HYPER-PARAMETER TUNING IN DEEP NEURAL NETWORK LEARNING

Tiffany Zhan

USAOT, Las Vegas, Nevada, USA

ABSTRACT

Deep learning has been increasingly used in various applications such as image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series. In deep learning, a convolutional neural network (CNN) is regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The full connectivity of these networks makes them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include penalizing parameters during training or trimming connectivity. CNNs use relatively little pre-processing compared to other image classification algorithms. Given the rise in popularity and use of deep neural network learning, the problem of tuning hyper-parameters is increasingly prominent tasks in constructing efficient deep neural networks. In this paper, the tuning of deep neural network learning (DNN) hyper-parameters is explored using an evolutionary based approach popularized for use in estimating solutions to problems where the problem space is too large to get an exact solution.

KEYWORDS

Deep Learning, Convolutional Neural Network, Deep Neural Network Learning, Hyper-Parameters.

1. DEEP LEARNING

In deep learning, a convolutional neural network (CNN) is a class of artificial neural network (ANN), most commonly applied to image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series [1-10]. CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks makes them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme [11, 12].

Convolutional networks were inspired by biological processes with relatively little pre-processing compared to other image classification algorithms [13]. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage. A CNN consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a CNN, the input is a tensor with a shape: (number of inputs) × (input height) × (input width) × (input channels). After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape: (number of inputs) × (feature map height) × (feature map width) × (feature map channels). Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus [14]. Each convolutional neuron processes data only for its receptive field. Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2 × 2 are commonly used. Global pooling acts on all the neurons of the feature map [15]. Each neuron in a neural network computes an output value by applying a specific function to the input values received from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning consists of iteratively adjusting these biases and weights.

2. HYPER PARAMETERS

Constructing an efficient DNN for given applications is not a trivial task. It involves significant domain knowledge and efforts in exploring the properties of the data. For example, how sparse the data is, how many training or test samples are available, the definition of the data and it's types, and the data representation power [16]. The goal of DNN is to be able to represent the available data as precisely as possible while avoiding the problem of over-fitting, thus constructing an efficient DNN should fit the data but not over-fit the data, which means that time must be spent determining when the data is being over-fit and when it is not. Fortunately, DNNs have several parameters that define their overall structure. These parameters are referred to as hyper-parameters in that they are used to define the structure of the DNN rather than are parameters to be used by the DNN [17].

There are many parameters to define DNNs, for example, the number of layers in the DNN, the number of nodes within a given layer, the algorithms used between layers, the overall algorithm used for the network, the optimization techniques [18] involved, the activation functions, the number of epochs, the size of batches, the number of folds, etc. Since there are variety of hyper-parameters, a critical question is what hyperparameters to be used for given problems. Exploring possible configurations of hyper-parameters will significantly impact the results. To complicate matters even further those parameters are often only reasonably transferable to another problem if the problems themselves can be mapped into the same problem space. Even if we may reasonably transfer some parameters to similar problems, there are often unique qualities within the datasets that differentiate them to a degree where one DNN configuration is not necessarily the most efficient for the next DNN configuration. The hyper-parameter tuning has been often based on intuition rather than scientific rationale because one experience with a set of parameters does not necessarily directly translate into another researcher's experience. This paper will explore the scientific evolutionary principles using genetic algorithms for hyper-parameter tuning.

3. Genetic Algorithms

Genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection [19, 20]. In a genetic algorithm, a population of candidate solutions (called phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible [20, 21]. The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is recombined or possibly randomly mutated to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. A typical genetic algorithm involves a genetic representation of the solution domain, a fitness function to evaluate the solution domain. Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

The initial population in GA is generated randomly, allowing the entire range of possible solutions (the search space). During each successive generation, a portion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. The fitness function is defined over the genetic representation and measures the quality of the represented solution. Genetic operators include Crossover (genetic algorithm) and Mutation (genetic algorithm). Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms [22, 23, 24, 25, 26]. This generational process is repeated until a termination condition has been reached. Common terminating conditions include a solution is found that satisfies minimum criteria, fixed number of generations reached, allocated budget (computation time/money) reached, the highest-ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results.

The best individual is kept and by the end of the GA run the best individual is the one with the optimal result. In this case optimal refers to the fitness function of the individual, which is the function that defines how good an individual is. Extrapolating this towards DNN we can then say that if we can express an individual as the set of hyper-parameters of a DNN then the optimal DNN configuration is the one which evaluates to the best outcome. The best outcome for a DNN is the one that achieves the best accuracy and the best learning rate. By treating the parameters that define a DNN as an optimization problem we map this problem to a Genetic Algorithm to find the best possible result within a given time.

4. Research Problem

During the confirmation of DNNs, we need to determine which parameters to use. In this context, I capture a small subset of the parameters in a DNN and attempt to optimize them. It is possible to model a DNN with much greater complexity but in order to verify the results it will take much longer than the computational resources available to complete.

The research problem is defined as follows: using a DNN and given datasets, can a GA be used to generate better results in a reasonable time period? Here a reasonable period refers to under one hour of time and the parameters explored are the batch sizes, number of layers, nodes within layers, and activations used within layers.

5. ALGORITHMIC SOLUTION

Since the number of possible combinations of these parameters is extremely large, a upper bound has been placed on these values in order to comply with the requirement that a result can be found within a reasonable time period. The activations used are those available within the Python Scikit-Learn framework. Although some activations are probably unnecessary there was no filtering done to prevent this. Instead, the GA identifies these as being poor performers and filter them out of the results. The batch sizes could potentially rise to half of the entire dataset but these instead were reduced to a more reasonable range. The reasonable range was determined experimentally ahead of time by noting that extremely large values would often result in extremely poor accuracy while at the same time extremely smaller batch sizes would runs slowly and result in poor accuracy.

Algorithm A shows the procedure to build and run a GA. Algorithm B shows the procedure of creating and running a model.

Algorithm A

```
Population Initialization
    while generations < 25 do
        for all individuals in population
                do Calculate fitness
        end for
        select N best fitness individuals to create population
        for all individuals in best do
            if random < Mutation Probability then
                mutate individual
            end if
        end for
        for all individual A, individual B in best do
            if random < Crossover Probability then
                crossover individual A, individual B
            end if
        end for
        generations++
    end while
    output best individual
```

Algorithm B

```
CREATEMODEL (individual)
    model = Sequential
    add input layer to model
    for all layers in individual do
        if random < Layer Probability then
```

```
            add random layer to model
        end if
    end for
    add output layer to model
    return model

RUNMODEL (individual)
    $train, test \quad data.split()$
    compile model from train, test
    $model \quad CreateModel(individual)$
    run model
    return model accuracy
```

6. EXPERIMENTS

The DNN and GA are implemented using Python Scikit-Learn framework on Windows. I used the Distributed Evolutionary Algorithms in Python (DEAP) which is an evolutionary computation framework for rapid prototyping and testing of ideas [27]. It incorporates the data structures and tools required to implement most common evolutionary computation techniques such as genetic algorithm, genetic programming, evolution strategies, particle swarm optimization, differential evolution, traffic flow [27, 28, 29, 30] and estimation of distribution algorithm. The benchmarks dataset MNIST and CIFAR-10 are used, which make use of categorical cross entropy as the loss function as well as the adam optimizer. For the GA, the population was limited to 25 in order to limit the computational running time.

Both datasets show a clear advantage to using the GA as the best accuracy achieved was better than on a basic CNN with no hidden layer. The best performing GA had around batch size of 100 whereas the worse performing had a batch size of 500. This indicates that the batch size is closely related to the data itself rather than a generic value and is a great candidate for hyper-parameter tuning.

7. CONCLUSION

The genetic algorithm is beneficial to be used for the purposes of hyper-parameters tuning in deep neural network learning. It can enhance the chances to identify an efficient architecture as well as optimize the performance to discover the best parameters for a given problem. Instead of researching what the current best solution is, it is wise to allow the genetic algorithm to do the work instead. Rather than a human applying an educated guess at what will perform better, the genetic algorithm will do the same thing without requiring additional effort to review the results. It is therefore a good approach to tuning hyper-parameters, particularly in a domain where there is not sufficient intuitive knowledge. However, the genetic algorithm suffers from high computational costs and in some cases is not realistic if computational resources are limited.

REFERENCES

[1] Valueva, M., Nagornov, N., Lyakhov, P., Valuev, G., Chervyakov, N. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. Mathematics and Computers in Simulation. Elsevier BV. 177: 232–243.

[2] Zhang, W. (1988). Shift-invariant pattern recognition neural network and its optical architecture. Proceedings of Annual Conference of the Japan Society of Applied Physics.

[3] Avilov, O., Rimbert, S., Popov, A. Bougrain, L. (2020). Deep Learning Techniques to Improve Intraoperative Awareness Detection from Electroencephalographic Signals. The 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). Montreal, QC, Canada: IEEE. 2020: 142–145.

[4] Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A. (2017). Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks. 2017 IEEE 19th Conference on Business Informatics (CBI). Thessaloniki, Greece: IEEE: 7–12. doi:10.1109/CBI.2017.23. ISBN 978-1-5386-3035-8. S2CID 4950757.

[5] Matusugu, M., Katsuhiko, M., Yusuke, M., Yuji, K. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network (PDF). Neural Networks. 16 (5): 555–559. doi:10.1016/S0893-6080(03)00115-1. PMID 12850007. Retrieved 17 November 2013.

[6] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press. p. 326.

[7] Habibi, H. (2017). Guide to convolutional neural networks: a practical application to traffic-sign detection and classification. Heravi, Elnaz Jahani. Cham, Switzerland. ISBN 9783319575490. OCLC 987790957.

[8] Venkatesan, R., & Li, B. (2017). Convolutional Neural Networks in Visual Computing: A Concise Guide. CRC Press. ISBN 978-1-351-65032-8.

[9] Ciresan, D., Meier, U., Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition. New York, NY: Institute of Electrical and Electronics Engineers (IEEE). pp. 3642–3649.

[10] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. Nature. 521 (7553): 436–444. Bibcode:2015 Natur.521..436L. doi:10.1038/nature14539. PMID 26017442. S2CID 3074096.

[11] LeCun, Y., Bengio, Y. (1995). Convolutional networks for images, speech, and time series. In Arbib, Michael A. (ed.). The handbook of brain theory and neural networks (Second ed.). The MIT press. pp. 276–278.

[12] Patrick, L. Viard-Gaudin, C., Barba, D. (2006). A Convolutional Neural Network Approach for Objective Video Quality Assessment (PDF). IEEE Transactions on Neural Networks. 17 (5): 1316–1327. doi:10.1109/TNN.2006.879766. PMID 17001990. S2CID 221185563. Retrieved 17 November 2013.

[13] Viebke, A., Memeti, S., Pllana, S., Abraham, A. (2019). CHAOS: a parallelization scheme for training convolutional neural networks on Intel Xeon Phi. The Journal of Supercomputing. 75 (1): 197–227. arXiv:1702.07908. doi:10.1007/s11227-017-1994-x. S2CID 14135321.

[14] Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. 1: 1097–1105 – via ACM.

[15] Haotian, J., Zhong, L., Qianxiao, L. (2021). Approximation Theory of Convolutional Architectures for Time Series Modelling. International Conference on Machine Learning. arXiv:2107.09355.

[16] Passos, D., & Mishra, P. (2022). A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks. Chemometrics and Intelligent Laboratory Systems, 104520.

[17] Gonzales-Martínez, R., Machacuay, J., Rotta, P., & Chinguel, C. (2022). Hyperparameters Tuning of Faster R-Cnn Deep Learning Transfer for Persistent Object Detection in Radar Images. IEEE Latin America Transactions, 20(4), 677-685.

[18] Thavasimani, K. & Srinath, N. (2022). Optimal Hyperparameter Tuning using custom genetic algorithm on deep learning to detect twitter bots. Journal of Engineering Science and Technology, 17(2), 1532-1549.

[19] Gerges, F., Zouein, G., Azar, D. (2018). Genetic Algorithms with Local Optima Handling to Solve Sudoku Puzzles. Proceedings of the 2018 International Conference on Computing and Artificial Intelligence. ICCAI 2018. New York, NY, USA: Association for Computing Machinery: 19–22. doi:10.1145/3194452.3194463. ISBN 978-1-4503-6419-5. S2CID 44152535.

[20] Ting, C. (2005). On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection. Advances in Artificial Life: 403–412. ISBN 978-3-540-28848-0.

[21] Deb, K. & Spears, W. (1997). C6.2: Speciation methods. Handbook of Evolutionary Computation. Institute of Physics Publishing. S2CID 3547258.

[22] Patrascu, M., Stancu, A.F., Pop, F. (2014). HELGA: a heterogeneous encoding lifelike genetic algorithm for population evolution modeling and simulation. Soft Computing. 18 (12): 2565–2576. doi:10.1007/s00500-014-1401-y. S2CID 29821873.

[23] Srinivas, M., Patnaik, L. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms (PDF). IEEE Transactions on System, Man and Cybernetics. 24 (4): 656–667. doi:10.1109/21.286385.

[24] Zhang, J., Chung, H., Lo, W. L. (2007). Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms. IEEE Transactions on Evolutionary Computation. 11 (3): 326–335. doi:10.1109/TEVC.2006.880727. S2CID 2625150.

[25] Fraser, A. & Burnell, D. (1970). Computer Models in Genetics. New York: McGraw-Hill. ISBN 978-0-07-021904-5.

[26] Fogel, D. B., ed. (1998). Evolutionary Computation: The Fossil Record. New York: IEEE Press. ISBN 978-0-7803-3481-6.

[27] Fortin, F., De Rainville, F., Gardner, F., Gagné, C., Parizeau, M. (2012). DEAP: Evolutionary Algorithms Made Easy. Journal of Machine Learning Research. 13: 2171–2175.

[28] Gonzales-Martínez, R., Machacuay, J., Rotta, P., & Chinguel, C. (2022). Hyperparameters Tuning of Faster R-Cnn Deep Learning Transfer for Persistent Object Detection in Radar Images. IEEE Latin America Transactions, 20(4), 677-685.

[29] Shankar, K., Kumar, S., Dutta, A. K., Alkhayyat, A., Jawad, A. J. A. M., Abbas, A. H., & Yousif, Y. K. (2022). An Automated Hyperparameter Tuning Recurrent Neural Network Model for Fruit Classification. Mathematics, 10(13), 2358.

[30] Elhoseny, M., Metawa, N., Sztano, G., & El-Hasnony, I. M. (2022). Deep Learning-Based Model for Financial Distress Prediction. Annals of Operations Research, 1-23.