Architecture Search in Deep Neural Networks

Intro: The Evolution of Neural Architecture Search

Traditional machine learning practitioners rely on domain-specific knowledge to extract and compose relevant features. Consequently, practitioners spend most of their time on the laborious process of hand-crafting features to pass into their algorithms.

Deep learning abstracts feature extraction into the learning mechanism itself, by teaching the early layers of neural networks to automatically learn optimal feature representations, which are then processed by the network's later layers. This advance is nothing short of a breakthrough: practitioners no longer need to hand-craft features, and can instead expect reasonable results by simply feeding raw data into deep networks.

Deep learning has effectively shifted the burden of machine learning from feature-engineering to network-engineering. While we no longer need domain specific knowledge, we now need to understand network architectures intimately. A quick look at recent machine learning conference papers reveals overwhelming focus on neural network architecture design, with breakthrough papers chiefly proposing architecture adjustments such as stacking various network models in particular manners, or variants on "going deeper." While progress is being made in this field, the machine learning community is combating a problem of interpretability: tuning network depths, widths, and components is not an intuitive process.

Neural architecture search automates the interpretability problem of network design -- I believe it is well-poised to solve many of the most pressing problems facing the machine learning community today.

THESIS: *Machine Learning is not what it used to be: rather than the art of interpretably engineering features, we are now faced with the art of tuning "magical" (uninterpretable) network topologies and network components. However, In the same way that feature tuning was automated via deep learning, network tuning has the potential to be automated via neural architecture search.*

Neural Architecture Search

TOPIC SENTENCE: We define neural architecture search as the process of *learning* a network topology. Methods proposed for optimizing over neural network topologies include Bayesian Optimization, Genetic Algorithms, and Reinforcement Learning. Reinforcement Learning is currently the most effective and promising (in terms of performance and rate of advances) method.

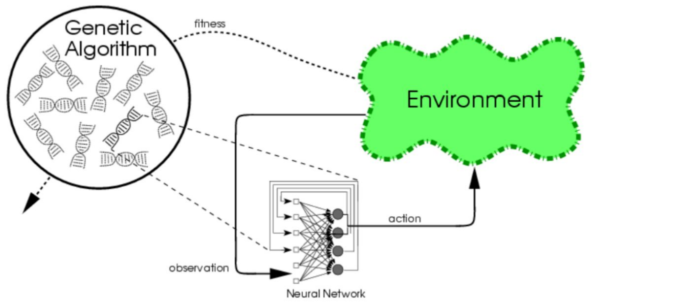


Figure 1: **Evolving Neural Networks.** population of genetic neural networks encodings (genotypes) is first created. At each iteration of evolution (generation), each genotype is decoded into a neural network (phenotype), which is evaluated in the task, resulting in a fitness value for the genotype. Crossover and mutation among the genotypes with the highest fitness is then used to generate the next generation.

The first attempt at neural architecture search used genetic algorithms [cite]. Neuroevolution (2008) uses genetic algorithms to jointly optimize over the weights and topology of a network, for maximal performance on a particular task (See Figure 1). *Large-Scale Evolution of Image Classifiers* (2017) [cite] shows that genetic algorithms are able to develop network topologies which can contend with today's state-of-the-art hand-engineered network architectures.

Neural Architecture Search with Reinforcement Learning (2017)

Neural Architecture Search with Reinforcement Learning (N.A.S.R.L.) [cite] optimizes over network architecture topology using reinforcement learning. First, a language to translate between variable-length strings and network architectures is defined (an encoding of network architectures as variable-length strings).

Then, to optimize over network architecture, a recurrent "controller" network

- produces a variable-length string (adhering to the aforementioned language standards),

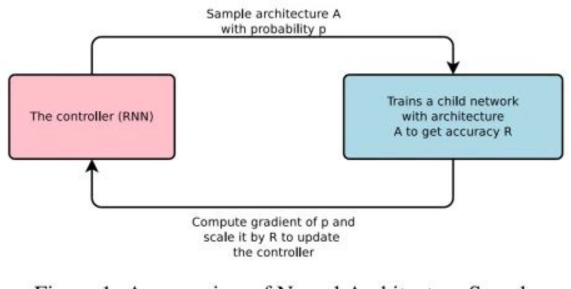

Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller

Figure 1: An overview of Neural Architecture Search.

- decodes it into a “child” network architecture, trains this child network,
- evaluates the trained child network to produce some reward score R, and finally,
- uses R as a reward signal to make a policy gradient [cite] update (using REINFORCE) to the controller network’s parameters, with the objective of optimizing the controller’s expected reward.

The controller then outputs another variable-length string, and the cycle continues until the child network’s architecture converges to some fixed topology.

Extensions on Neural Architecture Search, Defining DSLs, and Computational Concerns

TOPIC SENTENCE:
Neural architecture search can be extended to learn other components of deep neural networks, so long as we can define an appropriate domain-specific language. To render this task feasible, we also need to improve upon neural architecture search’s computational efficiency.

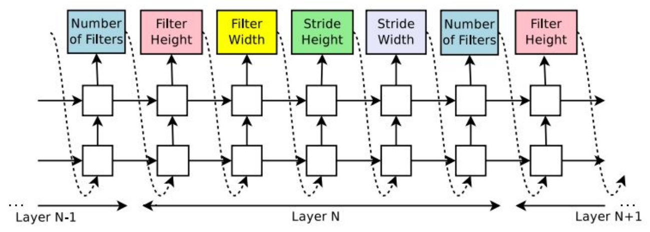


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

In the same way that we defined variable-length strings in neural architecture search in order to represent a network architecture’s topology, we can define proxies for many other pieces of deep neural networks, such as network layers, and even optimization schemes. For example: when defining a simple convolutional neural network, each subsequent index of the string might represent a particular hyperparameter of a particular convolutional layer, such as filter width or stride height (See Figure).

We call the set of rules for defining the aforementioned variable-length strings a “Domain-Specific Language”. We can generalize this definition to an string representing any arbitrary network component. A Domain-Specific Language (DSL) is a language that can encode some aspect of a deep neural network as a set of learnable parameters and translate between this set of parameters and the component. For more information, see Section 3 of [cite].

Neural architecture search is flexible, except for in the DSL (which requires detailed characterization). Thus, if we can define an appropriate DSL for a neural network component, we can then easily learn an optimal version of that component using neural architecture search.

LSTM Cells

N.A.S.R.L. experiments not only with optimization of network topology, but also with optimization of a particular network component: the content of an LSTM cell. The authors design a novel LSTM cell which outperforms the traditional LSTM cell and generalizes across tasks. [cite]

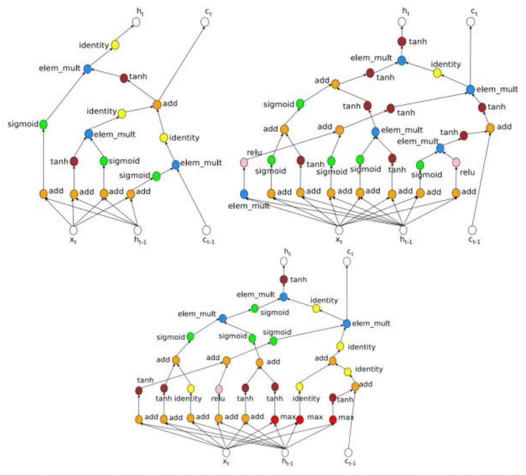


Figure 8: A comparison of the original LSTM cell vs. two good cells our model found. Top left: LSTM cell. Top right: Cell found by our model when the search space does not include *max* and *sin*. Bottom: Cell found by our model when the search space includes *max* and *sin* (the controller did not choose to use the *sin* function).

Optimizers

Google Brain recently expanded upon the N.A.S.R.L. architecture [cite] to allow for learning optimal gradient update rules (in place of traditional optimizers such as vanilla SGD, Adam, RMSProp, etc..) for the child network.The overall Neural architecture search scheme remains largely unchanged, and the true novelty in this work lies in its definition of an effective DSL for optimizers. To see this in the Figure, note that the optimization scheme is largely the same as that used in N.A.S.R.L., save for the replacement of A (architecture) with U (update rule).

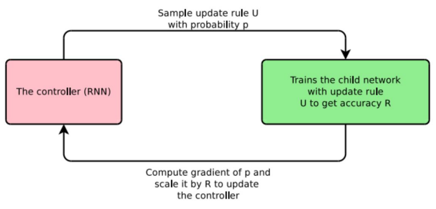
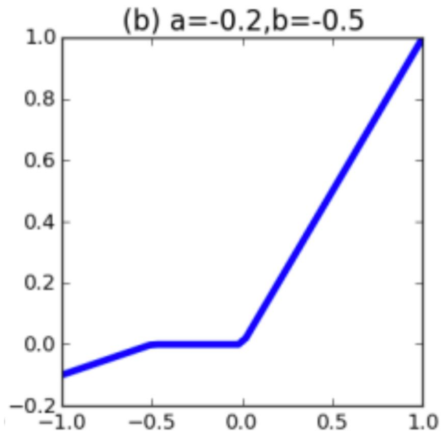

Sample update rule U with probability p
The controller (RNN)
Trains the child network with update rule U to get accuracy R
Compute gradient of p and scale it by R to update the controller

Figure 1. An overview of Neural Optimizer Search.

We can see the definition of the update-rule (optimizer) DSL in the equation below. The update rule is represented as a string describing:
“1) the first operand to select,
2) the second operand to select,
3) the unary function to apply on the first operand,
4) the unary function to apply on the second operand and
5) the binary function to apply to combine the outputs of the unary functions.” [cite]

Activation Functions

We can also use neural architecture search to optimize over one of the most obscure components of neural network design: activation functions. The activation-function DSL problem was tackled first naively, in [cite] by parametrizing activation functions as piecewise linear functions (sums of hinge functions; see Figure) and optimizing for performance of networks containing them by performing gradient ascent in the piecewise linear function's parameter space. The next attempt at learning nonlinearities improves upon this DSL by nonparametrically defining learnable activation functions in terms of a Fourier basis expansion. This approach offers greater flexibility in the space of functions which can be represented, yet correspondingly leads to limitations in exploring this now-massive space.



While to the author's knowledge there do not exist any attempts as using N.A.S.R.L. for activation function search, we believe that this is a promising research direction, particularly for learning uninterpretable activation functions.

DSLs as a search-space spanning problem

There is a tradeoff to the DSL problem. When defining a domain specific language that parametrizes a deep network component, we want a language that has arbitrary expressiveness, yet which will define parameters over which we can efficiently optimize. In other words, we want to define a language space expansive enough to span the space of all possible novelties, yet within which we can still efficiently learn.

For example, when defining a DSL for activation functions, we seek to define a language capable of both *parameterizing* and *learning* all monotonic squashing functions. This balance is difficult to strike. For example, the piecewise linear DSL [cite] spans only a small subspace of the space of all monotonic squashing functions, yet is easily learnable. On the other hand, the Fourier basis expansion DSL [cite] spans a much larger subspace, yet is not as easily learnable.

With this tradeoff in mind, we motivate research into definitions of DSLs which define expressive, yet learnable function spaces. Can you think of a DSL able to intelligently encode the space of all deep neural network regularization schemes?

Computational Concerns

Current reinforcement learning methods for optimization over network topologies and components are highly computationally taxing and thus research in the area is slow and limited to industry giants with access to thousands of GPUs.

To truly understand the computational burden of neural architecture search, note that in N.A.S.R.L., each gradient update to the controller requires training the “child” network to convergence. This scheme requires *significant* resources, as each gradient update can require 5-6 hours on modern GPUs. For example, Google Brain’s original N.A.S.R.L. experiment was trained using 800 GPUs over several weeks. Compute resources at this scale are unavailable to 99.9% of the machine learning community.

There is no currently known solution to neural architecture search’s computational burden. However, the work in [cite] points towards a couple promising directions. First, in Section 3.3, by expanding on the parallelization scheme described in [cite], the authors see initial results showing that the distributed training scheme can converge in less than a day using 100 CPUs. This work also points towards using more advanced policy gradient methods to update the controller, such as TRPO [cite].

With computational costs in mind, we motivate research into efficient neural architecture search training schemes, in order to allow for widespread research in the field by those without access to industry-level GPU clusters.

Conclusion

In conclusion, we see promise in using machine learning itself to optimize the structure, components, and all else related to training deep learning models.

We particularly motivate further work in: (1) Study of language encodings for representing deep neural network components and topologies; (2) Study of computationally tractable methods for reinforcement-learning based neural architecture search.

Cheers,
Riley