Local Stochastic Differentiable Architecture Search for Memetic Neuroevolution Algorithms

Joshua Karns
Rochester Institute of Technology
Rochester, New York, USA
josh@karns.dev

Travis Desell
Rochester Institute of Technology
Rochester, New York, USA
tjdvse@rit.edu

ABSTRACT

Even the most efficient approaches to neural architecture search can be very computationally expensive, which leaves little room for inefficiencies. Unfortunately, evolutionary approaches to neural architecture search (NAS) - neuroevolution - often suffer from these inefficiencies due in part to their massive, intractable search spaces. In the past several years, differentiable approaches to neural architecture search have garnered significant attention for their efficiency and performance. In this work, we suggest an approach to create a hybrid algorithm: a neuroevolutionary metaheuristic which employs a local differentiable stochastic neural architecture search during the fitness evaluation step. Our preliminary results demonstrate that this approach is successful in selecting recurrent neural network memory cells.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; • **Mathematics of computing** → **Stochastic processes**; **Evolutionary algorithms**.

KEYWORDS

differentiable neural architecture search, recurrent neural networks, metaheuristics

ACM Reference Format:
Joshua Karns and Travis Desell. 2023. Local Stochastic Differentiable Architecture Search for Memetic Neuroevolution Algorithms. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583133.3596368

1 INTRODUCTION

The manual design of performant neural network architectures is a difficult and labor intensive task. It is no surprise then that a wide assortment of techniques have popped up over the past few decades to automate this gruelling process, particularly with the advent of deep learning. These neural architecture search (NAS) algorithms must strike a balance between search efficiency and completeness. There are two levers that are usually reached for to control this: search space and performance evaluations strategy (PES).

A large search space opens the door to more performant candidate architectures but can also mean an incredibly computationally expensive search, without a guarantee that the search space can successfully be navigated to find such networks. Similarly, a high-fidelity PES means information the algorithm receives regarding candidate architectures is highly accurate, however this performance metric may be very expensive to compute. Most algorithms attempt to strike a balance with their choice of search space and/or PES.

Evolutionary approaches to NAS are, for all their virtues, relatively inefficient. Algorithms derivative of NEAT [9] in particular have incredibly intricate and large search spaces, evolving networks at the node and edge level. This leads to a search that can be slow, as networks grow by very small modifications from an initially minimal network, and a less than ideal strategy for selecting weights (as networks are not trained). Many techniques have been employed to bolster the efficiency of NEAT-like algorithms; one such approach is to use a local search during the performance evaluation step. For example, EXAMM [8] is a metaheuristic which uses backpropagation to efficiently train network weights before the fitness calculation is done. This takes the burden of discovering weights off of the evolutionary algorithm and places it in the hands of an incredibly efficient technique.

On the other end of the spectrum are one-shot / differentiable approaches to NAS, like DARTS [7] and SNAS [10]. Designed to be extremely efficient, both of these algorithms train a single network where nodes are connected with multiple operations; the outputs of these various operations are mixed using learned weights that can be thought of as architectural parameters. These architectural parameters are differentiable and can be trained along with normal network weights. So, only a single network needs to be trained. The search space however is fixed and must be provided as an input to the search, subject to hardware constraints (like VRAM on a GPU).

In this work, we do some preliminary investigation into the possibility of integrating a SNAS-style local search into an existing evolutionary metaheuristic: EXAMM [8]. EXAMM evolves recurrent neural network and randomly selects recurrent memory cell types. Our experiments demonstrate that SNAS-style architecture search can be used to select memory cell types by comparing the learned choices SNAS makes with a grid search. This leaves the door open for full integration of SNAS into EXAMM, fully removing the burden of selecting recurrent memory cell types from the evolutionary algorithm and placing it in the hands of a more efficient differentiable approach.

GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0120-7/23/07…$15.00
https://doi.org/10.1145/3583133.3596368

2 RELATED WORK

Differentiable neural architecture search (DNAS) is not a new concept, but was largely popularized by Liu *et al.* in a 2018 paper which presented an algorithm called DARTS: **D**ifferentiable **A**rchi**t**ecture **S**earch [7]. One of they key innovations presented by this paper is their relaxation of a discrete search space into a continuous and differentiable search space. The DARTS search space is depicted in Figure 1a. All candidate operations to connect two nodes in the search space are evaluated simultaneously and summed together using a weighted average. The weights used for this average are obtained by pushing a vector of weights through a Softmax operation, yielding a probability vector. These weights are referred to as architectural parameters as they are a learnable (*i.e.* differentiable) representation of neural architecture. They can be trained along with the rest of the weights using stochastic gradient descent.

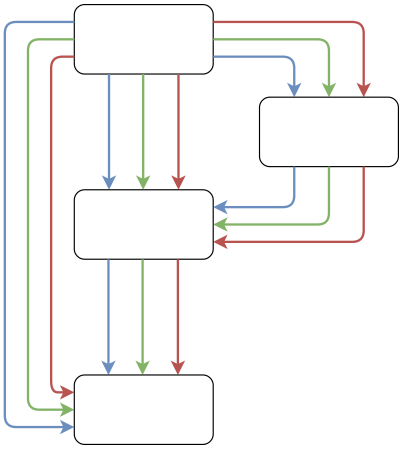
Stacking these learnable decisions on top of one another yields a search space: all possible sub-graphs (with up to one edge connecting any two nodes). Performing the actual search is as simple as training the macro-network and updating the weights as usual; the architecture search itself is done via gradient descent. The final step is discretization: obtaining a final output network. If you interpret the architectural parameters (after they're pushed through Softmax) as the likelihoods of an operation being the optimal, obtaining the most likely model would be as simple as removing all but the most likely operations connecting nodes. A trained DARTS search space is shown in Figure 1b; discretization would involve removing all but the thickest line connecting any two nodes.

Subsequent works have pointed out some failure modes of DARTS. Zela *et al.* explore this, identifying 12 engineered search spaces in which DARTS converges to degenerate architectures: architectures composed largely of parameter-less operations like skip connections and even noise operations [11]. Xie *et al.* claim that the relaxation used in DARTS introduces an intractable bias preventing the training process from maximizing the true objective: the expected performance of the derived architectures [10]. To address this, they propose an algorithm they call Stochastic Neural Architecture Search (SNAS).

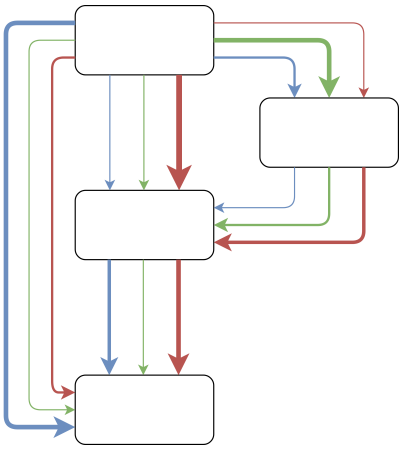
SNAS is similar to DARTS where node values are a mixture of operations, but the source of the probability vector used to weight the operations is different. Rather than using a fixed set of learnable weights, the vectors are sampled from a probability distribution. The authors pose that the probability $p(Z;\alpha)$ of a given architecture Z can be fully factorized into a set of probability distributions $\{\, p(Z_{i,j};\alpha) \mid (i,j) \in E \,\}$ where E is the set of edges. Then, iteratively sampling these architectures from $p(Z;\alpha)$ and training them (both the network weights and the distribution weights α) should optimize for the expected performance of architectures sampled from $p(Z;\alpha)$.

$$p(Z^k_{i,j}) = \frac{exp((ln(\alpha^k_{i,j}) + G^k_{i,j})/\lambda)}{\sum_{l=0}^{n} exp((ln(\alpha^l_{i,j}) + G^l_{i,j})/\lambda)} \tag{1}$$

The distribution they use is a continuous relaxation of the categorical distribution presented by Jang *et al.* called the Gumbel-Softmax distribution [5]. The equation used for sampling this distribution is described in Equation 1, specifically showing the probability of operation k connecting node i to node j is calculated, where



(a) A DARTS search space. Different colored edges represent different candidate operations. The actual networks this space represents are networks with only one of the three candidate operations connecting any two nodes.



(b) A trained DARTS search space. Thicker edges represent operations with a larger learned weight, and thinner values smaller weights.

Figure 1: Depicts a simple DARTS search space before (1a) and after (1b) training.

$G \sim Gumbel(0,1)$ and α is the matrix of architectural weights. It is quite similar to Softmax in appearance, and similarly outputs a probability vector. But unlike Softmax it is not deterministic (*i.e.* the same inputs will produce different outputs). Softmax is roughly the MLE of the Gumbel-Softmax distribution, though the parameterization is slightly different.

This distribution fits the problem well as it is parameterized with a temperature λ that allows for the distribution to be continuously annealed into a categorical distribution. That is: as λ approaches 0, a Gumbel-Softmax distribution approaches the categorical distribution. Moreover, Madison *et al.* proved that the relaxation is unbiased when converged. More noteworthy in the application of

this distribution to neural networks is that it can be made differentiable using the reparameterization trick [6], meaning a SNAS search space can be trained with stochastic gradient descent.

SNAS is efficient and works well in practice, but is usually only applied to a fixed search space. If $p(Z;\alpha)$ is fully factorizable as the authors assume, then modifying the search space should not be a problem. That is the approach being proposed in this work: leverage the ideas of SNAS to improve the efficiency of NE algorithms. SNAS can be leveraged to shift the burden of searching for operations onto the optimizer used for SGD. Such an algorithm would no longer be evolving neural networks. Rather, they would be evolving search spaces with corresponding probability distributions.

Hybrid neuroevolution algorithms go back well over two decades, the discussion of which can be seen in a 1999 journal article by Yao *et al.*. In particular, they weigh the use of backpropagation to search for weights rather than just using evolution. There was no discussion of hybrid approaches to searching architecture, however.

SNAS is an approach would fit well into existing metaheuristics which use some sort of SGD to evaluate networks, and would in effect create a hybrid approach which does search for architecture. One such metaheuristic is the Evolutionary eXploration of Augmenting Memory Models [8]. Based on NEAT [9], this algorithm evolves recurrent neural networks (RNNs) for time series data prediction tasks. When adding nodes, the evolutionary algorithm must make a decision on what sort of node to use - a plain node, or one of several types of memory cells (*e.g.* LSTM [4], MGU [12]). Usually the node type is randomly selected from a uniform distribution, but after adding only a few nodes the number of possible permutations of node types ends up being massive.

In this work we propose the use of a SNAS-style search to determine node types, rather than relying on the evolutionary search. When evaluating genomes, EXAMM trains them with backpropagation and updates their weights; without modification this process can also be applied to additional architectural weights. It is unclear whether SNAS can be applied to (1) node and edge-level NEAT-style search space and (2) recurrent memory cells. The experiments we perform in this work demonstrate that both are possible; our experiments show well performing memory cells are disproportionately selected by the Gumbel-Softmax distribution.

3 METHODOLOGY

EXAMM supports six different types of recurrent memory cells by default: simple (normal node with a recurrent connection), LSTM [4], MGU [12], GRU [1], UGRNN [2], and Delta [8] cells. In order to learn the optimal cell using SNAS, we propose a new cell type: the SNAS cell. This cell contains all cell types and feeds the input to all of them. The output is then computed as a mixture of all cell outputs according the the mixing ratios obtained by sampling $p(Z;\alpha)$. This cell is depicted in Figure 2.

There is only one sample drawn from $p(Z;\alpha)$ per forward pass, meaning the mixing ratios are the same for every time step. This decision was made as it was thought that it would be difficult for networks to propagate information through time if the architecture were potentially unstable between timesteps.

In SNAS, the mixing ratios $Z_{i,j}$ for operations connecting nodes i and j are made to be one-hot [10]. This means they are only using

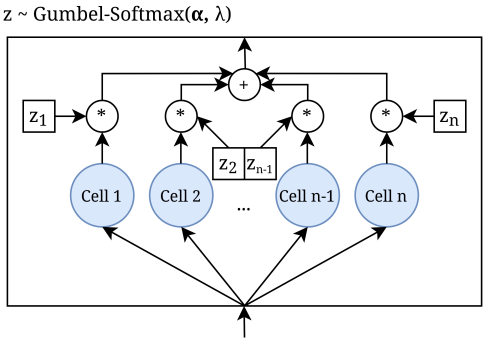


Figure 2: Shows the inner architecture of a SNAS cell. Note that z in the diagram corresponds to $Z_{i,k}$ for $Z \sim P(Z;\alpha)$.

the most likely operation, according to the sample $Z \sim P(Z;\alpha)$. In this work, the number of non-zero values in $Z_{i,j}$ is a hyperparameter k. For k values less than the number of candidate operations, all but the top-k values are zeroed out and $Z_{i,j}$ is re-normalized to be a probability vector.

4 EXPERIMENTS

The experiments done in this work are a sanity check of sorts to ensure that SNAS can learn the appropriate memory cell types. To do this, we perform a small grid search over all six node types to obtain a ground truth ranking. Then, we repeatedly perform a search using a SNAS-node and observe which node type it learned to be the most likely. We tabulate these results to obtain a probability for each node of SNAS having determined it to be the most likely.

The grid search begins with a very simple neural architecture - input nodes, a single hidden layer with a single node, and a several output nodes. We replace the hidden node with the appropriate node type and train it. This was done 20 times for each node type, starting with a different weight initialization each time. We can then use the average of the validation error for each node type to rank the nodes from worst to best. All of these experiments used 400 iterations of backpropagation.

For the SNAS experiments, the hidden node was replaced with a SNAS cell and the model was then trained. We observe which node was determined to be the most probable, and tabulate the results for the 20 repeats of the experiment. We do this for several hyper-parameter configurations, tweaking k (as in k-hot Z values) and the number of iterations of backpropagation used to train the SNAS cells.

All experiments used the Adagrad optimizer [3] to update the weights, with a learning rate of 0.01. A burn in period was allowed for the memory cells as well - that is, the architectural weights were not updated until the cell weights had gotten some time to train. The function used to calculate the architectural weight learning rate is shown in Figure 3.

As was mentioned in the Related Work Section, the Gumbel-Softmax distribution takes in a temperature parameter. The choice of the temperature value / schedule can have a significant impact on training behavior, particularly when the k value used is > 1. In

Table 1: Average validation MSE of the cells ± one standard deviation, along with their rankings.

Node type	Simple	LSTM	GRU	MGU	Delta	UGRNN
Avg. MSE	0.30518 ± 0.064	0.34037 ± 0.123	0.30036 ± 0.082	0.30179 ± 0.067	0.35095 ± 0.061	0.30642 ± 0.071
Rank	3	5	1	2	6	4

Table 2: Summary of the experiments which employed the SNAS cell. Each row corresponds to a hyperparameter configuration which was repeated 20 times. The probabilities correspond to the proportion of those 20 repeats in which a given cell type was learned to be the best by the SNAS cell. The most likely cell type in each row shown in bold.

BP ITERS	k	p(SIMPLE)	p(LSTM)	p(GRU)	p(MGU)	p(DELTA)	p(UGRNN)
100	1	0.05	0.1	0.25	**0.3**	0.15	0.15
100	2	0.05	**0.45**	0.15	0.15	0.0	0.2
100	6	0.05	0.15	0.2	0.2	0.15	**0.25**
200	1	0.05	0.3	0.05	**0.35**	0.1	0.15
200	2	0.1	0.2	0.15	**0.25**	0.1	0.2
200	6	0.05	0.15	0.1	**0.25**	0.2	**0.25**
300	1	0.05	0.05	0.25	0.1	0.05	**0.5**
300	2	0.0	0.1	**0.4**	0.2	0.15	0.15
300	6	0.0	0.1	0.25	**0.3**	0.15	0.2
400	1	0.1	0.15	**0.3**	0.15	0.05	0.25
400	2	0.0	0.15	**0.3**	**0.3**	0.05	0.2
400	6	0.0	0.25	0.1	**0.35**	0.05	0.25

$$\text{arch-lr}(p) = \begin{cases} 0.0 & \text{if} \quad 0 \leq p < 0.33 \\ 0.5 + 0.5 * \frac{0.66-p}{0.33} & \text{if} \quad 0.33 \leq p < 0.66 \\ 1 & \text{else} \end{cases}$$

Figure 3: Function used to calculate the learning rate for architectural parameters. $arch - lr$ is a function of p which represents the proportion of training (*i.e.* backpropagation iterations) that has been completed.

$$\lambda(p) = \begin{cases} 1.33 & \text{if} \quad 0 \leq p < 0.33 \\ 1.33 - 1.15 * \frac{0.66-p}{0.33} & \text{if} \quad 0.33 \leq p < 0.66 \\ 0.18 & \text{else} \end{cases}$$

Figure 4: Function used to calculate the temperature throughout the experiments. λ is a function of p which represents the proportion of training (*i.e.* backpropagation iterations) that has been completed.

this work, a fixed temperature schedule was used and it is depicted in Figure 4.

The specific dataset used in these experiments is a time series regression dataset provided in the EXAMM codebase [8]. It consists of 12 data files coming from aircraft's flight data recorders. Each data file contains 5500 consecutive entries of per-second aircraft sensor readings, including information about the aircraft's engine and its position, orientation, and velocity. The networks were set to predict eight engine parameters one time step into the future.

5 RESULTS

Our results are summarized in Tables 1 and 2. We observe that SNAS appears to have a strong preference for memory cells over simple neurons, despite the fact that the simple neuron performed the 3rd best out of the six node types. There is a fair preference for GRU and MGU nodes, which hold first and second place respectively. Of the 12 experimental groups, GRU and MGU nodes were more likely to be learned to be the most probably by SNAS than any other node type in 9.

6 DISCUSSION

The magnitude of preference for GRU and MGU units as shown in Table 2 is not remarkably strong but given the high variance in cell performance shown in Table 1, it is impressive it was able to distinguish the cells from one another at all. Still, the probabilities are much higher than if cells were being selected at random; the probabilities would be $\frac{1}{6} = 0.1\overline{666}$ if SNAS were learning randomly and uniformly.

There was a clear preference against simple neurons - the reason for this should be investigated further, particularly since the simple neurons were found to be the 3rd best performing operation in our grid search.

The effect of using more iterations of backpropagation doesn't seem to be particularly impactful beyond 200 iterations. The value of k also doesn't appear to be particularly impactful in this small scale. These preliminary tests were ran in part because of concerns regarding hyperparameter configuration for SNAS, but it appears that SNAS may perform well in many different configurations.

Future work will include further grid search studies that are more realistic. Rather than just evaluating a trivial neural network with a single hidden node, RNNs generated by EXAMM can be used

instead. The various memory cells can be placed in the same place in the network and then evaluated. We can then place a SNAS memory cell in the same position and observe which memory cell it learns to be the most probable, and compare this with our grid search. This phase of research will be a good opportunity to investigate the importance of hyperparameter tuning since it will be occurring in a more realistic use case.

The ultimate goal of this line of work will be to only add SNAS cells to genomes during mutations. Then, the networks will be able to learn the most likely cell type for every node added to the network. This will in a sense change the nature of the evolutionary algorithm, though. Rather than evolving neural networks directly, we are evolving joint probability distributions over a search space. This should however reduce the complexity of the evolutionary search, forking over the task of selecting memory cells to the efficient and differentiable SNAS cells.

REFERENCES

[1] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. arXiv:1409.1259 [cs.CL]

[2] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. 2017. Capacity and Trainability in Recurrent Neural Networks. arXiv:1611.09913 [stat.ML]

[3] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12, null (jul 2011), 2121–2159.

[4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (nov 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[5] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. arXiv:1611.01144 [stat.ML]

[6] Diederik P Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes. arXiv:1312.6114 [stat.ML]

[7] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. https://doi.org/10.48550/ARXIV.1806.09055

[8] Alexander Ororbia, Ahmed Ahmed Elsaid, and Travis Desell. 2019. Investigating Recurrent Neural Network Memory Structures using Neuro-Evolution. arXiv:1902.02390 [cs.NE]

[9] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[10] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: Stochastic Neural Architecture Search. https://doi.org/10.48550/ARXIV.1812.09926

[11] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2019. Understanding and Robustifying Differentiable Architecture Search. https://doi.org/10.48550/ARXIV.1909.09656

[12] Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. 2016. Minimal Gated Unit for Recurrent Neural Networks. arXiv:1603.09420 [cs.NE]