible model in the population. Given the formula and the purpose of the CNNs, the objective of the EA is then to minimize the value of γ and find a CNN with the lowest fitness possible.

3.4.2 Parent Selection

The parent selection is done by picking individuals that have the lowest fitness (since we are minimizing the objective). We first initialize an empty list that will contain our parent population. We then select an individual with the lowest "γ" in the current population. In addition, we keep track of their fitness for survival selection in later stages. We then replace their fitness in the original fitness list to a "+infinity" value to avoid duplication in selecting the other parents. This process is repeated for "θ" times and each individual is appended to parent population.

3.4.3 Reproduction

In the next step, we randomly pick 2 parents "P_a" and "P_b" from the parent population for the reproduction. The child is then generated by using the "one-point crossover" approach. To explain in more detail, we store the setting of each configuration of the CNN in a list. When the parents are selected, we take these two lists and generate a new list by taking the "one-point crossover" of them [16]. We repeat the reproduction process by an arbitrary number of "κ" to generate the children.

3.4.4 Mutation

For mutation, we access the configuration list of a child and randomly change one value in the CNN settings (e.g. the number of neurons in the "*Linear*" layers, The kernel+padding size, etc.). This new child is then trained for an arbitrary number of epochs=5 and evaluated based on the fitness function. These finesses are further used in the survival selection process.

3.4.5 Survival Selection

We establish the survival selection by taking a subset of the best parents with the generated children. This method is referred to as "elitism" and is a typical scheme applied in similar EA studies [4].

4 Experiments

4.1 Hyperparameters And Experiment Setup

The goal of the experiment is to determine the best CNN architecture given the general design of the model that performs with a higher accuracy (lower classification error) on our testing image of the digits dataset. However, several hyperparameters need to be set prior to the start of the experiment to ensure reliability. These hyperparameters include the population size "$N = 100$", the number of generations $\alpha = 100$, the number of parents in parent selection $\theta = 6$, the number of generated the children for the selected parents "$\kappa = N - \theta$", the number of epochs for the training "$epochs = 5$", the learning rate of "$lr = 0.001$", the weight decay value of "$wd = 0.00001$", the lambda value for penalty calculation of "$\lambda = 0.01$" and an arbitrary number of epochs to stop the training after no progress is made of "$MaxPatience = 20$". These settings are considered in order to improve the performance of the model based on prior research and experimental intuition.

We use the "Adamax" optimizer for the model. "Adamax" is a variant of the "Adam" optimizer with the difference that "Adam" scales the individual weights' gradients using the L2-norm of the gradients, while "Adamax" uses the infinite-order norm. The motivation behind using "Adamax" is that it makes the model more robust to noise in the gradients (this optimizer is usually preferable for sparsely updated parameters) [11]. Additionally, we use a weight decay for the regularization in the optimizer settings. The motivation behind using this method is to prevent overfitting. In addition, we also apply "early stopping" by setting a "patience" limit for the model to stop the training if the loss is not improved for an arbitrary number of $epochs = 3$.

4.2 Final CNN Selection

After finalizing the configurations, we will run the EA algorithm on the initial population and monitor its behaviour. This architecture is displayed in figure 1. The best fitness of each 10 generation (e.g. generations 10,20,30,etc.) are tracked and printed to better comprehend the generation's progress. At the end of the final generation, the architecture with the best fitness is chosen. This model is then trained from scratch for a number of 1000 epochs with an early stopping method (if the loss does not improve much after 20 epochs, the training is stopped to avoid overfitting) and evaluated on the testing image. The results are then compared with the previous model in assignment 4 to finalize the experiment.

```
best individual of the last generation (overall) is: ClassifierNeuralNet(
  (LogSoftMax): LogSoftmax(dim=1)
  (nll): NLLLoss()
  (classcnn): Sequential(
    (0): Reshape()
    (1): Conv2d(1, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (2): Tanh()
    (3): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1, ceil_mode=False)
    (4): Flatten()
    (5): Linear(in_features=2048, out_features=70, bias=True)
    (6): ELU(alpha=1.0)
    (7): Linear(in_features=70, out_features=10, bias=True)
    (8): LogSoftmax(dim=1)
  )
)
```

Figure 1: The architecture of the best model chosen by the EA algorithm.

5 Results and discussion

5.1 Plots and tables

We run the EA algorithm for 100 iterations (generations) and store the best fitness and individual model for further analysis. The EA convergence to the best fitness gradually which is displayed in plot 2. The final results of the generations finesses are displayed in table 1.

Generation	**B**est Fitness
0	0.07549528059388431
10	0.04508690973150452
20	0.04508690973150452
30	0.04508690973150452
40	0.0421977660457757
50	0.0421977660457757
60	0.0421977660457757
70	0.0421977660457757
80	0.0421977660457757
90	0.0421977660457757
100	0.0421977660457757

Table 1: Best fitness values of each generation.

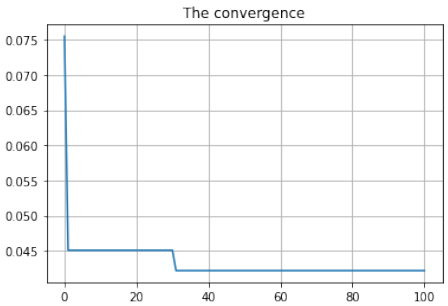

Figure 2: The convergence of the EA algorithm with regards to the fitness value).

It is evident from these results that the EA algorithm identifies the best model with the architecture 1 in the 100^{th} generation and this model has a final fitness $ce + penalty = 4.2\%$, which is the classification error considering the number of weights of the model. The final loss and classification error of this model on the testing data is equal to $nll = 0.35\%$ and $ce = 7.6\%$. Since we previously designed a MLP and a CNN model in assignment 4 for the digit dataset, we can compare the newly identified model respectively. The MLP model had a loss and classification error of $nll = 0.5\%$ and $ce = 9.6\%$, and the MLP model had a loss and classification error of $nll = 0.31\%$ and $ce = 5.3\%$. It can be concluded from the results that the CNN model of assignment 4 has the best results overall, and the model chosen by the EA algorithm comes in second place. The numerical comparison between the models are displayed in table 2 and the plot comparison between the models are displayed in figure 3.

Model	**F**inal Loss	**F**inal Classification Error
CNN Assignment 5	0.0035668942885644216	0.07606263982102908
CNN Assignment 4	0.003172011700119215	0.053691275167785234
MLP Assignment 4	0.005835608431290194	0.09619686800894854

Table 2: Final results of models on the testing dataset.

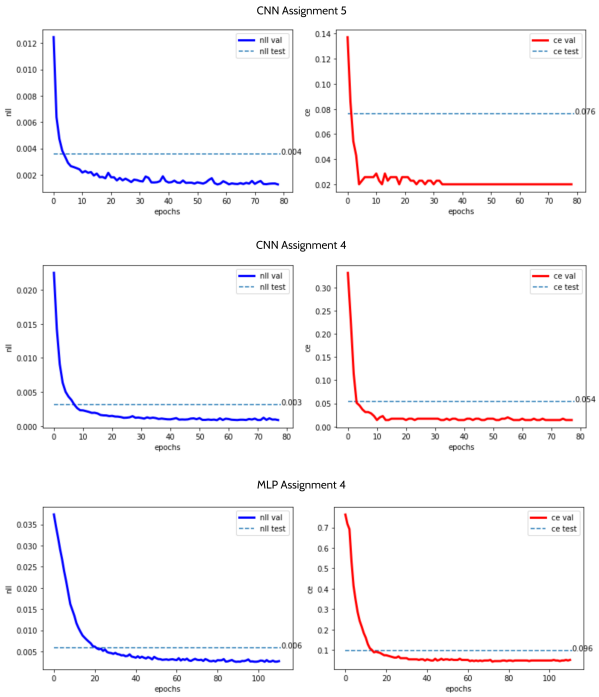

Figure 3: The plots for the loss and classification error of all models).

In addition, we plot the confusion matrix of the CNN model of assignment 5 based on the results from the testing dataset in figure 4.

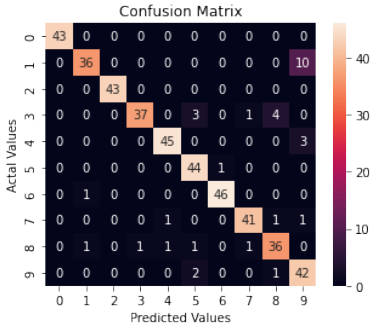

Figure 4: The confusion matrix of the best model chosen by the EA algorithm on the testing dataset.

5.2 Conclusion and future works

It is important to keep in mind that there are some advantages and disadvantages that need to be considered when conducting such experiment:

- Disadvantages: the EA algorithm identifies the best architecture by maintaining the general design of $Conv2D \rightarrow Activation \rightarrow Pooling \rightarrow Flatten \rightarrow Linear1 \rightarrow Activation \rightarrow Linear2 \rightarrow Softmax$. Therefore, there may exist better architecture that deviate from this general design and perform better than the chosen model. This is evident from table 2 since the CNN model from assignment 4 has a different general design and has a lower classification error on the testing data. Another disadvantage is that the fitness of the models depend on the number of epochs that they are trained for. Due to time and CPU power limitations, we chose a small number of 5 epochs for training and the EA may have been able to select a better solution if this number was greater (This is merely a hypothesis and needs to be further investigated to clarify). In addition, our dataset contains very small images of 1*8*8 pixels and the EA algorithm will take much longer to complete if the images have bigger size.
- Advantages: If we were to test different architectures to find the best model, the experiment would take very long since there exists 4500 possible configurations for this general design. However, by using the EA

algorithm we are able to achieve our goal very fast (since the current dataset images are small in size). In addition, this method could be adjusted to find the best architecture for different inputs by merely changing the general design in the primary function (it is important to keep in mind that the running time depends heavily on the size of images) and importing different datasets accordingly.

The experiment could be further optimized by numerous changes such as increasing the size of each training/validation/testing datasets, increasing the training epochs in the EA algorithm, adjusting the general design by adding/removing convolution and linear layers, changing the mutation/recombination/survival selection techniques in the EA and etc. Additional, there are several methods of data prepossessing which has proven to be effective and it can enhance the performance of the CNN greatly in future experiments [7].

References

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186. URL https://ieeexplore.ieee.org/abstract/document/8308186.

[2] A. L. Blum and R. L. Rivest. *Training a 3-node neural network is NP-complete*, pages 9–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993. ISBN 978-3-540-47568-2. doi: 10.1007/3-540-56483-7_20. URL https://doi.org/10.1007/3-540-56483-7_20.

[3] T. H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. PCANet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, 24(12):5017–5032, dec 2015. doi: 10.1109/tip.2015.2475625. URL https://ieeexplore.ieee.org/document/7234886v.

[4] H. Du, Z. Wang, W. Zhan, and J. Guo. Elitism and distance strategy for selection of evolutionary algorithms. *IEEE Access*, 6:44531–44541, 2018. doi: 10.1109/ACCESS.2018.2861760.

[5] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008. doi: 10.1007/s12065-007-0002-4. URL http://infoscience.epfl.ch/record/112676.

[6] E. Galván and P. Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *CoRR*, abs/2006.05415, 2020. doi: 10.1109/TAI.2021.3067574. URL https://arxiv.org/abs/2006.05415.

[7] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1:1–22, 2016. doi: https://doi.org/10.1186/s41044-016-0014-0. URL https://bdataanalytics.biomedcentral.com/articles/10.1186/s41044-016-0014-0#citeas.

[8] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013. doi: 10.1109/ICASSP.2013.6638947. URL http://arxiv.org/abs/1303.5778.

[9] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[10] J. Lederer. Activation functions in artificial neural networks: A systematic overview. *CoRR*, abs/2101.09957, 2021. URL https://arxiv.org/abs/2101.09957.

[11] R. Llugsi, S. E. Yacoubi, A. Fontaine, and P. Lupera. Comparison between adam, adamax and adam w optimizers to implement a weather forecast based on neural networks for the andean city of quito. In *2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6, 2021. doi: 10.1109/ETCM53643.2021.9590681.

[12] L. J. Miranda. Understanding softmax and the negative log-likelihood. *ljvmiranda921.github.io*, 2017. URL https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[14] M. Sun, Z. Song, X. Jiang, J. Pan, and Y. Pang. Learning pooling for convolutional neural network. *Neurocomputing*, 224:96–104, 2017. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2016.10.049. URL https://www.sciencedirect.com/science/article/pii/S092523121631290.

[15] G. Wahba. Practical approximate solutions to linear operator equations when the data are noisy. *SIAM Journal on Numerical Analysis*, 14(4):651–667, 1977. doi: 10.1137/0714044. URL https://doi.org/10.1137/0714044.

[16] F. A. Zainuddin, M. F. Abd Samad, and D. Tunggal. A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. *International Journal of Advanced Science and Technology*, 29(6s):759–769, 2020.