# NAS-Bench-NLP: Neural Architecture Search Benchmark for Natural Language Processing

**Nikita Klyuchnikov**[1]*     **Ilya Trofimov**[1]     **Ekaterina Artemova**[2]

**Mikhail Salnikov**[1]     **Maxim Fedorov**[1]     **Evgeny Burnaev**[1]

[1]Skolkovo Institute of Science and Technology,
[2]National Research University Higher School of Economics

## Abstract

Neural Architecture Search (NAS) is a promising and rapidly evolving research area. Training a large number of neural networks requires an exceptional amount of computational power, which makes NAS unreachable for those researchers who have limited or no access to high-performance clusters and supercomputers. A few benchmarks with precomputed neural architectures performances have been recently introduced to overcome this problem and ensure more reproducible experiments. However, these benchmarks are only for the computer vision domain and, thus, are built from the image datasets and convolution-derived architectures. In this work, we step outside the computer vision domain by leveraging the language modeling task, which is the core of natural language processing (NLP). Our main contribution is as follows: we have provided search space of recurrent neural networks on the text datasets and trained 14k architectures within it; we have conducted both intrinsic and extrinsic evaluation of the trained models using datasets for semantic relatedness and language understanding evaluation; finally, we have tested several NAS algorithms to demonstrate how the precomputed results can be utilized. We believe that our results have high potential of usage for both NAS and NLP communities.

## 1 Introduction

NAS has matured as a recognized research field, numbering a few well-known difficulties, including the complicatedness of reproducibility and enormous computation costs. Reproducibility issues with NAS methods arise due to the variance in search spaces and experimental pipelines. Since NAS requires multiple runs of neural network training and black-box optimization, adherence to experimental protocol becomes of critical importance. Yang et al. [44] notice: most researchers do not strictly follow the experimental protocol or do not make enough ablation studies, leaving obscure reasons for the effectiveness of various methods. For instance, Li and Talwalkar [26] have been unable to exactly reproduce state-of-the-art (SOTA) NAS methods due to the lack of source code and data. They have shown that random search with early stopping can achieve performance close to SOTA.

To address the reproducibility issue, a few recent works have proposed benchmarks. A NAS benchmark consists of numerous architectures trained and evaluated on a downstream task, with

---

*corresponding author, e-mail: `nikita.klyuchnikov@skolkovotech.ru`

evaluation results stored for further utilization. The architectures and metrics thereby can be queried from the benchmark, so time- and energy-consuming training procedures can be avoided. Benchmarks play an important role in facilitating NAS research. However, the community still lacks the variety of benchmarks that allow fully reproducible NAS experiments, as Lindauer and Hutter pointed out [27]. This justifies the need for the development of new and diverse NAS benchmarks.

Two largest benchmarks, NAS-Bench-101 [45] and NAS-Bench-201 [11], consist of convolutional and feedforward networks used for computer vision tasks. However, there are no NAS benchmarks that cover recurrent neural networks (RNN) or their modifications and natural language processing tasks. Thus, NAS applications to NLP have attracted fewer studies.

Many applications benefit from novel architectures and innovative design choices. For example, the quality of the core task in the natural language processing domain, language modeling [35, 4], was significantly improved by using Highway layers [21, 48] or tying input and output embeddings [36]. Residual connections [16] and dense connections [19] were adopted from computer vision architectures and impacted on machine translation [7], text classification [43], and machine reading comprehension [40]. Such novel architectures can be mass-produced computationally employing NAS, which designs new architectures via solving an optimization problem. Efficient NAS methods are still to be researched, and in order to facilitate this process, we need to have reproducible benchmarks with well-defined frameworks.

Creating a NAS benchmark is a challenging task; it includes several steps that have to be carefully designed and performed. First, the search space needs to be defined. Second, the datasets have to be selected. Finally, architectures from the search space need to be trained and evaluated according to both the objective function and to the downstream tasks' metrics.

Our contributions are as follows:

1. We have presented **the first RNN-derived NAS benchmark** designed for NLP tasks. Our benchmark has been derived from a **novel search space** that comprises various RNN modifications, including LSTM and GRU cells (Section 3).
2. We have trained **over 14k architectures for the language modeling task**, and assessed the overall quality of each architecture in terms of language modeling and two additional setups. We have conducted an intrinsic evaluation of learned static word embeddings by applying word similarity tests and an extrinsic evaluation by measuring the performance in downstream tasks, such as GLUE benchmarks (Section 4).
3. We have introduced a framework for benchmarking and conducted a thorough comparison of different NAS algorithms within it (Section 5). In particular, the framework provides a convenient proper way to simulate and measure the training wall time of a NAS process.
4. We have **released all learned architectures**, which allows architecture comparison in many aspects. A representative subset of architectures is provided with **metrics for language modeling and downstream tasks**.

The source code and links to precomputed files are available in this repository:
`https://github.com/fmsnew/nas-bench-nlp-release`

## 2 Related work

**NAS benchmarks**

Two earliest benchmarks were released during 2019: NAS-Bench-101 [45] and NAS-HPO-Bench [24]. NAS-Bench-101 [45] comprises 423k unique convolutional architectures trained on CIFAR-10 image dataset. NAS-HPO-Bench [24] latter comprises joint hyperparameters and architecture search space with a total size of 62k unique feedforward architectures and hyperparameters configurations evaluated on four regression datasets. Both benchmarks entirely cover their search spaces within the defined constraints. However, each network cell is of very low complexity, that limits sophisticated feature engineering because instead of using e.g., graph neural networks, one can merely encode architectures with their graphs adjacency matrices.

Two later projects elaborated on the NAS-Bench-101 benchmark. NAS-Bench-201 [11] by Dong and Yang proposed a framework and ran ten popular NAS algorithms within it. This benchmark contains 15.6k architectures trained on three image classification datasets, however, cell sizes are even smaller: they have only five nodes compared to the maximum of seven nodes in NAS-Bench-101. Zela et al.

[47] built a one-shot NAS framework on the top of NAS-Bench-101 that can reuse the underlying computations. Several one-shot NAS methods were adjusted to query approximate instances from NAS-Bench-101. However, this extension works only with sub-spaces of the original NAS-Bench-101 search space.

**NAS for RNN**

A few previous studies have attempted to optimize existing conventional recurrent cells and concluded that they are not necessarily optimal, moreover, the significance of their components is unclear [14, 22].

Greff et al. [14] conducted a study of eight modifications of LSTM architecture (Long-Short Term Memory [18]) on several tasks. They concluded that the ordinary LSTM performs reasonably well on all datasets, and no modification improves its performance significantly. However, some of the modifications also look promising due to the lower number of parameters and similar performance.

Jozefowich et al [22] tried to find an architecture that outperforms LSTM using evolutionary approach applied to GRU (Gated Recurrent Unit [9]) and LSTM. They evaluated over ten thousand modifications and did not find any that would consistently outperform LSTM on various datasets, although a few instances had superior performance on particular datasets. The authors concluded that architectures that dramatically outperform LSTM could not be easily found in the local area around the vanilla configuration.

There are also the works that introduce algorithms to increase efficiency of NAS, in particular they include application of those algorithms for designing RNN cells from scratch [49, 28, 34]. The experimental results have shown that these methods are capable of obtaining competitive results with SOTA.

## 3 Description

The benchmark in this work has the following components in its foundation: datasets, search space, training and evaluation protocols. In this section we describe all of them.

**Datasets**

We use Penn Tree Bank (PTB) [31] to train a sample of networks from the search space. We also use the WikiText-2 [30] dataset to train a stratified subsample of networks based on their performance on the PTB dataset. The second dataset is larger and is more realistic since it preserves the letter case, punctuation, and numbers. Statistics of these datasets are shown in Table 1.

Table 1: Statistics of language modeling datasets.

| Dataset | PTB | WikiText-2 |
|---|---|---|
| Tokens | 1.086M | 2.552M |
| Vocab size | 10000 | 33278 |
| OoV rate[2] | 0.049 | 0.032 |

**Macro-level of the search space (AWD-LSTM)**

The macro-level of each model and the training procedure are borrowed from AWD-LSTM [29] as it has a relatively simple structure and has comparable to SOTA performance. The network consists of three stacked cells with weightdrop regularizations in each and locked dropouts between them and input/output layers, as well as a dropout applied to input embedding (see Figure 1).
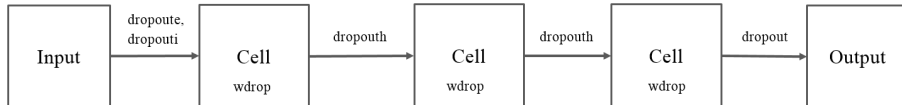


Figure 1: AWD-LSTM macro-level. See definitions of dropouts and other parameters of the architecture in the original repository [1].

**Micro-level of the search space (recurrent cells)**

We define a search space for cells (micro-level of models) to include all conventional recurrent cells as particular instances (see examples in Figure 2). Cell computations are encoded as attributed graphs:

---

[2]Out of Vocabulary (OoV) rate — share of tokens outside of the model's vocabulary; all OoV tokens are replaced with a service token <unknown>.

each node is associated with an operation, and edges encode its inputs. The following operations are available:

- Linear: $f(x_1, .., x_n) = W_1 x_1 + .. + W_n x_n + b$,
- Blending (element wise): $f(z, x, y) = z \odot x + (1 - z) \odot y$,
- Element wise product and sum,
- Activations: Tanh, Sigmoid, and LeakyReLU.

We impose some constraints on possible instances: number of nodes $\leq 24$, number of hidden states $\leq 3$, number of linear input vectors $n \leq 3$.



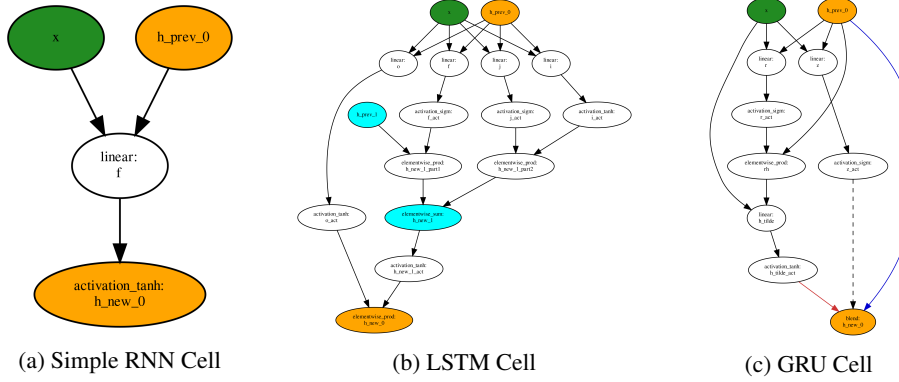(a) Simple RNN Cell      (b) LSTM Cell      (c) GRU Cell

Figure 2: Examples of conventional RNN cells. Colors of nodes highlight the corresponding previous and new hidden states, green color also highlights the input vector. Black dashed, blue and red edges indicate blending arguments $z$, $x$ and $y$ respectively.

**Generation procedure**

We used the following graph generation procedure: initial nodes correspond to the input vector and hidden states; at each step, a node is added, an operation associated to the node is randomly chosen and depending on the operation, connections are made with previous nodes; after all, nodes are added, new hidden states are randomly selected among them; next, redundant nodes, that do not lead to the new hidden states in the computational graph, are removed; finally, the architecture is accepted if the input vector and hidden state nodes are in the graph (not redundant), and no hidden state node is directly connected to the new hidden states in order to avoid numeric explosions.

In addition, we manually added three architectures to the generated sample as baselines: RNN (Simple RNN Cell), LSTM, and GRU (see Figure 2).

**Training process**

We generated 14322 architectures for training on the PTB dataset, 4114 of them were trained three times with different random seeds, the others were trained once; also, 289 out of them were trained on WikiText-2 based on stratified perplexities for PTB.

First, we found a trade-off between training time and validation performance on PTB after 50 iterations by varying sizes of the hidden states and batch size for AWD-LSTM (Figure 3a). The chosen pair was $nhid = 600$ and $batch\_size = 20$ because such a network almost converges to the same perplexity as the original AWD-LSTM, where $nhid = 1150$, but within a half of the original training time. Then we selected a random subset of architectures and performed a grid search of dropout values for each of them on PTB. Figure 3b shows the performance of all configurations for the selected architectures. The following configuration showed the best validation perplexity on average: $dropout = 0.1$, $dropouth = 0.25$, $dropouti = 0.4$, $dropoute = 0.0$, and $wdrop = 0.1$. We used that configuration and fixed other training settings for all architectures on both datasets.

**Evaluation**

We logged the following models metrics for each epoch: wall time and train/validation/test log perplexity $\log\_PPL(p) = -\sum_x p(x) \log p(x)$, where $p$ is a discrete probability distribution of words. In addition, the total number of trainable parameters and their final values were stored to evaluate architectures on downstream tasks.
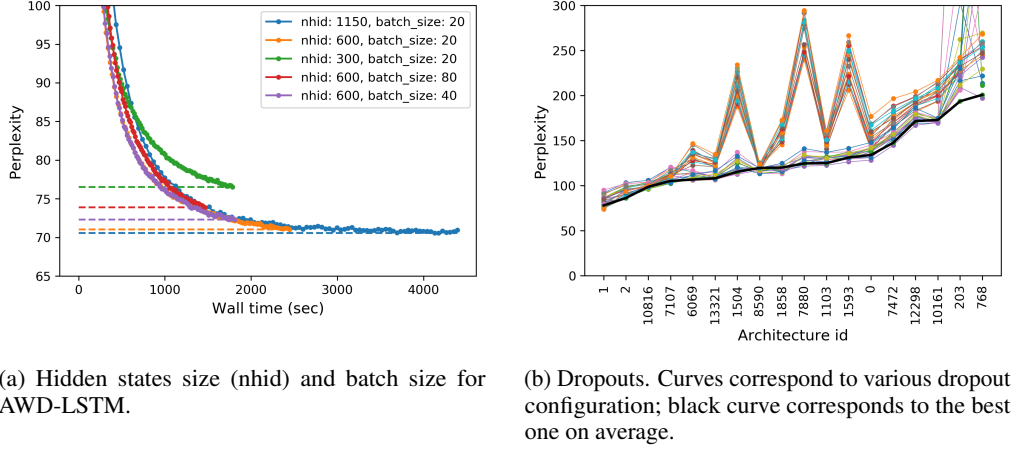
4

(a) Hidden states size (nhid) and batch size for AWD-LSTM.

(b) Dropouts. Curves correspond to various dropout configuration; black curve corresponds to the best one on average.

Figure 3: Hyper-parameters selection.

**Hardware**

We use the following hardware specifications to precompute architectures: HPC cluster Zhores [46] with Tesla V100-SXM2 with 16Gb of memory on each GPU.

# 4 Analysis

## 4.1 Search space evaluation

The complete search space is extremely large; for example, there are approximately $10^{53}$ connected non-attributed graphs with $\leq 24$ nodes and $\leq 3$ input edges.

Some architectures from the generated sample (around 23%) turned out to suffer from numerical explosions that occurred when, for example, there were no activations between corresponding hidden states.

Figure 4a shows the relationship among three metrics: number of parameters, training wall time, and test perplexity. According to the plot, there is no clear correlation between the test perplexity and the number of parameters or training wall time; LSTM and GRU also look like typical representatives of the generated sample. However, based on figure 4b, which shows the distribution of the best test perplexity achieved by each architecture, LSTM and GRU architectures have the top performance in terms of the test perplexity, whereas RNN looks average.
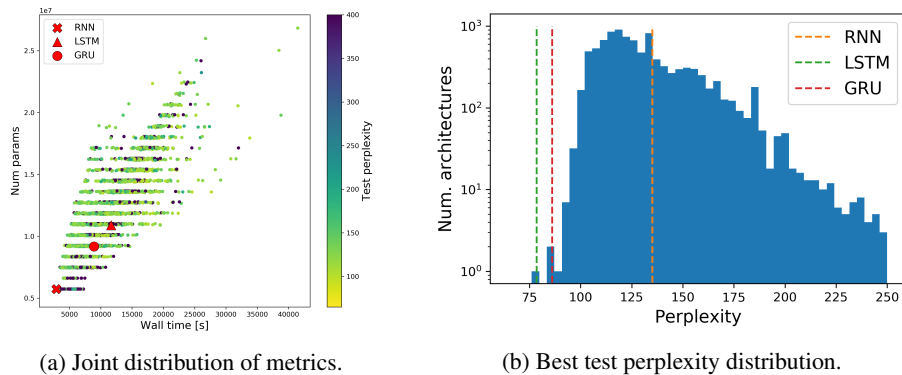


(a) Joint distribution of metrics.

(b) Best test perplexity distribution.

Figure 4: Architectures metrics on PTB.

5

We have investigated how the validation perplexity at different training stages correlates with the final testing perplexity. Figure 5 shows the ranking of architectures based on their final testing perplexities w.r.t. ranking obtained with validation perplexities at 5, 10, 25, and 50 epochs.
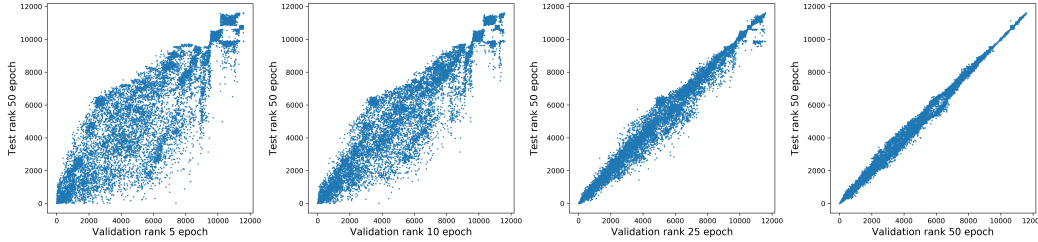


Figure 5: The ranking of architectures perplexity for different epochs on test and validation sets of PTB: lower rank corresponds to lower perplexity.

Figure 6 shows the correlation of architectures performance on PTB and WikiText-2. The plot suggests good transfer properties of NAS for RNN, that is, architectures that perform well on one dataset, will also perform well on another.

We have investigated the sparsity of the generated sample of architectures. Figure 7 shows the histogram of upper-bounds on Graph Edit Distances (GED) between 1000 random pairs of architectures, which also take into consideration the difference in operations associated with each node. To calculate these values, we used consecutive GED approximations [2] (implemented in NetworkX package [15]) for the limited time.

Table 2 compares the top architectures in more detail. LSTM has the best performance, GRU also achieves top-3 performance, however, the generated sample also contains a few competitive examples that are substantially different from LSTM and GRU. Their architectures are provided in Appendix A, Figure 11.

**Parameterization of architectures**

We used graph2vec [33] method to create the characteristic features for architectures, which can be useful for some NAS methods like Bayesian Optimization [13]. With this method, architectures were embedded in 10- and 50-dimensional spaces. In order to verify that the obtained features were sensible, we used them to 1) classify the flawed architectures, the ones that experienced problems during the training; and 2) predict the final testing log perplexity. We split corresponding datasets into equal training and testing parts and used XGBoost [8] for both problems. ROC AUC metric for the classification (1) was 0.98, whereas $R^2$-score for the regression task (2) was 0.012 (see also Figure 8). We also trained and evaluated task (2) on architectures with log perplexity $\leq 6$, and obtained $R^2$-score 0.24.

Table 2: Detailed comparison of top performing architectures and ordinary RNN on PTB dataset.

| Architecture | Num. params | Test perplexity |
|--------------|-------------|-----------------|
| LSTM | 10.9M | 78.5 |
| Top-2 | 9.2M | 84.7 |
| GRU | 9.2M | 86.1 |
| Top-4 | 11.0M | 90.6 |
| RNN | 5.73M | 135.1 |

## 4.2 Word embedding evaluation

Approaches to language model evaluation fall into two major categories: intrinsic and extrinsic evaluation [39]. Intrinsic evaluation tests the semantic relationships between words [3, 32], and, thus leverages word embeddings only. The whole language model is not involved in the evaluation. The first layer solely, i.e., static (fixed) word embeddings, is subjected to evaluation. Extrinsic evaluation treats the language model as dynamic (contextualized) word embeddings and feeds them as an input layer to a downstream task. It measures the performance metrics for the task to assess the quality of the language model. To fully assess trained architectures, we subject them to intrinsic and extrinsic evaluation, following the best practices from the NLP community. For the sake of time, we did not assess all architectures. Instead, we used a stratified sample of architectures according to the perplexity values.
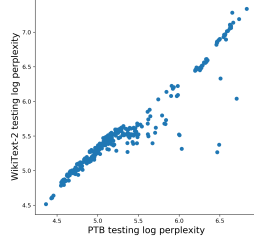
Figure 6: Correlation of architectures' test log perplexities on PTB and WikiText-2.
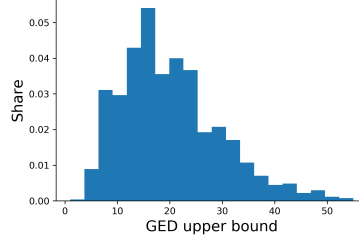


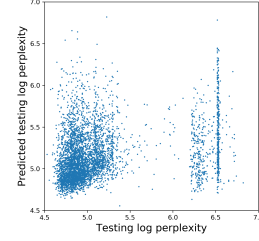Figure 7: Histogram of upper bounds of graph edit distances between 1000 random pairs of architectures.



Figure 8: True vs predicted final testing log perplexity based on graph2vec features.

**Intrinsic evaluation**

We picked two benchmark datasets to evaluate semantic relationships between word pairs: WordSimilarity-353 [12], consisting of similatiry judgments for 353 word pairs on a scale of 0 to 10. For example, word pair "book" and "paper" are scored with 7.5. SimLex-999 [17] is a larger and more challenging benchmark, consisting of similarity judgments for 999 word pairs. To evaluate word embeddings for each word pair, cosine similarity between word vectors is computed. The resulting similarity values are correlated with the judgments using Spearman's and Pearson's correlation coefficients. We used the benchmarks distributed by the gensim framework[3]. To ensure a fair comparison, we train two word2vec models (in particular, Skip-gram with negative sampling, SGNS) [32] on PTB and WikiText-2 independently and use them as baselines for both intrinsic and extrinsic evaluation.

Figure 9 shows the results of intrinsic evaluation. When trained on PTB, most of our architectures overcame SGNS by a large margin, judging from the evaluation on both benchmarks. However, this evaluation is overestimated as the intersection of the PTB vocabulary and both benchmarks are rather small. Similar patterns are observed when both SGNS and our architectures are trained on WikiText-2 and compared on WordSim-353. However, as SimLex-999 is more challenging than WordSim-353, only a half of the architectures manages to beat the SGNS baseline. As expected, in all settings, the lower the perplexity is, the higher the performance of the models is.
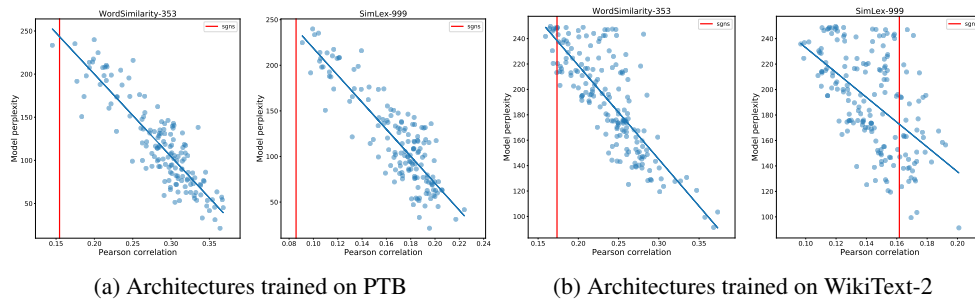


(a) Architectures trained on PTB

(b) Architectures trained on WikiText-2

Figure 9: OX: performance of 150 random architectures on WordSimilarity-353 and SimLex-999 (measured by Pearson correlation coefficient), OY: model perplexety, red line: SGNS performance on WordSimilarity-353 and SimLex-999 (measured by Pearson correlation coefficient).

**Extrinsic evaluation**

We used the General Language Understanding Evaluation (GLUE) benchmark [41], a collection of ten diverse tasks aimed at the evaluation of language models performance. The GLUE score is computed as an average of all performance measures for ten tasks, multiplied by 100.

We follow the GLUE evaluation pipeline [41]: for each task, our architectures encode the input sentences to the vectors, which are passed further to a classifier. We adjust the Jiant toolkit [42] to process our architectures.

---

[3]`https://radimrehurek.com/gensim/`

Finally, we evaluate two baselines: 1) the average bag-of-words using SGNS embeddings and 2) an LSTM encoder with SGNS embeddings in the same setting. When trained on PTB, only a few architectures perform better than simple bag-of-words baselines. None outperforms the LSTM baseline. Due to the small size of these models vocabulary, we do not observe any dependence between the perplexity values and the GLUE score. The architectures that pose a larger vocabulary, trained on WikiText-2, cope with GLUE tasks much better, and almost 20% of them beat both baselines, achieving mean GLUE at the level of 42.

To conclude, our results show that the architectures reach baseline performance and even exceed it in several NLP benchmarks if compared to strong baselines trained under the same conditions. However, these architectures do not achieve the same performance as the recent Transformer-based models, such as BERT [23], T5[37], or ELECTRA [10].

## 5  NAS Benchmark

We prepared an environment that simulates NAS processes and does the proper measurements of metrics. The environment can perform the following tasks:

- train an architecture for the specified number of epochs (the environment automatically simulates checkpoints and continuation of the training process);
- return architecture metrics at the specific training epoch (the architecture must be trained until the requested epoch);
- return total simulated wall time;
- return the testing log perplexity of the best configuration (architecture and epoch) based on validation perplexity.

For benchmarking NAS algorithms on our search space, we tested the following methods within the environment:

- Random Search in two modes: low-fidelity mode (RS 10E), which trains 5X networks for 10 epochs, and high-fidelity mode, which trains 1X networks for 50 epochs (RS 50E);
- Hyperbands (HB) is a feature-agnostic multi-fidelity method [25];
- Bayesian Optimization using 10-dimensional (BO 10D) and 50-dimensional (BO 50D) graph2vec features (Sec. 4, parameterization of architectures) and a bagged [6] XGBoost regressor to estimate uncertainty of predictions;
- Regularized Evolution (RE) [38] using graph2vec features;
- Hyperopt with the Tree-structured Parzen Estimator (TPE); [5];
- SMAC [20] using 10-dimensional graph2vec features.

Performance of each method was measured with regret vs. total training time, where regret $r$ at the moment $t$ is $r(t) = L(t) - L^*$, $L(t)$ is the final testing log perplexity of the best architecture according to validation perplexity found so far by the moment $t$, and $L^*$ is the lowest testing log perplexity in the whole dataset (in our benchmark $L^* = 4.36$, achieved by LSTM architecture). For each method, we report the average regret over 30 trials in Figure 10. Hyperbands achive the lowest final regret, while BO follows next.
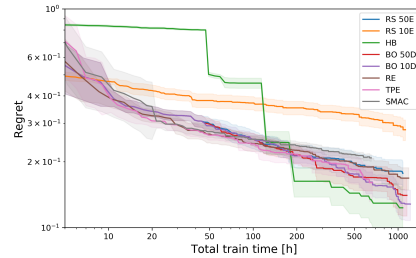


Figure 10: Performance of various NAS methods. Shades of curves correspond to 95%-confidence interval for mean values.

## 6  Discussion

The proposed benchmark is in a different vein than the previous ones. Firstly, it has a much more complex search space, however, at the price of being a very sparse sample from it. Secondly, as the analysis has shown, the distribution of performance metrics (perplexity) is not skewed towards the optimum as in Nas-Bench-101, moreover, hand-crafted architectures like LSTM and GRU seem to have a streamlined performance, hardly achievable by random instances from the search space. We believe these peculiarities of our benchmark will bring diversity and new challenges to the neural architecture search community. For example, larger architectures, that are more realistic, pose a challenge on feature engineering, since simple approaches like flat encoding of adjacency matrices of

architectures' graphs would suffer from the curse of dimensionality. In this work, we used graph2vec approach to obtain better features in a small dimensional space, but we leave a space for further experiments with graph neural networks and other graph-encoding techniques.

## 7 Conclusion

In this work, we introduced a novel benchmark for the search of recurrent neural architectures for language modeling. The complexity of recurrent cells opens new opportunities to experimenting with sophisticated feature engineering methods. With the data we generated, we have found that the performance of architectures highly correlates on different datasets. The results also extend the findings of previous works, that GRU and LSTM architectures generally have the top performance among others. While previously such conclusions were made based on analysis of local neighborhoods of those architectures, our work confirms them on a global scale; however, we have also found a few different architectures with the similar performance. We hope this benchmark will bring new insights regarding the performance of various recurrent architectures and better NAS methods.

## 8 Acknowledgements

## References

[1] Lstm and qrnn language model toolkit. `https://github.com/salesforce/awd-lstm-lm`. Accessed: 2020-06-02.

[2] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, 2015.

[3] M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, 2014.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[5] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[6] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[7] D. Britz, A. Goldie, M.-T. Luong, and Q. Le. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, 2017.

[8] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[9] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[10] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2019.

[11] X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[12] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414, 2001.

[13] P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[14] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[15] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] F. Hill, R. Reichart, and A. Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

[18] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[20] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[21] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[22] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350, 2015.

[23] J. D. M.-W. C. Kenton and L. K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[24] A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

[25] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[26] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[27] M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. *arXiv preprint arXiv:1909.02453*, 2019.

[28] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[29] S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[30] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[31] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[32] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[33] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[34] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[35] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281, 1998.

[36] O. Press and L. Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, 2017.

[37] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[38] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[39] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 298–307, 2015.

[40] Y. Tay, A. T. Luu, S. C. Hui, and J. Su. Densely connected attention propagation for reading comprehension. In *Advances in Neural Information Processing Systems*, pages 4906–4917, 2018.

[41] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, 2018.

[42] A. Wang, I. F. Tenney, Y. Pruksachatkun, P. Yeres, J. Phang, H. Liu, P. M. Htut, , K. Yu, J. Hula, P. Xia, R. Pappagari, S. Jin, R. T. McCoy, R. Patel, Y. Huang, E. Grave, N. Kim, T. Févry, B. Chen, N. Nangia, A. Mohananey, K. Kann, S. Bordia, N. Patry, D. Benton, E. Pavlick, and S. R. Bowman. `jiant` 1.3: A software toolkit for research on general-purpose text understanding models. `http://jiant.info/`, 2019.

[43] S. Wang, M. Huang, and Z. Deng. Densely connected cnn with multi-scale feature attention for text classification. In *IJCAI*, pages 4468–4474, 2018.

[44] A. Yang, P. M. Esperança, and F. M. Carlucci. Nas evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*, 2019.

[45] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114, 2019.

[46] I. Zacharov, R. Arslanov, M. Gunin, D. Stefonishin, A. Bykov, S. Pavlov, O. Panarin, A. Maliutin, S. Rykovanov, and M. Fedorov. "zhores"—petaflops supercomputer for data-driven modeling, machine learning and artificial intelligence installed in skolkovo institute of science and technology. *Open Engineering*, 9(1):512–520.

[47] A. Zela, J. Siems, and F. Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv preprint arXiv:2001.10422*, 2020.

[48] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4189–4198. JMLR. org, 2017.

[49] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

# A    The top architectures

Figure 2 already contains two top architectures - LSTM and GRU. Two more randomly generated architectures with competitive performance (test perplexity) are provided in Figure 11; they are substantially different from LSTM and GRU.
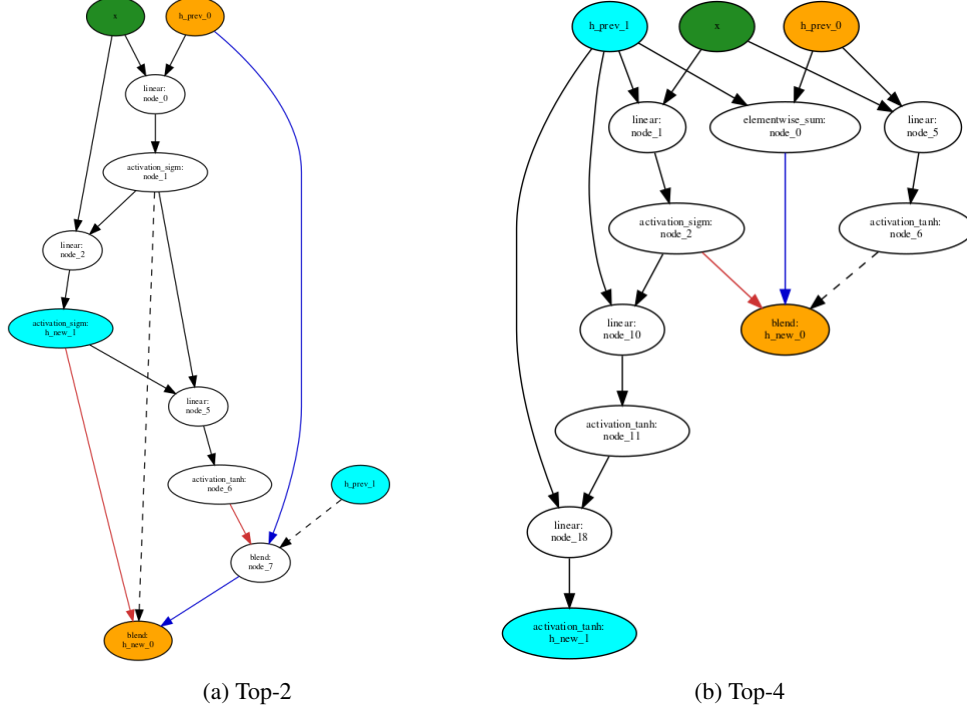


(a) Top-2                                    (b) Top-4

Figure 11: The top non-conventional architectures according to test perplexity.

# B    Additional analysis of NAS methods

Figure 12 is complementary to Figure 10, it shows the cumulative distribution of the final testing regrets among multiple random seeds. BO 50D, TPE and HB manage to find the best architecture (regret = 0) within 1000 hours in approximately 20% runs.
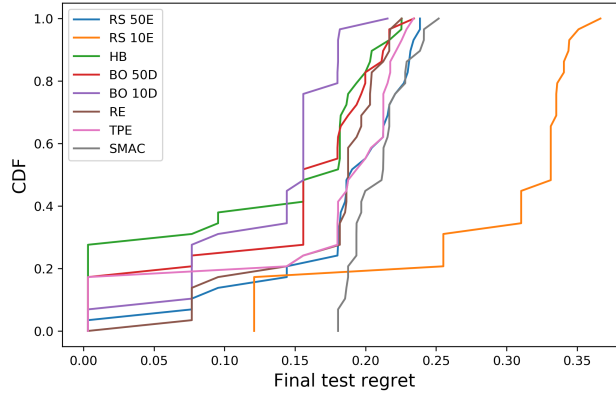


Figure 12: Distribution of the final testing regrets w.r.t. various seeds.

# C  Extended intrinsic evaluation

Figure 13 shows metrics of the intrinsic evaluation of all the 14k architectures (trained on PTB) from the search space. Figure 13b is an extension of the Figure 9, the latter one was calculated based on a stratified sample of the architectures. The red baseline indicates SGNS performance. Most of the architectures overcome this baseline. There are interesting clusters of architectures on Figure 13b that have high perplexity (around 400), yet almost the same correlation scores as architectures with low perplexity (0.15 on average). Such cases can take place due to degenerate cell structures, that do not allow networks to learn predictions well.



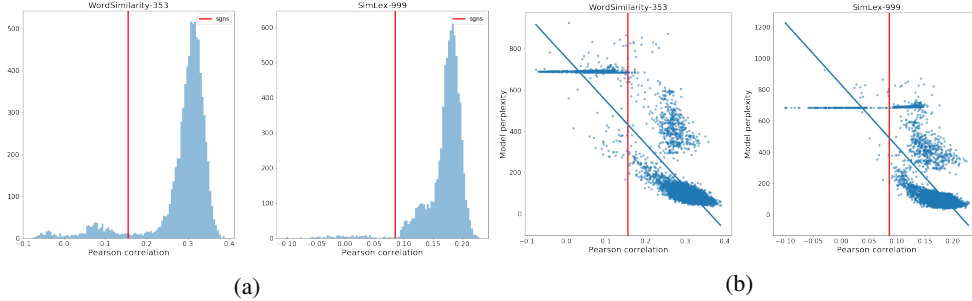(a)                                            (b)

Figure 13: Left: Histograms of WordSimilarity-353 and SimLex-999. Right: Scatter-plots of WordSimilarity-353 and SimLex-999 vs. perplexity. WordSimilarity-353 and SimLex-999 are measured by Pearson correlation coefficient

# D  Detailed extrinsic evaluation

Table 3 shows detailed results of testing 150 random architectures, trained on Wikitext-2, on GLUE benchmark. The results are lower than the baseline, adopted from [41]. The possible explanation for this is due to the vocabulary size. The baseline, which has been trained on the GLUE datasets from scratch, thus possess all GLUE vocabulary. The architectures are pretrained on WikiText-2, hence their vocabulary omits part of the GLUE vocabulary. For some simpler paraphrase tasks, such as QQP and MRPC, the architectures get close to baselines. This evidences the architectures are capable for solving tasks, that require understanding of semantics, but fail to capture more complex phenomena, such as natural language inference (NLI and RTE tasks) and language acceptability (CoLA).

Table 3: GLUE scores on separate tasks. Score shows the average metric on all tasks multiplied by 100.

|  | mean | std | min | median | max | baseline [41] |
|---|---|---|---|---|---|---|
| cola_mcc | 0.000 | 0.005 | -0.046 | 0.000 | 0.025 | 0.35 |
| sst_accuracy | 0.524 | 0.039 | 0.471 | 0.509 | 0.638 | 0.9 |
| mrpc_f1 | 0.805 | 0.036 | 0.501 | 0.812 | 0.817 | 0.84 |
| mrpc_accuracy | 0.680 | 0.024 | 0.478 | 0.684 | 0.711 | 0.78 |
| sts-b_pearsonr | 0.001 | 0.076 | -0.166 | -0.004 | 0.309 | 0.79 |
| sts-b_spearmanr | -0.001 | 0.076 | -0.164 | -0.010 | 0.312 | 0.79 |
| qqp_f1 | 0.532 | 0.086 | 0.000 | 0.542 | 0.613 | 0.66 |
| qqp_accuracy | 0.488 | 0.117 | 0.368 | 0.437 | 0.668 | 0.865 |
| mnli_accuracy | 0.363 | 0.025 | 0.328 | 0.360 | 0.438 | 0.769 |
| qnli_accuracy | 0.503 | 0.012 | 0.494 | 0.496 | 0.545 | 0.798 |
| rte_accuracy | 0.527 | 0.000 | 0.527 | 0.527 | 0.527 | 0.592 |
| wnli_accuracy | 0.527 | 0.058 | 0.338 | 0.563 | 0.634 | 0.651 |
| glue-diagnostic_all_mcc | 0.003 | 0.030 | -0.057 | 0.000 | 0.094 | 0.28 |
| score | 38.1 | 1.8 | 34.7 | 37.9 | 43.6 | 69.1 |