FENAS: Flexible and Expressive Neural Architecture Search

Ramakanth Pasunuru and **Mohit Bansal**
UNC Chapel Hill
{ram, mbansal}@cs.unc.edu

Abstract

Architecture search is the automatic process of designing the model or cell structure that is optimal for the given dataset or task. Recently, this approach has shown good improvements in terms of performance (tested on language modeling and image classification) with reasonable training speed using a weight sharing-based approach called Efficient Neural Architecture Search (ENAS). In this work, we propose a novel architecture search algorithm called Flexible and Expressible Neural Architecture Search (FENAS), with more flexible and expressible search space than ENAS, in terms of more activation functions, input edges, and atomic operations. Also, our FENAS approach is able to reproduce the well-known LSTM and GRU architectures (unlike ENAS), and is also able to initialize with them for finding architectures more efficiently. We explore this extended search space via evolutionary search and show that FENAS performs significantly better on several popular text classification tasks and performs similar to ENAS on standard language model benchmark. Further, we present ablations and analyses on our FENAS approach.

1 Introduction

Architecture search enables automatic ways of finding the best model architecture and cell structures for the given task or dataset, as opposed to the traditional approach of manually tuning among different architecture choices. Recently, this idea has been successfully applied to the tasks of language modeling and image classification (Zoph and Le, 2017; Zoph et al., 2018; Cai et al., 2018; Liu et al., 2018a,b). The first approach of architecture search involved an RNN controller which samples a model architecture and uses the validation performance of this architecture trained on the given dataset as feedback (or reward) to sample the next architecture. However, this process is computationally very expensive, making it infeasible to run on a single GPU in a reasonable amount of time. Some recent attempts have made architecture search more computationally feasible (Negrinho and Gordon, 2017; Baker et al., 2017), with further performance improvements by Pham et al. (2018) who introduced Efficient Neural Architecture Search (ENAS) and achieved strong results on language modeling and image classification tasks.

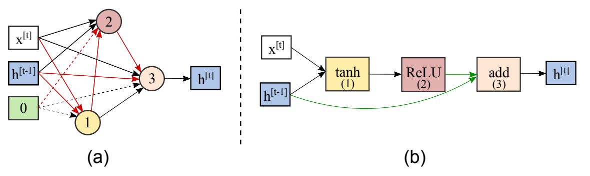


Figure 1: An example showing the recurrent cell in FENAS search space. (a) The DAG represents a recurrent cell with the red edges representing the flow of information; (b) The recurrent cell constructed from (a).

In this work, we present a new architecture search approach called Flexible and Expressible Neural Architecture Search (FENAS) with less restrictive and more flexible search space than ENAS. FENAS search space has more number of activation functions (e.g., skip-based $\tanh$, ReLU) and new atomic-level operations (e.g., addition, element-wise multiplication), as shown in Fig. 1. Importantly, unlike ENAS, FENAS can represent previous well-known human-designed architectures such as the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) in its search space, allowing it to have flexible number of input edges. Unlike ENAS, we do not use weight-sharing strategy during the architecture search, but instead use evolutionary search (Real et al., 2019) and initialize the population with known human-designed RNN architectures to search the space efficiently.

We conduct several experiments on a standard language modeling benchmark (PTB) and text classification tasks from GLUE benchmark (Wang et al., 2019). To the best of our knowledge, we are the first ones to compare NAS methods on the full GLUE benchmark. Comparing our FENAS approach with the previous NAS approaches, FENAS performs similarly on PTB and significantly better on several downstream GLUE tasks. Finally, we provide various advantages of FENAS over ENAS, and also analyze the learned FENAS cell structure for PTB, e.g., learned cell has fewer skip-connections and less network complexity.

2 Related Work

Neural architecture search (NAS) (Zoph and Le, 2017) has been shown to achieve better performance than the human-designed deep networks for image classification (Liu et al., 2018b; Ahmed and Torresani, 2018; Chen et al., 2018; Liu et al., 2019; Ying et al., 2019; Hu et al., 2019; Cai et al., 2019; Xie et al., 2019) and language modeling (Zoph and Le, 2017; Pham et al., 2018; Liu et al., 2019; Cai et al., 2018; Li and Talwalkar, 2019). Several sampling strategies have been explored for finding the NAS optimal cell in the context of reinforcement learning (Zoph and Le, 2017; Zoph et al., 2018; Pham et al., 2018), evolutionary algorithms (Xie and Yuille, 2017; Real et al., 2017; Lu et al., 2019; Miikkulainen et al., 2019; Real et al., 2019; So et al., 2019), and performance predictors (Rusu et al., 2016).

Early NAS approach (Zoph and Le, 2017) took many days and thousands of GPU hours to train on simple datasets like PTB (Marcus et al., 1994) and CIFAR-10 (Krizhevsky and Hinton, 2009). Recently, a weight-sharing strategy among search space parameters has been proposed by Pham et al. (2018), which reduced resource requirements to a few GPU days. Later, various variations of this approach have been proposed, e.g., Liu et al. (2019) replaced RL with gradient descent. Li and Talwalkar (2019) and Sciuto et al. (2020) showed that a simple random search approach can give best results. In our work, we propose a new approach with a better search space making NAS flexible and expressible in comparison to Pham et al. (2018).

3 FENAS Method Details

Similar to ENAS, our method has two stages. In stage-1, we search for an optimal cell, and in stage-2, we train a model using the optimal cell structure. For the rest of this section, we describe our method and the search approach for learning optimal cell.

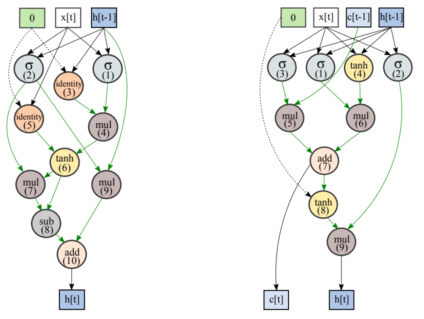
3.1 Search Space

ENAS's search space is restrictive, i.e., every node has only one input from the previous nodes (we refer to Pham et al. (2018) for more details). In our work, we introduce Flexible and Expressive Neural Architecture Search (FENAS), which assumes that every node has one or two inputs from the previous nodes and also has three levels of operational functions (details in next paragraph), and hence is more flexible and expressible than the ENAS. Next, we describe the FENAS cell in detail.

At a structural level, FENAS is similar to ENAS cell, where it has edges that represent weights and nodes which represent functions (see Fig. 1). Unlike the ENAS cell, FENAS has more number of node functions which are divided into three types: (1) atomic functions ($\mathrm{addition}$, $\mathrm{subtraction}$, and $\mathrm{element\text{-}wise\text{-}product}$); (2) activation functions (tanh, ReLU, $\mathrm{identity}$, and $\mathrm{sigmoid}$); and (3) skip-based activation functions ($\mathrm{tanh\text{-}skip}$, $\mathrm{ReLU\text{-}skip}$, and $\mathrm{sigmoid\text{-}skip}$). In comparison to ENAS, atomic functions and skip connection-based activation functions are new in FENAS. Note that ENAS uses skip connections at every computational node, whereas we allow FENAS to self-learn which computational nodes require skip connections. Edge weights are used when the nodes choose skip-activation functions. Nodes with activation functions can choose to have edge weights or not (means just identity function); in other cases, edge weights are replaced with the identity function (these are green edges in Fig 1(b)). Let $x(t)$ and $h(t-1)$ be the inputs to the FENAS cell at time step t, and $h(t)$ is the corresponding output from the FENAS cell. Let h_k^t be the node k output at time step t of the cell. Let h_i^t and h_j^t be the outputs of nodes i and j, where i,$j < k$, then the node functions are described as follows:

- addition ($+$): $h_k^t = h_i^t + h_j^t$
- subtraction ($-$): $h_k^t = h_i^t - h_j^t$
- element-wise-product ($\odot$): $h_k^t = h_i^t \odot h_j^t$
- activations:
  $h_k^t = f_a(w_{i\to k}h_i^t + w_{j\to k}h_j^t + b_{i,j\to k})$
- activations with no edge weights:
  $h_k^t = f_a(h_i^t + h_j^t)$
- skip-activations:
  $\hat{h}_k^t = f_a(w_{i\to k}h_i^t + w_{j\to k}h_j^t + b_{i,j\to k})$
  $c_k^t = \mathrm{sigmoid}(w^c_{i\to k}h_i^t + w^c_{j\to k}h_j^t + b^c_{i,j\to k})$
  $h_k^t = (1 - c_k^t)\cdot(h_i^t + h_j^t) + c_k^t \cdot \hat{h}_k^t$

where, f_a is any of the four activation func-



(a) GRU cell (b) LSTM cell

Figure 2: GRU/LSTM represented in FENAS space.

tions (ReLU, tanh, identity, sigmoid), and $w_{i \to k}$ ($w^c_{i \to k}$) and $w_{j \to k}$ ($w^c_{j \to k}$) are the edge weights (skip-weights) from nodes i and j, respectively to node k. FENAS also has an additional 'zero' node so as to allow single input to the node. Hence, every node has one or two input parent nodes (unlike one parent node in ENAS). Architectures with more than two input nodes can be derived by increasing the node count. Also, FENAS architecture is flexible such that its search space contains known architectures. For example, Fig. 2a presents the GRU cell represented in the FENAS search space, where the inputs are $x(t)$ and $h(t-1)$. FENAS's search space requires 10 computational nodes to represent the GRU cell. Note that even though ENAS has two inputs, it cannot represent GRU cell in its search space because of the skip connections and single input to its computational nodes, suggesting that it has a restrictive search space, and our FENAS approach has more expressive power than ENAS. FENAS can also represent the popular LSTM cell (Fig. 2b) by extending to 3-input nodes ($x(t)$, $h(t-1)$, and $c(t-1)$), and two outputs ($h(t)$ and $c(t)$). For this, we allow our approach to consider three inputs and also sample a computational node at the end which represents the output $c(t)$.

3.2 Evolutionary Search for FENAS

In this work, we use evolutionary search (ES) algorithm to find the optimal cell. For this, we follow the approach proposed in the previous work (Real et al., 2019). During the ES, a population of P trained models are kept throughout the search phase, where initially, the population is initialized with random architectures. In this setup, all the architectures that are possible in the FENAS search space are possible and equally likely. At each cycle, we sample S random models from the population where each of them is drawn uniformly at random with replacement. The model with the highest validation fitness in these S models is considered as the next parent. A new architecture is constructed which is a mutation of the selected parent architecture, we call it the child model. In FENAS, the mutation is a simple random change in one of the computational node operation. This child architecture is trained, evaluated, and added to the population. In order to keep the population size fixed, we remove the oldest model in the population when a new child model is added, this process is otherwise called as aging evolution. Real et al. (2019) suggested that aging evolution approach allows to explore the search space better by not focusing on good models too early. After the end of the cycles, the architecture for the best trained model during the whole search process is selected as the optimal.

Another advantage of ES with FENAS is that it has human-designed cells (LSTM and GRU) in its search space, and we can use these architectures as one of the models in the initial population of the ES, to start from a better state (experimental validation in Sec. 5.1). We also tried RL based weight-sharing (WS) strategy similar to ENAS during stage-1, but did not get expected results,[1] partly due to the reasoning discussed in Sciuto et al. (2020) that even though ENAS is computationally very efficient, its WS approach does not converge to local optima.

4 Experimental Setup

4.1 Datasets

Penn Treebank. The Penn Treebank (PTB) is a standard English language modeling benchmark dataset (Marcus et al., 1994). We use the standard pre-processing steps following Zaremba et al. (2014); Pham et al. (2018), which include lowercase, removing numbers and punctuation. The vocabulary size is capped at 10,000 unique tokens.

GLUE Tasks. We choose all the 9 tasks from GLUE benchmark (Wang et al., 2019):[2] QNLI (Rajpurkar et al., 2016), RTE (Dagan et al., 2005; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), MNLI (Williams et al., 2018), WNLI (Levesque et al., 2012), CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), STS-B (Cer et al., 2017), MRPC (Dolan and Brockett, 2005), and QQP.[3] We use the standard splits from

[1]We achieved a test perplexity score of 59.2 with RL search on PTB, while our evolution search (ES) based approach achieved a better test perplexity score of 56.8 (see Table 1).

[2]`https://gluebenchmark.com/tasks`

[3]`https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs`

Architecture	Param (million)	Valid PPL	Test PPL
LSTM+DropConnect (2018)	24	60.7	58.8
LSTM+MoS (2018)	22	58.1	56.0
NAS (2017)	25	N/A	64.0
ENAS (2018)	24	N/A	55.8
ENAS* (2018)	24	68.3	63.1
ENAS† (2019)	24	60.8	58.6
DARTS (2019)	23	58.1	55.7
Random Search WS (2019)	23	57.8	55.5
FENAS (ours)	24	58.9	56.8

Table 1: Results on Penn Treebank (PTB). $\star$ are the results obtained using the code publicly released by Pham et al. (2018). †: results obtained by Liu et al. (2019).

GLUE benchmark (Wang et al., 2019).

4.2 Metrics

For the language modeling tasks, we report the perplexity (PPL) as the performance measure. For GLUE tasks, we report the accuracy for MNLI, QNLI, RTE, WNLI, and SST-2, accuracy and F1 for MRPC and QQP, Matthews correlation (Matthews, 1975) for CoLA, and Pearson/Spearman correlation for STS-B.[4]

4.3 Training Details

In all our experiments, our hyperparameter choices are based on validation perplexity for the language modeling tasks and based on validation accuracy for the text classification tasks. We do not perform any extensive hyperparameter search. We manually tune only dropout in the range [0.1, 0.5] for very few tasks. We use 9 computational nodes in all of our FENAS models. In stage-1, for both tasks, we use evolution search algorithm (Real et al., 2019) with a population size of 100, sample size of 25, and a total of 5000 cycles for learning the FENAS optimal cell structure.

Language Models. In stage-1 evolution search, the child model hidden size and word embedding size are set to 300. We train each child model for 20 epochs with a learning rate of 0.001 using Adam optimizer (Kingma and Ba, 2015). We clip the norm of the gradient at 0.25, use l_2 regularization weighted by 8e-6, tie word embeddings and softmax weights (Inan et al., 2017), and use variational dropout (Gal and Ghahramani, 2016) for both stages. In stage-2, we use a hidden size of 900 and word embedding size of 900, and other settings such as stage-2 optimizer, learning rate, dropout are same as in previous work (Pham et al., 2018).

[4]https://www.scipy.org/index.html

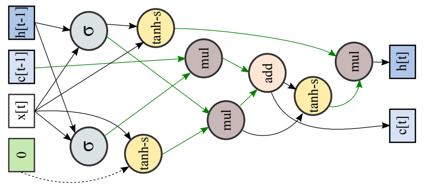


Figure 3: Learned cell structure on PTB dataset.

Text Classification Models. All the baseline models on the GLUE benchmark have same settings apart from the vocabulary size. Each model has a two layer bidirectional LSTM-RNN with a hidden size of 1500, and a word embedding size of 300 which are initialized with glove embeddings. The classifier is an MLP with a hidden size of 256. In all our models, we use Adam optimizer with a learning rate of 0.0001 and a dropout of 0.2, and keep the maximum length of RNN to 50. We use a batch size of 64. We refer to Appendix A for more training details on FENAS and ENAS approaches.

5 Results and Analysis

5.1 Language Model on Penn Treebank

Table 1 presents the performance of various state-of-the-art language models (both manually-designed LSTM-based and architecture search based models) on the standard Penn Treebank (PTB) dataset. ENAS, DARTS, and Random Search WS models use the same weight-sharing strategy with different search approach in the stage-1 to learn the optimal cell. Our FENAS method performs similar w.r.t. these ENAS models.

Computational Complexity. FENAS search space is larger than ENAS because of more activation functions and more inputs to the computational nodes. Stage-1 search process for learning the optimal cell takes 8 and 0.5 GPU days on Nvidia Tesla P100s for FENAS and ENAS, respectively. For stage-2, the training time of FENAS is similar to the ENAS approach.

Random Search Baseline. It has been shown that an architecture sampled uniformly from ENAS search space can also perform reasonably well (Li and Talwalkar, 2019; Liu et al., 2019). In fact, a random search with weight-sharing approach performed best on PTB (see Table 1). For FENAS random baseline, we uniformly sampled 5 random architectures from FENAS search space and trained them on PTB. The average perplexity of these 5 architectures is 126.67, which is substantially lower

Architecture	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	WNLI	AVG
LSTM	13.9	86.5	77.7	86.7	72.9	66.7	77.0	56.8	56.3	66.1
ENAS-RL	13.8	86.4	76.2	**87.1**	76.8	67.1	78.7	58.5	56.3	66.8
ENAS-RS	15.0	87.1	76.0	85.7	76.2	**67.5**	78.3	58.5	56.3	66.7
FENAS	**17.5**	**87.2**	**78.4**	**87.1**	**77.8**	67.4	**79.2**	**59.9**	**57.7**	**68.0**

Table 2: Results on GLUE task development sets. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson correlation. For CoLA, we report Matthews correlation. For all other tasks we report accuracy.

Architecture	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	WNLI	AVG
LSTM	17.1	86.9	71.0/78.9	83.2/62.7	67.8/65.6	64.9/65.8	77.4	52.1	65.1	64.3
ENAS-RL	14.7	84.1	74.5/82.6	83.8/63.0	72.6/70.7	66.0/66.6	78.5	51.0	65.1	64.8
ENAS-RS	16.7	85.6	73.7/81.6	81.9/61.5	72.5/70.4	66.9/67.5	78.8	53.1	65.1	65.3
FENAS	16.4	86.6	71.0/78.9	84.9/63.7	73.2/71.0	66.6/66.0	79.1	52.7	65.1	65.6

Table 3: Results on GLUE task test sets, obtained from `https://gluebenchmark.com/`.

w.r.t. best learned cell, emphasizing the importance of having a good search algorithm for FENAS.

LSTM-RNN Initialization. In all our models, we use LSTM cell in the initial population of the evolutionary search. To show the advantage of including human-designed cells, we perform an additional experiment where we do not include the LSTM cell, and observe that the search process is 24% slower in finding the best architecture.

Learned Cell Structure. Fig. 3 presents our learned FENAS cell on PTB. This cell has some similar computational nodes as LSTM cell. Interestingly, it does not have any ReLU activation function, unlike ENAS cell (Pham et al., 2018). Also, FENAS cell uses skip connection only 2 times (nodes with '-s'), and have roughly equal number of edges with and without learnable weights, accounting for its low network complexity.

5.2 Text Classification on GLUE Tasks

We move beyond language modeling tasks for NAS research and present novel results for several NAS methods on the full set of more realistic downstream GLUE benchmark tasks. We use the BiLSTM model as discussed in Wang et al. (2019) for all GLUE tasks, and do not include any attention methods or external contextual information to fairly only evaluate the influence of cell structures on model's performance. We replace the LSTM-RNN cell in this BiLSTM model with ENAS and FENAS cells to fairly compare all of them. Table 2 & 3 present the performance of LSTM baseline, and our implementations of ENAS with RL search (ENAS-RL) (Pham et al., 2018) and ENAS with random search (ENAS-RS) (Li and Talwalkar, 2019), and our FENAS on 9 GLUE tasks.[5] We observe that FENAS significantly outperforms ENAS and the LSTM baseline on many GLUE datasets.[6] To the best of our knowledge, this is the first detailed comparison of diverse NAS methods on the full GLUE benchmark and we hope this will encourage further comparison by future work.

Computational Complexity. The search time varies across GLUE tasks, but the average search time is 4 and 0.8 GPU days on Nvidia Tesla P100s for FENAS and ENAS models, respectively.

6 Conclusion

We presented a new architecture search algorithm (FENAS) which has more activation functions and more inputs to the computational nodes than the previous best algorithm (ENAS), thus achieving more flexible and expressible architectures. Our FENAS approach is also able to reproduce the well-known LSTM and GRU architectures, and is also able to initialize with them for finding architectures more efficiently. We also present the first detailed comparison of several NAS methods on the full GLUE benchmark, and achieve significant improvements on several text classification tasks.

Acknowledgments

We thank the reviewers for their helpful comments. This work was supported by DARPA YFA17-D17AP00022, ONR Grant N00014-18-1-2871, Amazon ML Research Award, and Microsoft PhD Fellowship. The views in the article are those of the authors and not of the funding agency.

[5]In terms of parameters, FENAS model size is always lower than ENAS and baseline, more details in Appendix A.

[6]FENAS vs. non-FENAS difference is stat. signif. with $p<0.05$ for CoLA, STS-B, QNLI, and RTE, based on bootstrap test (Noreen, 1989; Efron and Tibshirani, 1994).

References

Karim Ahmed and Lorenzo Torresani. 2018. Maskconnect: Connectivity learning by gradient descent. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 349–365.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *ICLR*.

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, pages 6–4. Venice.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient architecture search by network transformation. In *AAAI*.

Han Cai, Ligeng Zhu, and Song Han. 2019. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. In *SemEval workshop at ACL*.

Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. 2018. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pages 8699–8710.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

Hanzhang Hu, John Langford, Rich Caruana, Saurajit Mukherjee, Eric J Horvitz, and Debadeepta Dey. 2019. Efficient forward architecture search. In *Advances in Neural Information Processing Systems*, pages 10122–10131.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *ICLR*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. In *Conference on Uncertainty in Artificial Intelligence*.

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018a. Progressive neural architecture search. In *ECCV*.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018b. Hierarchical representations for efficient architecture search. In *CVPR*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *ICLR*.

Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. NSGA-NET: a multi-objective genetic algorithm for neural architecture search. In *GECCO*.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics.

Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *ICLR*.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel

Duffy, et al. 2019. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier.

Renato Negrinho and Geoff Gordon. 2017. Deeparchitect: Automatically designing and training deep architectures. In *CVPR*.

Eric W Noreen. 1989. *Computer-intensive methods for testing hypotheses*. Wiley New York.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *ACL*.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. 2020. Evaluating the search phase of neural architecture search. In *ICLR*.

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642.

Alex Wang, Amapreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*.

Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. SNAS: stochastic neural architecture search. In *ICLR*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. 2018. Breaking the softmax bottleneck: A high-rank RNN language model. In *ICLR*.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*.

A More Training Details on Text Classification Models

All the baseline models on the GLUE benchmark have same settings apart from the vocabulary size. Each model has a two layer bidirectional LSTM-RNN with a hidden size of 1500, and a word embedding size of 300 which are initialized with glove embeddings. The classifier is an MLP with a hidden size of 256. In all our models, we use Adam optimizer with a learning rate of 0.0001 and a dropout of 0.2, and keep the maximum length of RNN to 50. We use a batch size of 64. We use a vocabulary size of 5,000 for RTE, STS-B, and CoLA tasks, 40,000 for QQP and MNLI, 30,000 for QNLI, 10,000 for MRPC, 14,300 for SST-2, and 1,300 for WNLI.

For the ENAS models, we use 9 computational nodes and only one RNN layer. We use same settings in both stage-1 and stage-2. We use each model's performance metric as reward for the controller in the stage-1 search process. Rest of the settings are same as the baseline models.

We use different settings for stage-1 and stage-2 of FENAS models. This is because to keep the memory and computational complexity tractable when we do evolutionary search in stage-1, where we sample multiple child models in parallel. In stage-1, we use a hidden size of 1000 for large tasks (QNLI, MNLI, and QQP), and a hidden size of 300 for the rest of the tasks. We observe that the cells learned using models with smaller hidden size in stage-1 can not transfer its best performance to large hidden size models that we use in stage-2, especially for large tasks. For this reason, we use a larger hidden size in stage-1 for large tasks. We further only use 2000 examples in stage-1 for large tasks to find the optimal cell. In stage-2, we keep the hidden size such that the overall model size is lower than that of ENAS and LSTM baseline. We use 9 computational nodes in order to accommodate LSTM architecture in the FENAS search space. Rest of the hyperparameters are same as the ENAS baseline.