Sara Boyd
Lauren Gillespie
Elyssa Sliheet
May 1, 2019

Neural Architecture Search

Abstract

Neural architecture search is a new, popular technique for automatically discovering successful deep neural architectures for a wide variety of domains. Neural architecture search has been implemented with many different types of search algorithms, including Q-learning and evolutionary strategies. We attempt to implement neural architecture search using a simple mu + lambda evolutionary strategy to optimize accuracy on the popular Reuters and MNIST datasets. We evolve architectures with significantly better performance than the baseline architecture and comparable performance to hand-optimized architectures of similar size. In future work, we hope to also evolve the layers of the networks beyond the hyperparameter tuning we currently perform.

Introduction

Deep neural networks have recently made huge strides in achieving human-level performance on a wide range of previously intractable tasks such as image recognition and sentiment analysis.[1, 2] However, what drives much of this success is network architecture, as can be seen in the success of the VGG16 network family.[1] Network architecture of deep neural networks is defined by the number, type, and arrangement of neuron layers in the network, along with their respective hyperparameters.[3] Many successful deep learning models, such as VGG16 and LeNet are hand-designed architectures derived by experts, but the sheer number of parameters found in increasingly deep neural structures makes hand-tuning networks manually increasingly infeasible.[1, 4]

One increasingly popular alternative is neural architecture search, where successful architectures are discovered automatically.[5] Neural architecture search has been successful in a wide range of domains in discovering novel, successful architectures for tasks such as image recognition and text classification[q learning, ea paper] Based on the success of these previous approaches, this work attempts to automatically discover good networks for two popular deep learning datasets, the Reuters newswire and MNIST image datasets, using a simple evolutionary algorithm.[6, 4, 7] These datasets are of particular interest, as the Reuters dataset is a classic AI benchmark dataset from the 90's, while MNIST is a more recent, more complex image recognition dataset that has been a benchmark for state-of-the-art deep networks in recent years.[8, 4]

Our goal is not state-of-the-art performance but rather a proof of concept for small networks in a relatively simple domain with limited compute resources. We wanted to show significant improvement in network accuracy on these datasets over a default feedforward structure and also comparable behavior to known successful architectures. Ultimately, we were able to do just that. In a few generations and with a small population, we discovered architectures with significant performance increases over default architectures through hyperparameter modification with an N +1 evolutionary algorithm.[7] We saw a significant increase in accuracy over the default in both the Reuters and MNIST dataset, and accuracy comparable to known architectures for the Reuters dataset.

Background/Related Work

With the explosion of interest in neural architecture search in recent years, there are countless examples of successful neural architecture search. We choose to focus on two specific methods of optimization, reinforcement learning and evolutionary search. These specific models are of interest as we have experience with them from in class and their implementations are relatively simple and easy to compare.

Reinforcement Learning

Baker et. al. uses reinforcement learning, specifically a Q-learning method called MetaQNN, to find an optimal convolutional neural network architecture.[8] In this work, the goal is to construct a novel Q-learning agent to discover CNN architectures that perform well on a given machine learning task with no human intervention. The learning agent is given the task of sequentially picking layers of a CNN model. The space which the agent searches is a finite but large space of model architectures. The agent learns through random exploration and slowly begins to exploit its findings to select higher performing models using the ε- greedy strategy. The agent receives the validation accuracy on the given machine learning task as the reward for selecting an architecture. MetaQNN was evaluated on three databases- CIFAR10, SVHN, and MNIST. The authors compare their results with six existing architectures that are designed with standard convolution, pooling, and fully-connected layers and argue that they outperform these results. They also compare MetaQNN with “state-of-the-art” methods and achieve comparable results.

Evolutionary Algorithms

Evolutionary algorithms have also been used in recent years to optimize neural network structures effectively. Liang et. al. utilizes a modified version of the popular evolutionary algorithm Neural Evolution of Augmenting Topologies (NEAT) called CoDeepNEAT to evolve deep neural architectures for chest x-ray and wikipedia comments classification [11]. CoDeepNEAT specifically interprets the nodes of the evolved NEAT networks as layers of a deep neural network, and was able to achieve state-of-the-art results on both of these datasets [11]. Furthermore, they found that architecture search is more successful than pure hyperparameter optimization [11]. A

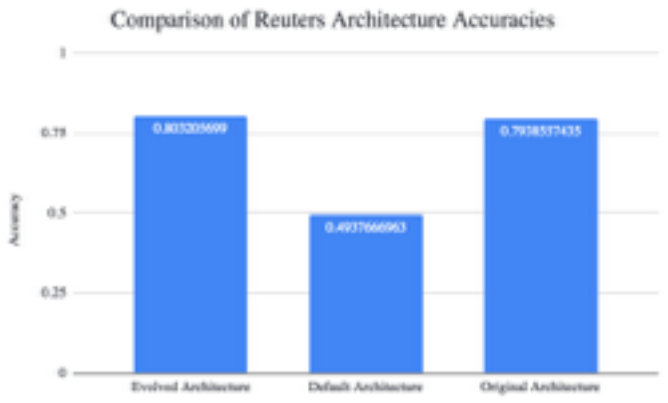

Figure 1. Comparison of network accuracies of the evolved, default, and original architecture for Reuters dataset. The evolved architecture as (activation function 1, dropout, activation function 2) is relu, .666, softmax. The default architecture is linear, 0.0, linear and the original architecture is relu, 0.5, softmax

group from Google Brain also developed a simple tournament selection evolutionary algorithm to evolve deep neural networks for both the CIFAR-10 and 100 datasets [12]. Their work evolved networks across a wide range of parameters including learning rate, link weights, inserting convolutional layers, deleting convolutional layers, stride, channel number, and filter size [12]. As impressive as these results are, our work cannot achieve these levels of success as the compute used in these experiments rely on GPU parallelization and use tens of hours of compute time [11, 12]. Our goal in this work is not state-of-the-art; rather it is to assure that significant performance increase of neural networks can be achieved over defaults with a simple evolutionary algorithm given a small population and evolution time on two common datasets as a proof-of-concept.

Domain

This work utilizes the Reuters and MNIST datasets as performance indicators for the neural networks we evolve. Both domains are well-established as popular benchmarks in the learning and deep learning communities and showing performance improvements on these datasets assures that indeed our simple approach can be compared to state-of-the-art and potentially scaled up. Another reason for choosing these datasets is that they are built into the Keras library and can be easily imported into Keras code, making using them very simple to do [9].

Reuters Text Categorization Dataset

The Reuters dataset is a collection of text newswires from Reuters in 1987 [9]. The dataset consists of 11,228 newswires, or feeds of newspaper and magazines articles, and over 46 topics, or categories, that each article can be characterized under. [9] Text categorization involves placing a piece of text into a category based on its contents; in this case an article is categorized into one of 46 prespecified categories like say “coconut” or “gold” [9]. What makes the Reuters dataset so interesting is that it was one of the first widely distributed machine learning training dataset used as a field-wide benchmark [9]. It also was extremely challenging dataset to categorize when it was first released, due to overlap of categories and ambiguousness [8]. We are using this dataset since it is so popular and is a common first dataset to train on for simple deep neural architectures.

MNIST Categorization Dataset

MNIST is a dataset of grayscale images of handwritten digits ranging from 0 to 9 from LeCun et. al [4]. There are 60,000 images to train the network with that are 28x28 pixels in size as well as 10,000 images set aside to test the

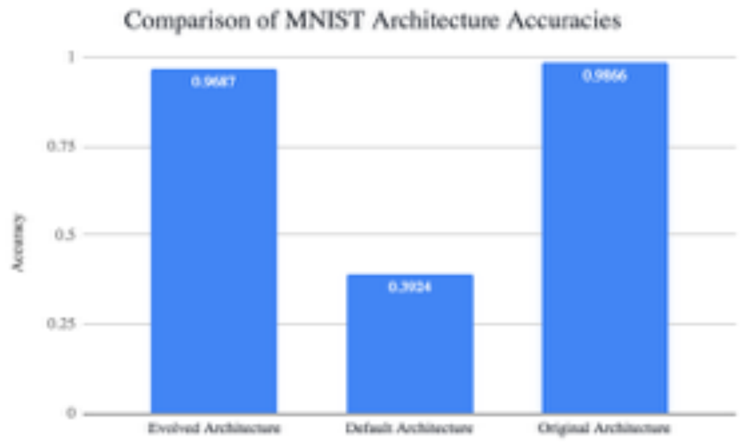

Figure 2. Comparison of network accuracies of the evolved, default, and original architecture for MNIST dataset. The evolved architecture as (activation function 1, dropout 1, activation function 2, dropout 2, activation function 3) is selu, ,783, elu, .478 and softmax. The default architecture is linear, 0.0, linear, 0.0, linear and the original architecture is relu, 0.25, relu, 0.5, softmax.

accuracy of the network [4]. The digits are also centered and size-normalized for ease of training. MNIST is a popular dataset as it is the first dataset that a deep neural network was able to perform to human levels of accuracy with LeNet [4]. It also has been successfully learned by a wide range of machine learning algorithms, which makes it a good benchmark to compare different machine learning methods to one another. We chose to use this dataset for that specific reason, and also because it is more complex and of a different type of classification than the Reuters dataset.

Approach/Methods

We used the TensorFlow library and Keras API to aid in developing and training our NAS program. TensorFlow is a Python library that enabled us to construct the model for our network [9]. Keras is an API that allows us to easily modify the layers of the model to generate new random networks. It works with TensorFlow in the backend to train networks using importable datasets [9].

First Attempt

We initially attempted to manually make changes to the network trained on the IMDB dataset using a tutorial found on TensorFlow's website [10]. We tried to see how different combinations of activation functions would affect the fitness. The activation functions available to us with Keras include elu, selu, softplus softsign, tanh, sigmoid, hard_sigmoid, linear, and exponential. The next modification we made was adding another layer to the network and then testing different combinations of activation functions. A few of our hand generated networks performed slightly better than the default network from the tutorial. However, majority performed comparable or worse when compared to the network from the tutorial. We quickly came to the conclusion that with the numerous possible combinations of activation functions and number of layers, this process of testing handmade networks would be infeasible for finding an optimal network.

Final Approach

After abandoning our initial approach, we decided to build a very simple default network with all linear activation and no dropout based on the networks provided in the Keras tutorials, and compare simple evolved architectures to said network. We decided to use a simple mu+lambda evolutionary strategy to create the network architecture and keep the elite member of each generation [7]. We decided to only modify the hyperparameters dropout and activation function and not the architecture, as modifying

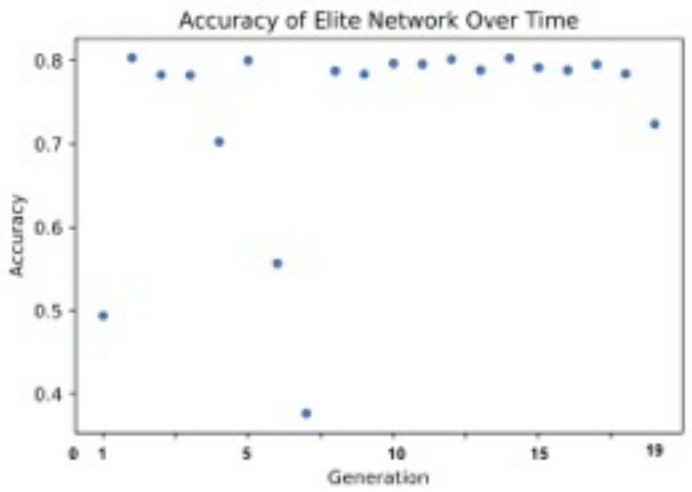

Figure 3. Evolved network accuracies by generation. The accuracy of the best network besides the elite network is displayed here. Interestingly, most of the elite networks from the randomly generated architectures perform better than the default.

architecture is complicated and is a huge search space. The networks we are altering are feed forward, as opposed to recurrent neural networks. The dropout rate is a randomly generated number between 0 and 1 and represents the number of links. We also compared against the original architecture provided in the starter code to see if our evolved networks could reach similar performance.

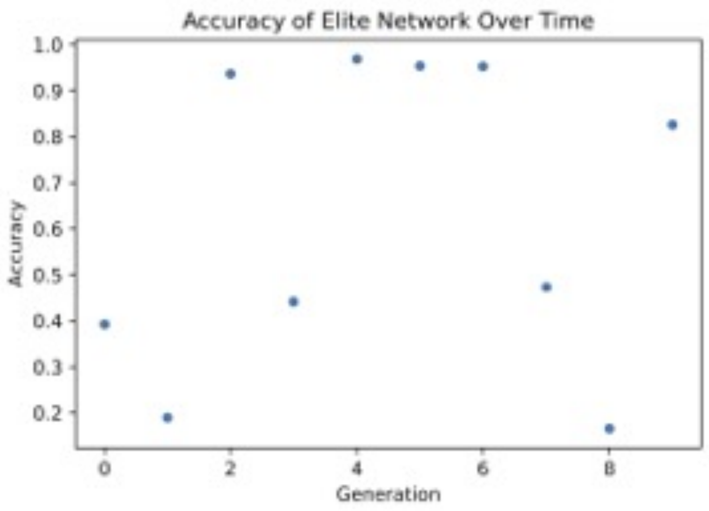

Figure 4. Evolved network accuracies by generation for the MNIST dataset. Here, the accuracies are more varied, but the smaller population size and fewer generations have more of an impact.

Experimental Procedure

For both the Reuters and MNIST datasets, for the final approach we modified the hyperparameters of two simple deep neural networks using a simple mu+lambda evolutionary algorithm.

Reuters Dataset

For the Reuters dataset, we took the original network from the Keras Reuters tutorial, which is a feed-forward network with a dense 512x1000 layer with a relu activation function and .5 dropout, another dense linear layer of 46 neurons with a softmax activation function.[9] From this original network, we created the default network for our experiments, which is a feed-forward network with the same 512x1000 dense layer with a linear activation function, no dropout and an output layer of 46 neurons with a linear activation function. Our evolutionary algorithm is an mu + lambda strategy, with an elitism parameter of 1. The fitness parameter was the accuracy of the network on the test data after 5 epochs of training via backpropagation on the training data with a batch size of 32. The mutation function is an asexual operator. A random activation function for the first and second layer is selected with replacement from this list of functions: tanh, softmax, exponential Linear Unit, scaled exponential Linear Unit, softplus, softsign, rectified exponential linear unit, sigmoid, hard sigmoid, exponential, and linear. Furthermore, the dropout rate between the two layers is selected from a Gaussian distribution randomly. This means the network genomes are represented by three separate evolvable hyperparameters that are modified by evolution. For each generation, the elite network from the previous generation is kept and the rest of the population is filled in with randomly generated genomes. The population size was 5 networks that were evolved for 20 generations.

MNIST Dataset

For the MNIST dataset, we took the original network from the Keras MNIST tutorial, which is a feed-forward network with a 2D convolutional layer with 32 channels, a kernel size of 2, a dropout rate of .25, a relu activation function, and flattened [9]. The next layer is a dense linear layer of 128 neurons, relu activation function, and a dropout rate of .5. Finally there is also an

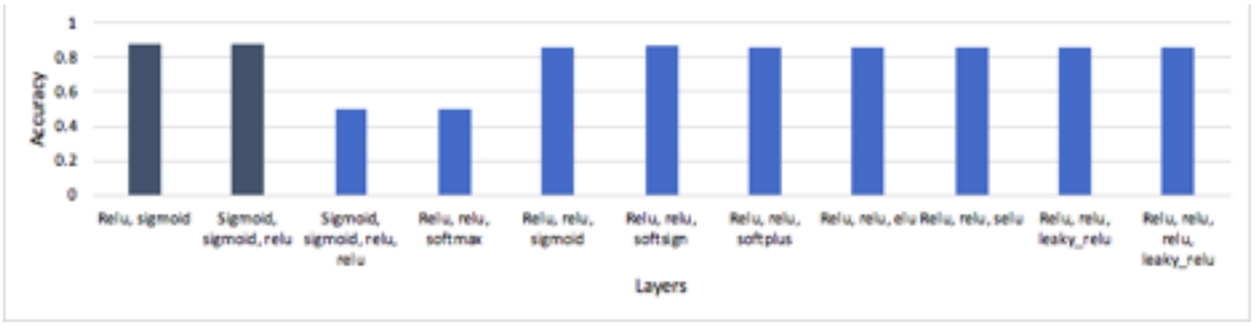

Table 1: The accuracies of architectures with different number of layers

output layer of 10 neurons with an activation function of softmax. The default network we compared against had linear as the activation function for all layers and a dropout rate of 0. For the MNIST experiments, there were 5 parameters in the genome: the first activation function, the first dropout, the second activation function, the second dropout, and the third activation function. Like in Reuters, to create a child genome, activations were randomly selected from the aforementioned list of activation functions with replacement, and the dropout was randomly selected from a Gaussian distribution. We also used an mu + lambda evolutionary strategy with an elitism parameter of 1. We kept the default kernel size of 2. The fitness parameter for this strategy was also the accuracy of the network on the test data after 5 epochs of training via backpropagation on the training data. Of course, to calculate the highest accuracy rates requires a significant amount of time (due to the size of the two datasets) to train the models we generate at each iteration. For this reason the number of generations and population size are relatively small compared to other research. For the MNIST dataset we ran 10 generations on a population of size 5.

Results

Our initial approach of manually manipulating the number of layers and activation functions did not yield interesting results. The reason for this is because we were trying to manipulate an architecture that was already optimized. The IMDB dataset was initially trained on a model architecture with 2 activation layers, the first was relu and the second was sigmoid, with an accuracy of 0.87208, the dark bar in Table 1.

To modify the number of layers we investigated the accuracy of the model with 2 to 4 activation functions, the values are depicted in Table 1. We were able to get accuracies comparable to the optimized network with 3 layers and sigmoid, sigmoid, relu activation functions and also with a relu sigmoid architecture, but the search was by hand which defeats the point of automated architecture search.

We then switched over to our final mu + lambda strategy. After switching to our final approach, we did indeed see a significant increase in network performance over the default architecture using an evolved neural architecture search strategy. In figure 1, we can see that for the Reuters dataset, the evolved network had significantly higher performance over the linear default architecture and comparable performance to the optimized original architecture. This evolved network had a relu activation function for the first layer, a dropout rate of 0.666 and used softmax as its second activation function. For the MNIST dataset, as seen in Figure 2 we also see

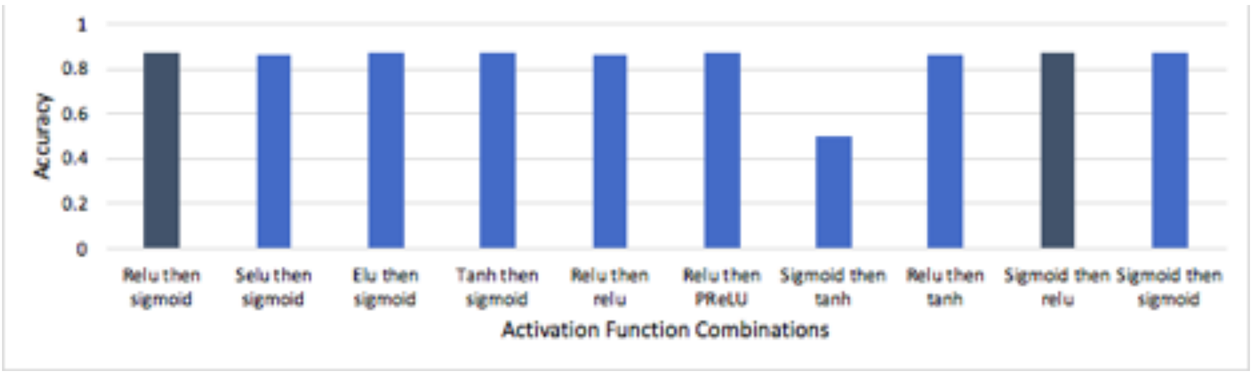

Table 2: The accuracies of architectures with different activation functions.

a significant increase in performance by the evolved architecture over the linear default and comparable performance to the original optimized architecture as well. This architecture had selu, 0.783 dropout, elu, 0.479 dropout, and softmax as its linear architecture. The network performance over time can also be seen for the Reuters dataset in Figure 3. The points displayed are the optimal architectures excluding the elite architecture for each generation. There is no correlation between accuracy and generation because all children are randomly generated. It is interesting to note though that generally, in a randomly selected population of 4 child networks per generation, for most generations at least one of these networks has comparable behavior to the optimal network. This indicates that there are many local optima in the architecture landscape that exhibit successful behavior and indicate that increasing the population size or number of generations may not lead to a significant increase in network accuracy. This has implications for neural architecture search because it indicates that extensive search algorithms that require large compute, like seen in the previous work section, may not actually be necessary in order to discover decent behavior for text classification domains. However, as can be seen in Figure 4, this hypothesis doesn't hold as strongly for the MNIST image classification domain.There is a lot more variation across generations in terms of performance of the best randomly generated child network. However, the MNIST runs had a smaller population size, which indicates that potentially population size correlates more strongly with increase in performance than number of evolved generations does.

Discussion and Future Work

Ultimately, our results show that significant increase in network performance can be obtained using a simple evolutionary algorithm with little compute resources, and can even discover behavior comparable to hand-optimized networks. Further analysis of our elite networks across generations indicate that random search discovers good architectures a lot of the time, indicating that large compute may not be necessary in order to discover good network behavior in certain domains like text classification. Furthermore, comparing our MNIST and Reuters results indicate that a larger population size may correlate to discovering more high accuracy architectures. Our method of neural architecture search was successful though given the parameters of the project.

In future work, we would like to expand our code to also evolve the number, type, and shape of the network layers on top of the hyperparameters we currently search. Given previous results from Liang et. al. we think that evolving the network architecture will return even better results than just modifying the hyperparameters [the first ea paper]. We would also like to run the experiments for longer with more compute, maybe even with GPU parallelization. Finally, we would like to extend our work into other different domains, maybe even comparing against domains used in the state-of-the-art work mentioned in Previous Works.

Conclusion

This project successfully attempts to use evolutionary techniques to perform simple neural architecture search on two classic classification domains. Using a mu + lambda strategy, we evolved hyperparameters for deep neural networks trained using Keras and Tensorflow on the Reuters and MNIST datasets and compared their accuracies to default structures. Our evolved networks produced significantly better behavior than the defaults and comparable behavior to hand-optimized networks. Future work includes evolving the number and types of layers alongside the hyperparameters that we already tune.

References

[1] Simonyan, K. and Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition in *ICLR*, (San Diego, CA, 2015), 1-14.

[2] Zhang, Lei & Wang, Shuai & Liu, Bing. (2018). Deep Learning for Sentiment Analysis : A Survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1* (2018), 1-34.

[3] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. A survey of deep neural network architectures and their applications. *Neurocomputing*. *234* (April 2017). 11-26.

[4] LeCun, Y., Cortes, C., and Burges, C. The MNIST Database of Handwritten Digits. Retrieved April 16, 2019 from http://yann.lecun.com/exdb/mnist/.

[5] Elsken, T., Metzen, J., and Hutter, F. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*. *20* (2019). 1-21.

[6] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.

[7] Ter-Sarkisov, A. and Marsland, S, Convergence Properties of ($\mu + \lambda$) Evolutionary Algorithms. in *AAAI Conference on Artificial Intelligence,* (San Francisco, CA, 2011), 1816-1817.

[8] Baker, B., Gupta, O., Naik, N., Raskar, R., Designing Neural Network Architectures Using Reinforcement Learning. Published as a conference paper at ICLR 2017.

[9] Keras. The Python Deep Learning library. Retrieved April 16, 2019 from https://keras.io/.

[10] TensorFlow. Retrieved April 16, 2019 from https://www.tensorflow.org/.

[11] Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K., and Miikkulainen, R.. Evolutionary Neural AutoML for Deep Learning. In *Proceedings of Conference on Genetic and Evolutionary Computation 2019*.

[12] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y., Tan, J., Le, Q., and Kurakin, A. Large-Scale Evolution of Image Classifiers. In *Proceedings of 34th International Conference on Machine Learning 2017*.

I have acted with honesty and integrity in producing this work and am unaware of anyone who has not. Lauren Gillespie, Sara Boyd, Elyssa Sliheet