Analysis of Heuristics for Neural Architecture Search
MAT496H1S: Mathematics of Deep Learning (Reading)

Robert Wu RUPERT@CS.TORONTO.EDU
Department of Computer Science
University of Toronto

Abstract

Neural architecture search (NAS) algorithms are a handy tool in the automated construction of effective deep learning models in information processing tasks such as classification, clustering, and generation. This report, in part, summarizes some conventional NAS algorithms, including those based on reinforcement learning and evolutionary algorithms. It also reviews kernel methods based on Gaussian Processes (GP) and the neural tangent kernel (NTK) as a mathematical heuristic for the model performance estimation that is usually derived from an expensive training process. And finally, this report surveys the recent works on proxies or surrogates with mathematical correspondence in model ranking and pruning. Comparisons are made for image classification tasks such as CIFAR-10, with newly constructed networks or those sampled from an architectural dataset like the NAS-Bench variants. As heuristics are generally associated with trading some accuracy/quality for substantial cuts in cost, it remains to be seen if kernel and proxy heuristics can soon approach state-of-the-art performance in practical applications.

1. Introduction

Artificial neural networks (ANNs) are the modern standard in information processing. They have been used in several useful applications, such as classification, segmentation, semantic analysis, and generative methods. *Neural architecture search (NAS)*, a subset of automated machine learning, has been a very useful tool in aiding in the design and optimization of ANNs. Even early reinforcement learning-based algorithms (Zoph and Le, 2016) (Pham et al., 2018) demonstrated respectable performance that had taken thousands of researchers several years to achieve. However, the computational resources required to reach practical benchmarks has held NAS back and left much potential on the table.

While the emergence of ever-improving models demonstrates optimistic advancement in deep learning. In general, these methods are far too slow and do not surpass the human-engineered states of the art (SOTA) when applied to real-world tasks. The challenges with NAS algorithms lie in the complex nature of performant models: better models for real-world problems typically have many more parameters arranged in complex structures. ANNs can be described as computational graphs of vertices (known as cells or layers) and edges (connections). Barring recurrent architectures such as long short-term memory (LSTM) units (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs) (Cho et al., 2014), these graphs are usually acyclic. It is intuitive that the sheer universe of different architectures and associated hyperparameters possible for any ANN depth L is intractibly large. The computational burden of evaluating even a fraction of these models is usually greater than manually designing a suitable model.

In the past few years, there have been many efforts to make NAS more efficient. Several contributions have involved augmenting existing algorithms with concepts such as *stacking cells* (Zoph et al., 2017) and *weight sharing* (Pham et al., 2018) (Liu et al., 2018b) (Su et al., 2021). The re-examination of *kernel methods* – a rather old concept – such as the neural tangent kernel (NTK) (Jacot et al., 2018) in evaluating ANNs has also attracted much attention and hinted at possible applications to NAS. And recently, *proxy methods* focusing on efficient *model ranking* have emerged as cheap heuristics for model pruning and the classical exploration vs. exploitation trade-off. This report paper surveys several of these innovations and speculates on further applications.

2. Neural Architecture Search (NAS)

The following description of NAS is intentionally high-level and generic to reflect the diversity of recent algorithms. When constructing an ANN, the cells/layers and connections in between are selected to dictate the flow of signals and gradients in training. NAS algorithms automated this sampling process to produce *child networks* that are evaluated, the performance of which informs how the next generation of networks should be constructed.

Neural Operation Search Spaces Let neural operation cells ω (vertices in a graph) be drawn from an arbitrary universe (or set) $\mathcal{S}$. In the popular problem of image classification, $\mathcal{S}$ might contain cells such as convolutional, normalization, and pooling layers. The operations sampled by an algorithm might be repeating with independent parameters and training gradients, so they would conveniently be expressed as a bag rather than a subset. The number of possible bags of size L is equal to $L|\mathcal{S}|$.

Making Connections In addition to sampling cells for a child network, a NAS algorithm is also responsible for making connections (edges in a graph) between said cells. These can be represented simply using conventional directed binary edge maps of size $\mathcal{O}(L^2)$ where the child network has L cells. Since each cell can have a (skip) connection to any later cell, there are $\mathcal{O}(2^{L^2})$ possible edge maps.

Hyperparameter Optimization (HPO) When designing a child network and running experiments, the hyperparameter optimization (HPO) process typically applies to a conventional set of values, which might include learning rate, batch size, and number of iterations/epochs. HPO can be regarded as orthogonal to the NAS process (as siblings under the umbrella of automated learning), but some neural cells will have associated hyperparameters; convolutions, for example, have properties such as filter size, dilation, padding, and stride. Either way, HPO will need to be considered when performing NAS.

Network Evaluation In order for NAS to be useful, there must be some measure of (predicted) quality, also known as *performance estimation*. Traditionally, child networks would be trained directly on the training set, and the validation error would serve as the performance estimation; this was intuitive since the training/validation sets would contain the same kind of data as the test set. However, this is a very naïve approach to the problem. Kernel methods are old but have, as an alternative, shown impressive results given the computational resources used. Several proxies have also been proposed, not to directly estimate performance, but to efficiently rank child networks with a fraction of the resources; methods discussed in this survey include training regime reduction (Zhou et al., 2020) or analysis of non-linear activation patterns.

3. Datasets

In modern NAS literature, there exist datasets for tasks (historical data, language corpora, image databases, etc.) as well as datasets of model architectures for those task datasets (which will be discussed later).

3.1 Task Datasets

Unsurprisingly, most NAS methods are designed for and implemented on conventional datasets. Image classification remains one of the most popular tasks; commonly featured datasets include CIFAR-10/100 (Krizhevsky, 2009), MNIST (LeCun et al., 2010), and ImageNet (Deng et al., 2009) (and its variants such as ImageNet-2012 (Russakovsky et al., 2015) and ImageNet-16-120 (Chrabaszcz et al., 2017)). But NAS can also be applied to recurrent architectures for sequential data tasks such as language modelling; Liu et al. (2018b) demonstrate NAS on Penn TreeBank (PTB) (Taylor et al., 2003) and WikiText-2 (Merity et al., 2016).

3.1.1 Synthetic Datasets

In cases where common tasks would be too complex for a proof-of-concept, a synthetic dataset may be used as a toy problem. In addition to CIFAR-10, Simon et al. (2021) also performed experiments using simple geometric structures such as unit circles, (hyper)cubes and (hyper)spheres as target functions. These examples can be briefly and respectively defined (Simon et al., 2021):

- Let $\mathcal{X} := \left\{\cos\left(\frac{2\pi j}{M}\right), \sin\left(\frac{2\pi j}{M}\right)\right\}_{j=1}^{M}$ be discrete points on a unit circle where $M = |\mathcal{X}|$.
- Let $\mathcal{X} := \{-1, 1\}^d$ be M discrete points of a d-dimensional hypercube where $M = 2^d$.
- Let $\mathbb{S}^d := \{x \in \mathbb{R}^{d+1} | x^2 = 1\}$ be the domain of a d-dimensional hypersphere.

While these synthetic datasets are seldom used and usually uninteresting (due to their unrealistic and theoretical character), they might serve as a better standard baseline or toy problem for future NAS research. Perhaps specific datasets like CIFAR-10 or PTB mask the lack of generalization of some NAS algorithms.

3.2 Architectural Datasets

Because NAS involves sampling many child network architectures, it can be useful to keep reference datasets of models and their performance estimations/results. In fact, some recent works such as Mellor et al. (2021) rely heavily on such datasets in scoring and evaluating their heuristic methods. To paraphrase Mellor et al. (2021),

- NAS-Bench 101: 423,624 child networks trained on CIFAR-10 (Ying et al., 2019).
- NAS-Bench 201: 15,625 child networks trained on CIFAR-10/100 and ImageNet-16-120 (Dong and Yang, 2020).
- NATS-Bench: An extension of NAS-Bench 201 where the original 15,625 child networks are in a topology search space, and which includes 32,768 other child networks whose cells vary in channels (width).
- NDS: Instead of comparing search *algorithms*, NDS compares networks trained on search *spaces* from AmoebaNet (Shah et al., 2018), DARTS (Liu et al., 2018b), ENAS (Pham et al., 2018), NASNet (Zoph and Le, 2016), and PNAS (Liu et al., 2018a).
- NAS-Bench-301: A surrogate on NAS-Bench 101 and DARTS search spaces (Siems et al., 2020).

4. Conventional NAS Algorithms

NAS algorithms are diverse, but a common trait of early algorithms is their performance estimation: most methods relied on explicit training on a similar task to estimate performance on the evalution task.

Evolutionary Algorithms (EA) These algorithms are meant to emulate biological evolution. Given any arbitrary set of child networks at time step t (a generation), evolutionary algorithms (EA) such as Real et al. (2019) perform mutations of some degree on each child network (for example, change a single cell or connection) to create multiple mutants. After evaluating the mutants against the originals, the variants with the best performance are (deterministically or stochastically) retained for the next generation at time $t+1$. Interestingly, Grey (2017) follows an EA approach in their high-level YouTube explainer.

Reinforcement Learning (RL) If sampling neural cells and making connections are regarded as actions, reinforcement learning (RL) becomes a very useful paradigm. Zoph and Le (2016) first introduced NAS using reinforcement learning: an LSTM-based controller performs sampling and is trained using the classical REINFORCE algorithm (Williams, 1992) with a accuracy-based (performance estimation) reward (Figure 1). Weight sharing was introduced in Pham et al. (2018) wherein instances of the same neural cell would share the same preserved parameters. Wu and Jain (2021) extended this further by replacing the LSTM-based controller with a Q-Learning or multi-armed bandit controller.

Other Methods There exist many other algorithms such as progressive (Liu et al., 2018a) and differentiable (Liu et al., 2018b) NAS, but this survey will not discuss them in detail.

4.1 Weight Sharing

Retaining common parameters has proven to be an effective way reduce training costs. However, Saxena et al. (2021) has shown that weight sharing has shown to have severe vulnerabilities related to search spaces and gradients.

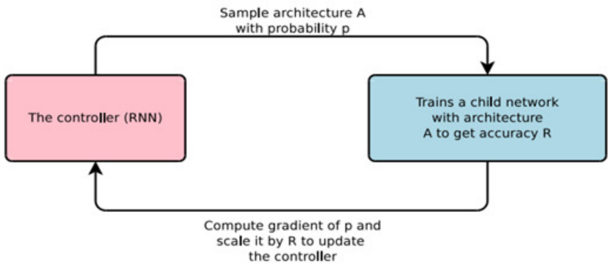
Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller

Figure 1: Controller training process in RL-based algorithms.

4.2 Computational Burden

It should be noted that most conventional algorithms are still very slow. This is not necessarily a criticism of entire existing algorithms, but a highlight of the common performance estimation bottleneck. They all rely on some form of expensive training process to estimate practical or test model performance. Search times can take as long as 3150 GPU *days* (Real et al., 2019); even an algorithm like ENAS labeled "efficient" might still take 16 hours (Pham et al., 2018). Comprehensive comparisons of search cost can be found in Zhou et al. (2020) and Mellor et al. (2021). Instead of or in addition to weight sharing, other methods such as kernel methods and proxies have been examined to further speed up NAS algorithms.

5. Kernel Methods

To avoid the explicit training that conventional NAS methods require, kernel methods can be used to analyze the theoretical behaviour. Given a training dataset $\mathcal{X}$ with pairs $(x_i, y_i) \in \mathcal{X}, i \in [1, n]$, and a query point x' (perhaps in the test dataset), a kernel machine that predicts $\hat{y}$ in binary classifcation might look like the following.

$$\hat{y} := \mathbb{I}\left(\sum_{i=1}^{n} a_i y_i \mathbf{K}(x_i, x') + b_i\right) \tag{1}$$

Where a_i, b_i are trainable parameters for data point x_i and $\mathbf{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a similarity function between data points. Note that this solution can be computed analytically without depth or time. Kernel methods are well-founded mathematical tools, but have been largely overlooked in machine learning in favour of deep neural architectures (Schölkopf and Smola, 2001). Thanks to Jacot et al. (2018) and many works spawned therefrom, kernel methods have resurfaced as impressive heuristic alternatives to deep learning training techniques.

5.1 Gaussian Processes (GP)

The correspondence between Gaussian Processes (GP) and a single-layer neural network is well-known (Williams, 1996). Given the data $x \in \mathbb{R}^{d_0}$, let the activations $z^1 := [z_i^1]_{i\in[1,N]}$ in a hidden layer of width N be defined as,

$$z_i^1(x) := \sum_{j=1}^{N} W_{i,j}^1 x_j^1(x) + b_i^1, \quad x_j^1(x) := \phi\left(\sum_{k=1}^{d_0} W_{j,k}^0 x_k + b_j^0\right) \tag{2}$$

Where x_i^l and z_i^l, respectively, are the post-nonlinearity and post-affine transformation for the i'th component of activations in layer l (Lee et al., 2017). And the kernel function $\mathbf{K}$ was then derived as,

$$\begin{aligned}\mathbf{K}^1(x,x') &\equiv \mathbb{E}[z_i^1(x)z_i^1(x')] \\ &= \sigma_w^2\mathbb{E}[x_i^1(x)x_i^1(x)] + \sigma_b^2 \\ &\equiv \sigma_w^2 C(x,x') + \sigma_b^2\end{aligned}$$

where $b_i \sim \mathcal{N}(0,\sigma_b^2), w_i \sim \mathcal{N}(0,\sigma_w^2)$. Lee et al. (2017) extended these definitions to deeper networks using induction. The forward-pass recurrence relation for layer l can be defined,

$$z_i^l(x) := \sum_{j=1}^{N} W_{i,j}^l x_j^l(x) + b_i^l, \quad x_j^l(x) := \phi(z_j^{l-1}(x')) \tag{3}$$

Given the GP $\mathcal{GP}$, the kernel function was generalized and compacted with a deterministic function F_ϕ parameterized by the nonlinearity ϕ (Lee et al., 2017).

$$\begin{aligned}\mathbf{K}^l(x,x') &\equiv \mathbb{E}[z_i^l(x)z_i^l(x')] \\ &= \sigma_w^2\mathbb{E}_{z_i^{l-1}\sim\mathcal{GP}(o,K^{l-1})}[z_i^{l-1}(x)z_i^{l-1}(x')] + \sigma_b^2 \\ &= \sigma_w^2 F_\phi(K^{l-1}(x,x'), K^{l-1}(x,x), K^{l-1}(x',x')) + \sigma_b^2\end{aligned}$$

5.2 Neural Tangent Kernel (NTK)

For a network of depth L, let $F^{(L)} : \mathbb{R}^P \to \mathcal{F}$ be a realization function that maps parameters $\theta \in \mathbb{R}^P$ to functions $f_\theta \in \mathcal{F}$. Given a cost function $C : \mathcal{F} \to \mathbb{R}$, let the composite function $C \circ F^{(L)} : \mathbb{R}^P \to \mathbb{R}$ be represented as $C|_{f_\theta}$ for a function f_θ. With respect to the NTK $\Theta^{(L)}(\theta)$ (instead of $\mathbf{K}$), the network function f_θ will evolve along the (negative) kernel gradient $\partial_t f_{\theta(t)}$ with the definition,

$$\begin{aligned}\Theta^{(L)}(\theta) &= \sum_{p=1}^{P} \partial_{\theta_p} F^{(L)} \otimes \partial_{\theta_p} F^{(L)} \\ \partial_t f_{\theta(t)} &= -\nabla_{\Theta^{(L)}} C|_{f_\theta(t)}\end{aligned}$$

The initialization scheme and training regime are further detailed in Jacot et al. (2018). The theoretical result is that in the infinite-width limit, an ANN can be described as the constant positive-definite NTK $\Theta_\infty^{(L)}$ and that traditional ANN gradient descent corresponds to kernel gradient descent (Jacot et al., 2018).

5.3 Convolutional Neural Tangent Kernel (CNTK)

Jacot et al. (2018) presented interesting theoretical results, but the NTK will need to be applied to real datasets to be practical. Arora et al. (2019) define a variant of the NTK to correspond to convolutional neural networks (CNNs) in classification on CIFAR-10. Let P, Q be the length and width of each image x from CIFAR-10 $\mathcal{X}$, and $C^{(l)}$ be the number of channels at layer l (so that $C^{(0)}$ is the number of channels in the image data). Then the images fall in the domain $x \in \mathcal{X} \subset \mathbb{R}^{P \times Q \times C^{(0)}}$. Let q be the convolution filter size. For two images x, x' and channels $\alpha = 1, \ldots, C^{(0)}$, let $\mathbf{K}^{(0)}_{(\alpha)}(x, x') := x_{(\alpha)} \otimes x'_{(\alpha)}$. Then for pixel indices $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, define the following covariance,

$$\left[\mathbf{\Sigma}^{(0)}(x, x')\right]_{i,j,i',j'} := \sum_{\alpha=1}^{C^{(0)}} \operatorname{tr}\left(\left[\mathbf{K}^{(0)}_{(\alpha)}(x, x')\right]\right) \tag{4}$$

In subsequent layers $l \in [L]$, for indices $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, define the following covariance matrix $\mathbf{\Lambda}^{(l)}_{(i,j,i',j')}$, kernel matrix $\mathbf{K}^{(l)}(x, x') \in \mathbb{R}^{P \times Q \times P \times Q}$ and its derivative $\dot{\mathbf{K}}^{(l)}(x, x') \in \mathbb{R}^{P \times Q \times P \times Q}$, and covariance entry $\left[\mathbf{\Sigma}^{(l)}(x, x')\right]_{i,j,i',j'} \in \mathbb{R}^{P \times Q \times P \times Q}$.

$$\begin{aligned}
\mathbf{\Lambda}^{(l)}_{(i,j,i',j')} &:= \begin{pmatrix} \left[\mathbf{\Sigma}^{(l-1)}(x, x')\right]_{i,j,i,j} & \left[\mathbf{\Sigma}^{(l-1)}(x, x')\right]_{i,j,i',j'} \\ \left[\mathbf{\Sigma}^{(l-1)}(x, x')\right]_{i',j',i,j} & \left[\mathbf{\Sigma}^{(l-1)}(x, x')\right]_{i',j',i',j'} \end{pmatrix} \\
\mathbf{K}^{(l)}(x, x') &:= \frac{c_\sigma}{q^2} \cdot \mathbb{E}_{(u,v) \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Lambda}^{(l)}_{(i,j,i',j')}\right)}[\sigma(u)\sigma(v)] \\
\dot{\mathbf{K}}^{(l)}(x, x') &:= \frac{c_\sigma}{q^2} \cdot \mathbb{E}_{(u,v) \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Lambda}^{(l)}_{(i,j,i',j')}\right)}[\dot{\sigma}(u)\dot{\sigma}(v)] \\
\left[\mathbf{\Sigma}^{(l)}(x, x')\right]_{i,j,i',j'} &:= \operatorname{tr}\left(\left[\mathbf{K}^{(l)}(x, x')\right]_{D_{i,j,i',j'}}\right)
\end{aligned}$$

where $D_{i,j,i',j'}$ is defined as the pixels surrounding (i, j) and (i', j') as per the convolutional filter, and c_σ is defined in Arora et al. (2019). Then the following recursive definition computes the CNTK where $l = 1, \ldots, L-1$ and $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$.

$$\begin{aligned}
\mathbf{\Theta}^{(0)}(x, x') &:= \mathbf{\Sigma}^{(0)}(x, x') \\
\left[\mathbf{\Theta}^{(l)}(x, x')\right]_{i,j,i',j'} &:= \operatorname{tr}\left(\left[\dot{\mathbf{K}}^{(l)}(x, x') \odot \mathbf{\Theta}^{(l-1)}(x, x') + \mathbf{K}^{(l)}(x, x')\right]_{D_{i,j,i',j'}}\right) \\
\mathbf{\Theta}^{(L)}(x, x') &:= \dot{\mathbf{K}}^{(L)}(x, x') \odot \mathbf{\Theta}^{(L-1)}(x, x') + \mathbf{K}^{(L)}(x, x')
\end{aligned}$$

And finally the CNTK value is $\operatorname{tr}(\mathbf{\Theta}((L)))(x, x')$. Importantly, the preliminary results show that each CNTK usually achieves lower performance than its conventional CNN counterpart. The best result (77.43% accuracy) is found in the *global average pooling (GAP)* variant of the CNTK, and is below a CNN with GAP, which achieved an accuracy of 83.30% under their training regime Arora et al. (2019). However, these results are still impressive considering that no training actually takes place.

5.3.1 Potential Application to NAS

Arora et al. (2019) make an explicit suggestion of applying the CNTK to NAS algorithms. In summary, one can derive performance estimation of the CNTK on a validation (sub)set after computing it on a small training subset. This could be used to directly choose the final child networks in a new cheap NAS algorithm (Arora et al., 2019). Alternatively, this estimation can be integrated into existing algorithms, either as a reward surrogate for RL-based algorithms, or as the mutation selection process in EA.

5.4 Enhancing Convolutional Neural Tangent Kernels (CNTK II)

To improve upon Arora et al. (2019), the same authors went on to propose data and kernel augmentations in Li et al. (2019); they also distinguish regression with respect to the CNN Gaussian Process kernel (CNN-GP) $\mathbf{\Sigma}$ from that with respect to the CNTK $\mathbf{\Theta}$ under their different training conditions (Li et al., 2019). In particular, they define regression of the two kernels without GAP, designated with "FC" (for fully-connected layers),

$$\mathbf{\Sigma}_{\text{FC}}(x, x') := \operatorname{tr}\left(\mathbf{\Sigma}^{(L)}(x, x')\right)$$
$$\mathbf{\Theta}_{\text{FC}}(x, x') := \operatorname{tr}\left(\mathbf{\Theta}^{(L)}(x, x')\right)$$

As well as with GAP,

$$\mathbf{\Sigma}_{\text{GAP}}(x, x') := \frac{1}{P^2Q^2} \sum_{i,j,i',j' \in [P]\times[Q]\times[P]\times Q} \left[\mathbf{\Sigma}^{(L)}(x, x')\right]_{i,j,i',j'}$$
$$\mathbf{\Theta}_{\text{GAP}}(x, x') := \frac{1}{P^2Q^2} \sum_{i,j,i',j' \in [P]\times[Q]\times[P]\times Q} \left[\mathbf{\Theta}^{(L)}(x, x')\right]_{i,j,i',j'}$$

5.4.1 Data Augmentation

Li et al. (2019) propose two data augmentation schemes on dataset $D := \{(x_i, y_i)\}_{h=1}^N$:

Translation For $(i, j) \in [P] \times [Q]$, the translation operator $\mathcal{T}_{i,j} : \mathbb{R}^{P\times Q\times C} \to \mathbb{R}^{P\times Q\times C}$ transforms each image x and the full dataset $D \to D^{\mathcal{T}}$, respectively.

$$[\mathcal{T}_{i,j}(x)]_{i',j',c} = [x]_{i'+i,j'+j,c}, \quad D_{\mathcal{T}} := \{(T_{i,j}(x_i), y_i)\}_{(i,j,c)\in[P]\times[Q]\times[N]} \tag{5}$$

Horizontal Flip For $(i, j) \in [P] \times [Q]$, the flip operator $\mathcal{F} : \mathbb{R}^{P\times Q\times C} \to \mathbb{R}^{P\times Q\times C}$ transforms each image x and the full dataset $D \to D^{\mathcal{F}}$, respectively.

$$[\mathcal{F}(x)]_{i,j,c} = [x]_{P+1-i,j,c}, \quad D_{\mathcal{F}} := \{(F(x_i), y_i)\} \tag{6}$$

It was shown that computing the original kernels on the augmented datasets $D_{\mathcal{T}}$, $D_{\mathcal{F}}$ is equivalent to computing the augmented kernels on the original dataset D (Li et al., 2019).

5.4.2 Local Average Pooling (LAP)

Full translation of the dataset images results in unrealistic images. Moderating this global effect is termed *local average pooling (LAP)*. Importantly, hyperparameter c is introduced to control the size of translation windows; a smaller c value results in more local translations, whereas maximizing c is equivalent to GAP. Details are relegated to Li et al. (2019).

5.4.3 Ablations & Results

Experiments were carried out on CIFAR-10 and Fashion-MNIST (Xiao et al., 2017). Ablations were performed with $L = 5, 8, 11, 14$, $c = 4\gamma$ where $\gamma \in [0, 8]$ and with/out horizontal flipping. The authors additionally performed preprocessing involving normalization and transformation of image patches. The best results are summarized in Table 1. (Li et al., 2019)

Dataset	**Kernel**	**PreProc**	**Flip**	$(\mathbf{L}, \mathbf{c})$	**TestAcc**
CIFAR-10	CNTK	No	No	$(5, 12)$	80.11%
CIFAR-10	CNTK	No	Yes	$(8, 16)$	81.40%
CIFAR-10	CNN-GP	No	No	$(8, 16)$	80.78%
CIFAR-10	CNN-GP	No	Yes	$(8, 16)$	82.20%
CIFAR-10	CNTK	Yes	No	$(5, 16)$	86.77%
CIFAR-10	CNTK	Yes	Yes	$(5, 16)$	88.36%
CIFAR-10	CNN-GP	Yes	No	$(5, 16)$	87.28%
CIFAR-10	CNN-GP	Yes	Yes	$(8, 12)$	88.92%
Fashion-MNIST	CNTK	No	No	$(5, 4)$	93.76%
Fashion-MNIST	CNTK	No	Yes	$(5, 4)$	94.07%
Fashion-MNIST	CNN-GP	No	No	$(11, 4)$	93.63%
Fashion-MNIST	CNN-GP	No	Yes	$(8, 4)$	93.79%

Table 1: Results consolidated across ablations of datasets, kernel, preprocessing, flipping, and the best test accuracy and associated hyperparameters (L, c) (Li et al., 2019).

5.5 Approximation to Kernel Machines in the Wild

There have also been attempts to approximate general deep learning models with kernel machines. Domingos (2020) proposes what they call the *path kernel.* Given data points x, x', a prediction function y and parameters w, a path kernel can be expressed as an integral of the dot product of the model's gradients over path $c(t)$ at times t,

$$\mathbf{K}(x, x') := \int_{c(t)} \nabla_w y(x) \cdot \nabla_w y(x') dt \tag{7}$$

Thereafter, the tangent kernel and path kernel associated with function $f_w(x)$ can be defined,

$$\begin{aligned} \mathbf{K}^g_{f,v}(x, x') &:= \nabla_w f_w(x) \cdot \nabla_w f_w(x') \\ \mathbf{K}^p_{f,c}(x, x') &:= \int_{c(t)} \mathbf{K}^g_{f,w(t)}(x, x') dt \end{aligned}$$

where $c(t)$ is the curve in parameter space $v = w(t)$ is the vector of parameters at time t. The resulting limit is shown in Domingos (2020).

$$\begin{aligned} \lim_{\epsilon \to 0} y &= y_0 - \sum_{i=1}^{m} \bar{L}'(y_i^*, y_i) \mathbf{K}^p_{f,c}(x, x_i) \\ &= \sum_{i=1}^{m} a_i \mathbf{K}^p_{f,c}(x, x_i) + b \end{aligned}$$

where $\mathbf{K}(x, x_i) = \mathbf{K}^p_{f,c}(x, x_i)$, $a_i = -\bar{L}'(y_i^*, y_i)$, and $b = y_0$. This was highly controversial because it uses the query point x in the determination of the weights themselves. Consequently, the results in Domingos (2020) do not appear to include true kernel machines. In any event, this work only claimed to approximate the general deep ANN, and did not provide experimental results.

5.6 Generalization: A Theory About Kernels

Simon et al. (2021) sought to provide a mathematical framework for the concept of learnability: how well can a kernel like the NTK or wide ANN learn tasks and adapt to new ones? To paraphrase, they showed that they could describe the inductive bias of a kernel as a fixed budget of learnability; approximate the mean/covariance of the predicted function; obtain a new result regarding the hardness of the parity problem; and study network overfitting and robustness. Given the known "zero-sum game" nature of all target functions, the resulting implication is that there is a well-defined limit to learnability for any network, which can only be increased by introducing more data. The mathematical reasoning behind learnability is beyond the scope of the survey, but is detailed in Simon et al. (2021). This would appear to hamper kernel methods in general.

6. Proxy & Pruning

But perhaps quickly estimating the true performance is not the only way to substantially speed up NAS. Indeed, most conventional algorithms were selecting from a set of child networks to move forward; child networks were selected not for their *relative* performance, not necessarily their raw performance. More loosely, ranking the child networks is the critical process for which performance estimation was conducted.

6.1 Reducing the Training Regime

Hyperparameters such as width (number of channels), data resolution, training epochs, and sample ratio (from the dataset) are generally scaled to improve the training regime. However, one might foresee that scaling any of these values too high may result in overfitting and/or may not be necessary to achieve the same performance. A more likely scenario is that reduced training regimes might suffer in raw performance but maintain rank consistency.

Zhou et al. (2020) explores this idea for CNNs in their reductions experiments and subsequent evaluation of rank consistency. Let c, r, e, s represent the aforementioned number of channels, resolution of input images, training epochs, and dataset sample ratio, respectively; and let (c_a, r_b, e_d, s_g) define a proxy training regime with reduction factors a, b, d, g. Note that the reduced setting requires $\frac{1}{2^{a+b+d}}$ as many FLOPs.

A commonly used metric for rank consistency is the Spearman Coefficient ρ_{sp}. For a collection of K entries where d_i is the absolute difference between the original and new ranks of entry i, ρ_{sp} can be expressed as,

$$\rho_{\text{sp}} := 1 - \frac{6\sum_{i=1}^{K} d_i^2}{K(K^2-1)} \tag{8}$$

where higher values are better. This metric is directly applied to ANNs in Zhou et al. (2020) when comparing child network rankings between the baseline and proxy training regimes. Results are promising: the best experiments on SOTA models achieved impressive 2.62% and 25.2% test error rates on CIFAR-10 and ImageNet with the cost of 8 GPU days. Integrating training reduction into existing NAS algorithms also displayed competitive results (Zhou et al., 2020).

6.2 Activation Map Kernel

An innovative idea is to revisit the concept of activations. Nonlinearities like ReLU, tanh, and sigmoid exist primarily to create binary signals in the hidden layers of ANNs, and secondarily to scale the information thereafter. Mellor et al. (2021) investigates this line of reasoning using a scoring mechanism to describe how distinctively **untrained** child networks (they provide a baseline performance that trained networks typically only improve on) would behave in the forward-passes of different data points. This begins with constructing binary codes $\mathbf{c}_i$ of the activation maps of a child network for a training example $\mathbf{x}_i$. Hamming distance $d_H(\mathbf{c}_i, \mathbf{c}_j)$ is used to describe how dissimilar the codes are for data points $\mathbf{x}_i$, $\mathbf{x}_j$. Then these distance metrics are arranged in a kernel matrix $\mathbf{K}_H$.

$$\begin{aligned}
d_H(\mathbf{c}_i, \mathbf{c}_j) &:= \langle \mathbf{c}_i, \mathbf{c}_i \rangle \\
\mathbf{K}_H &:= \begin{bmatrix} N - d_H(\mathbf{c}_1, \mathbf{c}_1) & \dots & N - d_H(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ N - d_H(\mathbf{c}_N, \mathbf{c}_1) & \dots & N - d_H(\mathbf{c}_N, \mathbf{c}_N) \end{bmatrix} \\
&= N(I_N) - \begin{bmatrix} d_H(\mathbf{c}_1, \mathbf{c}_1) & \dots & d_H(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ d_H(\mathbf{c}_N, \mathbf{c}_1) & \dots & d_H(\mathbf{c}_N, \mathbf{c}_N) \end{bmatrix}
\end{aligned}$$

where N is the number of activations and I_N is the identity matrix of size N. The score $s = \log |\mathbf{K}_H|$ is then computed as the logarithm of the determinant of this kernel. Out of curiosity, I propose swapping Hamming distance d_H for Euclidean distance d_E, which in the binary case, is equivalent to the square-root of Hamming distance. Without normalization, this would more variable scores and might lead to better differentiability.

$$\begin{aligned}
d_E(\mathbf{c}_i, \mathbf{c}_j) &:= \sqrt{\langle \mathbf{c}_i, \mathbf{c}_i \rangle} \\
\mathbf{K}_E &= N(I_N) - \begin{bmatrix} d_E(\mathbf{c}_1, \mathbf{c}_1) & \dots & d_E(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ d_E(\mathbf{c}_N, \mathbf{c}_1) & \dots & d_E(\mathbf{c}_N, \mathbf{c}_N) \end{bmatrix}
\end{aligned}$$

Mellor et al. (2021) scored untrained child networks from the NAS-Bench variants (mostly NAS-Bench 201), as well as NDS and NATS-Bench. Under the training regime of 500 runs, search time was a mere 3 seconds per ten samples, making this a very scalable solution. The largest experiment presented in Mellor et al. (2021) shows a 92.96% test accuracy on CIFAR-10 in just 306.19 seconds, which magnitudes faster than past methods.

6.3 Zero-Cost Proxies

Another approach is to score based on saliency – in other words, importance or notability. Abdelfattah et al. (2021) discusses both per-parameter and network-level saliency. Three metrics proposed are based on per-parameter saliency $\mathcal{S}_p(\theta)$.

$$\texttt{snip} : \mathcal{S}_p(\theta) = \left| \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta \right|, \quad \texttt{grasp} : \mathcal{S}_p(\theta) = -\left(H \frac{\partial \mathcal{L}}{\partial \theta} \right) \odot \theta, \quad \texttt{synflow} : \mathcal{S}_p(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta \tag{9}$$

where $\mathcal{L}$ is the loss function of the ANN with parameters θ, H is the Hessian, $\mathcal{S}_p$ is the per-parameter saliency and $\odot$ is the Hadamard product (Abdelfattah et al., 2021). The authors also included the `fisher` saliency metric which is based on per-activation saliency.

$$\texttt{fisher} : \mathcal{S}_z(z) = \left(\frac{\partial \mathcal{L}}{\partial z} z\right)^2, \quad \mathcal{S}_n = \sum_{i=1}^{M} \mathcal{S}_z(z_i) \tag{10}$$

Additionally, Abdelfattah et al. (2021) included the Jacobian covariance derived from Mellor et al. (2021) as `jacob_cov` and a majority `vote` thereof with `snip` and `synflow`. The best results reported indicate a 94.22% test accuracy on CIFAR-10 classifcation. The authors indicate that `synflow` outperforms Zhou et al. (2020) in rank consistency and is the most robust and consistent across all datasets and measures.

6.4 Hierarchical Strategy & Explore-then-Exploit

All of the proxies discussed share a common but powerful potential application: they can be used in a hierarchical search strategy. Recall that in HPO or any search problem, an effective strategy is to explore the space widely early on, and exploit later when the optima have been localized. This is known as the classical exploration vs. exploitation trade-off, except here, we can dynamically adjust. This is especially useful in EA, as Zhou et al. (2020) demonstrates with their performance binning process and stochastic sampling and mutations. Analysing the results of our heuristics also provides guidance in how to make the trade-off, which also becomes appealing in RL-based algorithms in determining how much performance estimation should be muted or amplified in reward expressions.

7. Comparison & Conclusion: Back to Kernels?

Unfortunately, most of these algorithms were trained under vastly different conditions, and some – like Li et al. (2019) – have ambiguous runtime claims. Thus, these methods are difficult to compare. The proxy/pruning methods show a lot of promise in how they can be improved or combined with other algorithms. `synflow` in Abdelfattah et al. (2021) has eclipsed Zhou et al. (2020), while Mellor et al. (2021) has more potential.

Kernel machines have higher upfront computational cost than iterative training, and they fall short of SOTA performance. But they can still be of use if they needn't exactly model the potential performance of networks. It's likely that NTK-like methods can still model *relative performance* fairly well. If future experiments show that these kernel machines mostly preserve rank consistency, they can be used to accelerate the performance estimation stage of NAS. This survey thus motivates further study into the reliability and robustness of the NTK (and its derivatives) as a cheap model evaluator and ranker that is more faithful than proxy/pruning methods.

8. Acknowledgements

Thanks to Prof. Vardan Papyan for supervising this reading course, and Rohan Jain and Zhanghan (Neo) Yin for the guiding and stimulating discussions in our weekly reading sessions. I am also grateful to the Department of Mathematics at the University of Toronto for giving me this *exceptional* opportunity.

References

Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-cost proxies for lightweight nas. In *International Conference on Learning Representations (ICLR)*, 2021.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/dbc4d84bfcfe2284ba11beffb853a8c4-Paper.pdf`.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL `http://arxiv.org/abs/1409.1259`.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017. URL `http://arxiv.org/abs/1707.08819`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Pedro Domingos. Every model learned by gradient descent is approximately a kernel machine. *CoRR*, abs/2012.00152, 2020. URL `https://arxiv.org/abs/2012.00152`.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=HJxyZkBKDr`.

Colin Gregory Palmer Grey. How machines learn. YouTube, 2017. URL `https://www.youtube.com/watch?v=R9OHn5ZF4Uo`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes, 2017. URL `https://arxiv.org/abs/1711.00165`.

Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S. Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels, 2019. URL `https://arxiv.org/abs/1911.00809`.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 19–35, Cham, 2018a. Springer International Publishing. ISBN 978-3-030-01246-5.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, 2021.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL `http://arxiv.org/abs/1802.03268`.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789, Jul. 2019. doi: 10.1609/aaai.v33i01.33014780. URL `https://ojs.aaai.org/index.php/AAAI/article/view/4405`.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Nayan Saxena, Robert Wu, and Rohan Jain. Towards one shot search space poisoning in neural architecture search. *CoRR*, abs/2111.07138, 2021. URL `https://arxiv.org/abs/2111.07138`.

Bernhard Schölkopf and A.J. Smola. *Smola, A.: Learning with Kernels - Support Vector Machines, Regularization, Optimization and Beyond. MIT Press, Cambridge, MA*, volume 98. 01 2001.

Syed Shah, Wenji wu, Qiming Lu, Liang Zhang, Sajith Sasidharan, Phil DeMar, Chin Guok, John Macauley, Eric Pouyoul, Jin Kim, and Seo-Young Noh. Amoebanet: An sdn-enabled network service for big data science. *Journal of Network and Computer Applications*, 119, 06 2018. doi: 10.1016/j.jnca.2018.06.015.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *CoRR*, abs/2008.09777, 2020. URL `https://arxiv.org/abs/2008.09777`.

James B. Simon, Madeline Dickens, and Michael R. DeWeese. A theory of the inductive bias and generalization of kernel regression and wide neural networks, 2021. URL `https://arxiv.org/abs/2110.03922`.

Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. K-shot nas: Learnable weight-sharing for nas with k-shot supernets. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9880–9890. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/su21a.html`.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. *The Penn Treebank: An Overview*, pages 5–22. Springer Netherlands, Dordrecht, 2003. ISBN 978-94-010-0201-1. doi: 10.1007/978-94-010-0201-1_1. URL `https://doi.org/10.1007/978-94-010-0201-1_1`.

Christopher Williams. Computing with infinite networks. In M.C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996. URL `https://proceedings.neurips.cc/paper/1996/file/ae5e3ce40e0404a45ecacaaf05e5f735-Paper.pdf`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Robert Wu and Rohan Jain. Qenas: Q-learning for efficient neural architecture search. Technical report, Department of Computer Science, University of Toronto, 2021.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL `http://arxiv.org/abs/1708.07747`.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *CoRR*, abs/1902.09635, 2019. URL `http://arxiv.org/abs/1902.09635`.

Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. *CoRR*, abs/2001.01233, 2020. URL `http://arxiv.org/abs/2001.01233`.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. URL `http://arxiv.org/abs/1611.01578`.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL `http://arxiv.org/abs/1707.07012`.