Learning Architectures from an Extended Search Space for Language Modeling

Yinqiao Li[1], **Chi Hu**[1], **Yuhao Zhang**[1], **Nuo Xu**[1], **Yufan Jiang**[1], **Tong Xiao**[1,2]*, **Jingbo Zhu**[1,2], **Tongran Liu**[3], **Changliang Li**[4]
[1]NLP Lab, Northeastern University, Shenyang, China
[2]NiuTrans Research, Shenyang, China
[3]CAS Key Laboratory of Behavioral Science, Institute of Psychology, CAS, Beijing, China
[4]Kingsoft AI Lab, Beijing, China
li.yin.qiao.2012@hotmail.com,
{huchinlp,yoohao.zhang}@gmail.com,
{xunuo0629,jiangyufan2018}@outlook.com,
{xiaotong,zhujingbo}@mail.neu.edu.com,
liutr@psych.ac.cn,lichangliang@kingsoft.com

Abstract

Neural architecture search (NAS) has advanced significantly in recent years but most NAS systems restrict search to learning architectures of a recurrent or convolutional cell. In this paper, we extend the search space of NAS. In particular, we present a general approach to learn both intra-cell and inter-cell architectures (call it ESS). For a better search result, we design a joint learning method to perform intra-cell and inter-cell NAS simultaneously. We implement our model in a differentiable architecture search system. For recurrent neural language modeling, it outperforms a strong baseline significantly on the PTB and WikiText data, with a new state-of-the-art on PTB. Moreover, the learned architectures show good transferability to other systems. E.g., they improve state-of-the-art systems on the CoNLL and WNUT named entity recognition (NER) tasks and CoNLL chunking task, indicating a promising line of research on large-scale pre-learned architectures.

1 Introduction

Neural models have shown remarkable performance improvements in a wide range of natural language processing (NLP) tasks. Systems of this kind can broadly be characterized as following a neural network design: we model the problem via a pre-defined neural architecture, and the resulting network is treated as a black-box family of functions for which we find parameters that can generalize well on test data. This paradigm leads to many successful NLP systems based on well-designed architectures. The earliest of these makes use of recurrent neural networks (RNNs) for representation learning (Bahdanau et al., 2015; Wu et al., 2016), whereas recent systems have successfully incorporated fully attentive models into language generation and understanding (Vaswani et al., 2017).

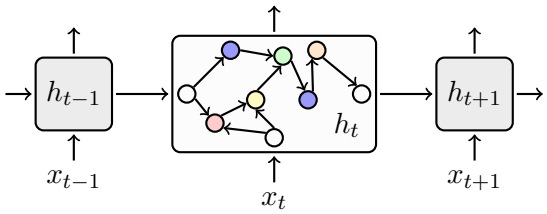
In designing such models, careful engineering of the architecture plays a key role for the state-of-the-art though it is in general extremely difficult to find a good network structure. The next obvious step is toward automatic architecture design. A popular method to do this is neural architecture search (NAS). In NAS, the common practice is that we first define a search space of neural networks, and then find the most promising candidate in the space by some criteria. Previous efforts to make NAS more accurate have focused on improving search and network evaluation algorithms. But the search space is still restricted to a particular scope of neural networks. For example, most NAS methods are applied to learn the topology in a recurrent or convolutional cell, but the connections between cells are still made in a heuristic manner as usual (Zoph and Le, 2017; Elsken et al., 2019).

Note that the organization of these sub-networks remains important as to the nature of architecture design. For example, the first-order connectivity of cells is essential to capture the recurrent dynamics in RNNs. More recently, it has been found that additional connections of RNN cells improve LSTM models by accessing longer history on language modeling tasks (Melis et al., 2019). Similar results appear in Transformer systems. Dense connections of distant layers help in learning a deep Transformer encoder for machine translation (Shen et al., 2018). A natural question that arises is: can we learn the connectivity of sub-networks for better architecture design?

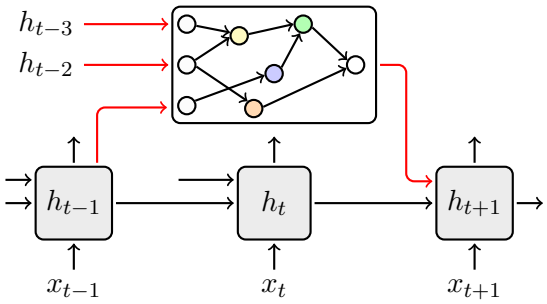
In this paper, we address this issue by enlarging the scope of NAS and learning connections among

*Corresponding author.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 6629–6639



(a) Connections in a cell



(b) Connections among cells

Figure 1: Examples of intra and inter-cell architectures.

sub-networks that are designed in either a handcrafted or automatic way (Figure 1). We call this the Extended Search Space method for NAS (or ESS for short). Here, we choose differentiable architecture search as the basis of this work because it is efficient and gradient-friendly. We present a general model of differentiable architecture search to handle arbitrary search space of NAS, which offers a unified framework of describing intra-cell NAS and inter-cell NAS. Also, we develop a joint approach to learning both high-level and low-level connections simultaneously. This enables the interaction between intra-cell NAS and inter-cell NAS, and thus the ability of learning the full architecture of a neural network.

Our ESS method is simple for implementation. We experiment with it in an RNN-based system for language modeling. On the PTB and WikiText data, it outperforms a strong baseline significantly by 4.5 and 2.4 perplexity scores. Moreover, we test the transferability of the learned architecture on other tasks. Again, it shows promising improvements on both NER and chunking benchmarks, and yields new state-of-the-art results on NER tasks. This indicates a promising line of research on large-scale pre-learned architectures. More interestingly, it is observed that the inter-cell NAS is helpful in modeling rare words. For example, it yields a bigger improvement on the rare entity recognition task (WNUT) than that on the standard NER task (CoNLL).

2 Related work

NAS is a promising method toward AutoML (Hutter et al., 2018), and has been recently applied to NLP tasks (So et al., 2019; Jiang et al., 2019; Li and Talwalkar, 2019). Several research teams have investigated search strategies for NAS. The very early approaches adopted evolutionary algorithms to model the problem (Angeline et al., 1994; Stanley and Miikkulainen, 2002), while Bayesian and reinforcement learning methods made big progresses in computer vision and NLP later (Bergstra et al., 2013; Baker et al., 2017; Zoph and Le, 2017). More recently, gradient-based methods were successfully applied to language modeling and image classification based on RNNs and CNNs (Liu et al., 2019a). In particular, differentiable architecture search has been of great interest to the community because of its efficiency and compatibility to off-the-shelf tools of gradient-based optimization.

Despite of great success, previous studies restricted themselves to a small search space of neural networks. For example, most NAS systems were designed to find an architecture of recurrent or convolutional cell, but the remaining parts of the network are handcrafted (Zhong et al., 2018; Brock et al., 2018; Elsken et al., 2019). For a larger search space, Zoph et al. (2018) optimized the normal cell (i.e., the cell that preserves the dimensionality of the input) and reduction cell (i.e., the cell that reduces the spatial dimension) simultaneously and explored a larger region of the space than the single-cell search. But it is still rare to see studies on the issue of search space though it is an important factor to NAS. On the other hand, it has been proven that the additional connections between cells help in RNN or Transformer-based models (He et al., 2016; Huang et al., 2017; Wang et al., 2018, 2019). These results motivate us to take a step toward the automatic design of inter-cell connections and thus search in a larger space of neural architectures.

3 Inter-Cell and Intra-Cell NAS

In this work we use RNNs for description. We choose RNNs because of their effectiveness at preserving past inputs for sequential data processing tasks. Note that although we will restrict ourselves to RNNs for our experiments, the method and discussion here can be applied to other types of models.

3.1 Problem Statement

For a sequence of input vectors $\{x_1, ..., x_T\}$, an RNN makes a cell on top of every input vector. The RNN cell receives information from previous cells and input vectors. The output at time step t is defined to be:

$$h_t = \pi(\hat{h}_{t-1}, \hat{x}_t) \tag{1}$$

where $\pi(\cdot)$ is the function of the cell. $\hat{h}_{t-1}$ is the representation vector of previous cells, and $\hat{x}_t$ is the representation vector of the inputs up to time step t. More formally, we define $\hat{h}_{t-1}$ and $\hat{x}_t$ as functions of cell states and model inputs, like this

$$\hat{h}_{t-1} = f(h_{[0,t-1]}; x_{[1,t-1]}) \tag{2}$$
$$\hat{x}_t = g(x_{[1,t]}; h_{[0,t-1]}) \tag{3}$$

where $h_{[0,t-1]} = \{h_0, ..., h_{t-1}\}$ and $x_{[1,t-1]} = \{x_1, ..., x_{t-1}\}$. $f(\cdot)$ models the way that we pass information from previous cells to the next. Likewise, $g(\cdot)$ models the case of input vectors. These functions offer a general method to model connections between cells. For example, one can obtain a vanilla recurrent model by setting $\hat{h}_{t-1} = h_{t-1}$ and $\hat{x}_t = x_t$, while more intra-cell connections can be considered if sophisticated functions are adopted for $f(\cdot)$ and $g(\cdot)$.

While previous work focuses on searching for the desirable architecture design of $\pi(\cdot)$, we take $f(\cdot)$ and $g(\cdot)$ into account and describe a more general case here. We separate two sub-problems out from NAS for conceptually cleaner description:

- **Intra-Cell NAS**. It learns the architecture of a cell (i.e., $\pi(\cdot)$).
- **Inter-Cell NAS**. It learns the way of connecting the current cell with previous cells and input vectors (i.e., $f(\cdot)$ and $g(\cdot)$).

In the following, we describe the design and implementation of our inter-cell and intra-cell NAS methods.

3.2 Differentiable Architecture Search

For search algorithms, we follow the method of differentiable architecture search (DARTS). It is gradient-based and runs orders of magnitude faster than earlier methods (Zoph et al., 2018; Real et al., 2019). DARTS represents networks as a directed acyclic graph (DAG) and search for the appropriate architecture on it. For a DAG, the edge $o^{i,j}(\cdot)$ between node pair (i, j) performs an operation to transform the input (i.e., tail) to the output (i.e., head). Like Liu et al. (2019a)'s method and others, we choose operations from a list of activation functions, e.g., sigmoid, identity and etc[1]. A node represents the intermediate states of the networks. For node i, it weights vectors from all predecessor nodes ($j < i$) and simply sums over them. Let s_i be the state of node i. We define s_i to be:

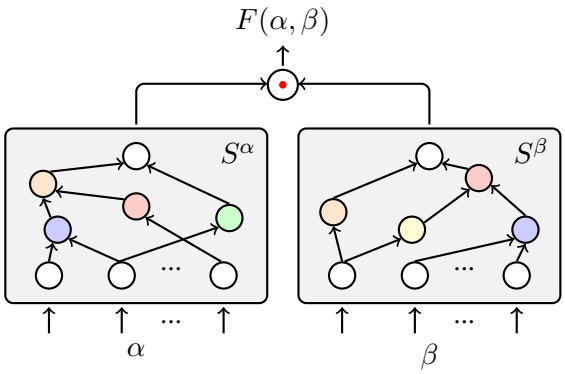


Figure 2: Formalizing intra and inter-cell NAS as learning function $F(\cdot)$.

$$s_i = \sum_{j<i} \sum_k \theta_k^{i,j} \cdot o_k^{i,j}(s_j \cdot W_j) \tag{4}$$

where W_j is the parameter matrix of the linear transformation, and $\theta_k^{i,j}$ is the weight indicating the importance of $o_k^{i,j}(\cdot)$. Here the subscript k means the operation index. $\theta_k^{i,j}$ is obtained by softmax normalization over edges between nodes i and j: $\theta_k^{i,j} = \exp(w_k^{i,j}) / \sum_{k'} \exp(w_{k'}^{i,j})$. In this way, the induction of discrete networks is reduced to learning continuous variables $\{\theta_k^{i,j}\}$ at the end of the search process. This enables the use of efficient gradient descent methods. Such a model encodes an exponentially large number of networks in a graph, and the optimal architecture is generated by selecting the edges with the largest weights.

The common approach to DARTS constraints the output of the generated network to be the last node that averages the outputs of all preceding nodes. Let s_n be the last node of the network. We have

$$s_n = \frac{1}{n-1} \sum_{i=1}^{n-1} s_i \tag{5}$$

Given the input vectors, the network found by DARTS generates the result at the final node s_n.

[1] We also consider a special activation function "drop" that unlinks two nodes.

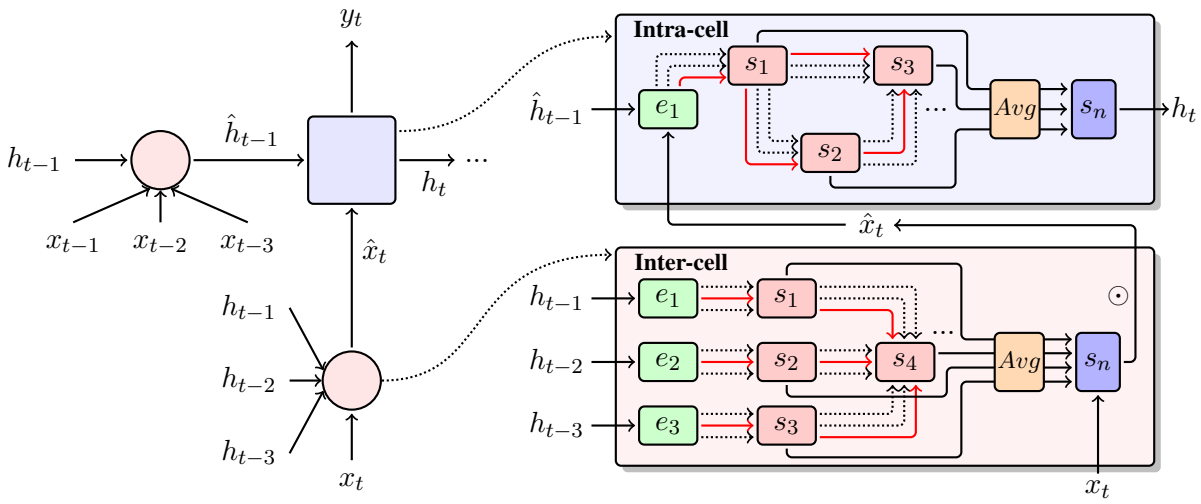


Figure 3: An example of intra-cell and inter-cell NAS in RNN models.

Here we present a method to fit this model into intra and inter-cell NAS. We re-formalize the function for which we find good architectures as $F(\alpha; \beta)$. α and β are two groups of the input vectors. We create DAGs on them individually. This gives us two DAGs with s^α and s^β as the last nodes. Then, we make the final output by a Hadamard product of s^α and s^β, like this,

$$F(\alpha; \beta) = s^\alpha \odot s^\beta \tag{6}$$

See Figure 2 for the network of an example $F(\alpha; \beta)$. This method transforms the NAS problem into two learning tasks. The design of two separate networks allows the model to group related inputs together, rather than putting everything into a "magic" system of NAS. For example, for the inter-cell function $f(\cdot)$, it is natural to learn the pre-cell connection from $h_{[0,t-1]}$, and learn the impact of the model inputs from $x_{[1,t-1]}$. It is worth noting that the Hadamard product of s^α and s^β is doing something very similar to the gating mechanism which has been widely used in NLP (Dauphin et al., 2017; Bradbury et al., 2017; Gehring et al., 2017). For example, one can learn s^β as a gate and control how much s^α is used for final output. Table 1 gives the design of α and β for the functions used in this work.

Another note on $F(\alpha; \beta)$. The grouping reduces a big problem into two cheap tasks. It is particularly important for building affordable NAS systems because computational cost increases exponentially as more input nodes are involved. Our method instead has a linear time complexity if we adopt a reasonable constraint on group size, leading to a possibility of exploring a much larger space during the architecture search process.

Function	α	β
$\pi(\cdot)$	$\{\hat{h}_{t-1}, \hat{x}_t\}$	1
$f(\cdot)$	$h_{[0,t-1]}$	$x_{[1,t-1]}$
$g(\cdot)$	$x_{[1,t]}$	$h_{[0,t-1]}$

Table 1: α and β for different functions

3.3 The Intra-Cell Search Space

The search of intra-cell architectures is trivial. Since $\beta = 1$ and $s^\beta = 1$ (see Table 1), we are basically performing NAS on a single group of input vectors $\hat{h}_{t-1}$ and $\hat{x}_t$. We follow Liu et al. (2019a)'s work and force the input of networks to be a single layer network of $\hat{h}_{t-1}$ and $\hat{x}_t$. This can be described as

$$e_1 = \tanh(\hat{h}_{t-1} \cdot W^{(h)} + \hat{x}_t \cdot W^{(x)}) \tag{7}$$

where $W^{(h)}$ and $W^{(x)}$ are parameters of the transformation, and tanh is the non-linear transformation. e_1 is the input node of the graph. See Figure 3 for intra-cell NAS of an RNN models.

3.4 The Inter-Cell Search Space

To learn $\hat{h}_{t-1}$ and $\hat{x}_t$, we can run the DARTS system as described above. However, Eqs. (2-3) define a model with a varying number of parameters for different time steps, in which our architecture search method is not straightforwardly applicable. Apart from this, a long sequence of RNN cells makes the search intractable.

Function JOINTLEARN ($rounds$, w, W)

1: **for** i in range(1, $rounds$) **do**
2: **while** intra-cell $model$ not converged **do**
3: Update intra-cell $w^{(intra)}$ and W
4: **while** inter-cell $model$ not converged **do**
5: Update inter-cell $w^{(inter)}$ and W
6: Derive $architecture$ based on w
7: **return** $architecture$

Figure 4: Joint search of intra-cell and inter-cell architectures. w = edge weights, and W = model parameters.

For a simplified model, we re-define $f(\cdot)$ and $g(\cdot)$ as:

$$f(h_{[0,t-1]}; x_{[1,t-1]}) = f'(h_{t-1}; x_{[t-m,t-1]}) \quad (8)$$

$$g(x_{[1,t]}; h_{[0,t-1]}) = g'(x_t; h_{[t-m,t-1]}) \quad (9)$$

where m is a hyper-parameter that determines how much history is considered. Eq. (8) indicates a model that learns a network on $x_{[t-m,t-1]}$ (i.e., $\beta = x_{[t-m,t-1]}$). Then, the output of the learned network (i.e., s^{β}) is used as a gate to control the information that we pass from the previous cell to the current cell (i.e., $\alpha = \{h_{t-1}\}$). Likewise, Eq. (9) defines a gate on $h_{[t-m,t-1]}$ and controls the information flow from x_t to the current cell.

Learning $f'(\cdot)$ and $g'(\cdot)$ fits our method well due to the fixed number of input vectors. Note that $f'(\cdot)$ has m input vectors $x_{[t-m,t-1]}$ for learning the gate network. Unlike what we do in intra-cell NAS, we do not concatenate them into a single input vector. Instead, we create a node for every input vector, that is, the input vector $e_i = x_{t-i}$ links with node s_i. We restrict s_i to only receive inputs from e_i for better processing of each input. This can be seen as a pruned network for the model described in Eq. (4). See Figure 3 for an illustration of inter-cell NAS.

4 Joint Learning for Architecture Search

Our model is flexible. For architecture search, we can run intra-cell NAS, or inter-cell NAS, or both of them as needed. However, we found that simply joining intra-cell and inter-cell architectures might not be desirable because both methods were restricted to a particular region of the search space, and the simple combination of them could not guarantee the global optimum.

This necessitates the inclusion of interactions between intra-cell and inter-cell architectures into the search process. Generally, the optimal inter-cell architecture depends on the intra-cell architecture used in search, and vice versa. A simple method that considers this issue is to learn two models in a joint manner. Here, we design a joint search method to make use of the interaction between intra-cell NAS and inter-cell NAS. Figure 4 shows the algorithm. It runs for a number of rounds. In each round, we first learn an optimal intra-cell architecture by fixing the inter-cell architecture, and then learn a new inter-cell architecture by fixing the optimal intra-cell architecture that we find just now.

Obviously, a single run of intra-cell (or inter-cell) NAS is a special case of our joint search method. For example, one can turn off the inter-cell NAS part (lines 4-5 in Figure 4) and learn intra-cell architectures solely. In a sense, the joint NAS method extends the search space of individual intra-cell (or inter-cell) NAS. Both intra-cell and inter-cell NAS shift to a new region of the parameter space in a new round. This implicitly explores a larger number of underlying models. As shown in our experiments, joint NAS learns intra-cell architectures unlike those of the individual intra-cell NAS, which leads to better performance in language modeling and other tasks.

5 Experiments

We experimented with our ESS method on Penn Treebank and WikiText language modeling tasks and applied the learned architecture to NER and chunking tasks to test its transferability.

5.1 Experimental Setup

For language modeling task, the monolingual and evaluation data came from two sources.

- Penn Treebank (PTB). We followed the standard preprocessed version of PTB (Mikolov et al., 2010). It consisted of 929k training words, 73k validation words and 82k test words. The vocabulary size was set to 10k.
- WikiText-103 (WT-103). We also used WikiText-103 (Merity et al., 2017) data to search for a more universal architecture for NLP tasks. This dataset contained a larger training set of 103 million words and 0.2 million words in the validation and test sets.

Dataset	Method	Search Space		Params	Perplexity		Search Cost
		intra-cell	inter-cell		valid	test	(GPU days)
PTB	AWD-LSTM (Merity et al., 2018c)	-	-	24M	61.2	58.8	-
	Transformer-XL (Dai et al., 2019)	-	-	24M	56.7	54.5	-
	Mogrifier LSTM (Melis et al., 2019)	-	-	23M	**51.4**	**50.1**	-
	ENAS (Pham et al., 2018)	✓	-	24M	60.8	58.6	0.50
	RS (Li and Talwalkar, 2019)	✓	-	23M	57.8	55.5	2
	DARTS†	✓	-	23M	55.2	53.0	0.25
	ESS	-	✓	23M	54.1	52.3	0.5
	ESS	✓	✓	23M	**47.9**	**45.6**	0.5
WT-103	QRNN (Merity et al., 2018a)	-	-	151M	32.0	33.0	-
	Hebbian + Cache (Rae et al., 2018)	-	-	-	29.9	29.7	-
	Transformer-XL (Dai et al., 2019)	-	-	151M	**23.1**	**24.0**	-
	DARTS†	✓	-	151M	31.4	31.6	1
	ESS	✓	✓	156M	**28.8**	**29.2**	1.5

Table 2: Comparison of language modeling methods on PTB and WikiText-103 tasks (lower perplexity is better). †Obtained by training the corresponding architecture using our setup.

NER and chunking tasks were also used to test the transferability of the pre-learned architecture. We transferred the intra and inter-cell networks learned on WikiText-103 to the CoNLL-2003 (English), the WNUT-2017 NER tasks and the CoNLL-2000 tasks. The CoNLL-2003 task focused on the newswire text, while the WNUT-2017 contained a wider range of English text which is more difficult to model.

Our ESS method consisted of two components, including recurrent neural architecture search and architecture evaluation. During the search process, we ran our ESS method to search for the intra-cell and inter-cell architectures jointly. In the second stage, the learned architecture was trained and evaluated on the test dataset.

For architecture search on language modeling tasks, we applied 5 activation functions as the candidate operations, including drop, identity, sigmoid, tanh and relu. On the PTB modeling task, 8 nodes were equipped in the recurrent cell. For the inter-cell architecture, it received 3 input vectors from the previous cells and consisted of the same number of the intermediate nodes. By default, we trained our ESS models for 50 rounds. We set $batch = 256$ and used 300 hidden units for the intra-cell model. The learning rate was set as 3×10^{-3} for the intra-cell architecture and 1×10^{-3} for the inter-cell architecture. The BPTT (Werbos, 1990) length was 35. For the search process on WikiText-103, we developed a more complex model to encode the representation. There were 12 nodes in each cell and 5 nodes in the inter-cell networks. The batch size was 128 and the number of hidden units was 300 which was the same with that on the PTB task. We set the intra-cell and inter-cell learning rate to 1×10^{-3} and 1×10^{-4}. A larger window size ($= 70$) for BPTT was applied for the WikiText-103. All experiments were run on a single NVIDIA 1080Ti.

After the search process, we trained the learned architectures on the same data. To make it comparable with previous work, we copied the setup in Merity et al. (2018b). For PTB, the size of hidden layers was set as 850 and the training epoch was 3,000. While for the WikiText-103, we enlarged the number of hidden units to 2,500 and trained the model for 30 epochs. Additionally, we transferred the learned architecture to NER and chunking tasks with the setting in Akbik et al. (2019). We only modified the batch size to 24 and hidden size to 512.

5.2 Results

5.2.1 Language Modeling tasks

Here we report the perplexity scores, number of parameters and search cost on the PTB and WikiText-103 datasets (Table 2). First of all, the joint ESS method improves the performance on language modeling tasks significantly. Moreover, it does not introduce many parameters. Our ESS method achieves state-of-the-art result on the PTB task. It outperforms the manually designed Mogrifier-LSTM by 4.5 perplexity scores on the test set. On

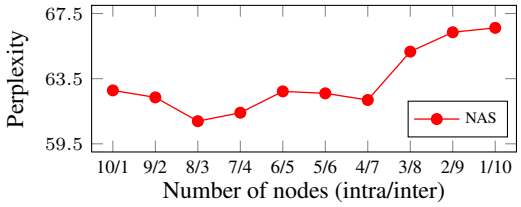


Figure 5: Perplexity on the validation data (PTB) vs. number of nodes in intra and inter-cell.

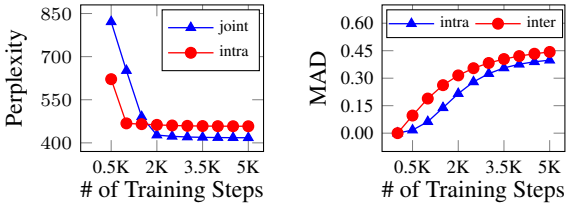


Figure 6: Perplexity on the validation data (PTB) and Mean Absolute Deviation (MAD) between edge weights and uniform distribution vs. number of training steps.

the WikiText task, it still yields a +2.4 perplexity scores improvement over the strong NAS baseline (DARTS) method. These results indicate that ESS is robust and can learn better architectures by enlarging the scope of search space.

Also, we find that searching for the appropriate connections among cells plays a more important role in improving the model performance. We observe that the intra-cell NAS (DARTS) system underperforms the inter-cell counterpart with the same number of parameters. It is because the well-designed intra-cell architectures (e.g., Mogrifier-LSTM) are actually competitive with the NAS structures. However, the fragile connections among different cells greatly restrict the representation space. The additional inter-cell connections are able to encode much richer context.

Nevertheless, our ESS method does not defeat the manual designed Transformer-XL model on the WikiText-103 dataset, even though ESS works better than other RNN-based NAS methods. This is partially due to the better ability of Transformer-XL to capture the language representation. Note that RNNs are not good at modeling the long-distance dependence even if more history states are considered. It is a good try to apply ESS to Transformer but this is out of the scope of this work.

5.2.2 Sensitivity Analysis

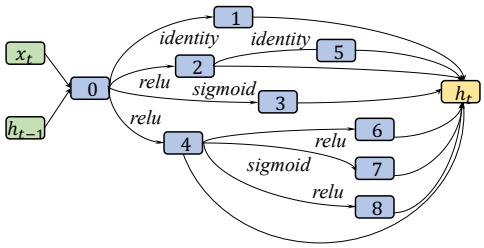
To modulate the complexity of the intra and inter-cell, we study the system behaviors under different numbers of intermediate nodes (Figure 5). Fixing the number of model parameters, we compare these systems under different numbers of the intra and inter-cell nodes. Due to the limited space, we show the result on the PTB in the following sensitivity analysis. We observe that an appropriate choice of node number (8 nodes for intra-cell and 3 nodes for inter-cell) brings a consistent improvement. More interestingly, we find that too many nodes for inter-cell architecture do not improve the model representation ability. This is reasonable because more inter-cell nodes refer to considering more history in our system. But for language modeling, the current state is more likely to be relevant to most recent words. Too many inputs to the gate networks raise difficulties in modeling.

Word	Count	Δloss	Word	Count	Δloss
mcmoran	11	-0.74	the	59421	-0.009
cie.	9	-0.66	<unk >	53299	-0.004
mall	13	-0.65	<eos >	49199	-0.010
missile	23	-0.55	N	37607	-0.008
siemens	12	-0.51	of	28427	-0.008
baldwin	9	-0.51	to	27430	-0.004
nfl	21	-0.49	a	24755	-0.013
prime-time	17	-0.47	in	21032	-0.015

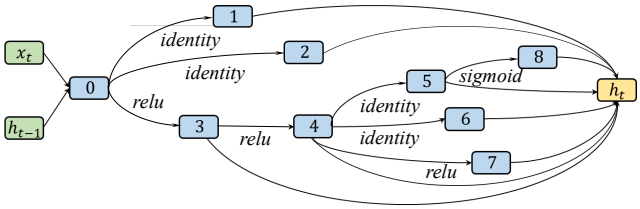
Table 3: Difference in word loss (normalized by word counts) on validation data when searching intra and inter-cell jointly. The left column contains the words with eight best improvements (larger absolute value of Δloss) and right column presents the most frequent words in the validation data.

We observe that our ESS method leads to a model that is easier to train. The left part in Figure 6 plots the validation perplexity at different training steps. The loss curve of joint ESS significantly goes down as the training proceeds. More interestingly, our joint learning method makes the model achieve a lower perplexity than the intra-cell NAS system. This indicates better networks can be obtained in the search process. Additionally, the convergence can be observed from the right part in Figure 6. Here we apply Mean Absolute Deviation (MAD) to define the distance between edge weights and initial uniform distribution. It is obvious that both the intra and inter-cell architectures change little at the final searching steps.

In order to figure out the advantage of inter-cell connections, we detail the model contribution on each word on the validation data. Specifically, we compute the difference in word loss function (i.e.,



(a) An intra-cell architecture found by using inter-cell connections



(b) An intra-cell architecture found without using inter-cell connections

Figure 7: Comparison of intra-cell architectures found by using and not using additional inter-cell connections

Models	F1
LSTM-CRF (Lample et al., 2016)	90.94
LSTM-CRF + ELMo (Peters et al., 2018)	92.22
LSTM-CRF + Flair (Akbik et al., 2019)	93.18
GCDT + $\text{BERT}_{\text{LARGE}}$ (Liu et al., 2019b)	93.47
CNN Large + ELMo (Baevski et al., 2019)	**93.50**
DARTS + Flair (Jiang et al., 2019)	93.13
I-DARTS + Flair (Jiang et al., 2019)	93.47
ESS	91.78
ESS + Flair	**93.62**

Table 4: F1 scores on CoNLL-2003 NER task. Bi-LSTM

Models	F1
Cross-BiLSTM-CNN (Aguilar et al., 2018)	45.55
Flair (Akbik et al., 2019)	**50.20**
DARTS + Flair†	50.34
ESS	48.85
ESS + Flair	**52.18**

Table 5: F1 scores on WNUT-2017 NER task. †Obtained by training the corresponding architecture using our setup.

Models	F1
NCRF++ (Yang and Zhang, 2018)	95.06
BiLSTM-CRF + IntNet (Xin et al., 2018)	95.29
Flair (Akbik et al., 2019)	96.72
GCDT + $\text{BERT}_{\text{LARGE}}$ (Liu et al., 2019b)	**97.30**
DARTS + Flair†	96.59
ESS	95.51
ESS + Flair	**97.22**

Table 6: F1 scores on CoNLL-2000 chunking task. †Obtained by training the corresponding architecture using our setup.

log perplexity) between methods with and without inter-cell NAS. The words with eight best improvements are shown in the left column of Table 3. We observe that the rare words in the training set obtain more significant improvements. In contrast, the most frequent words lead to very modest decrease in loss (right column of Table 3). This is because the connections between multiple cells enable learning rare word representations from more histories. While for common words, they can obtain this information from rich contexts. More inputs from previous cells do not bring much useful information.

Additionally, we visualize the learned intra-cell architecture in Figure 7(a). The networks are jointly learned with the inter-cell architecture. Compared with the results of intra-cell NAS (Figure 7(b)), the learned network is more shallow. The inter-cell architectures have deeper networks. This in turn reduces the need for intra-cell capacity. Thus a very deep intra-cell architecture might not be necessary if we learn the whole model jointly.

5.2.3 Transferring to Other Tasks

After architecture search, we test the transferability of the learned architecture. In order to apply the model to other tasks, we directly use the architecture searched on WikiText-103 and train the parameters with the in-domain data. In our experiments, we adapt the model to CoNLL-2003, WNUT-2017 NER tasks and CoNLL-2000 chunking task.

For the two NER tasks, it achieves new state-of-the-art F1 scores (Table 4 and Table 5). ELMo, Flair and $\text{BERT}_{\text{LARGE}}$ refer to the pre-trained language models. We apply these word embeddings to the learned architecture during model training process. For the chunking task, the learned architecture also shows greater performance than other NAS methods (Table 6). Moreover, we find that our pre-learned neural networks yield bigger improvements on the WNUT-2017 task. The difference of the two NER tasks lies in that the WNUT-2017 task is a long-tail emerging entities recognition task. It focuses on identifying unusual, previously-unseen entities in the context of emerging discussions. As we discuss in the previous part of the section, the additional inter-cell NAS is good at learning the representations of rare words. Therefore, it makes

sense to have a bigger improvement on WNUT-2017.

6 Conclusions

We have proposed the Extended Search Space (ESS) method of NAS. It learns intra-cell and inter-cell architectures simultaneously. Moreover, we present a general model of differentiable architecture search to handle the arbitrary search space. Meanwhile, the high-level and low-level sub-networks can be learned in a joint fashion. Experiments on two language modeling tasks show that ESS yields improvements of 4.5 and 2.4 perplexity scores over a strong RNN-based baseline. More interestingly, it is observed that transferring the pre-learned architectures to other tasks also obtains a promising performance improvement.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Nos. 61876035 and 61732005), the National Key R&D Program of China (No. 2019QY1801) and the Opening Project of Beijing Key Laboratory of Internet Culture and Digital Dissemination Research. The authors would like to thank anonymous reviewers for their comments.

References

Gustavo Aguilar, Adrian Pastor López-Monroy, Fabio González, and Thamar Solorio. 2018. Modeling noisiness to recognize named entities using multitask neural networks on social media. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1401–1412, New Orleans, Louisiana. Association for Computational Linguistics.

Alan Akbik, Tanja Bergmann, and Roland Vollgraf. 2019. Pooled contextualized embeddings for named entity recognition. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page 724–728.

Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. 1994. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, 5(1):54–65.

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5359–5368, Hong Kong, China. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 115–123.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2018. SMASH: one-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 933–941.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Efficient multi-objective neural architecture search via lamarckian evolution. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceed-*

ings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pages 1243–1252.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. 2018. *Automated Machine Learning: Methods, Systems, Challenges*. Springer. In press, available at http://automl.org/book.

Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2019. Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3583–3588, Hong Kong, China. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 260–270.

Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 129.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Yijin Liu, Fandong Meng, Jinchao Zhang, Jinan Xu, Yufeng Chen, and Jie Zhou. 2019b. GCDT: A global context enhanced deep transition architecture for sequence labeling. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2431–2441.

Gábor Melis, Tomáš Kočiský, and Phil Blunsom. 2019. Mogrifier lstm.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018a. An analysis of neural language modeling at multiple scales. *CoRR*, abs/1803.08240.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018b. An analysis of neural language modeling at multiple scales. *CoRR*, abs/1803.08240.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018c. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Stephen Merity, Bryan McCann, and Richard Socher. 2017. Revisiting activation regularization for language rnns.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4092–4101.

Jack W. Rae, Chris Dyer, Peter Dayan, and Timothy P. Lillicrap. 2018. Fast parametric learning with activation memorization. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4225–4234.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. volume 33, page 4780–4789. Association for the Advancement of Artificial Intelligence (AAAI).

Yanyao Shen, Xu Tan, Di He, Tao Qin, and Tie-Yan Liu. 2018. Dense information flow for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1294–1303.

David R. So, Quoc V. Le, and Chen Liang. 2019. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning, ICML*

2019, 9-15 June 2019, Long Beach, California, USA, pages 5877–5886.

Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Qiang Wang, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. 2018. Multi-layer representation fusion for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 3015–3026.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yingwei Xin, Ethan Hart, Vibhuti Mahajan, and Jean-David Ruvini. 2018. Learning better internal structure of words for sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2584–2593, Brussels, Belgium. Association for Computational Linguistics.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia. Association for Computational Linguistics.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2423–2432.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8697–8710.