From Nodes to Networks: Evolving Recurrent Neural Networks

Aditya Rawal
aditya.rawal@uber.com

Risto Miikkulainen
risto@cs.utexas.edu

Abstract

Gated recurrent networks such as those composed of Long Short-Term Memory (LSTM) nodes have recently been used to improve state of the art in many sequential processing tasks such as speech recognition and machine translation. However, the basic structure of the LSTM node is essentially the same as when it was first conceived 25 years ago. Recently, evolutionary and reinforcement learning mechanisms have been employed to create new variations of this structure. This paper proposes a new method, evolution of a tree-based encoding of the gated memory nodes, and shows that it makes it possible to explore new variations more effectively than other methods. The method discovers nodes with multiple recurrent paths and multiple memory cells, which lead to significant improvement in the standard language modeling benchmark task. Remarkably, this node did not perform well in another task, music modeling, but it was possible to evolve a different node that did, demonstrating that the approach discovers customized structure for each task. The paper also shows how the search process can be speeded up by training an LSTM network to estimate performance of candidate structures, and by encouraging exploration of novel solutions. Thus, evolutionary design of complex neural network structures promises to improve performance of deep learning architectures beyond human ability to do so.

1 Introduction

Recent studies on metalearning methods such as neural architecture search and evolutionary optimization have shown that LSTM performance can be improved by complexifying it further (Zoph & Le, 2016; Miikkulainen et al., 2018). This paper develops a new method along these lines, recognizing that a large search space where significantly more complex node structures can be constructed could be beneficial. The method is based on a tree encoding of the node structure so that it can be efficiently searched using genetic programming. Indeed, the approach discovers significantly more complex structures than before, and they indeed perform significantly better: Performance in the standard language modeling benchmark, where the goal is to predict the next word in a large language corpus, is improved by 6 perplexity points over the standard LSTM (Zaremba et al., 2014), and 0.9 perplexity points over reinforcement-learning based neural architecture search (Zoph & Le, 2016).

These improvements are obtained by constructing a homogeneous layered network architecture from a single gated recurrent node design. A second innovation in this paper shows that further improvement can be obtained by constructing such networks from multiple different designs. As a first step, allocation of different kinds of LSTM nodes into slots in the network is shown to improve perfor-

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

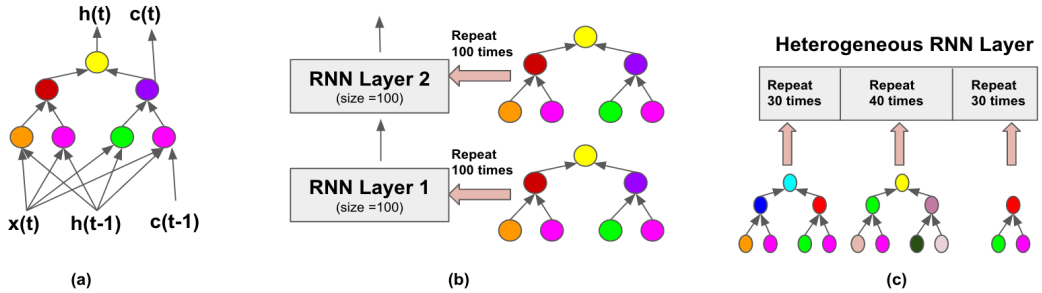

Figure 1: (a)Tree based representation of the recurrent node. Tree outputs $h(t)$ and $c(t)$ are fed as inputs in the next time step. (b) In standard recurrent network, the tree node is repeated several times to create each layer in a multi-layered network. Different node colors depict various element activations. (c) The heterogeneous layer consists of different types of recurrent nodes.

mance by another 0.5 perplexity points. This result suggests that further improvements are possible with more extensive network-level search.

A third contribution of this paper is to show that evolution of neural network architectures in general can be speeded up significantly by using an LSTM network to predict the performance of candidate neural networks. After training the candidate for a few epochs, such a Meta-LSTM network predicts what performance a fully trained network would have. That prediction can then be used as fitness for the candidate, speeding up evolution fourfold in these experiments. A fourth contribution is to encourage exploration by using an archive of already-explored areas (described in Appendix). The effect is similar to that of novelty search, but does not require a separate novelty objective, simplifying the search.

Interestingly, when the recurrent node evolved for language modeling was applied to another task, music modeling, it did not perform well. However, it was possible to evolve another solution for that task that did. As a fifth contribution, the results in this paper demonstrate that it is not simply the added complexity in the nodes that matter, but that it is the right kind, i.e. complexity customized for each task.

Thus, evolutionary optimization of complex deep learning architectures is a promising approach that can yield significant improvements beyond human ability to do so.

2 Methods

Evolving recurrent neural networks is an interesting problem because it requires searching the architecture of both the node and the network. As shown by recent research (Zoph & Le, 2016) (Zilly et al., 2016), the recurrent node in itself can be considered a deep network. In this paper, Genetic Programming (GP) is used to evolve such node architectures. First, the overall network architecture is fixed i.e. constructed by repeating a single evolved node to form a layer (Figure1(b)). Second, a heterogeneous network is evolved by combining several different types of nodes into a layer (Figure1(c)). In the future more complex coevolution approaches are also possible.

Evaluating the evolved node and network is costly. Training the network for 40 epochs takes two hours on a 1080 NVIDIA GPU. A sequence to sequence model called meta-LSTM is developed to speed up evaluation.

2.1 Genetic Programming for Recurrent Nodes

As shown in Figure1(a), a recurrent node can be represented as a tree structure, and GP can therefore be used to evolve it. However, standard GP may not be sufficiently powerful to do it. In particular, it does not maintain sufficient diversity in the population. Similar to the GP-NEAT approach by Tujillo et al. (2015), it can be augmented with ideas from NEAT speciation.

There are three kind of mutation operations in the experiments: (1) Mutation to randomly replace an element with an element of the same type, (2) Mutation to randomly inserts a new branch at a random position in the tree. The subtree at the chosen position is used as child node of the newly

created subtree. (3) Mutation to shrink the tree by choosing a branch randomly and replacing it with one of the branch's arguments (also randomly chosen). Speciation and crossover are described in Section A.5.

2.2 Meta-LSTM for Fitness Prediction

To overcome costly evaluation and to speed up evolution, a Meta-LSTM framework for fitness prediction was developed. Meta-LSTM is a sequence to sequence model (Sutskever et al., 2014) that consists of an encoder LSTM and a decoder LSTM. Validation perplexity of the first 10 epochs is provided as sequential input to the encoder, and the decoder is trained to predict the validation loss at epoch 40. Training data for these models is generated by fully training sample networks (i.e. until 40 epochs). The loss is the mean absolute error percentage at epoch 40. The hyperparameter values of the Meta-LSTM were selected based on its performance in the validation dataset.

Recent approaches to network performance prediction include Bayesian modeling (Klein et al. (2017)) and regression curve fitting (Baker et al., 2017). The learning curves for which the above methods are deployed are much simpler as compared to the learning curves of structures discovered by evolution. Note that Meta-LSTM is trained separately and only deployed for use during evolution. Thus, networks can be partially trained with a $4\times$ speedup, and assessed with near-equal accuracy as with full training.

3 Experiments

Neural architectures were constructed for the language modeling task, using Meta-LSTM as the predictor of training performance. In the first experiment described below, homogeneous networks were constructed from single evolved nodes, and in the second, heterogeneous networks that consisted of multiple evolved nodes (see Section A.2 for heterogeneous network evolution).

The best evolved architecture for language modeling is then transferred to another sequence modeling task - music prediction.

3.1 Natural Language Modeling Task

Experiments focused on the task of predicting the next word in the Penn Tree Bank corpus (PTB), a well-known benchmark for language modeling (Marcus et al., 1993). LSTM architectures in general tend to do well in this task, and improving them is difficult (Zaremba et al., 2014; Jozefowicz et al., 2015; Gal, 2015). Network training setup during architecture search and subsequent hyperparameter tuning is the same as in Zoph & Le (2016).

The best evolved node is shown Figure2. As shown in Table 1, the evolved node (called GP Node evolution in the table) achieves a test performance of 68.2 for 20 Million parameter configuration on Penn Tree Bank. This is 2.8 perplexity points better than the test performance of the node discovered by NAS (Zoph(2016) in the table) in the same configuration. Evolved node also outperforms NAS in the 32 Million configuration (68.1 v/s. 66.5). Recent work has shown that sharing input and output embedding weight matrices of neural network language models improves performance (Press & Wolf, 2016). The experimental results obtained after including this method are marked as shared embeddings in Table 1.

3.2 Music Modeling Task

Similar to natural language modeling, this task involves next-step musical note prediction. The input to the network is a piano-roll representation and its output is of the same form. The dataset piano-midi.de is used as the benchmark data. This dataset holds 307 pieces of classical piano music from various composers.

Note, this setup is similar to that of Ycart & Benetos (2017). The goal of this experiment is not to achieve state-of-the-art results but to perform apples-to-apples comparison between LSTM nodes and evolved nodes (discovered for language) in a new domain i.e. music.

In this transfer experiment, three networks were constructed: the first with LSTM nodes, the second with NAS nodes, and the third with evolved nodes. All the three networks were trained under the

Table 1: Single Model Perplexity on Test set of Penn Tree Bank. Node evolved using GP outperforms the node discovered by NAS (Zoph & Le, 2016) in various configurations.

Model	Parameters	Test Perplexity
Gal (2015) - Variational LSTM	66M	73.4
Zoph & Le (2016)	20M	71.0
GP Node Evolution	20M	68.2
Zoph & Le (2016)	32M	68.1
GP Node Evolution	32M	66.5
Zilly et al. (2016) , shared embeddings	24M	66.0
Zoph & Le (2016), shared embeddings	25M	64.0
GP Evolution, shared embeddings	25M	63.0
Heterogeneous, shared embeddings	25M	62.2
Zoph & Le (2016), shared embeddings	54M	62.9

Table 2: F1 scores computed on Piano-Midi dataset. LSTM outperforms both the evolved node and NAS node for language, but not the node evolved specifically for music, demonstrating that the approach discovers solutions customized for the task.

Model	F1 score
LSTM	0.548
Zoph & Le (2016)	0.48
GP Evolution (Language)	0.49
GP Evolution (Music)	0.599

same setting as described in Ycart & Benetos (2017). The F1 score of each of the three models is shown in Table 2. LSTM nodes outperform both NAS and evolved nodes. This result is interesting because both NAS and evolved nodes significantly outperformed LSTM nodes in the language-modeling task. This result suggests that NAS and evolved nodes are custom solution for a specific domain, and do not necessarily transfer to other domains. However, the framework developed for evolving recurrent nodes for natural language can be transferred to the music domain as well. The setup is the same i.e. at each generation a population of recurrent nodes represented as trees will be evaluated for their performance in the music domain. The validation performance of the network constructed from the respective tree node will be used as the node fitness. The performance measure of the network in music domain is the F1 score, therefore, it is used as the network fitness value.

The evolution parameters are the same as those used for language modeling. The results from evolving custom node for music are shown in Table 2. The custom node (GP Evolution (Music)) achieves an improvement of five points in F1 score over LSTM (Figure 3). Thus, evolution was able to discover custom structure for the music modeling domain as well.

4 Discussion and Future Work

Population based architecture search as presented in this paper is powerful since it can harness more extensive exploration than other meta-learning techniques such as reinforcement learning, Bayesian parameter optimization, and gradient descent. Remarkably, the node that performed well in language modeling performed poorly in music modeling, but evolution was able to discover a different node that performed well in music. Apparently, the approach discovers regularities in each task and develops node structures that take advantage of them, thus customizing the nodes separately for each domain. Analyzing what those regularities are and how the structures encode them is an interesting direction of future work.

The GP-NEAT evolutionary search method in this paper is run in the same search space used by NAS (Zoph & Le, 2016), resulting in significant improvements. In a recent paper (Pham et al., 2018), the NAS search space was extended to include recurrent highway connections as well, improving the results further. An interesting direction of future work is thus to extend the GP-NEAT search space in a similar manner; similar improvements should result.

References

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Practical neural network performance prediction for early stopping. *CoRR*, abs/1705.10823, 2017. URL http://arxiv.org/abs/1705.10823.

F. Francone, M. Conrads, W. Banzhaf, and P. Nordin. Homologous crossover in genetic programming. In *GECCO*, 1999.

Y. Gal. A theoretically grounded application of dropout in recurrent neural networks. 2015.

R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *In Proceedings of the 32nd International Conference on Machine Learning*, pp. 2342–2350, 2015.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *ICLR*, 2017.

J. Lehman. Evolution through the search for novelty. 2012.

M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2), 1993.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. In Robert Kozma, Cesare Alippi, Yoonsuck Choe, and Francesco Carlo Morabito (eds.), *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam: Elsevier, 2018. URL http://nn.cs.utexas.edu/?miikkulainen:chapter18.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing, 2018. URL http://arxiv.org/abs/1802.03268. cite arxiv:1802.03268.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint*, arxiv/1608.05859, 2016.

I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

L. Tujillo, L. Munoz, E. Lopez, and S.Silva. neat genetic programming: Controlling bloat naturally. In *Information Sciences*, 2015.

Adrien Ycart and Emmanouil Benetos. A study on lstm networks for polyphonic music sequence modelling. In *ISMIR*, 2017.

W. Zaremba, I. Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint*, arxiv/1409.2329, 2014.

Julian G. Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *CoRR*, abs/1607.03474, 2016. URL http://arxiv.org/abs/1607.03474.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. 2016. URL https://arxiv.org/pdf/1611.01578v1.pdf.

A Appendix

A.1 Evolved Solutions

The best evolved recurrent node for language modeling is showing in Figure 2 and the best evolved node for music is shown in Figure 3.

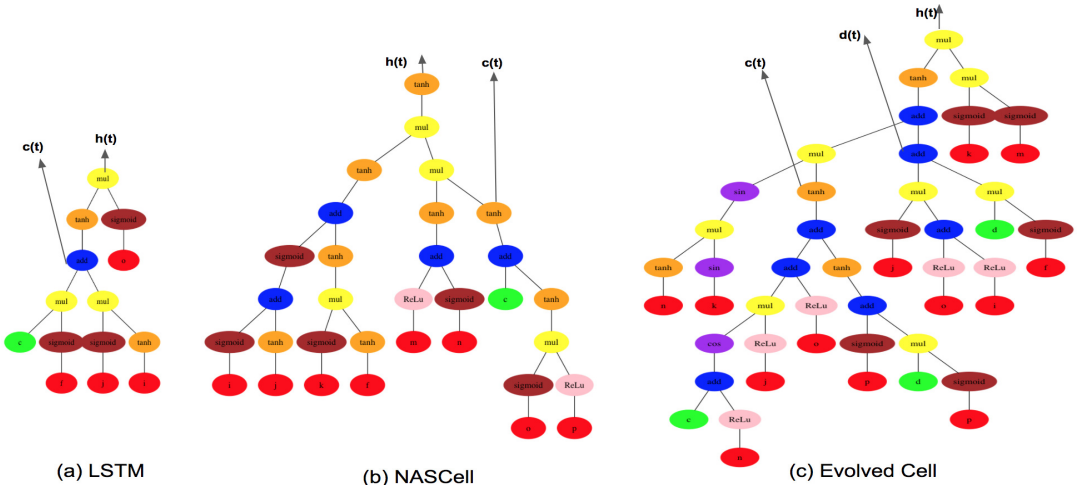

Figure 2: (a) Comparing Evolved recurrent node with NASCell and LSTM. The green input elements denote the native memory cell outputs from the previous time step (c, d). The red colored inputs are formed after combining the node output from the previous time step $h(t-1)$ and the new input from the current time step $x(t)$. In all three solutions, the memory cell paths include relatively few non-linearities. The evolved node utilizes the extra memory cell in different parts of the node. GP evolution also reuses inputs unlike the NAS and LSTM solution. Evolved node also discovered LSTM like output gating.

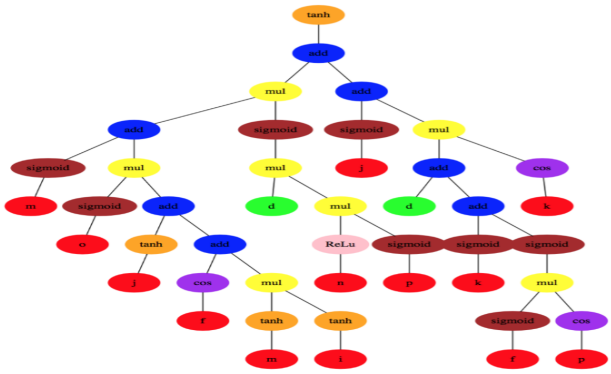

Figure 3: Evolved Node for Music. The node evolved to solve the music task is very different from the node for the natural language task. For example, this node only uses a single memory cell (green input d in the figure) unlike the language node that used both c and d. This results indicates that 'architecture does matter' and that custom evolved solution perform better than hand-designed ones.

A.2 Evolving Heterogeneous Recurrent Networks

Top 10% of the population from 10 runs of Experiment 1 was collected into a pool 100 nodes. Out of these, 20 that were the most diverse, i.e. had the largest tree distance from the others, were selected for constructing heterogeneous layers (as shown in Figure1(c)). Nodes were chosen from this pool randomly to form 2000 such networks. Meta-LSTM was again used to speed up evaluation.

After hyperparameter tuning, the best network (for 25 Million parameter configuration)achieved a perplexity of 62.2, i.e. 0.8 better than the homogeneous network constructed from the best evolved node. This network is also 0.7 perplexity point better than the best NAS network double its size (54 Million parameters). Interestingly, best heterogeneous network was also found to be more robust to hyperparameter changes than the homogeneous network. This result suggests that diversity not only improves performance, but also adds flexibility to the internal representations. The heterogeneous network approach therefore forms a promising foundation for future work, as discussed next.

A.3 Search Space: Node

GP evolution of recurrent nodes starts with a simple fully connected tree. During the course of evolution, the tree size increases due to insert mutations and decreases due to shrink mutations. The maximum possible height of the tree is fixed at 15. However, there is no restriction on the maximum width of the tree.

The search space for the nodes is more varied and several orders of magnitude larger than in previous approaches. More specifically, the main differences from the state-of-the-art Neural Architecture Search (NAS) (Zoph & Le, 2016) are: (1) NAS searches for trees of fixed height 10 layers deep; GP searches for trees with height varying between six (the size of fully connected simple tree) and 15 (a constraint added to GP). (2) Unlike in NAS, different leaf elements can occur at varying depths in GP. (3) NAS adds several constraint to the tree structure. For example, a linear element in the tree is always followed by a non-linear element. GP prevents only consecutive non-linearities (they would cause loss of information since the connections within a cell are not weighted). (4) In NAS, inputs to the tree are used only once; in GP, the inputs can be used multiple times within a node.

Most gated recurrent node architectures consist of a single native memory cell (denoted by output c in Figure1(a)). This memory cell is the main reason why LSTMs perform better than simple RNNs. One key innovation introduced in this paper is to allow multiple native memory cells within a node. The memory cell output is fed back as input in the next time step without any modification, i.e. this recurrent loop is essentially a skip connection. Adding another memory cell in the node therefore does not effect the number of trainable parameters: It only adds to the representational power of the node.

A.4 Search Space: Network

Standard recurrent networks consist of layers formed by repetition of a single type of node. However, the search for better recurrent nodes through evolution often results in solutions with similar task performance but very different structure. Forming a recurrent layer by combining such diverse node solutions is potentially a powerful idea, related to the idea of ensembling, where different models are combined together to solve a task better.

In this paper, such heterogenous recurrent networks are constructed by combining diverse evolved nodes into a layer (Figure1(c)). A candidate population is created that consists of top-performing evolved nodes that are structurally very different from other nodes. The structure difference is calculated using the tree distance formula detailed previously. Each heterogenous layer is constructed by selecting nodes randomly from the candidate population. Each node is repeated 20 times in a layer; thus, if the layer size is e.g. 100, it can consist of five different node types, each of cardinality 20.

The random search is an initial test of this idea. In the future the idea is to search for such heterogenous recurrent networks using a genetic algorithm as well.

A.5 Speciation and Crossover

One-point crossover is the most common type of crossover in GP. However, since it does not take into account the tree structure, it can often be destructive. An alternative approach, called homologous crossover (Francone et al., 1999), is designed to avoid this problem by crossing over the common regions in the tree. Similar tree structures in the population can be grouped into species, as is often done in NEAT (Tujillo et al., 2015). Speciation achieves two objectives: (1) it makes homologous crossover effective, since individuals within species are similar, and (2) it helps keep the population diverse, since selection is carried out separately in each species. A tree distance metric proposed by Tujillo et al. (2015) is used to determine how similar the trees are.

$$\delta(T_i, T_j) = \beta \frac{N_{i,j} - 2n_{S_i,j}}{N_{i,j} - 2} + (1 - \beta) \frac{D_{i,j} - 2d_{S_i,j}}{D_{i,j} - 2}, \tag{1}$$

where:

n_{T_x} = number of nodes in GP tree T_x,

d_{T_x} = depth of GP tree T_x,

$S_{i,j}$ = shared tree between T_i and T_j,

$N_{i,j} = n_{T_i} + n_{T_j}$,

$D_{i,j} = d_{T_i} + d_{T_j}$,

$\beta \in [0, 1]$,

$\delta \in [0, 1]$.

On the right-hand side of Equation 1, the first term measures the difference with respect to size, while the second term measures the difference in depth. Thus, setting $\beta = 0.5$ gives an equal importance to size and depth. Two trees will have a distance of zero if their structure is the same (irrespective of the actual element types).

In most GP implementations, there is a concept of the left and the right branch. A key extension in this paper is that the tree distance is computed by comparing trees after all possible tree rotations, i.e. swaps of the left and the right branch. Without such a comprehensive tree analysis, two trees that are mirror images of each other might end up into different species. This approach reduces the search space by not searching for redundant trees. It also ensures that crossover can be truly homologous Figure4 (a).

The structural mutations in GP, i.e. insert and shrink, can lead to recycling of the same strcuture across multiple generations. In order to avoid such repetitions, an archive called Hall of Shame is maintained during evolution (Figure4(b)). This archive consists of individuals representative of stagnated species, i.e. regions in the architecture space that have already been discovered by evolution but are no longer actively searched. During reproduction, new offsprings are repeatedly mutated until they result in an individual that does not belong to Hall of Shame. Mutations that lead to Hall of Shame are not discarded, but instead used as stepping stones to generate better individuals. Such memory based evolution is similar to novelty search. However, unlike novelty search (Lehman, 2012), there is no additional fitness objective, simply an archive.

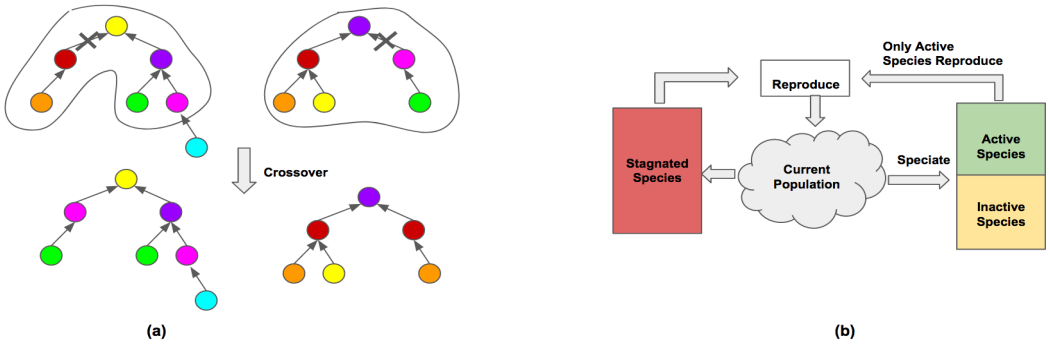

Figure 4: (a) Homologous crossover in GP - the two trees on the top look different but in-fact they are almost mirror images of each other. These two trees will therefore belong in the same species. The line drawn around the trees marks the homologous regions between the two. A crossover point is randomly selected and one point crossover is performed. The bottom two networks are the resultant offsprings. (b) An archive of stagnant species called Hall of Shame (shown in red) is built during evolution. This archive is looked up during reproduction, to make sure that newly formed offsprings do not belong to any of the stagnant species. At a time, only 10 species are actively evaluated (shown in green). This constraint ensures that active species get enough spawns to ensure a comprehensive search in its vicinity before it is added to the Hall of Shame. Offsprings that belong to new species are pushed into a inactive species list (shown in yellow) and are only moved to the active list whenever an active species moves to Hall of Shame.

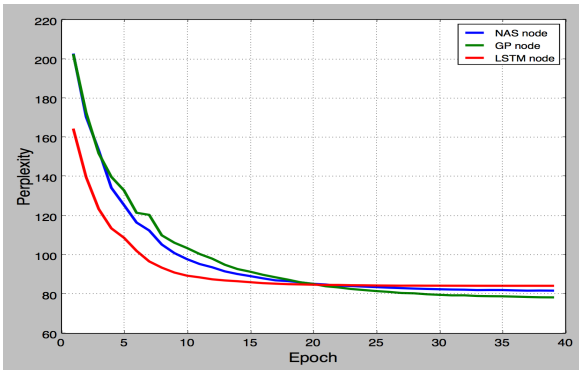

Figure 5: Learning curve comparison of LSTM node, NAS node and GP nodes. Y-axis is the validation perplexity (lower is better) and X-axis is the epoch number. Notice that LSTM node learns quicker than the other two initially but eventually settles at a larger perplexity value. This graph demonstrates that the strategy to determine network fitness using partial training (say based on epoch 10 validation perplexity) is faulty. A fitness predictor model like Meta-LSTM can overcome this problem.