Probabilistic Neural Architecture Search

Chen Zhao

(Arxiv 2019, `https://arxiv.org/pdf/1902.05116.pdf`)

1 Motivations and Contributions

- The trade-off between high computational cost and high memory cost has resulted in the usage of different types of surrogates (or proxies) during architecture search, architecture surrogates and dataset surrogates.
  - Architecture surrogates are small versions of the final network that are cheaper to store in memory and faster to train.
  - Dataset surrogates are datasets that can be used as a proxy to perform a quicker search. For example, one could perform architecture search on CIFAR-10 and "transfer" the resulting architecture (with some modifications, e.g. changing the number of filters) to ImageNet.
- While good performance on a surrogate task does not guarantee good performance on the final task (i.e. fully-sized network on the target dataset), the vast majority of architecture search methods use at least one, if not both, of these surrogates.
- The proposed procedure PARSEC (Probabilistic neural ARchitecture SEarCh) is memory-efficient.
- PARSEC transfers probability distributions over architectures learnt on small surrogates to larger networks and datasets, enabling us to further reduce the computational cost.
- PARSEC is computationally efficient and can be run in less than a day on a single GPU on CIFAR-10.
- PARSEC outperforms other methods that consider the same architecture search space, while drastically reducing the search time.

2 Architecture Search Space

In this work, we consider the same architecture search space as in DARTS. Neural network architectures are obtained by stacking a recurrent convolutional unit, denoted as cell. A cell is defined as a directed acyclic graph of N ordered nodes, $\{\mathbf{z}_1, ..., \mathbf{z}_N\}$. An intermediate node, $\mathbf{z}_k$, is defined as the sum of two of the previous nodes, $\mathbf{z}_i$ and $\mathbf{z}_j$ with $i, j < k$, after applying primitive operations $o_{i,k}$ and $o_{j,k}$, respectively:

$$\mathbf{z}_k = o_{i,k}(\mathbf{z}_i) + o_{j,k}(\mathbf{z}_j) \tag{1}$$

3 Deterministic Approach to Architecture Search

It consider an over-parametrized parent network containing all possible paths between nodes:

$$\mathbf{z}_k = \sum_{i,p} \alpha_{i,k}^p \cdot o_{i,k}^p(\mathbf{z}_i) \quad \text{where} \quad \sum_p \alpha_{i,k}^p = 1 \tag{2}$$

where indices i and p run over all possible inputs and operations for node k respectively. $o_{i,k}^p$ denotes primitive operation p on input node i to contribute to output node k, and $\alpha_{i,k}^p$ is the weight of the path.

4 Probabilistic Neural Architecture Search

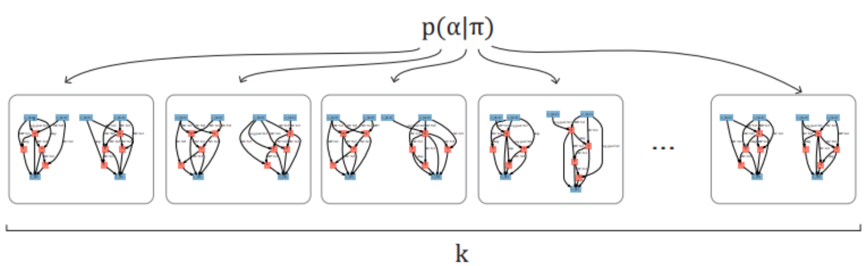

Figure 1: Pictorial representation of architecture sampling. Each sample consists of a normal and a reduction cell, which are shown side-to-side.

In PARSEC, child networks from the search space are directly obtained as samples from the specified architecture distribution, and thus neural architecture is reduced to inferring the probability distribution $p(\alpha|\pi)$ of high-performing architectures on the task at hand. Given a supervised task and denoting with $\mathbf{y}$ the targets and with $\mathbf{X}$ the input features, this is achieved by optimizing the continuous prior hyper-parameters π through an empirical Bayes Monte Carlo procedure. We optimize

$$p(\mathbf{y}|\mathbf{X}, \mathbf{v}, \pi) = \int p(\mathbf{y}|\mathbf{X}, \mathbf{v}, \alpha)p(\alpha|\pi)d\alpha \tag{3}$$

with respect to the network weights $\mathbf{v}$, which are shared across all child architectures (i.e., samples), and the prior hyper-parameters π, i.e., the architecture parameters.

We assume independent categorical distributions over each input/operation pair. Specifically, for normal and reduction cells we set

$$\begin{aligned} p(\boldsymbol{\alpha}^{(\mathrm{n})}|\boldsymbol{\pi}^{(\mathrm{n})}) &= \prod_{n=1}^{N}\prod_{i=1}^{2}\mathrm{Cat}(\boldsymbol{\alpha}_{n,i}^{(\mathrm{n})}|\boldsymbol{\pi}_{n,i}^{(\mathrm{n})}), \\ p(\boldsymbol{\alpha}^{(\mathrm{r})}|\boldsymbol{\pi}^{(\mathrm{r})}) &= \prod_{n=1}^{N}\prod_{i=1}^{2}\mathrm{Cat}(\boldsymbol{\alpha}_{n,i}^{(\mathrm{r})}|\boldsymbol{\pi}_{n,i}^{(\mathrm{r})}), \end{aligned}$$

where n runs over the nodes, i runs over the inputs of each node (in our search space each node has two inputs), $\pi_{n,i}^{(n)}$ and $\pi_{n,i}^{(r)}$ are the vectors of probabilities for all possible incoming node/operation pairs and finally, we introduced $\alpha^{(\cdot)} = \{\alpha_{n,i}^{(\cdot)}\}$ and $\pi^{(\cdot)} = \{\pi_{n,i}^{(\cdot)}\}$.

5 Importance-Weighted Monte Carlo empirical Bayes

We develop an importance weighted EB procedure for jointly optimizing π and v. We begin by introducing the following estimator:

$$\begin{aligned} p(\boldsymbol{y}|\mathbf{X},\boldsymbol{v},\boldsymbol{\pi}) &= \int p(\boldsymbol{y}|\mathbf{X},\boldsymbol{v},\boldsymbol{\alpha})p(\boldsymbol{\alpha}|\boldsymbol{\pi})\mathrm{d}\boldsymbol{\alpha} \\ &\approx \frac{1}{K}\sum_{k} p(\boldsymbol{y}|\mathbf{X},\boldsymbol{v},\boldsymbol{\alpha}_k), \end{aligned}$$

with $\alpha_k \sim p(\alpha|\pi)$. Note that the gradients can be written as

$$\begin{aligned} \nabla_{\boldsymbol{v},\boldsymbol{\pi}} \log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{v},\boldsymbol{\pi}) &= \frac{1}{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{v},\boldsymbol{\pi})}\int \nabla_{\boldsymbol{v},\boldsymbol{\pi}} p(\boldsymbol{y},\boldsymbol{\alpha}|\boldsymbol{X},\boldsymbol{v},\boldsymbol{\pi})\mathrm{d}\alpha \\ &= \frac{1}{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{v},\boldsymbol{\pi})}\int \nabla_{\boldsymbol{v},\boldsymbol{\pi}} \log p(\boldsymbol{y},\boldsymbol{\alpha}|\boldsymbol{X},\boldsymbol{v},\boldsymbol{\pi}) p(\boldsymbol{y},\boldsymbol{\alpha}|\boldsymbol{X},\boldsymbol{v},\boldsymbol{\pi})\mathrm{d}\boldsymbol{\alpha} \end{aligned}$$

Finally, taking an importance sampling estimator to the expectation, we obtain tractable gradient estimators for θ and π:

$$\begin{aligned}
\nabla_{\boldsymbol{v}} \log p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{v}, \boldsymbol{\pi}) &\approx \sum_{k=1}^{K} \tilde{\omega}_k \nabla_{\boldsymbol{v}} \log p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{v}, \boldsymbol{\alpha}_k) \triangleq \tilde{\nabla}_{\boldsymbol{v}}, \\
\nabla_{\boldsymbol{\pi}} \log p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{v}, \boldsymbol{\pi}) &\approx \sum_{k=1}^{K} \tilde{\omega}_k \nabla_{\boldsymbol{\pi}} \log p(\boldsymbol{\alpha}_k|\boldsymbol{\pi}) \triangleq \tilde{\nabla}_{\boldsymbol{\pi}},
\end{aligned}$$

where $\alpha_k \sim p(\alpha|\pi)$ and $\tilde{\omega}_k = \frac{p(\mathbf{y}|\mathbf{X},\mathbf{v},\alpha_\mathbf{k})}{\sum_j p(\mathbf{y}|\mathbf{X},\mathbf{v},\alpha_\mathbf{j})}$.

Algorithm 1 Importance weighted Monte-Carlo EB algorithm used for joint training of the network and architecture parameters.

Define train and search splits of the data
$\boldsymbol{\theta}, \boldsymbol{\pi} \leftarrow$ Initial values
Initialize $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$
repeat
  Sample batch $(\mathbf{X}^{(\mathrm{s})}, \boldsymbol{y}^{(\mathrm{s})})$ from search set
  for $k \in \{1, \ldots, K\}$ **do**
    Sample architecture $\boldsymbol{\alpha}_k$ from $p(\boldsymbol{\alpha}|\boldsymbol{\pi})$
    $\omega_k \leftarrow p(\boldsymbol{y}^{(\mathrm{s})}|\mathbf{X}^{(\mathrm{s})}, \boldsymbol{v}, \boldsymbol{\alpha})$
  end for
  Compute normalized weights $\tilde{\omega}_k = \frac{\omega_k}{\sum_{j=1}^{K} \omega_j}$
  Sample batch $(\mathbf{X}^{(\mathrm{s})}, \boldsymbol{y}^{(\mathrm{t})})$ from train set
  for $k \in \{1, \ldots, K\}$ **do**
    $g_{\boldsymbol{v},k} \leftarrow \nabla_{\boldsymbol{v}} \log p(\boldsymbol{y}^{(\mathrm{t})}|\mathbf{X}^{(\mathrm{t})}, \boldsymbol{v}, \boldsymbol{\alpha}_k)$
    $g_{\boldsymbol{\pi},k} \leftarrow \nabla_{\boldsymbol{\pi}} \log p(\boldsymbol{\alpha}_k|\boldsymbol{\pi})$
  end for
  Update $\boldsymbol{v}$ by $\tilde{\nabla}_{\boldsymbol{v}} = \sum_k \tilde{\omega}_k g_{\boldsymbol{w},k}$
  Update $\boldsymbol{\pi}$ by $\tilde{\nabla}_{\boldsymbol{\pi}} = \sum_k \tilde{\omega}_k g_{\boldsymbol{\pi},k}$
until Number of epochs is reached or convergence is achieved
Return: $(\boldsymbol{\theta}, \pi)$