
Neural Architecture Optimization

Renqian Luo^{†*}

University of Science and Technology of China
lrq@mail.ustc.edu.cn

Fei Tian[†]

Microsoft Research
fetia@microsoft.com

Tao Qin

Microsoft Research
taoqin@microsoft.com

Tie-Yan Liu

Microsoft Research
tyliu@microsoft.com

Abstract

Automatic neural architecture design has shown its potential in discovering powerful neural network architectures. Existing methods, no matter based on reinforcement learning or evolutionary algorithms (EA), conduct architecture search in a discrete space, which is highly inefficient. In this paper, we propose a simple and efficient method to automatic neural architecture design based on continuous optimization. We call this new approach neural architecture optimization (NAO). There are three key components in our proposed approach: (1) An encoder embeds/maps neural network architectures into a continuous space. (2) A predictor takes the continuous representation of a network as input and predicts its accuracy. (3) A decoder maps a continuous representation of a network back to its architecture. The performance predictor and the encoder enable us to perform gradient based optimization in the continuous space to find the embedding of a new architecture with potentially better accuracy. Such a better embedding is then decoded to a network by the decoder. Experiments show that the architecture discovered by our method is very competitive for image classification task on CIFAR-10 and language modeling task on PTB, outperforming or on par with the best results of previous architecture search methods with a significantly reduction of computational resources. Specifically we obtain 2.07% test set error rate for CIFAR-10 image classification task and 55.9 test set perplexity of PTB language modeling task. The best discovered architectures on both tasks are successfully transferred to other tasks such as CIFAR-100 and WikiText-2. Furthermore, combined with the recent proposed weight sharing mechanism, we discover powerful architecture on CIFAR-10 (with error rate 3.53%) and on PTB (with test set perplexity 56.3), with very limited computational resources (in 10 hours on one GPU) for both tasks.

1 Introduction

Automatic design of neural network architecture without human intervention has been the interests of the community from decades ago [13, 22] to very recent [48, 49, 28, 39, 8]. The latest algorithms for automatic architecture design usually fall into two categories: reinforcement learning (RL) [48, 49, 37, 3] based methods and evolutionary algorithm (EA) based methods [42, 35, 39, 28, 38]. In RL based methods, the choice of a component of the architecture is regarded as an action. A sequence of actions defines an architecture of a neural network, whose dev set accuracy is used as the reward. In EA based method, search is performed through mutations and re-combinations of architectural components, where those architectures with better performances will be picked to continue evolution.

^{*}The work was done when the first author was an intern at Microsoft Research Asia.

[†]The first and second author contribute equally to this work.

It can be easily observed that both RL and EA based methods essentially perform search within the discrete architecture space. This is natural since the choices of neural network architectures are typically discrete, such as the filter size in CNN and connection topology in RNN cell. However, directly searching the best architecture within discrete space is inefficient given the exponentially growing search space with the number of choices increasing. In this work, we instead propose to *optimize* network architecture by mapping architectures into a continuous vector space (i.e., network embeddings) and conducting optimization in this continuous space via gradient based method. On one hand, similar to the distributed representation of natural language [36, 25], the continuous representation of an architecture is more compact and efficient in representing its topological information; On the other hand, optimizing in a continuous space is much easier than directly searching within discrete space due to better smoothness.

We call this optimization based approach *Neural Architecture Optimization* (NAO), which is briefly shown in Fig. 1. The core of NAO is an encoder model responsible to map a neural network architecture into a continuous representation (the blue arrow in the left part of Fig. 1). On top of the continuous representation we build a regression model to approximate the final performance (e.g., classification accuracy on the dev set) of an architecture (the yellow surface in the middle part of Fig. 1). It is noteworthy here that the regression model is similar to the performance predictor in previous works [4, 27, 11]. What distinguishes our method is how to leverage the performance predictor: different with previous work [27] that uses the performance predictor as a heuristic to select the already generated architectures to speed up searching process, we directly *optimize* the module to obtain the continuous representation of a better network (the black arrow in the middle and bottom part of Fig. 1) by gradient descent. The optimized representation is then leveraged to produce a new neural network architecture that is predicted to perform better. To achieve that, another key module for NAO is designed to act as the decoder recovering the discrete architecture from the continuous representation (the red arrow in the right part of Fig. 1). The decoder is an LSTM model equipped with an attention mechanism that makes the exact recovery possible. The three components (i.e., encoder, performance predictor and decoder) are jointly trained in a multi task setting which is beneficial to continuous representation: the decoder objective of recovering the architecture further improves the quality of the architecture embedding, making it more effective in predicting the performance.

We conduct thorough experiments to verify the effectiveness of NAO, on both image classification and language modeling tasks. Using the same architecture space commonly used in previous works [48, 49, 37, 27], the architecture found via NAO achieves 2.07% test set error rate (with cutout [12]) on CIFAR-10. Furthermore, on PTB dataset we achieve 55.9 perplexity, also surpassing best performance found via previous methods on neural architecture search. Furthermore, we show that equipped with the recent proposed weight sharing mechanism in ENAS [37] to reduce the large complexity in the parameter space of child models, we can achieve improved efficiency in discovering powerful convolutional and recurrent architectures, e.g., both take less than 10 hours on 1 GPU. We will release our codes/models soon.

2 Related Work

Recently the design of neural network architectures has largely shifted from leveraging human knowledge to automatic methods, sometimes referred to as Neural Architecture Search (NAS) [42, 48, 49, 27, 37, 7, 39, 38, 28, 8, 7, 21], with the objective of discovering better neural network architectures without hand-crafted heuristics. As mentioned above, most of these methods are built upon one of the two basic algorithms: reinforcement learning (RL) [48, 49, 8, 3, 37, 9] and evolutionary algorithm (EA) [42, 39, 35, 38, 28]. For example, [48, 49, 37] use policy networks to guide the next-step architecture component. The evolution processes in [39, 28] guide the mutation and recombination process of candidate architectures. Some recent works [17, 18, 27] try to improve the efficiency in architecture search by exploring the search space incrementally and sequentially, typically from shallow to hard. Among them, [27] additionally utilizes a performance predictor to select the promising candidates. Similar performance predictor has been specifically studied in parallel works such as [11, 4]. Although different in terms of searching algorithms, all these works target at improving the quality of discrete decision in the process of searching architectures.

The most recent work parallel to ours is DARTS [29], which relaxes the discrete architecture space to continuous one by mixture model and utilizes gradient based optimization to derive the best

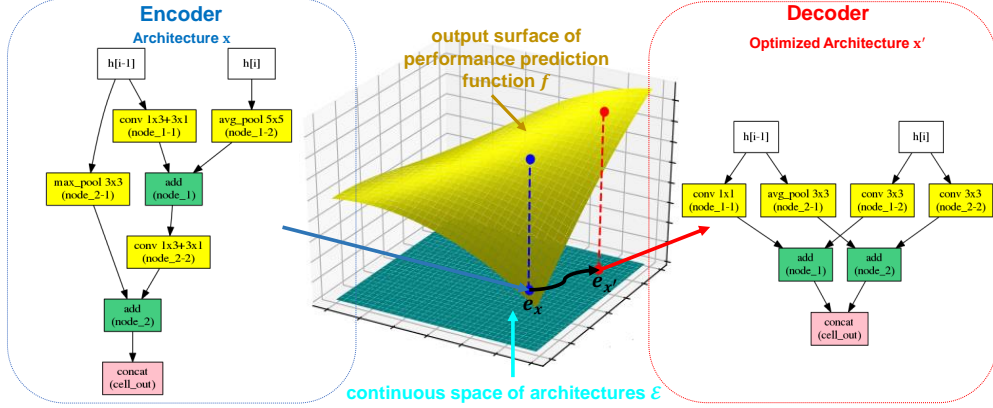


Figure 1: The general framework of NAO. Better viewed in color mode. The original architecture x is mapped to continuous representation e_x via encoder network. Then e_x is optimized into $e_{x'}$ via maximizing the output of performance predictor f . Afterwards $e_{x'}$ is transformed into a new architecture x' using the decoder network.

architecture. One one hand, both NAO and DARTS conducts continuous optimization via gradient based method; on the other hand, the continuous space in the two works are different: in DARTS it is the mixture weights and in NAO it is the embedding of neural architectures. The difference in optimization space leads to the difference in how to derive the best architecture from continuous space: DARTS simply assumes the best decision (among different choices of architectures) is the *argmax* of mixture weights while NAO uses a decoder to exactly recover the discrete architecture.

Another line of work with similar motivation to our research is using bayesian optimization (BO) to perform automatic architecture design [40, 21]. Using BO, an architecture’s performance is typically modeled as sample from a Gaussian process (GP). The induced posterior of GP, a.k.a. the acquisition function denoted as $a : \mathcal{X} \rightarrow R^+$ where \mathcal{X} represents the architecture space, is tractable to minimize. By solving $x_{next} = \arg \max_x a(x)$, the next architecture x_{next} to be evaluated is selected, which is assumed to have better performance. However, the effectiveness of GP heavily relies on the choice of covariance functions $K(x, x')$ which essentially models the similarity between two architectures x and x' . One need to pay more efforts in setting good $K(x, x')$ in the context of architecture design, bringing additional manual efforts whereas the performance might still be unsatisfactory [21]. As a comparison, we do not build our method on the complicated GP setup and empirically find that our model which is simpler and more intuitive works much better in practice.

3 Approach

We introduce the details of Neural Architecture Optimization in this section.

3.1 Architecture Space

Firstly we introduce the design space for neural network architectures, denoted as \mathcal{X} . For fair comparison with previous NAS algorithms, we adopt the same architecture space commonly used in previous works [48, 49, 37, 27, 39, 38].

For searching CNN architecture, we assume that the CNN architecture is hierarchical in that a cell is stacked for a certain number of times (denoted as N) to form the final CNN architecture. The goal is to design the topology of the cell. A cell is a convolutional neural network containing B nodes. Each of the nodes contains two branches, with each branch taking the output of one of the former nodes as input and applying an operation to it. The operation set includes 11 operators listed in Appendix. The node adds the outputs of its two branches as its output. The inputs of the cell are the outputs of two previous cells, respectively denoted as node -2 and node -1 . Finally, the outputs of all the nodes that are not used by any other nodes are concatenated to form the final output of the cell. Therefore, for each of the B nodes we need to: 1) decide which two previous nodes are used as

the inputs to its two branches; 2) decide the operation to apply to its two branches. We set $B = 5$ in our experiments as in [49, 37, 27, 38].

For searching RNN architecture, we use the same architecture space as in [37]. The architecture space is imposed on the topology of an RNN cell, which computes the hidden state h_t using input i_t and previous hidden state h_{t-1} . The cell contains B nodes and we have to make two decisions for each node, similar to that in CNN cell: 1) a previous node as its input; 2) the activation function to apply. For example, if we sample node index 2 and ReLU for node 3, the output of the node will be $o_3 = \text{ReLU}(o_2 \cdot W_3^h)$. An exception here is for the first node, where we only decide its activation function a_1 and its output is $o_1 = a_1(i_t \cdot W^i + h_{t-1} \cdot W_1^h)$. Note that all W matrices are the weights related with each node. The available activation functions are: tanh, ReLU, identity and sigmoid. Finally, the output of the cell is the average of the output of all the loose nodes, which are the nodes that are not chosen as inputs to any other nodes. In our experiments we set $B = 12$ as in [37].

We use a sequence consisting of discrete string tokens to describe a CNN or RNN architecture. Taking the description of CNN cell as an example, each branch of the node is represented via three tokens, including the node index it selected as input, the operation type and operation size. For example, the sequence “node-2 conv 3x3 node1 max-pooling 3x3 ” means the two branches of one node respectively takes the output of node -2 and node 1 as inputs, and respectively apply 3×3 convolution and 3×3 max pooling. For the ease of introduction, we use the same notation $x = \{x_1, \dots, x_T\}$ to denote such string sequence of an architecture x , where x_t is the token at t -th position and all architectures $x \in \mathcal{X}$ share the same sequence length denoted as T . T is determined via the number of nodes B in each cell in our experiments.

3.2 Components of Neural Architecture Optimization

The overall framework of NAO is shown in Fig. 1. To be concrete, there are three major parts that constitute NAO: the *encoder*, the *performance predictor* and the *decoder*.

Encoder. The encoder of NAO takes the string sequence describing an architecture as input, and maps it into a continuous space \mathcal{E} . Specifically the encoder is denoted as $E : \mathcal{X} \rightarrow \mathcal{E}$. For an architecture x , we have its continuous representation (a.k.a. embedding) $e_x = E(x)$. We use a single layer LSTM as the basic model of encoder and the hidden states of the LSTM are used as the continuous representation of x . Therefore we have $e_x = \{h_1, h_2, \dots, h_T\} \in \mathcal{R}^{T \times d}$ where $h_t \in \mathcal{R}^d$ is LSTM hidden state at t -th timestep with dimension d^3 .

Performance predictor. The performance predictor $f : \mathcal{E} \rightarrow \mathcal{R}^+$ is another important module accompanied with the encoder. It maps the continuous representation e_x of an architecture x into its performance s_x measured by dev set accuracy. Specifically, f first conducts mean pooling on $e_x = \{h_1, \dots, h_T\}$ to obtain $\bar{e}_x = \frac{1}{T} \sum_t h_t$, and then maps \bar{e}_x to a scalar value using a feed-forward network as the predicted performance. For an architecture x and its performance s_x as training data, the optimization of f aims at minimizing the least-square regression loss $(s_x - f(E(x)))^2$.

Considering the requirement of performance prediction, an important requirement for encoder is to guarantee the permutation invariance of architecture embedding: for two architectures x_1 and x_2 if they are symmetric (e.g., x_2 is formed via swapping two branches within a node in x_1), their embeddings should be close to produce the same performance prediction scores. To achieve that, we adopt a simple data augmentation approach which is inspired from the data augmentation method in computer vision (e.g., image rotation and flipping): for each (x_1, s_x) pair, we add an additional pair (x_2, s_x) where x_2 is symmetrical to x_1 , and use both pairs (i.e., (x_1, s_x) and (x_2, s_x)) to train the encoder and performance predictor. Empirically we found that acting in this way brings about 2% pairwise accuracy gain for training performance predictor.

Decoder. Similar to the decoder in the neural sequence-to-sequence model [41, 10], the decoder in NAO is responsible to decode out the string tokens in x , taking e_x as input and in an autoregressive manner. Mathematically the decoder is denoted as $D : \mathcal{E} \rightarrow x$ which decodes the string tokens x from its continuous representation: $x = D(e_x)$. We set D as an LSTM model with the initial hidden state $s_0 = h_T(x)$. Furthermore, attention mechanism [2] is leveraged to make decoding easier, which will output a context vector ctx_r combining all encoder outputs $\{h_t\}_{t=1}^T$ at each timestep r . The

³For ease of introduction, even though some notations have been used before (e.g., h_t in defining RNN search space), they are slightly abused here.

decoder D then induces a factorized distribution $P_D(x|e_x) = \prod_{r=1}^T P_D(x_r|e_x, x_{<r})$ on x , where the distribution on each token x_r is $P_D(x_r|e_x, x_{<r}) = \frac{\exp(W_{x_r}[s_r, ct_{x_r}])}{\sum_{x' \in V_r} \exp(W_{x'}[s_r, ct_{x_r}])}$. Here W is the output embedding matrix for all tokens, $x_{<r}$ represents all the previous tokens before position r , s_r is the LSTM hidden state at r -th timestep and $[\cdot]$ means concatenation of two vectors. V_r denotes the space of valid tokens at position r to avoid the possibility of generating invalid architectures.

The training of decoder aims at recovering the architecture x from its continuous representation $e_x = E(x)$. Specifically we would like to maximize $\log P_D(x|E(x)) = \sum_{r=1}^T \log P_D(x_r|E(x), x_{<r})$. It is worthy to mention that for the purpose of generating architecture from a continuous space, one can leverage variational autoencoder [6], or even directly generate the computational graph of an architecture using deep generative models for graph [45, 26]. In this work we use the vanilla LSTM model as a preliminary trial and find it works quite well in practice.

3.3 Training and Inference

We jointly train the encoder E , performance predictor f and decoder D by minimizing the combination of performance prediction loss L_{pd} , structure reconstruction loss L_{rec} and permutation invariance loss L_{pi} :

$$L = \lambda L_{pd} + (1 - \lambda) L_{rec} = \lambda \sum_{x \in X} (s_x - f(E(x)))^2 - (1 - \lambda) \sum_{x \in X} \log P_D(x|E(x)), \quad (1)$$

where X denotes all candidate architectures x (and their symmetrical counterparts) that are evaluated with the performance number s_x . $\lambda \in [0, 1]$ is the trade-off parameter. Furthermore, the performance prediction loss acts as a regularizer that forces the encoder not optimized into a trivial state to simply copy tokens in the decoder side, which is typically eschewed by adding noise in encoding x by previous works [1, 24].

When both the encoder and decoder are optimized to convergence, the inference process for better architectures is performed in the continuous space \mathcal{E} . Specifically, starting from an architecture x with satisfactory performance, we obtain a better continuous representation $e_{x'}$ by moving $e_x = \{h_1, \dots, h_T\}$ towards the gradient direction induced by f :

$$h'_t = h_t + \eta \frac{\partial f}{\partial h_t}, \quad e_{x'} = \{h'_1, \dots, h'_T\}, \quad (2)$$

where η is the step size. Such optimization step is represented via the black arrow in Fig. 1. $e_{x'}$ corresponds to a new architecture x' which is probably better than x since we have $f(e_{x'}) \geq f(e_x)$, as long as η is within a reasonable range (e.g., small enough). Afterwards, we feed $e_{x'}$ into decoder to obtain a new architecture x' assumed to have better performance⁴. We call the original architecture x as ‘seed’ architecture and iterate such process for several rounds, with each round containing several seed architectures with top performances. The detailed algorithm is shown in Alg. 1.

3.4 Combination with Weight Sharing

Recently the weight sharing trick proposed in [37] significantly reduces the computational complexity of neural architecture search. Different with NAO that tries to reduce the huge computational cost brought by the search algorithm, weight sharing aims to ease the huge complexity brought by massive child models via the one-shot model setup [5]. Therefore, the weight sharing method is complementary to NAO and it is possible to obtain better performance by combining NAO and weight sharing. To verify that, We add the weight sharing method to NAO by replacing the RL controller in ENAS [37] by NAO including encoder, performance predictor and decoder, with the other training pipeline of ENAS unchanged. The results are reported in the next section 4.

⁴If we have $x' = x$, i.e., the new architecture is exactly the same with the previous architecture, we ignore it and keep increasing the step-size value by $\frac{\eta}{2}$, until we found a different decoded architecture different with x .

Algorithm 1 Neural Architecture Optimization

Input: Initial candidate architectures set X to train NAO model. Initial architectures set to be evaluated denoted as $X_{eval} = X$. Performances of architectures $S = \emptyset$. Number of seed architectures K . Step size η . Number of optimization iterations L .

for $l = 1, \dots, L$ **do**

Train each architecture $x \in X_{eval}$ and evaluate it to obtain the dev set performances $S_{eval} = \{s_x\}, \forall x \in X_{eval}$. Enlarge S as $S = S \cup S_{eval}$.

Train encoder E , performance predictor f and decoder D via minimizing the loss in Eqn. (1), using X and S .

Pick K architectures with top K performances among X , forming the set of seed architectures X_{seed} .

For $x \in X_{seed}$, obtain a better representation $e_{x'}$ from e_x using Eqn. (2), based on encoder E and performance predictor f . Denote the set of enhanced representations as $E' = \{e_{x'}\}$.

Decode each x' from $e_{x'}$ using decoder, set X_{eval} as the set of new architectures decoded out: $X_{eval} = \{D(e_{x'}), \forall e_{x'} \in E'\}$. Enlarge X as $X = X \cup X_{eval}$.

end for

Output: The architecture within X with the best performance

4 Experiments

In this section, we report the empirical performances of NAO in discovering competitive neural network architectures, on two widely acknowledged benchmark dataset, the CIFAR-10 dataset for image recognition and the Penn Treebank (PTB) dataset for language modeling.

4.1 Results on CIFAR-10 Classification

CIFAR-10 contains 50k and 10k images for training and testing. We randomly choose 5000 images within training set as the dev set for measuring the performance of each candidate network in the optimization process of NAO. Standard data pre-processing and augmentation, such as whitening, randomly cropping 32×32 patches from unsampled images of size 40×40 , and randomly flipping images horizontally are applied to original training set. The CNN models are trained using SGD with momentum set to 0.9, where the arrange of learning rate follows a single period cosine schedule with $l_{max} = 0.024$ proposed in [30]. For the purpose of regularization, We apply stochastic drop-connect on each path, and an l_2 weight decay of 5×10^{-4} . All the models are trained with batch size 128.

The architecture encoder of NAO is an LSTM model with token embedding size and hidden state size respectively set as 32 and 96. The hidden states of LSTM are normalized to have unit length, i.e., $h_t = \frac{h_t}{\|h_t\|_2}$, to constitute the embedding of the architecture x : $e_x = \{h_1, \dots, h_T\}$. The performance predictor f is a one layer feed-forward network taking $\frac{1}{T} \sum_{t=1}^T h_t$ as input. The decoder is an LSTM model with an attention mechanism and the hidden state size is 96. The normalized hidden states of the encoder LSTM are used to compute the attention. The encoder, performance predictor and decoder of NAO are trained using Adam for 1000 epochs with a learning rate of 0.001. The trade-off parameters in Eqn. (1) is $\lambda = 0.9$. We run the evaluation-optimization step in Alg. 1 for three times (i.e., $L = 3$), with initial X set as 600 randomly sampled architectures, $K = 200$, forming $600 + 200 + 200 = 1000$ model architectures evaluated in total. The step size to perform continuous optimization is $\eta = 10$. Similar to previous works [48, 37], for all the architectures in NAO training phase (i.e., in Alg. 1), we set them to be small networks with $B = 5, N = 3, F = 32$ and train them for 25 epochs. We use 200 V100 GPU cards to complete all the process within 1 day. After the best cell architecture is found, we build a larger network by stacking such cells 6 times (set $N = 6$), and enlarging the filter size (set $F = 36, F = 64$ and $F = 128$), and train it on the whole training dataset for 600 epochs.

The detailed results are shown in Table 1, where we demonstrate the performances of best experts designed architectures (in top block), the networks discovered by previous NAS algorithm (in middle block) and by NAO (refer to its detailed architecture in Appendix), which we name as NAONet (in bottom block). We have several observations. (1) NAONet achieves the best test set error rate (2.98) among all architectures which is on par with AmoebaNet, but with less parameters. (2) Compared with the previous strongest architecture, the *AmoebaNet*, within smaller search space ($\#op = 11$), NAO

Model	B	N	F	#op	Error(%)	#params	M	GPU Days
DenseNet-BC [19]		100	40	/	3.46	25.6M	/	/
ResNeXt-29 [43]				/	3.58	68.1M	/	/
NASNet-A [48]	5	6	32	13	3.41	3.3M	20000	2000
NASNet-B [48]	5	4	N/A	13	3.73	2.6M	20000	2000
NASNet-C [48]	5	4	N/A	13	3.59	3.1M	20000	2000
Hier-EA [28]	5	2	64	6	3.75	15.7M	7000	300
AmoebaNet-A [38]	5	6	36	10	3.34	3.2M	20000	3150
AmoebaNet-B [38]	5	6	36	19	3.37	2.8M	27000	3150
AmoebaNet-B [38]	5	6	80	19	3.04	13.7M	27000	3150
AmoebaNet-B [38]	5	6	128	19	2.98	34.9M	27000	3150
AmoebaNet-B + Cutout [38]	5	6	128	19	2.13	34.9M	27000	3150
ENAS [37]	5	5	36	5	3.54	4.6M	/	0.45
PNAS [27]	5	3	48	8	3.41	3.2M	1280	225
DARTS + Cutout [29]	5	6	36	7	2.83	4.6M	/	4
NAONet	5	6	36	11	3.18	10.6M	1000	200
NAONet	5	6	64	11	2.98	28.6M	1000	200
NAONet + Cutout	5	6	128	11	2.07	128M	1000	200
NAONet-WS	5	5	36	5	3.53	3.1M	/	0.4

Table 1: Performances of different CNN models on CIFAR-10 dataset. ‘B’ is the number of nodes within a cell introduced in subsection 3.1. ‘N’ is the number of times the discovered normal cell is unrolled to form the final CNN architecture. ‘F’ represents the filter size. ‘#op’ is the number of different operation for one branch in the cell, which is an indicator of the scale of architecture space for automatic architecture design algorithm. ‘M’ is the total number of network architectures that are trained to obtain the claimed performance. ‘/’ denotes that the criteria is meaningless for a particular algorithm. ‘NAONet-WS’ represents the architecture discovered by NAO and the weight sharing method as described in subsection 3.4.

not only reduces the classification error rate ($3.34 \rightarrow 3.18$), but also needs an order of magnitude less architectures that are evaluated ($20000 \rightarrow 1000$). When setting the architectures to be deeper and wider, the performance of NAONet is on par with that of AmoebaNet-B, but with less parameters ($34.9M \rightarrow 28.6M$). Therefore, we can conclude that optimization in the continuous space is more efficient in discovering better architectures than previous SOTA NAS method. (3) Compared with PNAS [27] which similarly uses performance predictor, even though the architecture space of NAO is slightly larger ($\#op$ is larger), NAO is more efficient ($M = 1000$) and significantly reduces the error rate of PNAS ($3.41 \rightarrow 3.18$). (4) When accompanied with weight sharing, NAO achieves 3.55% error rate (by NAONet-WS) within only 10 hours. The improvement demonstrates that the performance predictor plays more effect via guiding the continuous optimization rather than simply set as a heuristic in ranking architecture candidates.

We furthermore conduct in-depth analysis towards the performances of NAO. In Fig. 2(a) we show the performances of performance predictor and decoder w.r.t. the number of training data, i.e., the number of evaluated architectures $|X|$. Specifically, among 600 randomly sampled architectures, we randomly choose 50 of them (denoted as X_{test}) and their corresponding performances (denoted as $S_{test} = \{s_x, \forall x \in X_{test}\}$) as test set. Then we train the encoder, performance predictor and decoder using the left 550 architectures (together with their performances) as training set. We vary the number of training data as (100, 200, 300, 400, 500, 550) and observe the corresponding accuracy of performance predictor f , as well as the decoder D . To evaluate f , we compute the pairwise accuracy on X_{test} and S_{test} computed via f , i.e., $acc_f = \frac{\sum_{x_1 \in X_{test}, x_2 \in X_{test}} \mathbb{1}_{f(E(x_1)) \geq f(E(x_2))} \mathbb{1}_{s_{x_1} \geq s_{x_2}}}{\sum_{x_1 \in X_{test}, x_2 \in X_{test}} 1}$, where $\mathbb{1}$ denotes the 0-1 indicator function. To evaluate decoder D , we compute the Hamming distance (denoted as $Dist$) between the sequence representation of decoded architecture using D and original architecture to measure their differences. Specifically the measure is $dist_D = \frac{1}{|X_{test}|} \sum_{x \in X_{test}} Dist(D(E(x)), x)$. As can be witnessed in Fig. 2(a), the performance predictor is able to achieve satisfactory performance (i.e., $> 78\%$ pairwise accuracy) with only roughly 500 evaluated architectures. Furthermore, the decoder D is powerful in that it can almost exactly recover the network architecture from its embedding, with averaged Hamming distance between the description strings of two architectures less than 0.5, which means

that on average the difference between the decoded sequence and the original one is less than 0.5 tokens (60 tokens in total).

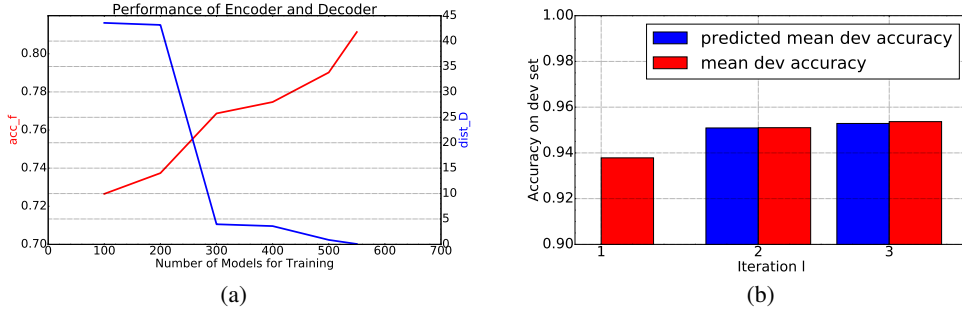


Figure 2: Left: the accuracy acc_f of performance predictor f (red line) and performance $dist_D$ of decoder D (blue line) on the test set, w.r.t. the number of training data (i.e., evaluated architectures). Right: the mean dev set accuracy, together with its predicted value by f , of candidate architectures set X_{eval} in each NAO optimization iteration $l = 1, 2, 3$. The architectures are trained for 25 epochs.

Furthermore, we would like to inspect whether the gradient update in Eqn.(2) really helps to generate better architecture representations that are further decoded to architectures via D . In Fig. 2(b) we show the average performances of architectures in X_{eval} discovered via NAO at each optimization iteration. Red bar indicates the mean of real performance values $\frac{1}{|X_{eval}|} \sum_{x \in X_{eval}} s_x$ while blue bar indicates the mean of predicted value $\frac{1}{|X_{eval}|} \sum_{x \in X_{eval}} f(E(x))$. We can observe that the performances of architectures in X_{eval} generated via NAO gradually increase with each iteration. Furthermore, the performance predictor f produces predictions well aligned with real performance, as is shown via the small gap between the paired red and blue bars.

4.2 Transferring the discovered architecture to CIFAR-100

To evaluate the transferability of discovered NAONet, we apply it to CIFAR-100. Similar to CIFAR-10, CIFAR-100 also contains 50k and 10k images for training and testing, but they are categorized into 100 classes. We use the best architecture discovered on CIFAR-10 and exactly follow the same training setting but without cutout. Meanwhile, we evaluate the performances of other automatically discovered neural networks on CIFAR-100 by strictly using the reported architectures in previous NAS papers [38, 37, 27]. All results are listed in Table 2. NAONet gets test error rate of 14.36, better than the previous SOTA obtained with cutout [12](15.20). The results show that our NAONet derived with CIFAR-10 is indeed transferable to more complicated task such as CIFAR-100.

Model	B	N	F	#op	Error (%)	#params
DenseNet-BC [19]	/	100	40	/	17.18	25.6M
Shake-shake [15]	/	/	/	/	15.85	34.4M
Shake-shake + cutout [12]	/	/	/	/	15.20	34.4M
NASNet-A [48]	5	6	32	13	19.70	3.3M
AmoebaNet-B [38]	5	6	128	19	17.66	34.9M
ENAS [37]	5	5	36	5	19.43	4.6M
PNAS [27]	5	3	48	8	19.53	3.2M
NAONet	5	6	128	11	14.36	128M

Table 2: Performances of different CNN models on CIFAR-100 dataset. ‘NAONet’ represents the best architecture discovered by NAO on CIFAR-10.

4.3 Results of Language Modeling on PTB

Penn Treebank (PTB) [31] is one of the most widely adopted benchmark dataset for language modeling task. We use the open-source code of ENAS [37] released by the authors and exactly follow their setups. Specifically, we apply variational dropout, l_2 regularization with weight decay of

Models and Techniques	#params	Test Perplexity	GPU Days
Vanilla LSTM [46]	66M	78.4	/
LSTM + Zoneout [23]	66M	77.4	/
Variational LSTM [14]	19M	73.4	/
Pointer Sentinel-LSTM [34]	51M	70.9	/
Variational LSTM + weight tying [20]	51M	68.5	/
Variational Recurrent Highway Network + weight tying [47]	23M	65.4	/
4-layer LSTM + skip connection + averaged weight drop + weight penalty + weight tying [32]	24M	58.3	/
LSTM + averaged weight drop + Mixture of Softmax + weight penalty + weight tying [44]	22M	56.0	/
NAS + weight tying [48]	54M	62.4	1e4 CPU days
ENAS + weight tying + weight penalty [37]	24M	58.6 ⁵	0.5
DARTS+ weight tying + weight penalty	23M	56.1	1
NAO + weight tying + weight penalty	24M	55.9	300
NAO-WS + weight tying + weight penalty	23M	56.3	0.4

Table 3: Performance of different models and techniques on PTB dataset. Similar to CIFAR-10 experiment, ‘NAO-WS’ represents NAO accompanied with weight sharing.

5×10^{-7} , and tying word embeddings and softmax weights. We train the models using SGD with an initial learning rate of 10.0, decayed by a factor of 0.9991 after every epoch starting at epoch 15. To avoid gradient explosion, we clip the norm of gradient with the threshold value 0.25.

The encoder in NAO is an LSTM with embedding size 64 and hidden size 128. The hidden state of LSTM is further normalized to have unit length. The performance predictor is a two-layer MLP with each layer size as 200, 1. The decoder is a single layer LSTM with attention mechanism and the hidden size is 128. The trade-off parameters in Eqn. (1) is $\lambda = 0.8$. The encoder, performance predictor, and decoder are trained using Adam with a learning rate of 0.001. We perform the optimization process in Alg 1 for two iterations (i.e., $L = 2$). We train the sampled RNN models for shorter time (600 epochs) during the training phase of NAO, and afterwards train the best architecture discovered yet for 2000 epochs for the sake of better result. We use 200 P100 GPU cards to complete all the process within 1.5 days.

We report all the results in Table 3, separated into three blocks, respectively reporting the results of experts designed methods (we list them just to make comparisons), architectures discovered via previous automatic neural architecture search methods, and our NAO. As can be observed, NAO successfully discovered an architecture that achieves quite competitive perplexity 55.9, surpassing previous NAS methods and is on par with the best performance from LSTM method with advanced manually designed techniques such as averaged weight drop [32].

4.4 Transferring the discovered architecture to WikiText-2

We also apply the best discovered RNN architecture on PTB to another language modelling task based on a much larger dataset WikiText-2 (WT2 for short in the following). We use embedding size 700, weight decay of 5×10^{-7} and variational dropout 0.15. Others unstated are the same as in PTB, such as weight tying. Table 4 shows the result that NAONet discovered by our method surpasses both ENAS and DARTS with test perplexity of 66.5.

5 Conclusion

We design a new automatic architecture design algorithm named as *neural architecture optimization* (NAO), which performs the optimization within continuous space rather than searching discrete decisions. The encoder, performance predictor and decoder together makes it more effective and efficient to discover better architectures and we achieve quite competitive results on both CIFAR-10

⁵The ENAS paper [37] reports 55.8, which is inconsistent with our reproduction based on the published codes released by authors. We thus adopt the number reported via [29] which is similar to our reproduction.

Models and Techniques	#params	Test Perplexity
Variational LSTM + weight tying [20]	28M	87.0
LSTM + continuous cache pointer [16]	-	68.9
LSTM [33]	33	66.0
4-layer LSTM + skip connection + averaged weight drop + weight penalty + weight tying [32]	24M	65.9
LSTM + averaged weight drop + Mixture of Softmax + weight penalty + weight tying [44]	33M	63.3
ENAS + weight tying + weight penalty [37] (searched on PTB)	33M	70.4
DARTS + weight tying + weight penalty (searched on PTB)	33M	66.9
NAO + weight tying + weight penalty (searched on PTB)	36M	66.5

Table 4: Performance of different models and techniques on WT2 dataset. ‘NAONet’ represents the best architecture discovered by NAO on PTB.

classification task and PTB language modeling task. For future work, first we would like to try other methods to further improve the performance of the discovered architecture, such as mixture of softmax [44] for language modeling. Second, we would like to apply NAO to discovering better architectures for more applications such as Neural Machine Translation.

6 Acknowledgement

We thank Hieu Pham for the discussion on some details of ENAS implementation, and Hanxiao Liu for the code base of language modeling task. We furthermore thank the anonymous reviewers for their constructive comments.

References

- [1] ARTETXE, M., LABAKA, G., AGIRRE, E., AND CHO, K. Unsupervised neural machine translation. In *International Conference on Learning Representations* (2018).
- [2] BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
- [3] BAKER, B., GUPTA, O., NAIK, N., AND RASKAR, R. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations* (2017).
- [4] BAKER, B., GUPTA, O., RASKAR, R., AND NAIK, N. Accelerating neural architecture search using performance prediction. In *International Conference on Learning Representations, Workshop Track* (2018).
- [5] BENDER, G., KINDERMANS, P.-J., ZOPH, B., VASUDEVAN, V., AND LE, Q. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning* (2018), pp. 549–558.
- [6] BOWMAN, S. R., VILNIS, L., VINYALS, O., DAI, A., JOZEFOWICZ, R., AND BENGIO, S. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning* (2016), pp. 10–21.
- [7] BROCK, A., LIM, T., RITCHIE, J., AND WESTON, N. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations* (2018).
- [8] CAI, H., CHEN, T., ZHANG, W., YU, Y., AND WANG, J. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873* (2017).
- [9] CAI, H., YANG, J., ZHANG, W., HAN, S., AND YU, Y. Path-level network transformation for efficient architecture search. *arXiv preprint arXiv:1806.02639* (2018).
- [10] CHO, K., VAN MERRIENBOER, B., GULCEHRE, C., BAHDANAU, D., BOUGARES, F., SCHWENK, H., AND BENGIO, Y. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Doha, Qatar, October 2014), Association for Computational Linguistics, pp. 1724–1734.

- [11] DENG, B., YAN, J., AND LIN, D. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351* (2017).
- [12] DEVRIES, T., AND TAYLOR, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552* (2017).
- [13] FAHLMAN, S. E., AND LEBIERE, C. The cascade-correlation learning architecture. In *Advances in neural information processing systems* (1990), pp. 524–532.
- [14] GAL, Y., AND GHAHRAMANI, Z. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems* (2016), pp. 1019–1027.
- [15] GASTALDI, X. Shake-shake regularization. *CoRR abs/1705.07485* (2017).
- [16] GRAVE, E., JOULIN, A., AND USUNIER, N. Improving neural language models with a continuous cache. *CoRR abs/1612.04426* (2016).
- [17] GROSSE, R. B., SALAKHUTDINOV, R., FREEMAN, W. T., AND TENENBAUM, J. B. Exploiting compositionality to explore a large space of model structures. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence* (Arlington, Virginia, United States, 2012), UAI’12, AUAI Press, pp. 306–315.
- [18] HUANG, F., ASH, J., LANGFORD, J., AND SCHAPIRE, R. Learning deep resnet blocks sequentially using boosting theory. *arXiv preprint arXiv:1706.04964* (2017).
- [19] HUANG, G., LIU, Z., VAN DER MAATEN, L., AND WEINBERGER, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 4700–4708.
- [20] INAN, H., KHOSRAVI, K., AND SOCHER, R. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462* (2016).
- [21] KANDASAMY, K., NEISWANGER, W., SCHNEIDER, J., POCZOS, B., AND XING, E. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191* (2018).
- [22] KITANO, H. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems Journal* 4 (1990), 461–476.
- [23] KRUEGER, D., MAHARAJ, T., KRAMÁR, J., PEZESHKI, M., BALLAS, N., KE, N. R., GOYAL, A., BENGIO, Y., COURVILLE, A., AND PAL, C. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305* (2016).
- [24] LAMPLE, G., CONNEAU, A., DENOYER, L., AND RANZATO, M. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations* (2018).
- [25] LE, Q., AND MIKOLOV, T. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning* (Beijing, China, 22–24 Jun 2014), E. P. Xing and T. Jebara, Eds., vol. 32 of *Proceedings of Machine Learning Research*, PMLR, pp. 1188–1196.
- [26] LI, Y., VINYALS, O., DYER, C., PASCANU, R., AND BATTAGLIA, P. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018).
- [27] LIU, C., ZOPH, B., SHLENS, J., HUA, W., LI, L.-J., FEI-FEI, L., YUILLE, A., HUANG, J., AND MURPHY, K. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559* (2017).
- [28] LIU, H., SIMONYAN, K., VINYALS, O., FERNANDO, C., AND KAVUKCUOGLU, K. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations* (2018).
- [29] LIU, H., SIMONYAN, K., AND YANG, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
- [30] LOSHCILLOV, I., AND HUTTER, F. SGDR: stochastic gradient descent with restarts. *CoRR abs/1608.03983* (2016).
- [31] MARCUS, M., KIM, G., MARCINKIEWICZ, M. A., MACINTYRE, R., BIES, A., FERGUSON, M., KATZ, K., AND SCHASBERGER, B. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology* (1994), Association for Computational Linguistics, pp. 114–119.

- [32] MELIS, G., DYER, C., AND BLUNSOM, P. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations* (2018).
- [33] MERITY, S., KESKAR, N. S., AND SOCHER, R. Regularizing and optimizing LSTM language models. *CoRR abs/1708.02182* (2017).
- [34] MERITY, S., XIONG, C., BRADBURY, J., AND SOCHER, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).
- [35] MIKKULAINEN, R., LIANG, J., MEYERSON, E., RAWAL, A., FINK, D., FRANCON, O., RAJU, B., SHAHRZAD, H., NAVRUZIAN, A., DUFFY, N., ET AL. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548* (2017).
- [36] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.
- [37] PHAM, H., GUAN, M. Y., ZOPH, B., LE, Q. V., AND DEAN, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).
- [38] REAL, E., AGGARWAL, A., HUANG, Y., AND LE, Q. V. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548* (2018).
- [39] REAL, E., MOORE, S., SELLE, A., SAXENA, S., SUEMATSU, Y. L., TAN, J., LE, Q. V., AND KURAKIN, A. Large-scale evolution of image classifiers. In *International Conference on Machine Learning* (2017), pp. 2902–2911.
- [40] SNOEK, J., LAROCHELLE, H., AND ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (2012), pp. 2951–2959.
- [41] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (2014), pp. 3104–3112.
- [42] XIE, L., AND YUILLE, A. Genetic cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)* (Oct. 2017), pp. 1388–1397.
- [43] XIE, S., GIRSHICK, R., DOLLÁR, P., TU, Z., AND HE, K. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* (2017), IEEE, pp. 5987–5995.
- [44] YANG, Z., DAI, Z., SALAKHUTDINOV, R., AND COHEN, W. W. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations* (2018).
- [45] YOU, J., YING, R., REN, X., HAMILTON, W. L., AND LESKOVEC, J. Graphrnn: A deep generative model for graphs. *arXiv preprint arXiv:1802.08773* (2018).
- [46] ZAREMBA, W., SUTSKEVER, I., AND VINYALS, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014).
- [47] ZILLY, J. G., SRIVASTAVA, R. K., KOUTNÍK, J., AND SCHMIDHUBER, J. Recurrent highway networks. In *International Conference on Machine Learning* (2017), pp. 4189–4198.
- [48] ZOPH, B., AND LE, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).
- [49] ZOPH, B., VASUDEVAN, V., SHLENS, J., AND LE, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018).

7 Appendix

7.1 Search Space

In searching convolutional cell architectures without weight sharing, following previous works of [38, 49], we adopt 11 possible ops as follow:

- identity

- 1×1 convolution
- 2×2 convolution
- 3×3 convolution
- 1×1 separable convolution
- 2×2 separable convolution
- 3×3 separable convolution
- 2×2 max pooling
- 3×3 max pooling
- 2×2 average pooling
- 3×3 average pooling

When using weight sharing, we use exactly the same 5 operators as [37]:

- identity
- 3×3 separable convolution
- 5×5 separable convolution
- 3×3 average pooling
- 3×3 max pooling

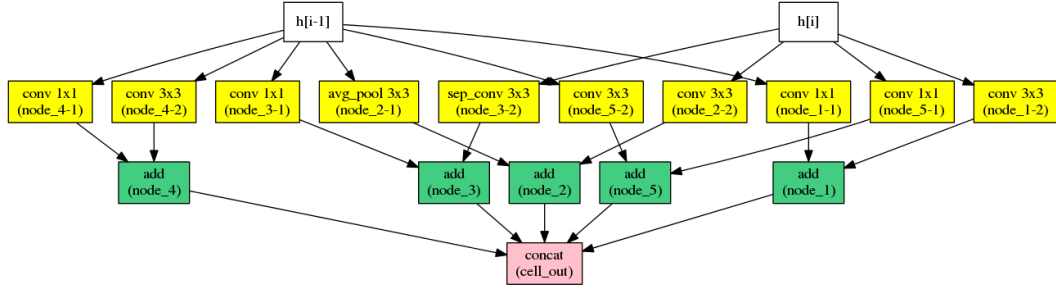
In searching for recurrent cell architectures, we exactly follow the search space of ENAS [37], where possible activation functions are:

- tanh
- relu
- identity
- sigmoid

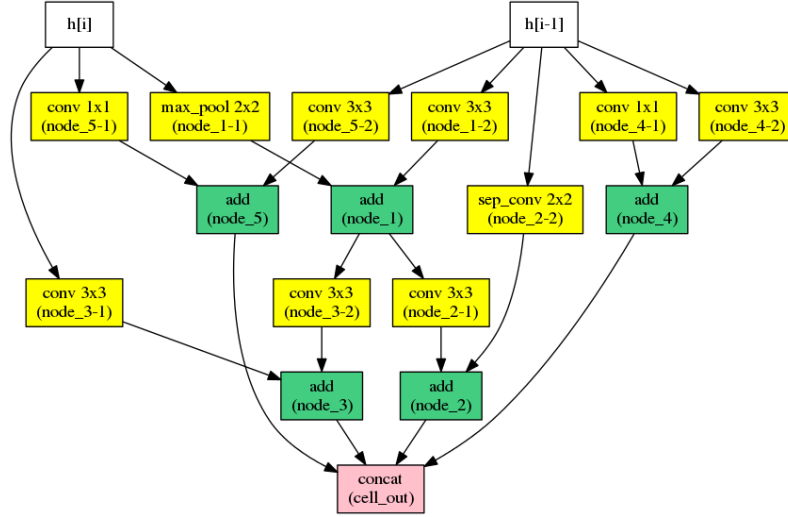
7.2 Best Architecture discovered

Here we plot the best architecture of CNN cells discovered by our NAO algorithm in Fig. 3.

Furthermore we plot the best architecture of recurrent cell discovered by our NAO algorithm in Fig. 4.



(a) Normal Cell



(b) Reduction Cell

Figure 3: Basic NAONet building blocks. NAONet normal cell (left) and reduction cell (right).

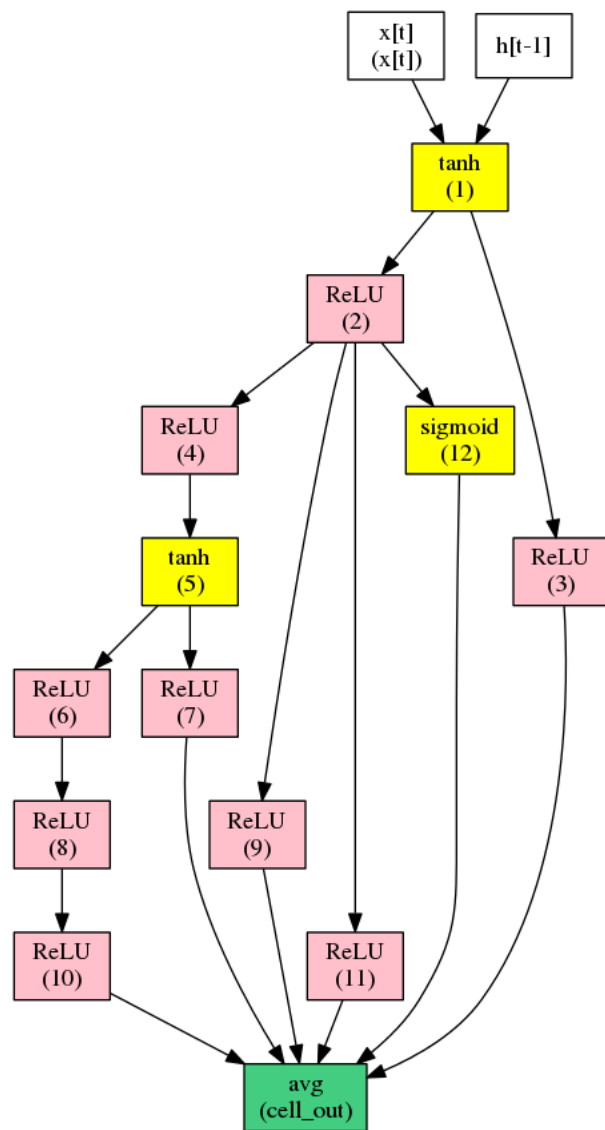


Figure 4: Best RNN Cell discovered by NAO for Penn Treebank.