NAS论文学习笔记

1）NAS with RL - 17 ICLR

neural architecture search with reinforcement learning
用强化学习RL架构进行NAS工作

贡献点

提出了以RNN作为控制器（Controller）用于NAS的RL架构

算法

最大化期望奖励

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

概率 P 累乘转为对数累加

$$\begin{aligned}\bigtriangledown_{\theta_c} J(\theta_c) &= \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\left[\bigtriangledown_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\right] \\ &\approx \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \bigtriangledown_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R_k\end{aligned}$$

为减小方差，引入基线（baseline）b:

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \bigtriangledown_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)\left(R_k - b\right)$$

生成简单CNN层示例

由RNN预测诸如层（filter）、跨步（stride）大小，通道数（channel）等超参数，一层一层从前往后预测

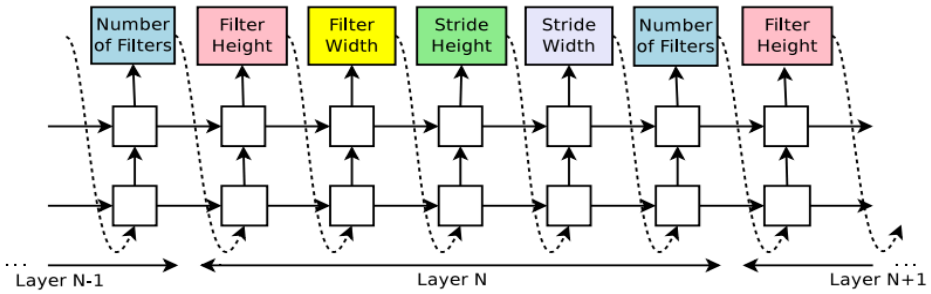


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

生成跳跃连接

简而言之，每个层（Layer）标记一个锚点（anchor point），每一对层（相邻层）决定是否形成跳跃连接，其中W_{prev}、W_{curr}和v是可训练的参数，h_k是RNN控制器到第k层时的隐藏状态（hiddenstate）

$$\mathrm{P}(\text{Layer j is an input to layer i}) = \mathrm{sigmoid}(v^{\mathrm{T}}\tanh(W_{prev} * h_j + W_{curr} * h_i))$$

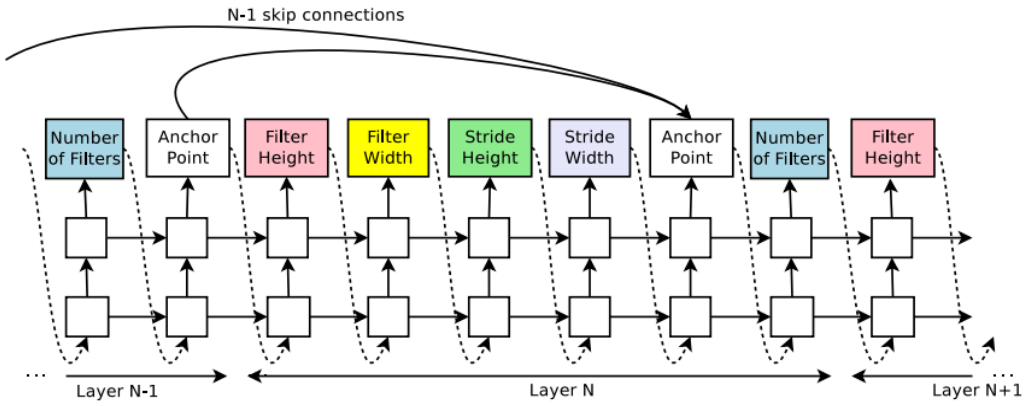


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

并行异步架构，加快训练

分布式架构，多台服务器一起上

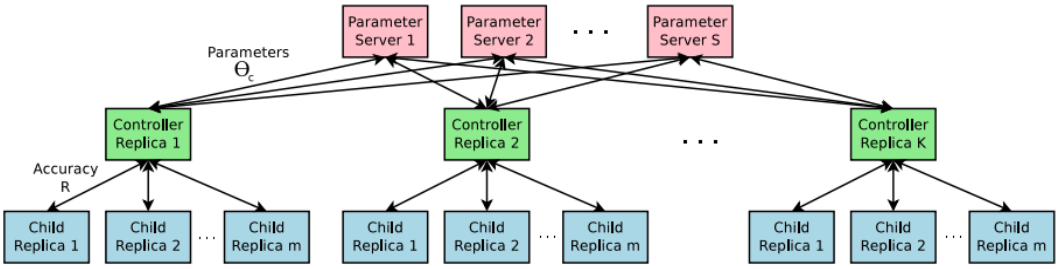


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to compute the gradients with respect to θ_c, which are then sent back to the parameter servers.

~~就算是并行计算，那也是钱砸出来的，试那么多个模型还是很慢啊~~

生成RNN架构

基本原理就是依次生成运算方法或激活函数，最后指定哪个函数归属哪个结点，比如下图中的1,0，那么 Add,Tanh 就属于 Tree Index 1， ElemMult,ReLU 就属于 Tree Index 0 。

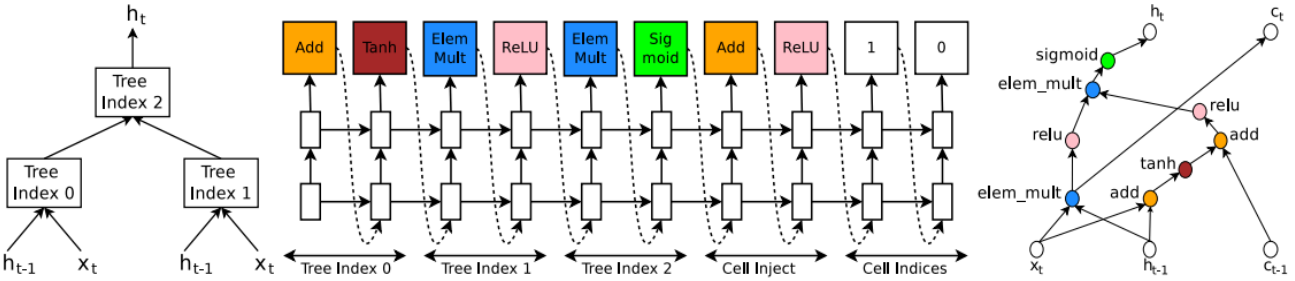
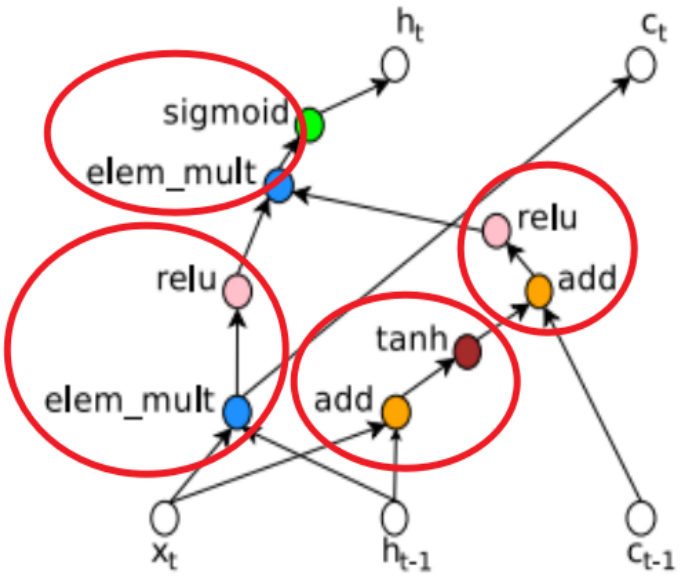


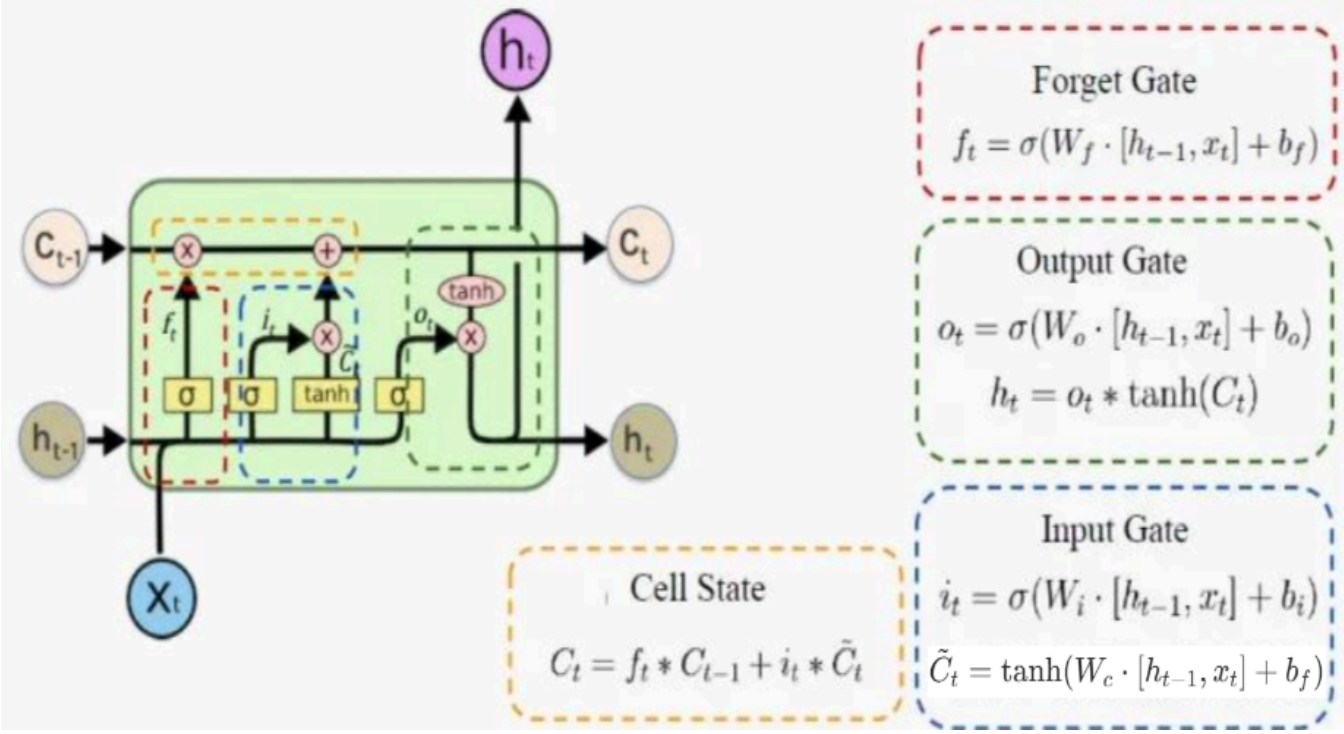
Figure 5: An example of a recurrent cell constructed from a tree that has two leaf nodes (base 2) and one internal node. Left: the tree that defines the computation steps to be predicted by controller. Center: an example set of predictions made by the controller for each computation step in the tree. Right: the computation graph of the recurrent cell constructed from example predictions of the controller.

划分块示意图如下，可以清楚的看到 Tree Index i 对应的架构，其中 Cell Inject 指的是上一个细胞状态C_{t-1}的处理方式：



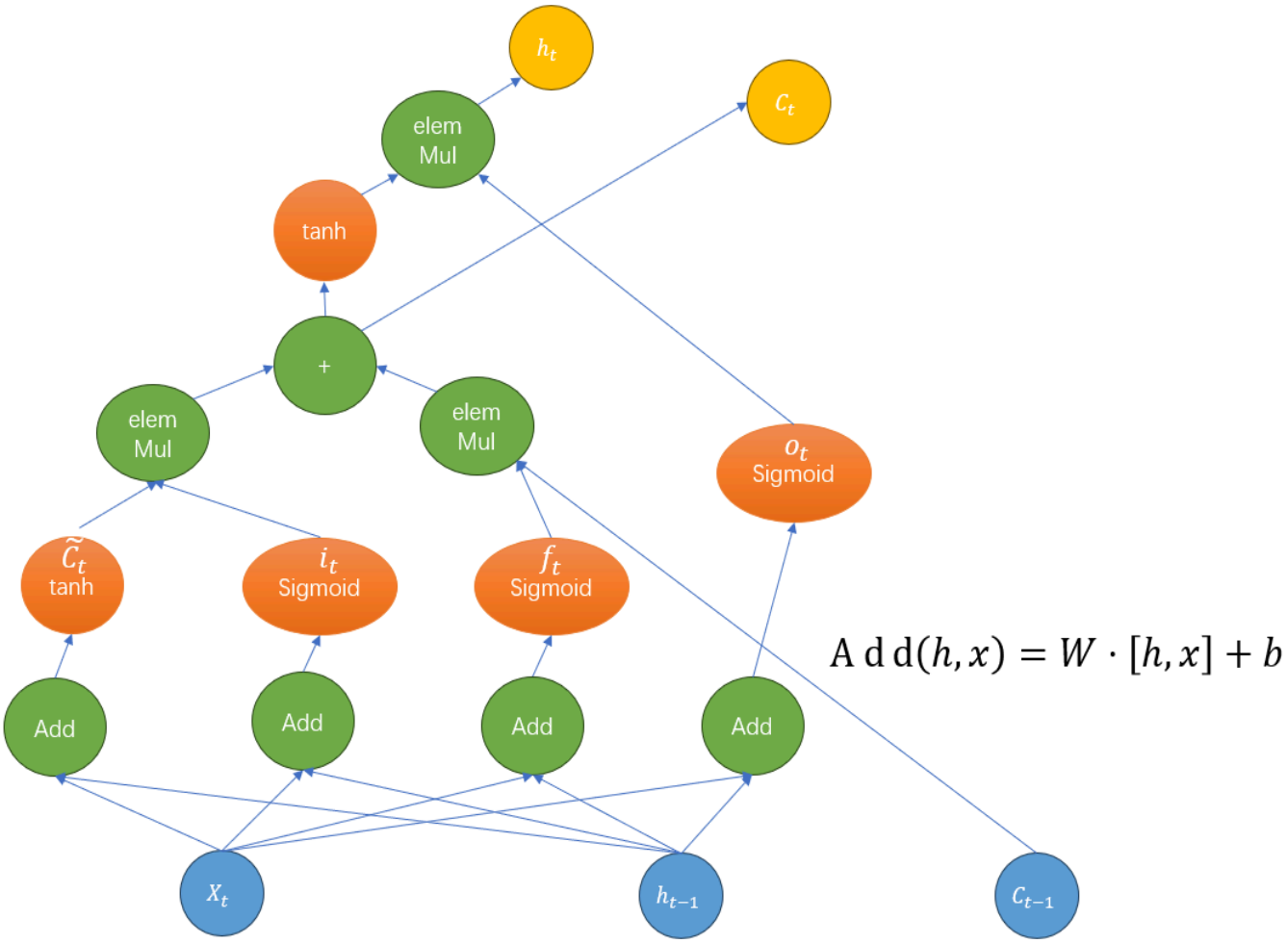
示例为"base 2"架构（实际训练是"base 8"架构）

其中细胞状态C_t是借鉴了LSTM的设计原理：



~~知乎上扒的，有点糊...~~

下图是笔者基于论文RNN设计原理整理的生成LSTM的示意图，至于看上去像是base多少看不太出来，maybe base 4



问题和缺点

大概就是跑的仍然比较慢，不过论文里没挂出来训练多久，只说同时用了800个GPU，都是之后引用的论文的挂出来的（NASNet那篇说500个GPU跑了28天 = 1w4 GPU days）。~~扬长避短~~

2）SMASH - 18 ICLR

SMASH: One-Shot Model Architecture Search through HyperNetworks
通过构建超网（HyperNet）实现单次模型架构搜索

算法：

Algorithm 1 SMASH

input Space of all candidate architectures, $\mathbb{R}_c$
  Initialize HyperNet weights H
  loop
    Sample input minibatch x_i, random architecture c and architecture weights $W = H(c)$
    Get training error $E_t = f_c(W, x_i) = f_c(H(c), x_i)$, backprop and update H
  end loop
  loop
    Sample random c and evaluate error on validation set $E_v = f_c(H(c), x_v)$
  end loop
  Fix architecture and train normally with freely-varying weights W

解释：

1. 首先训练超网H（**一个**输入为神经架构，输出为权重的特殊**神经网络**）

- 采样小批量输入x_i，随机架构c和随机权重$W = H(c)$
- 得到训练误差反向传播更新H

2. 随机采样大量架构c，找到一个最好的c_0

- 在验证集上评估找到最好的c_0

3. 正常训练c_0的权重，得到结果

贡献点

1. 如何采样架构？论文提出了Memory-Bank，将采样拓扑编码为二进制向量
2. 如何采样权重？采用超网（HyperNet）得到二进制拓扑向量所映射的权重空间（其实就是跑一遍超网H）

如何采样架构

memery-bank读取和写入概念（以ResNet,DenseNet等为例）：

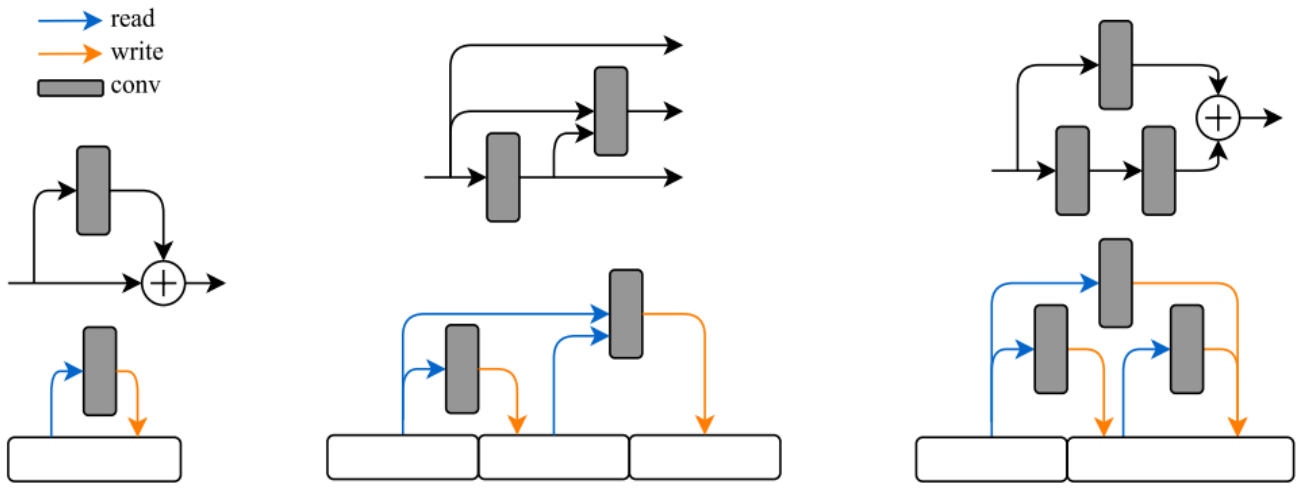


Figure 1: Memory-Bank representations of ResNet, DenseNet, and FractalNet blocks.

其中白色矩形为 Memery-Bank，用于存储神经网络架构（使用预编码规则的二进制向量存储）（只有一个大Memery-Bank供存储，不是1、3、2个Memery-Bank，~~笔者起初误解了~~）

读取:

采样（随机或启发式）网络架构，用于评估、训练或进一步搜索。

写入:

生成网络架构（随机、进化算法或其他方法得出），编码后存入。

以CNN为例，论文中采用了下右图的基本网络框架，其中trans为 $\mathbf{1}\times\mathbf{1}$卷积 $+$ 平均池化 构建的下采样（downsample）层，卷积的权重由学习得来。

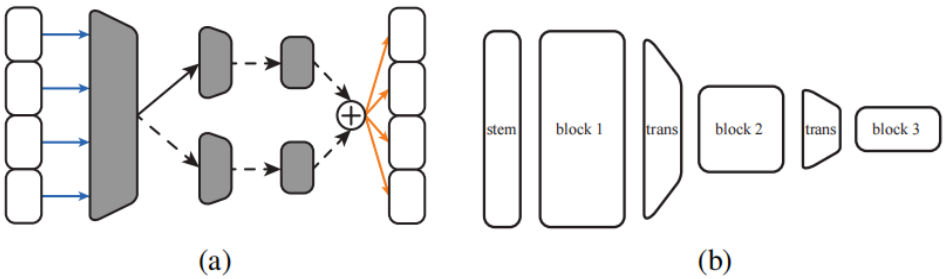


Figure 2: (a) Structure of one op: A 1x1 conv operating on the memory banks, followed by up to 2 parallel paths of 2 convolutions each. (b) Basic network skeleton.

上左图显示了每个块（block）中的一个操作，它包含一个 $\mathbf{1}\times\mathbf{1}$卷积，然后是可变数量且非线性交错的卷积。

~~实现细节看不太懂~~

问题和缺点

不难发现，由于是一次性训练的超网，同一个架构所用的是同一组权重，然而这可能并不合适，对于不同的架构，可能存在不同的特征和需求，使用同一组权重可能无法充分捕捉到每个架构的特性。这可能会导致某些架构的性能不足，尤其是在架构间有显著差异时。

3）NASNet - 18 CVPR

Learning Transferable Architectures for Scalable Image Recognition
提出了新颖的搜索空间，即常规单元和缩减单元（Normal Cell and Reduction Cell），采用这种搜索空间甚至使得采用强化学习方法只比随机搜索强一点。

贡献点

非常好**搜索空间**，常规单元和缩减单元（Normal Cell and Reduction Cell），使得训练得到的网络对数据集分辨率的变化具有较高的鲁棒性，具有可迁移、可扩展性。

算法

对于缩减单元（Reduction Cell）的解释

常规单元（Normal Cell）是不改变图像特征图的卷积单元，而缩减单元（Reduction Cell）则将特征图的长宽减半，具体操作是将第一个 operation 的步幅（stride）设置为2。

引用21年kbsAutoML综述（指 AutoML: A survey of the state-of-the-art ）的图来解释架构：

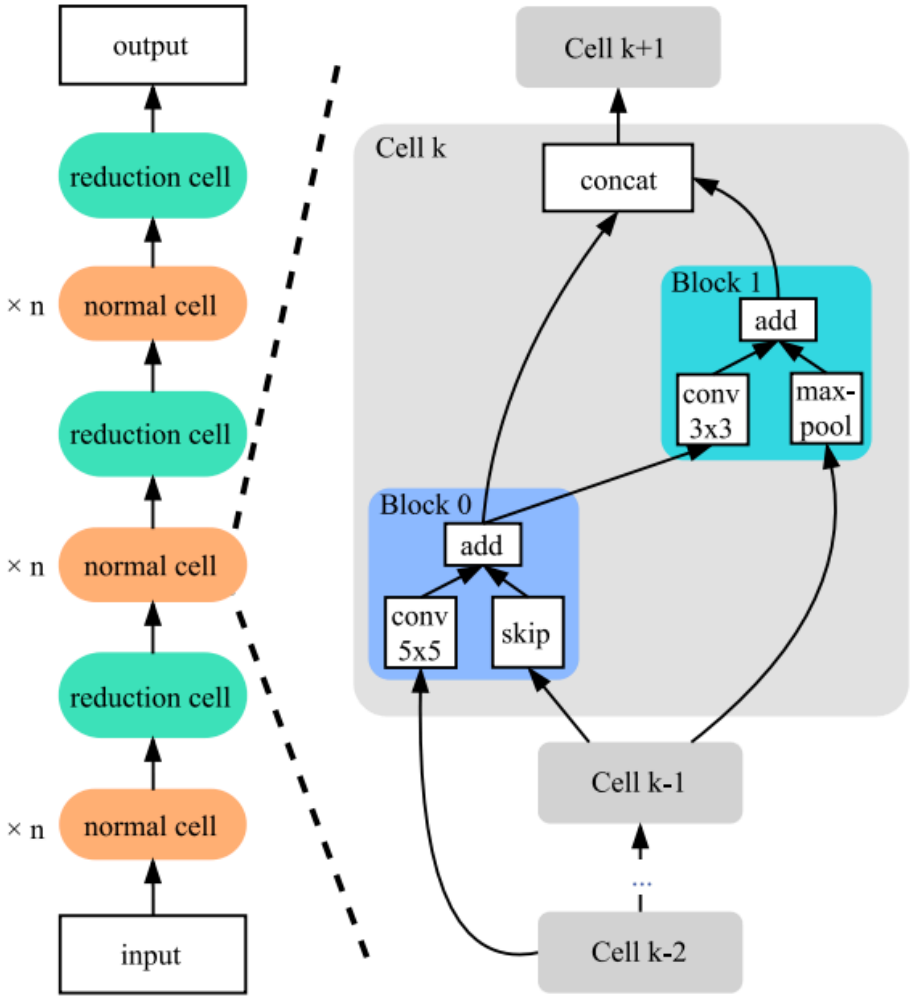


Fig. 7. (Left) Example of a cell-based model comprising three motifs, each with n normal cells and one reduction cell. (Right) Example of a normal cell, which contains two blocks. Each block has two nodes, and each node is specified with different operation and input. The reduction cell is similar in structure to the normal cell.

Normal Cell 和 Reduction Cell 的架构相似，每个 Cell 有 B 个 Block，每个 Cell 相同 - 缩小搜索空间，每个 Block 之间不同（当然小概率相同） - 保证灵活性，以下所有搜索方法都是基于 Block 的。

生成 Block

NasNet Search Space 大纲图

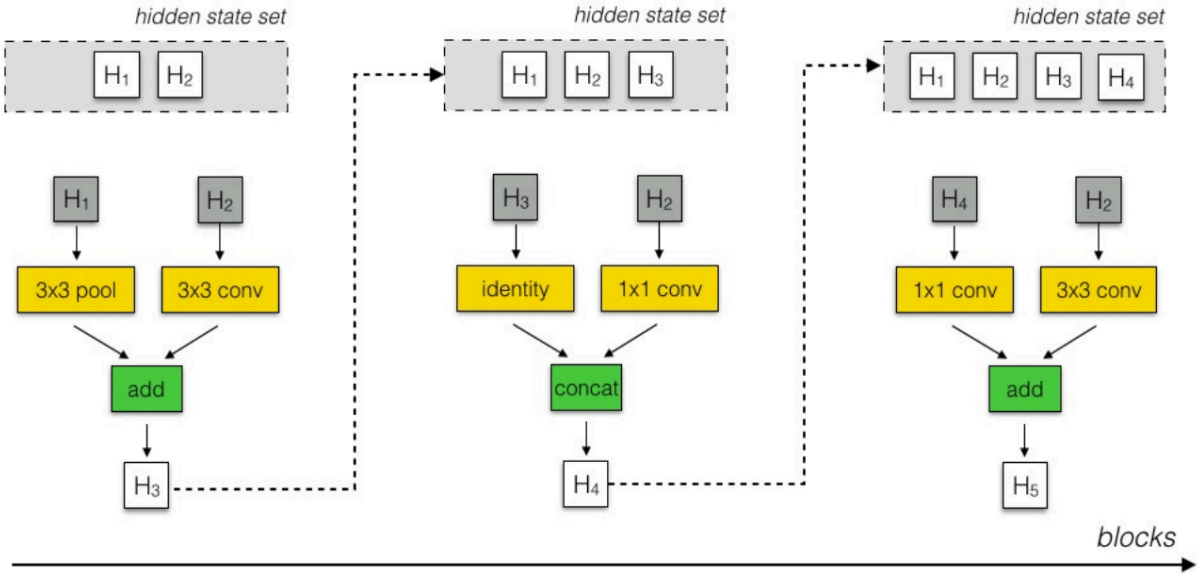


Figure 7. Schematic diagram of the NASNet search space. Network motifs are constructed recursively in stages termed blocks. Each block consists of the controller selecting a pair of hidden states (dark gray), operations to perform on those hidden states (yellow) and a combination operation (green). The resulting hidden state is retained in the set of potential hidden states to be selected on subsequent blocks.

hidden state set：定义一个隐藏状态集合，在第一个 Block 的初始状态下，内含两个隐藏状态 H_1, H_2，对于前两层（即前两个 Cell），他们由 Input Image 产生（~~具体也没说~~）；对于非第一层，他们由前两层产生。

生成 Block 只需要5个简单步骤：

1. 从 hidden state set 中挑第一个 h_1 （怎么挑？）
2. 从 hidden state set 中挑第二个 h_2
3. 为 h_1 找一个 operation_1（怎么找？）
4. 为 h_2 找一个 operation_2
5. 找一个能够组合 operation_1 和 operation_2 的输出来创建一个新 h_3 的方法，随后将 h_3 加入 hidden state set

以上“怎么挑”，“怎么找”都是RNN控制器需要训练的内容。

即如下的生成 one block of a convolutional cell 流程图

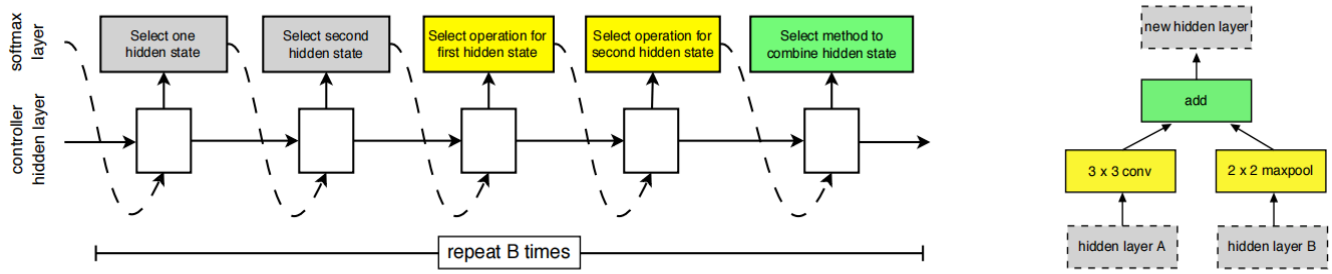


Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each block requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

其中候选 operation 如下：

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

统计：有 2 种 Cell，每个 Cell 有 B 个 Block，每个 Block 有 5 个 step，因此RNN预测器（Controller）总共需做 $\mathbf{2 \times 5B}$ 次预测。

有的同学可能听到这大概懂 Block 整个设计的流程了，也懂 Block 的不同是怎么来的了，可要是问**为什么要设计 hidden state** 呢?

还记得上文提到的灵活性吗，保证灵活性可不只是提高 Block 的多样性，还有**同一个 Cell 内部 Block 的拓扑结构**，即不同 Block 的连接方式，这可比单单 Block 的种类多了去了。而一个 Block 由小巧玲珑的两个输入一个输出组成，因此要找前面的两个层或输入图像（two lower layers or input image），其实就是既结合了不同 Block 之间的连接多样性，又实现了 Cell 的残差连接（通过 Block_0 实现），从而实现了较小的（相比于完全架构搜索（searching for an entire structure），见下图 - 引用自21kbsAutoML综述）搜索空间，妙哉妙哉~

Complexity. Searching for a cell structure is more efficient than searching for an entire structure. To illustrate this, let us assume that there are M predefined candidate operations, the number of layers for both entire and the cell-based structures is L, and the number of blocks in a cell is B. Then, the number of possible entire structures can be expressed as:

$$N_{entire} = M^L \times 2^{\frac{L\times(L-1)}{2}} \tag{2}$$

The number of possible cells is $(M^B \times (B+2)!)^2$. However, as there are two types of cells (i.e., normal and reduction cells), the final size of the cell-based search space is calculated as

$$N_{cell} = (M^B \times (B+2)!)^4 \tag{3}$$

Evidently, the complexity of searching for the entire structure grows exponentially with the number of layers. For an intuitive comparison, we assign the variables in Eqs. (2) and (3) the typical value in the literature, i.e., $M = 5, L = 10, B = 3$; then $N_{entire} = 3.44 \times 10^{20}$ is much larger than $N_{cell} = 5.06 \times 10^{16}$.

4）ENAS - 18 ICLR

Efficient Neural Architecture Search via Parameter Sharing

通过**参数共享**进行高效搜索

这也有一个超网，不过在论文中的表述是 A Large Computational Graph，即一张大型计算图G，其中任何神经架构都是这张计算图的子图（是Graph，如下图所示，区别于SMASH（Network）），每个边或节点保存的是一个神经架构（卷积或全连接层之类的）的权重，这样保证除了第一遍跑的时候是随机初始化，其余情况一开始直接拿这张大网G上的权重跑，这比随机初始化更快收敛，收敛后再在G上更新权重。

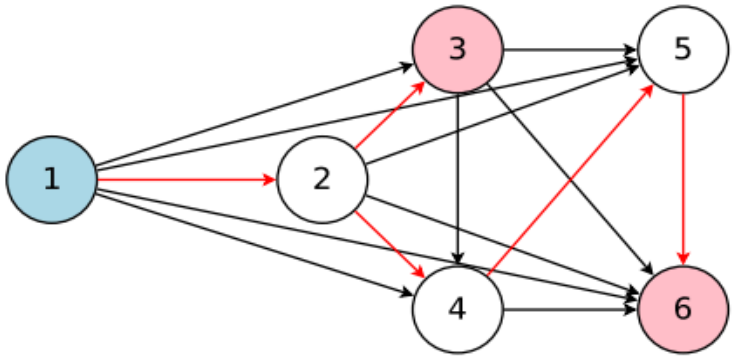


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

算法

采用的还是强化学习（RL）方法，其中控制器由一个LSTM组成，它的主要任务是什么呢，如上图所示，对于每个节点，它主要是决定之前需要连哪个节点，还有采样什么激活函数（activation function）。

两组参数**交替**训练：

1. LSTM 控制器（controller）的参数 θ
2. 子模型共享参数（就是那张大型计算图 G 的参数）ω

还是基于 Cell 的训练

RNN

Controller对每个结点进行前结点采样和激活函数采样，权重取 G 中的 $\mathbf{W}_{i,j}^{\mathbf{h}}$（$\mathbf{h}$ 代表隐藏状态），最后对所有汇点取平均后输出。

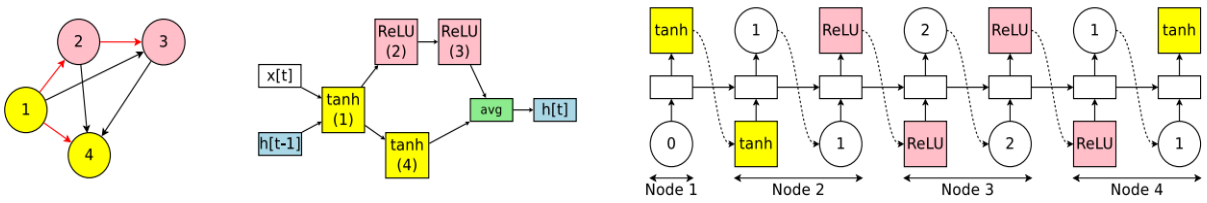


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell's output.

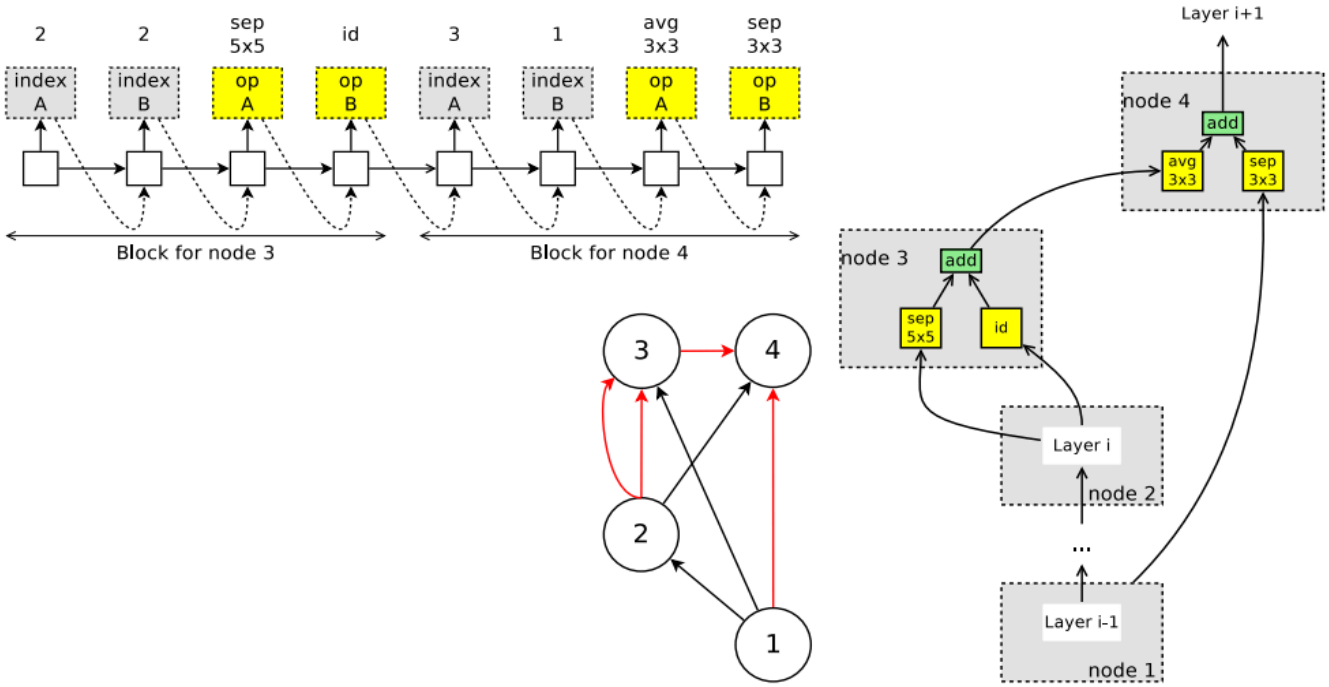
在Penn Treebank数据集上只跑了0.45个 GPU days（一张卡跑了10h）

CNN

先是试了试基于 Layer 的，理论可行，但是搜索空间大（候选操作 $M = 6$，层数 $L = 12$ 时，搜索空间大小 $M^L \times 2^{\frac{L(L-1)}{2}} = 1.6 \times 10^{29}$），收敛太慢

再是基于 Cell 的，这里引用了NASNet的搜索空间，即引入常规单元和缩减单元（Normal Cell and Reduction Cell）

过程也与之前类似，一个 Cell 含 B 个 Block，对于 node_i（$2 <= i < B$），为其挑选两个之前的节点作为输入，再选两个对应的操作（operation），将结果相加后输出。



缩减单元同理。

候选操作 $M = 5$，每单元块数 $B = 7$ 时，，搜索空间大小 $(M^B \times (B-2)!)^4 = 7.7 \times$

10^{27}，相比 1.6×10^{29} 小多了。（原文是 $M = 5, B = 7, (M \times (B-2)!)^4 = 1.3 \times 10^{11}$，感觉有点问题，应该是 $(M^B \times (B-2)!)^4$ ）

关于 $(M^B \times (B-2)!)^4$ 的解释：

1. 除前两个块外排列组合 $(B-2)!$
2. 每个块两条 input 的排列组合 $((B-2)!)^2$
3. 每条 input 都有一个对应的 operation $(M^B \times (B-2)!)^2$
4. 常规单元和缩减单元独立采样 $(M^B \times (B-2)!)^4$

考虑到不同的 operation 对应的权重不一样（比如 3×3 的权重和 5×5 的权重），无法共用，因此应该使用多个类似于 $\mathbf{W}_{i,j}^{\mathbf{sep\_conv\_3 \times 3}}$，$\mathbf{W}_{i,j}^{\mathbf{conv\_5 \times 5}}$ 的权重，而不是单个 $\mathbf{W}_{i,j}$

> **可分离卷积（Separable Convolution）Tips**:
> 其中 $\mathbf{sep\_conv\_3 \times 3}$ 代表 3×3 的可分离卷积，那么啥是分离卷积?
> 对于图像大小为 N，输入、输出通道数分别为 I、O，卷积核大小为 S
>
> 1. 传统卷积计算量
>
> $$\mathrm{N} \times \mathrm{I} \times \mathrm{O} \times \mathrm{S}$$
>
> 2. 分离卷积计算量（深度卷积 + 逐点卷积）
>
> - **深度卷积**:
>
> $$\mathrm{N} \times \mathrm{I} \times \mathrm{S}$$
>
> - **逐点卷积**:
>
> $$\mathrm{N} \times \mathrm{I} \times \mathrm{O}$$

In summary, 可分离卷积通过削弱传统卷积的灵活性，大大减少了计算量。