August 4th 2023
Institute For Computing In Research

Understanding Neural Network Architecture Using Evolutionary Algorithm Search

Ammar Mukadam
Supervised By Cole Agard

Abstract

Neural networks have revolutionized computer processing and analysis, enabling them to 'learn' using a multi-layer model. These models process input data, gradually training their internal layers by modifying their parameters to output accurate results. This paper focuses on optimizing the network's aptitude to 'learn' before it starts training by modifying the hyperparameters that govern its architecture. We then use an evolutionary algorithm search, a method that essentially replicates evolution in nature, to find the optimal combination of these hyperparameters.

1 Neural Network Background

Neural networks represent a fascinating example of how computers learn. They can take in a wide range of information and fulfill many tasks, including identification, classification, and predictions, producing remarkable results. For example, neural networks can excel at image recognition, taking in an image's pixels and classifying it as a specific object.

Neural networks are structured with multiple rows of neurons, with each neuron from the previous row intricately linked to all others in the following row. Input data is received into the first (input) layer, and the final (output) layer outputs the result or prediction, with the actual "learning" being done in the layers sandwiched between the two - the hidden layers. These layers act as the brain, processing the input and learning to identify the patterns that connect the input into meaningful output.

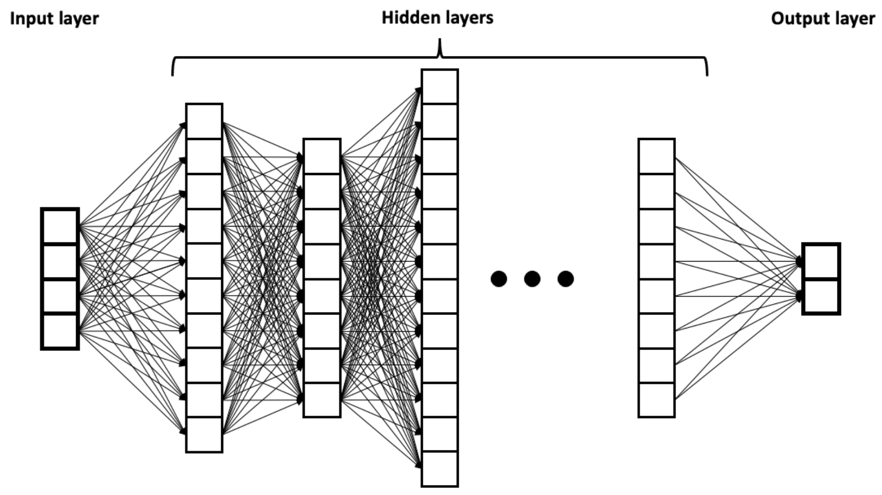

Figure 1: Neural Network Diagram
Source: Wikimedia Commons

This "learning" process is facilitated by the neural network's parameters: weights and biases. Weights determine the value of each neural connection to the final output, while bias represents the threshold for a neuron to activate based on the input it receives. This adds an extra level of customizability to each neuron's response, making the network capable of capturing complex patterns and relationships in the data [Nielsen (2015)].

To ensure the neural network is actively getting more accurate, we employ a loss function, essentially a way of understanding the network's progress towards consistently outputting the expected result. Using back-propagation, we backtrack from the output to the input neurons to identify how much each parameter contributed to the error. This is then used to determine the specific changes in the weights and biases needed to lower the loss, which is facilitated through an optimization algorithm called gradient descent. It minimizes the loss function by slowly modifying the parameters as the model trains, eventually leading to a trained neural network.

2 Network Architecture

Neural architecture focuses on the neural network structure; most critically, you may have noticed the arbitrary nature of the hidden layers. How many layers are in the hidden layers? How many neurons are in each of these layers? Additionally, the many other unique hyperparameters that govern neural networks before they even begin learning. The answer is that finding the optimal structure of the neural network varies greatly on the network. This architecture has a critical impact on the efficiency and accuracy of these neural networks, however, finding the optimal structure can be computationally expensive and difficult.

One common method for finding these hyperparameters is grid search, which represents the hyperparameter choices as a multi-dimensional grid and tests points sequentially throughout this grid. However, with every increasing hyperparameter, the number of necessary grid points increases exponentially, making comparing all these points unfeasible.

Several other methods have also been utilized to find the optimal architecture for each unique neural network, including relaxed problem space, traditional search, and Bayesian optimization. However, this paper's focus will be on evolutionary algorithm search. Before we dive deeper into the evolutionary algorithm search, it's important to focus on the specific hyperparameters we aim to optimize.

- **Number of Layers and Neurons:** The most evident hyperparameter of the evolutionary algorithm search is the number of layers and neurons per layer in the hidden layer. Too few layers/neurons and the neural network will not be able to capture intricate patterns, while too many of either can lead to overfitting the data, in which the model memorizes the data instead of learning.
- **Activation Function:** Another crucial hyperparameter is the network's activation function. The sum of all the weights and activations from a neuron's connections and its own bias is put into this activation function, which determines that neuron's activation. The activation functions we utilize are ReLU (Rectified linear unit), sigmoid, tanh (Hyperbolic Tangent), and SELU (Scaled Exponential Linear Units), each with different properties that allow the perceptrons to learn the nonlinearity in the dataset. For example, given the same input value, the sigmoid function will output a value between zero and one. In contrast, the tanh function will output a value between negative one and one as it follows a tangent function. The choice of activation functions is crucial and should be based on the network's task and desired output.
- **Learning Rate:** The learning rate determines the rate at which the network adjusts its weights in response to the results of its simulations. If it is set too high, the network will overstep the global minima of optimization in the loss function, and contrastingly if it is too low, the network will be stuck at a local minimum.
- **Dropout Layers:** Dropout layers are an essential hyperparameter to prevent overfitting the data. They work by excluding specific portions of neurons during the learning process, forcing different areas of neurons to learn independently. This encourages the network to understand and learn from the data rather than simply memorizing it [Lawrence et al. (1997)].

- **Cost Function:** The cost function is similar to the loss function, but instead of using it to evaluate each training example, it is used to evaluate the average of this loss function through all of the training examples. Common cost functions include mean squared error, which calculates the average square difference between the predicted and expected results [Chapados & Bengio (2001)]. Additionally, there is mean squared error which calculates the absolute value of these differences instead of squaring them. The choice of cost function depends on the neural network's task and the prominence of its errors. [Leung et al. (2003)]

3 Evolutionary Algorithm Search

The evolutionary algorithm search essentially replicates "survival of the fittest" in nature, where through evolution, the most optimal topology for the neural network will be found. Three distinct phases define the search.

Firstly, *initialization* creates a population of individuals, each with a chromosome that represents specific hyperparameters for the neural network's topology. Next, in the *selection* phase, individuals are mated, and their chromosomes are crossed over, resulting in new individuals with unique hyperparameters. Crucially, this process includes a dynamic chance of mutation where one of the hyperparameters in these individuals may change, increasing diversity to avoid being stuck at a local minimum of optimization.

These individuals' neural networks are then evaluated on data, which provides each individual's fitness. This evaluation process is facilitated through the package TensorFlow, which automatically creates the neural network with the individual's hyperparameters and evaluates its fitness. The nature of this fitness function is up to the user and the neural network's specific task, but it can focus on accuracy, F1 score, minimizing loss, completion time, or other metrics. This adds an extra layer of customizability, allowing us to search for hyperparameters that best fit our network's intended use case [Lambora et al. (2019)]. The specific fitness function used for a neural network aids in selecting the best architectures for further breeding. Finally, in the *termination* stage, the individuals are ranked by their fitness, and those at the bottom are deleted. The process is then repeated for a chosen number of generations, ultimately finding a near-optimal topology for that neural network.

However, there is still one process to be analyzed - the unique hyperparameters that govern the evolutionary algorithm search. How many individuals should start the simulation? How many generations to run? Additionally, what should be the mutation rate, death percentage, and number of parents mating? These genetic search hyperparameters significantly impact the search's efficiency and effectiveness, just as the neural network's hyperparameters influence its learning process. A narrower search can lead to premature convergence to a less optimal solution, while a search that is too broad is computationally expensive and lengthy. By tweaking these hyperparameters, we can find a suitable middle ground to set up the best search, therefore finding the best architecture and thus the best neural network for our task.

The current model is still a proof of concept, and the chosen search hyperparameters have been proven by researchers to be effective for small datasets. However, they are not guaranteed to find the optimal solution. For more complex datasets, using more starting individuals, increasing the number of generations, and reducing the death rate could likely produce better results. In the future, the model aims to implement Bayesian optimization, a black box optimization method that builds a probability model to learn from its guesses and find the optimal hyperparameters for the search. We plan to use the method to find a set of specific search hyperparameters that can be generalized and used for a large number of datasets.

4 Conclusion

Let's backtrack and take an overall view of our process. We optimize the hyperparameters of the genetic search, including the number of generations, mutation rate, etc., to find the best individual through this evolution simulation. This individual holds the key to the optimal architecture for our neural network, such as the number of hidden layers, learning rate, etc. Once we apply this hyperparameter architecture to our network, it is best set up to complete our task, and we have successfully optimized the network to learn.

5 Acknowledgements

Thank you to my mentor Cole Agard for directing me through the process and providing me access to his work. His guidance was invaluable in learning neural networks and in completing this project. Additionally, thank you to The Institute for Computing In Research for providing the structure and resources to complete the project.

References

Chapados, N. & Y. Bengio. 2001. Cost functions and model combination for var-based asset allocation using neural networks. *IEEE Transactions on Neural Networks*

12(4). 890–906. doi:10.1109/72.935098.

Ding, Shifei, Chunyang Su & Junzhao Yu. 2011. An optimizing bp neural network algorithm based on genetic algorithm. *Artificial intelligence review* 36. 153–162.

Lambora, Annu, Kunal Gupta & Kriti Chopra. 2019. Genetic algorithm- a literature review 380–384. doi:10.1109/COMITCon.2019.8862255.

Lawrence, Steve, C Lee Giles & Ah Chung Tsoi. 1997. Lessons in neural network training: Overfitting may be harder than expected 540–545.

Lee, Sanghyeop, Junyeob Kim, Hyeon Kang, Do-Young Kang & Jangsik Park. 2021. Genetic algorithm based deep learning neural network structure and hyperparameter optimization. *Applied Sciences* 11(2). 744.

Leung, F.H.F., H.K. Lam, S.H. Ling & P.K.S. Tam. 2003. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks* 14(1). 79–88. doi:10.1109/TNN.2002.804317.

Nabi, Javaid. 2019. Hyper-parameter tuning techniques in deep learning. *Towards Data Science* .

Nielsen, Michael A. 2015. Neural network and deep learning. *Determination Press* .

Yoo, YoungJun. 2019. Hyperparameter optimization of deep neural network using univariate dynamic encoding algorithm for searches. *Knowledge-Based Systems* 178. 74–83. doi:10.1016/j.knosys.2019.04.019.