XNAS: Neural Architecture Search with Expert Advice

Niv Nayman*, **Asaf Noy***, **Tal Ridnik***, **Itamar Friedman, Rong Jin, Lihi Zelnik-Manor**

Machine Intelligence Technology, Alibaba Group
{niv.nayman,asaf.noy,tal.ridnik,itamar.friedman,jinrong.jr,lihi.zelnik}
@alibaba-inc.com

Abstract

This paper introduces a novel optimization method for differential neural architecture search, based on the theory of prediction with expert advice. Its optimization criterion is well fitted for an architecture-selection, i.e., it minimizes the regret incurred by a sub-optimal selection of operations. Unlike previous search relaxations, that require hard pruning of architectures, our method is designed to dynamically wipe out inferior architectures and enhance superior ones. It achieves an optimal worst-case regret bound and suggests the use of multiple learning-rates, based on the amount of information carried by the backward gradients. Experiments show that our algorithm achieves a strong performance over several image classification datasets. Specifically, it obtains an error rate of 1.6% for CIFAR-10, 24% for ImageNet under mobile settings, and achieves state-of-the-art results on three additional datasets.

1 Introduction

In recent years tremendous efforts have been put into a manual design of high performance neural networks [22, 16, 40, 39]. An emerging alternative approach is replacing the manual design with automated Neural Architecture Search (**NAS**). NAS excels in finding architectures which yield state-of-the-art results. Earlier NAS works were based on reinforcement learning [55, 56], sequential optimization [24], and evolutionary algorithms [33], and required immense computational resources, sometimes demanding years of GPU compute time in order to output an architecture. More recent NAS methods reduce the search time significantly, e.g. via weight-sharing [30] or by a continuous relaxation of the space [25], making the search affordable and applicable to real problems.

While current NAS methods provide encouraging results, they still suffer from several shortcomings. For example, a large number of hyper-parameters that are not easy to tune, hard pruning decisions that are performed sub-optimally at once at the end of the search, and a weak theoretical understanding. This cultivates skepticism and criticism of the utility of NAS in general. Some recent works even suggest that current search methods are only slightly better than random search and further imply that some selection methods are not well principled and are basically random [23, 35].

To provide more principled methods, we view NAS as an online selection task, and rely on *Prediction with Experts Advice* (**PEA**) theory [4] for the selection. Our key contribution is the introduction of XNAS (eXperts Neural Architecture Search), an optimization method (section 2.2) that is well suited for optimizing inner architecture weights over a differentiable architecture search space (section 2.1). We propose a setup in which the experts represent inner neural operations and connections, whose dominance is specified by architecture weights.

[*] These authors contributed equally.

Preprint. Under review.

Our proposed method addresses the mentioned shortcomings of current NAS methods. For the mitigation of the hard pruning, we leverage the Exponentiated-Gradient (EG) algorithm [21], which favors sparse weight vectors to begin with, enhanced by a wipeout mechanism for dynamically pruning inferior experts during the search process. Additionally, the algorithm requires less hyper-parameters to be tuned (section 3.2.2), and the theory behind it further provides guidance for the choice of learning rates. Specifically, the algorithm avoids the decay of architecture weights [12], which is shown to promote selection of arbitrary architectures.

Additionally, XNAS features several desirable properties, such as achieving an optimal worst-case regret bound (section 3.1) and suggesting to assign different learning rates for different groups of experts. Considering an appropriate reward term, the algorithm is more robust to the initialization of the architecture weights and inherently enables the recovery of 'late bloomers', i.e., experts which may become effective only after a warm-up period (section 3.2.1). The wipeout mechanism allows the recovery of experts with a chance of being selected at the end of the process.

We compare XNAS to previous methods and demonstrate its properties and effectiveness over statistical and deterministic setups, as well as over 7 public datasets (section 4). It achieves state-of-the-art performance over 3 datasets, and top-NAS over rest, with significant improvements. For example, XNAS reaches 1.60% error over CIFAR-10, more than 20% improvement over existing NAS methods.

2 Proposed Approach

To lay out our approach we first reformulate the differentiable architecture search space of DARTS [25] in a way that enables direct optimization over the architecture weights. We then propose a novel optimizer that views NAS as an online selection task, and relies on PEA theory for the selection.

2.1 Neural Architecture Space

We start with a brief review of the PEA settings and then describe our view of the search space as separable PEA sub-spaces. This enables us to leverage PEA theory for NAS.

PEA Settings. PEA [4] refers to a sequential decision making framework, dealing with a decision maker, i.e. a forecaster, whose goal is to predict an unknown outcome sequence $\{y_t\}_{t=1}^T \in \mathcal{Y}$ while having access to a set of N experts' advises, i.e. predictions. Denote the experts' predictions at time t by $f_{t,1}, \ldots, f_{t,N} \in \mathcal{D}$, where $\mathcal{D}$ is the *decision space*, which we assume to be a convex subset of a vector space. Denote the forecaster's prediction $\{p_t\}_{t=1}^T \in \mathcal{D}$, and a non-negative loss function $\ell : \mathcal{D} \times \mathcal{Y} \to \mathbb{R}$. At each time step $t = 1, \ldots, T$, the forecaster observes $f_{t,1}, \ldots, f_{t,N}$ and predicts p_t. The forecaster and the experts suffer losses of $\ell_t(p_t) := \ell(p_t, y_t)$ and $\ell_t(f_{t,i}) := \ell(f_{t,i}, y_t)$ respectively.

The Search Space Viewed as Separable PEA Sub-spaces. We view the search space suggested by DARTS [25] as multiple separable sub-spaces of experts, as illustrated in Figure 1, described next. An architecture is built from replications of *normal* and *reduction cells* represented as a directed acyclic graph. Every node $x^{(j)}$ in the graph represents a feature map and each directed edge (j,k) is associated with a forecaster, that predicts a feature map $p^{(j,k)} := p^{(j,k)}(x^{(j)})$ given the input $x^{(j)}$. Intermediate nodes are computed based on all of their predecessors: $x^{(k)} = \Sigma_{j<k} p^{(j,k)}$. The output of the cell is obtained by applying a reduction operation (e.g. concatenation) to the intermediate nodes. During the search stage, every forecaster combines N experts' feature map predictions $\{f_i^{(j,k)}\}_{i=1}^N := \{f_i^{(j,k)}(x^{(j)})\}_{i=1}^N$ forming its own prediction,

$$p^{(j,k)} = \sum_{i=1}^{N} u_i^{(j,k)} f_i^{(j,k)} \quad ; \quad u_i^{(j,k)} = \frac{v_i^{(j,k)}}{\sum_{l=1}^{N} v_l^{(j,k)}} \quad ; \quad v_i^{(j,k)} \geq 0 \tag{1}$$

From now on, we will ignore the superscript indices (j,k) for brevity. Each expert represents a neural operation, e.g. convolution or max pooling, associated with network weights w_i, that receives an input x_t at time t, and outputs a feature map prediction. Thus a time index is attached, as each prediction $f_{t,i} := f_i(x_t)$ is associated with updated weights $w_{t,i}$.

Our architecture search approach is composed of two stages. In the *search stage*, the weights $w_{t,i}$ and $v_{t,i}$ are alternately optimized as described in section 2.2; then, in the *discretization stage*, a discrete child architecture is obtained as explained next.

The Discretization Stage. Once the architecture weights are optimized, the final discrete neural architecture is obtained by performing the following discretization stage, adopted from [25]: Firstly, the strongest two predecessor edges are retained for each intermediate node. The strength of an edge is defined as $\max_i u_i^{(k,j)}$. Lastly, every forecaster is replaced by the corresponding strongest expert.

2.2 XNAS: eXperts Neural Architecture Search

The differential space, described in section 2.1, enables direct optimization over the architecture weights via gradient-descent based techniques. Previous methods adopted generic optimizers commonly used for training the network weights. For example [25, 49, 6, 3] used adam [20], and [29] used SGD with momentum. While those optimizers excel in joint minimization of neural network losses when applied to network weights, NAS is a essentially a *selection* task, aiming to select a subset of experts out of a superset. The experts weights form a convex combination, as they compete over a forecaster's attention.

We argue that a generic alternate optimization of network weights and architecture weights, as suggested in previous works, e.g. [17, 25], is not suitable for the unique structure of the architecture space. Hence, we design a tailor-made optimizer for this task, inspired by PEA theory. In order to evaluate experts' performance, a loss is to be associated with each expert. However, an explicit loss is not assigned to each expert, as opposed to a back-propagated loss gradient. Therefore, we base our algorithm on a version of the Exponentiated-Gradient (EG) algorithm adapted for the NAS space. EG algorithms favor sparse weight vectors [21], thus fit well to online selection problems.

We introduce the XNAS (eXperts Neural Architecture Search) algorithm, outlined in Algorithm 1 for a single forecaster. XNAS alternates between optimizing the network weights $\boldsymbol{w}$ and the architecture weights $\boldsymbol{v}$ in a designated manner. After updating $\boldsymbol{w}$ (line 4) by descending the train loss ℓ_{train}, the forecaster makes a prediction based on the mixture of experts (line 5), then $\boldsymbol{v}$ are updated with respect to the validation loss ℓ_{val} according to the EG rule (lines 8-9). Next, the optimizer wipes-out weak experts (lines 11-12), effectively assigning their weights to the remaining ones. The exponential update terms, i.e. rewards, are determined by the projection of the loss gradient on the experts' predictions: $R_{t,i} = -\nabla_{p_t}\ell_{\text{val}}(p_t) \cdot f_{t,i}$. In section 3.1 we relate to the equivalence of this reward to the one associated with *policy gradient* search [45] applied to NAS.

Algorithm 1 XNAS for a single forecaster

```
1:  Input: The learning rate $\eta$,
    Loss-gradient bound $\mathcal{L}$ ,
    Experts predictions $\{f_{t,i}\}_{i=1}^{N}$ $\forall t = 1, \dots, T$
2:  Init: $I_0 = \{1, \dots, N\}$, $v_{0,i} \leftarrow 1$, $\forall i \in I_0$
3:  for rounds $t = 1, \dots, T$ do
4:      Update $\boldsymbol{\omega}$ by descending $\nabla_{\boldsymbol{\omega}}\ell_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{v})$
5:      $p_t \leftarrow \frac{\sum_{i \in I_{t-1}} v_{t-1,i} \cdot f_{t-1,i}}{\sum_{i \in I_{t-1}} v_{t-1,i}}$        #Predict
6:      $\{$loss gradient revealed: $\nabla_{p_t}\ell_{\text{val}}(p_t)\}$
7:      for $i \in I_{t-1}$ do
8:          $R_{t,i} = -\nabla_{p_t}\ell_{\text{val}}(p_t) \cdot f_{t,i}$      #Rewards
9:          $v_{t,i} \leftarrow v_{t-1,i} \cdot \exp\{\eta R_{t,i}\}$      #EG step
10:     end for
11:     $\theta_t \leftarrow \max_{i \in I_{t-1}} \{v_{t,i}\} \cdot \exp\{-2\eta\mathcal{L}(T - t)\}$
12:     $I_t \leftarrow I_{t-1} \setminus \{i \mid v_{t,i} < \theta_t\}$        #Wipeout
13: end for
```

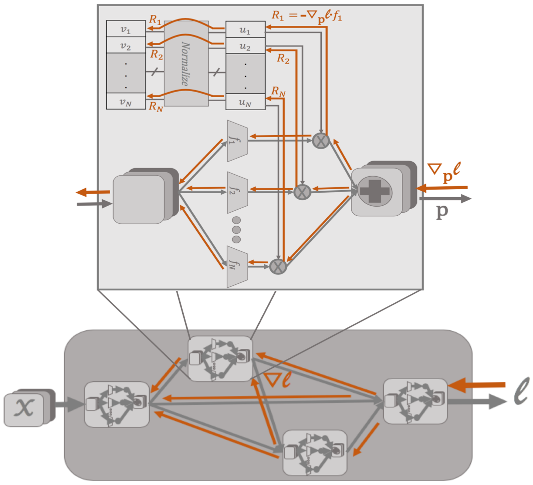


Figure 1: A visual illustration of the search space as separable experts sub-problems, equipped with XNAS forward and backward propagation. forward and backward passes are presented as grey and orange arrows respectively.

The purpose of the wipeout step is threefold. First, the removal of weak experts consistently smooths the process towards selecting a final architecture at the descretization stage described in section 2.1. Thus it mitigates the harsh final pruning of previous methods, which results in a *relaxation bias* addressed in [50, 29]. Second, it dynamically reduces the number of network weights, thus simplifying the optimization problem, avoiding over-fitting and allowing it to converge to a better solution. Last, it speeds-up the architecture search, as the number of graph computations decreases with the removal of experts.

3 Analysis and Discussion

3.1 Theoretical Analysis

In this section we analyse the performance of the proposed algorithm. For this purpose, we introduce the *regret* as a performance measure for NAS algorithms. Showing that the wipeout mechanism cannot eliminate the best expert (lemma 1), we provide theoretical guarantees for XNAS with respect to that measure (theorem 1). Proofs appear in Section 7 of the supplementary material for brevity. Relaying on the theoretical analysis, we extract practical instructions with regard to the choice of multiple learning rates. Finally we briefly discuss the equivalence of our reward to the one considered by the policy gradient approach applied to NAS.

The Regret as a Performance Measure. Denote the regret, the cumulative losses of the forecaster and of the ith expert at time t by,

$$\text{Regret}_t = L_t - \min_{i=1,\dots N} L_{t,i} \quad ; \quad L_t = \sum_{s=1}^{t} \ell_s(p_s) \quad ; \quad L_{t,i} = \sum_{s=1}^{t} \ell_s(f_{s,i}) \tag{2}$$

respectively. The regret measures how much the forecaster regrets not following the advice of the best expert in hindsight. This criterion suits our setup as we optimize a mixture of experts and select the best one by the end of the process.

In classical learning theory, statistical properties of the underlying process may be estimated on the basis of stationarity assumptions over the sequence of past observations. Thus effective prediction rules can be derived from these estimates [4]. However, NAS methods that alternately learn the architecture weights and train the network weights are highly non-stationary. In PEA theory, no statistical assumptions are made, as "simplicity is a merit" [14], and worst-case bounds are derived for the forecaster's performance. We obtain such bounds for the wipeout mechanism and the regret.

A Safe Wipeout. In XNAS (line 11), by the choice of the wipeout thresholds, experts with no chance of taking the lead by the end of the search are wiped out along the process. In a worse-case setup, a single incorrect wipeout might result in a large regret, i.e., linear in the number of steps T, due to a loss gap at each consecutive step. The following lemma assures that this cannot happen,

Lemma 1. *In XNAS, the optimal expert in hindsight cannot be wiped out.*

The wipeout effectively transfers the attention to leading experts. Define the *wipeout factor* and the *aggregated wipeout factor* as $\Gamma_t := 1 + \frac{\sum_{i \in I_{t-1} \setminus I_t} v_{t,i}}{\sum_{i \in I_t} v_{t,i}}$ and $\gamma_T := \prod_{t=1}^{T} \Gamma_t$, respectively.

Lemma 2. *The aggregated wipeout factor satisfies* $1 \leq \gamma_T < N$.

Equipped with both lemmas, we show that the wipeout may improve the EG regret bound for certain reward sequences.

Regret Bounds. Our main theorem guarantees an upper bound for the regret,

Theorem 1 (XNAS Regret Bound)**.** *The regret of the XNAS algorithm 1, with N experts and learning rate η, incurring a sequence of T non-negative convex losses of $\mathcal{L}$-bounded rewards, satisfies,*

$$\text{Regret}_T \leq \frac{\eta T \mathcal{L}^2}{2} + \frac{1}{\eta} \ln N - \frac{1}{\eta} \ln \gamma_T \tag{3}$$

As an input parameter of XNAS, the learning rate η cannot be determined based on the value of γ_T, since the later depends on the data sequence. Choosing the minimizer η^* of the first two terms of (3) fully known in advance, yields the following tight upper bound,

$$\eta^* = \sqrt{\frac{2\ln N}{T\mathcal{L}^2}} \quad ; \quad \mathrm{Regret}_T \leq \mathcal{L}\sqrt{2T\ln N}\left(1 - \frac{1}{2}\frac{\ln \gamma_T}{\ln N}\right) \tag{4}$$

The regret upper bound of XNAS is tight, as the lower bound can be shown to be of $\Omega(\mathcal{L}\sqrt{T\ln N})$ [13]. In addition, the wipeout related term reduces the regret in an amount which depends on the data sequences through γ_T, as it effectively contributes the attention of weak experts to the leading ones. For comparison, under the same assumptions, the worst-case regret bound of gradient-descent is of $O(\mathcal{L}\sqrt{TN})$ [14], while the one of Adam is linear in T [34]. An illustration of the relationship between the regret and the rate of correct expert selection appears in section 8.3 of the supplementary material, where XNAS is shown to achieve a better regret compared to a generic optimizer.

Multiple Learning Rates. Equation 4 connects the optimal theoretical learning rate η^* with the number of steps T, which is also the number of gradient feedbacks received by the experts. Since forecasters weights are being replicated among different cells, the number of feedbacks is different for normal and reduction cells (section 2.1). Explicitly, $T_c = d \times \mathcal{E} \times r_c$, where $T_c, d, \mathcal{E}, r_c$ are the effective horizon T, the validation set size, the number of epochs and the number of replications for cell type c respectively. We adopt the usage of multiple learning rates η_c^* in our experiments as upper bounds on the learning rates for minimizing the upper bound of the regret.

The Connection to Policy Gradient. We conclude this section by pointing out an interesting connection between *policy gradient* in NAS [55] and PEA theory. We refer to the PEA based reward term in line 8 of algorithm 1. This reward has been shown by [50] to be the same effective reward of a policy gradient method applied to the common NAS optimization criterion [55, 56, 30]. More precisely, consider the case where instead of mixing the experts' predictions, the forecaster is to sample a single expert at each step with probability $u_{t,i}$, specified in (1). Then the effective reward associated with the policy gradient will be exactly the derived PEA reward, $R_{t,i} = -\nabla_{p_t}\ell_t(p_t) \cdot f_{t,i}$. XNAS optimizes with respect to the same reward, while avoiding the sampling inefficiency associated with policy gradient methods.

3.2 Key Properties and Discussion

In this section we discuss some of the key properties of XNAS. For each of this properties we provide supporting derivations, illustrations and demonstrations appearing in section 8 of the supplementary material for brevity.

3.2.1 The Recovery of Late Bloomers and Robustness to Initialization

In this section we point out a key difference between our proposed update rule and the one used in previous works. We refer to Gradient Descent (GD) with softmax as updating the parameters $\alpha_{t,i} = \ln v_{t,i}$, by descending $\nabla_{\boldsymbol{\alpha_{t,i}}}\ell_t(p_t)$ respectively. Variants of GD with softmax, as used in [25] to optimize the architecture weights, suppress operations that are weak at the initial iterations, making it more difficult for them to "bloom" and increase their weights. This could be problematic, e.g. in the two following cases. First, consider the best expert starting with a poor weight which gradually rises. This could be the case when an expert representing a parameterized operation (e.g. a convolutional layer) competes with an unparameterized one (e.g. a pooling layer), as the first requires some period for training, as stated by [29, 17]. Second, consider a noisy setup, where the best expert in hindsight could receive some hard penalties before other inferior experts do. In NAS we deal with stochastic settings associated with the training data.

We inspect the update term of GD with softmax,

$$v_{t,i} = \exp\{\alpha_{t,i}\} = \exp\{\alpha_{t-1,i} - \eta\partial_{\alpha_{t-1,i}}\ell_t(p_t)\} = v_{t-1,i} \cdot \exp\{-\eta\partial_{\alpha_{t-1,i}}\ell_t(p_t)\} \tag{5}$$

Hence, the effective reward in this case is,

$$\tilde{R}_{t,i} := -\partial_{\alpha_{t-1,i}}\ell_t(p_t) = -\nabla_{p_t}\ell_t(p_t) \cdot u_{t-1,i}\,(f_{t,i} - p_t) \tag{6}$$

See derivations in section 8.4. The linear dependence on the expert's weight $u_{t-1,i}$ in (6) implies that GD with softmax makes it harder for an expert whose weight is weak at some point to recover and become dominant later on, as the associated rewards are attenuated by the weak expert's weight.

XNAS mitigates this undesirable behavior. Since for XNAS the update term (8) depends on the architecture weights only indirectly, i.e. through the prediction, the recovery of *late bloomers* is not discouraged, as demonstrated in section 8.1 of the supplementary material. From the very same reasons, XNAS is more robust to the initialization scheme compared to GD with softmax and its variants, as demonstrated in section 8.2 of the supplementary material. These advantages make XNAS more suitable for the NAS setup.

Note that while the XNAS enables the recovery of experts with badly initialized weights or with delayed rewards, the wipeout mechanism prevents inferior operations that start blooming too late from interfering, by eliminating experts with no chance of leading at the end.

Wipeout Factor. As mentioned in section 2.2, the wipeout mechanism contributes to both optimization process and search duration. A further reduction in duration can be achieved when the wipe-out threshold in line 11 of Algorithm 1 is relaxed with a parameter $0 < \zeta \leq 1$, being replaced by $\theta_t \leftarrow \max_{i \in I_{t-1}} \{v_{t,i}\} \cdot \exp\{-2\eta\mathcal{L}(T-t) \cdot \zeta\}$. This will lead to a faster convergence to a single architecture, with the price of a violation of the theoretical regret. As worst-case bounds tend to be over pessimistic, optimizing over ζ could lead to improved results. We leave that for future work.

3.2.2 Fewer Hyper Parameters

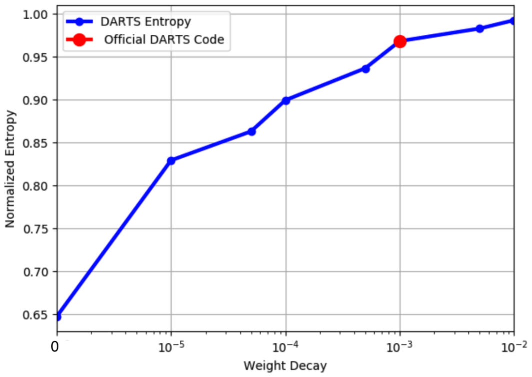
The view of the differentiable NAS problem as an optimization problem solved by variants of GD, e.g. Adam, introduces some common techniques for such schemes along with their corresponding hyper-parameters. Tuning these complicates the search process - the fewer hyper-parameters the better. We next discuss how XNAS simplifies and reduces the number of hyper-parameters.

Theoretically Derived Learning Rates. The determination of the learning rate has a significant impact on the convergence of optimization algorithms. Various scheduling schemes come up, e.g. [26, 38], as the later additionally suggests a way for obtaining an empirical upper bound on the learning rate. In section 3.1, multiple learning rates η_c^* are suggested for minimizing the regret bound (4), as $c \in \{N, R\}$ represents normal and reduction cells respectively. For example, for CIFAR10 with 50%:50% train-validation split, 50 search epochs, gradient clipping of 1, 6 normal cells and 2 reduction cells both of 8 experts for each forecaster, (4) yields $\eta_N^* =$7.5e-4 and $\eta_R^* =$1.3e-3.

Remark 1. *Note that the proposed learning rates minimize an upper bound of the regret* (4) *in the case of no wipeout, i.e. the worst case, as the extent of the wipeout cannot be known in advance. Hence the proposed learning rate provides an upper bound on the optimal learning rates and can be further fine-tuned.*

No Weight Decay. Another common technique involving hyper-parameters is weight decay, which has no place in the theory behind XNAS. We claim that the obviation of weight decay by XNAS makes sense. Regularization techniques, such as weight decay, reduce over-fitting of over-parametrized models when applied to these parameters [12]. No such effect is incurred when applying weight decay on the architecture parameters as they do not play the same role as the trained network parameters $\boldsymbol{w}$. Instead, weight decay encourages uniform dense solutions, as demonstrated in Figure 2, where the mean normalized entropy increases with the weight decay coefficient. The calculation of the mean normalized entropy is detailed in section 8.5 of the supplementary material. This observation could be associated with the suggestion of recent works [23, 35] that current search methods are only slightly better than random search. The density of results in a harder degradation in performance once *discretization stage* occurs (section 2.1), hence sparse solutions are much preferred over dense ones.

Figure 2: Mean normalized entropy vs weight decay. The red dot refers to DARTS' settings.



No Momentum. The theory behind XNAS obviates momentum [31] and ADAM's exponentially decay rates [20]. Since momentum requires more state variables and more computations, the resulting XNAS optimizer turns out to be simpler, faster and with a smaller memory footprint, compared to commonly used optimizers for NAS, e.g. ADAM [25, 49, 6, 3] and SGD with momentum [29].

4 Experiments and Results

In this section we will test XNAS on common image classification benchmarks, and show its effectiveness compared to the other state-of-the-art models.

We used the CIFAR-10 dataset for the main search and evaluation phase. In addition, using the cell found on CIFAR-10 we did transferability experiments on the well-known benchmarks ImageNet, CIFAR-100, SVHN, Fashion-MNIST, Freiburg and CINIC10.

4.1 Architecture Search on CIFAR-10

Using XNAS, we searched on CIFAR-10 in a small parent network for convolutional cells. Then we built a larger network by stacking the learned cells, trained it on CIFAR-10 and compared the results against other NAS methods.

We created the parent network by stacking 8 cells with 4 ordered nodes, each of which connected via forecasters to all previous nodes in the cell and also to the two previous cells outputs. Each forecaster contains seven operations: 3x3 and 5x5 separable and dilated separable convolutions, 3x3 max-pooling, 3x3 average-pooling and an identity. A cells output is a concatenation of the outputs of the four cells nodes.

The search phase lasts 50 epochs. We use the first-order approximation [25], relating to $\boldsymbol{v}$ and $\boldsymbol{\omega}$ as independent parameters which can be optimized separately. The train set is divided into two parts of equal sizes: one is used for training the operations weights $\boldsymbol{\omega}$ and the other for training the architecture weights $\boldsymbol{v}$. With a batch size of 96, one epoch takes 8.5 minutes in average on a single GPU[2], summing up to 7 hours in total for a single search. Figure 9 shows our learned normal and reduction cells, respectively.

4.2 CIFAR-10 Evaluation Results

We built the evaluation network by stacking 20 cells: 18 normal cells and 2 reduction cells. The reduction cells are placed after $1/3$ and $2/3$ of the network. After each reduction cell we double the amount of channels in the network. We trained the network for 1500 epochs using a batch size of 96 and SGD optimizer with nesterov-momentum. Our learning rate regime was composed of 5 cycles of power cosine annealing learning rate [17], with amplitude decay factor of 0.5 per cycle. For regularization we used cutout [9], scheduled drop-path [22], auxiliary towers [39], label smoothing [40] AutoAugment [7] and weight decay. To understand the effect of the network size on the final accuracy, we chose to test 3 architecture configurations, XNAS-Small, XNAS-Medium and XNAS-Large, with 36, 44 and 50 initial network channels respectively. Table 1 shows the performance of XNAS architectures compared to other state-of-the-art NAS methods. We can see from Table 1 that XNAS smallest network variant, XNAS-Small, outperforms on CIFAR-10 previous NAS methods by a large margin. Our largest network variant, XNAS-Large, is the second highest reported score on CIFAR-10 (without additional pre-train data), while having 4 times less parameters than top one [7]. In addition, XNAS is among the fastest NAS methods.

CIFAR-10	Test	Params	Search
Architecture	error	(M)	cost
AutoAugment [7]	1.48	26	20
NAONet-WS [27]	3.53	2.5	0.3
PNAS [24]	3.41	3.2	150
Amoeba-A [33]	3.34	3.2	3150
DSO-NAS [53]	2.95	3	1
DARTS(1nd) [25]	2.94	2.9	0.4
ENAS [30]	2.89	4.6	0.5
SNAS [50]	2.85	2.8	1.5
DARTS(2nd) [25]	2.67	3.4	1
NASNet-A [56]	2.65	3.3	1800
Amoeba-B [33]	2.55	2.8	3150
PDARTS [6]	2.50	3.4	0.3
NAONet [27]	2.11	128	200
ProxylessNAS [2]	2.08	5.7	10
ASAP + ET [29]	1.94	3.7	0.2
SharpDarts [17]	1.93	3.6	0.8
XNAS-Small	**1.81**	3.7	0.3
XNAS-Medium	**1.73**	5.6	0.3
XNAS-Large	**1.60**	7.2	0.3

Table 1: Classification errors of XNAS compared to state-of-the-art methods on CIFAR-10. The Search cost is measured in GPU days. Test error refers to top-1 test error (%). XNAS Small, Medium and Large refers to network architectures with XNAS cell and 36, 44 and 50 initial channels respectively.

[2] Experiments were performed using a NVIDIA GTX 1080Ti GPU.

4.3 Transferability Evaluation

Using the cell found by XNAS search on CIFAR-10, we preformed transferability tests on 6 popular classification benchmarks: ImageNet, CIFAR-100, Fashion-MNIST, SVHN, Freiburg and CINIC10.

ImageNet Results. Our ImageNet network was composed of 14 stacked XNAS cells, with two initial stem cells for downscaling. We used 46 initial channels so the total number of network FLOPs is below 600[M], similar to other ImageNet architectures with small computation regime [48]. We trained the network for 250 epochs with one cycle of power cosine learning rate and a nesterov-momentum optimizer. The results are presented in Table 2. We can see from Table 2 that XNAS transferability results on ImageNet are highly competitive, outperforming all previous NAS cells.

ImageNet Architecture	Test error	Params (M)	Search cost
SNAS [50]	27.3	4.3	1.5
ASAP [29]	26.7	5.1	0.2
DARTS [25]	26.7	4.9	1
NASNet-A [56]	26.0	5.3	1800
PNAS [24]	25.8	5.1	150
Amoeba-A [33]	25.5	5.1	3150
RandWire [48]	25.3	5.6	0
SharpDarts [17]	25.1	4.9	0.8
Amoeba-C [33]	24.3	6.4	3150
XNAS	**24.0**	5.2	0.3

Table 2: Transferability classification error of XNAS, compared to top NAS cells, on ImageNet. Test error refers to top-1 test error (%). Search cost is measured in GPU days.

Additional Results. We further tested XNAS transferability abilities on 5 smaller datasets: CIFAR-100 [42], Fashion-MNIST [47], SVHN [28], Freiburg [19] and CINIC10 [8]. We chose to use the XNAS-Small architecture, with similar training scheme to the one described in section 4.2. Table 3 shows the performance of our model compared to NAS methods. We can see that XNAS cell excels on the datasets tested. On CIFAR-100 it surpasses the next top cell by 1%, achieving the second highest reported score on CIFAR-100 (without additional pre-train data), second only to [7]. On Fashion-MNIST, Freiburg and CINIC10, to the best of our knowledge XNAS achieves a new state-of-the-art accuracy.

Datasets Architecture	CIFAR100 Error	FMNIST Error	SVHN Error	Freiburg Error	CINIC10 Error	Params (M)	Search cost
Known SotA	**10.7** [7]	3.65 [54]	**1.02** [7]	10.7 [29]	6.83 [29]	26 [7]	-
PDARTS [6]	15.9	-	-	-	-	3.6	0.3
NAONet-1 [27]	15.7	-	-	-	-	10.8	200
NAONet-2 [27]	14.7	-	-	-	-	128	200
PDARTS-L [6]	14.6	-	-	-	-	11	0.3
SNAS† [50]	16.5	3.72	1.98	14.7	7.13	2.8	1.5
PNAS† [24]	15.9	3.72	1.83	12.3	7.03	3.4	150
Amoeba-A† [33]	15.9	3.8	1.93	11.8	7.18	3.2	3150
NASNet† [56]	15.8	3.71	1.96	13.4	6.93	3.3	1800
DARTS† [25]	15.7	3.68	1.95	10.8	6.88	3.4	1
ASAP† [29]	15.6	3.71	1.81	10.7	6.83	2.5	0.2
XNAS-Small	13.6	**3.64**	1.72	**6.3**	**6.0**	3.7	0.3

Table 3: Classification errors of XNAS, compared to state-of-the-art NAS methods, on several datasets. Error refers to top-1 test error (%). The Search cost is measured in GPU days. Results marked with † are taken from [29], which tested and compared different cells on various datasets.

5 Related Work

Mentions of Experts in deep learning [32, 52, 11, 1] literature go decades back [18, 5], typically combining models as separable experts sub-models. A different concept of using multiple mixtures of experts as inner parts of a deep model, where each mixture has its own gating network, is presented in [10]. Following works build upon this idea and include a gating mechanism per mixture [51, 41], and some further suggest sparsity regularization over experts via the gating mechanism [37, 44]. These gating mechanisms can be seen as a dynamic routing, which activates a single or a group of experts in the network on a per-example basis. Inspired by these works, our methods leverage PEA principled methods for automatically designing neural network inner components.

Furthermore, optimizers based on PEA theory may be useful for the neural architecture search phase. Common stochastic gradient-descent (SGD) and a set of PEA approaches, such as follow-the-regularized-leader (FTRL), were shown by [36, 14, 43] to be equivalent. Current NAS methods [56, 55, 30, 25, 2, 46, 23, 17, 29] use Adam, SGD with Momentum or other common optimizers. One notion that is common in PEA principled methods is the regret [4]. PEA strategies aim to guarantee a small regret under various conditions. We use the regret as a NAS objective, in order to establish a better principled optimizer than existing methods [23, 35]. Several gradient-descent based optimizers, such as Adam, present a regret bound analysis, however, the worst-case scenario for Adam has non-zero average regret [34], i.e., it is not robust. Our optimizer is designated for selecting architecture weights while achieving an optimal regret bound.

6 Conclusion

In this paper we presented XNAS, a PEA principled optimization method for differential neural architecture search. Inner network architecture weights that govern operations and connections, i.e. experts, are learned via exponentiated-gradient back-propagation update rule. XNAS optimization criterion is well suited for architecture-selection, since it minimizes the regret implied by sub-optimal selection of operations with tendency for sparsity, while enabling late bloomers experts to warm-up and take over during the search phase. Regret analysis suggests the use of multiple learning rates based on the amount of information carried by the backward gradient. A dynamic mechanism for wiping out weak experts is used, reducing the size of computational graph along the search phase, hence reducing the search time and increasing the final accuracy. XNAS shows strong performance on several image classification datasets, while being among the fastest existing NAS methods.

Acknowledgements

We would like to thank the members of the Alibaba Israel Machine Vision Lab (AIMVL), in particular to Avi Mitrani, Avi Ben-Cohen, Yonathan Aflalo and Matan Protter for their feedbacks and productive discussions.

References

[1] R. Aljundi, P. Chakravarty, and T. Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3366–3375, 2017.

[2] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[3] F. P. Casale, J. Gordon, and N. Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.

[4] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[5] K. Chen, L. Xu, and H. Chi. Improved learning algorithms for mixture of experts in multiclass classification. *Neural networks*, 12(9):1229–1252, 1999.

[6] X. Chen, L. Xie, J. Wu, and Q. Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint arXiv:1904.12760*, 2019.

[7] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[8] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

[9] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[10] D. Eigen, M. Ranzato, and I. Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

[11] E. Garmash and C. Monz. Ensemble learning for multi-source neural machine translation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1409–1418, 2016.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[13] D. Haussler, J. Kivinen, and M. K. Warmuth. Tight worst-case loss bounds for predicting with expert advice. In *European Conference on Computational Learning Theory*, pages 69–83. Springer, 1995.

[14] E. Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[15] W. Hoeffding. A lower bound for the average sample number of a sequential test. *The Annals of Mathematical Statistics*, pages 127–130, 1953.

[16] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[17] A. Hundt, V. Jain, and G. D. Hager. sharpdarts: Faster and more accurate differentiable architecture search. *arXiv preprint arXiv:1903.09900*, 2019.

[18] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, Mar. 1991.

[19] P. Jund, N. Abdo, A. Eitel, and W. Burgard. The freiburg groceries dataset. *arXiv preprint arXiv:1611.05799*, 2016.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *information and computation*, 132(1):1–63, 1997.

[22] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[23] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[24] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[25] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[26] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[27] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pages 7827–7838, 2018.

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[29] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik-Manor. Asap: Architecture search, anneal and prune. *arXiv preprint arXiv:1904.04123*, 2019.

[30] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, , and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, 2018.

[31] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[32] C. E. Rasmussen and Z. Ghahramani. Infinite mixtures of gaussian process experts. In *Advances in neural information processing systems*, pages 881–888, 2002.

[33] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *International Conference on Machine Learning - ICML AutoML Workshop*, 2018.

[34] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[35] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

[36] S. Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[37] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[38] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[40] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[41] R. Teja Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8080–8089, 2018.

[42] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[43] T. Van Erven, W. Kotłowski, and M. K. Warmuth. Follow the leader with dropout perturbations. In *Conference on Learning Theory*, pages 949–974, 2014.

[44] X. Wang, F. Yu, R. Wang, Y.-A. Ma, A. Mirhoseini, T. Darrell, and J. E. Gonzalez. Deep mixture of experts via shallow embedding. *arXiv preprint arXiv:1806.01531*, 2018.

[45] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[46] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, 2018.

[47] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[48] S. Xie, A. Kirillov, R. Girshick, and K. He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019.

[49] S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[50] S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: Stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[51] B. Yang, G. Bender, Q. V. Le, and J. Ngiam. Soft conditional computation. *CoRR*, abs/1904.04971, 2019.

[52] B. Yao, D. Walther, D. Beck, and L. Fei-Fei. Hierarchical mixture of classification experts uncovers interactions between brain regions. In *Advances in Neural Information Processing Systems*, pages 2178–2186, 2009.

[53] X. Zhang, Z. Huang, and N. Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arxiv 1811.01567*, 2018.

[54] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

[55] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[56] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Supplementary Material

7 Proofs

7.1 Proof of Lemma 1

Proof: By contradiction, assume that expert j is being wiped-out at the iteration t based on line 11 in algorithm 1, and without loss of generality, $k \neq j$ is the leading expert at that time,

$$\begin{aligned} v_{t,k} &= \arg\max_i \{v_{t,i}\} \\ v_{t,j} &< v_{t,k} \cdot \exp\{-2\eta\mathcal{L}(T-t)\} \end{aligned} \tag{7}$$

Since expert j is the optimal one in hindsight, specifically,

$$v_{T,j} \geq v_{T,k} \tag{8}$$

However, since the loss is $\mathcal{L}$-bounded, the ratios between weights at time t, T are bounded as well,

$$v_{T,k} \geq v_{t,k} \cdot \exp\{-\eta\mathcal{L}(T-t)\} \tag{9}$$

$$v_{T,j} \leq v_{t,j} \cdot \exp\{\eta\mathcal{L}(T-t)\} \tag{10}$$

Recap inequalities (8)-(10),

$$\begin{aligned} v_{t,j} &\geq v_{T,j} \cdot \exp\{-\eta\mathcal{L}(T-t)\} \\ &\geq v_{T,k} \cdot \exp\{-\eta\mathcal{L}(T-t)\} \\ &\geq v_{t,k} \cdot \exp\{-2\eta\mathcal{L}(T-t)\} \end{aligned}$$

In contradiction to (7). □

7.2 Proof of Lemma 2

Proof: Left Hand Side. Due to the non-negativity of $\{v_{t,i}\}_{i=1}^N$ for all $t = 1, \ldots, T$, we have,

$$\Gamma_t = 1 + \frac{\sum_{i \in I_{t-1} \setminus I_t} v_{t-1,i}}{\sum_{i \in I_t} v_{t,i}} \geq 1 \tag{11}$$

Hence,

$$\gamma_T = \prod_{t=1}^{T} \Gamma_t \geq \prod_{t=1}^{T} 1 = 1 \tag{12}$$

Right Hand Side. Denote $|\mathcal{S}|$ as the cardinally of the set $\mathcal{S}$. Then,

$$\begin{aligned} \Gamma_t &= 1 + \frac{\sum_{i \in I_{t-1} \setminus I_t} v_{t,i}}{\sum_{i \in I_t} v_{t,i}} \\ &< 1 + \frac{|I_{t-1} \setminus I_t| \cdot \theta_t}{\sum_{i \in I_t} v_{t,i}} && (13) \\ &\leq 1 + \frac{|I_{t-1} \setminus I_t| \cdot \theta_t}{|I_t| \cdot \theta_t} && (14) \\ &= \frac{|I_t| + |I_{t-1} \setminus I_t|}{|I_t|} \\ &= \frac{|I_{t-1}|}{|I_t|} \end{aligned}$$

where (13) is since $v_{t,i} < \theta_t \, \forall i \in I_{t-1} \setminus I_t$ and (14) is since $v_{t,i} \geq \theta_t \, \forall i \in I_t$, according to line 12 of algorithm 1. Thus taking the telescopic product yields,

$$\gamma_T = \prod_{t=1}^{T} \Gamma_t < \prod_{t=1}^{T} \frac{|I_{t-1}|}{|I_t|} = \frac{|I_0|}{|I_T|} = \frac{N}{|I_T|} \leq N \tag{15}$$

Finally we have,

$$1 \leq \gamma_T < N \tag{16}$$

□

7.3 Proof of XNAS Regret Bound

Proof: First let us state an auxiliary lemma,
Lemma 3 (Hoeffding). *For any random variable X with* $\Pr\left(a \leq X \leq b\right) = 1$ *and* $s \in \mathbb{R}$,

$$\ln \mathrm{E}\left[e^{sX}\right] \leq s\mathrm{E}\left[X\right] + \frac{s^2(b-a)^2}{8}. \tag{17}$$

Proof can be found in [15].

We start with defining experts' auxiliary and accumulated-auxiliary losses following [4] ,

$$\ell'_t(f_{t,i}) \quad = \quad \nabla\ell(p_t) \cdot f_{t,i} = -R_{t,i} \tag{18}$$

$$L'_{T,i} \quad = \quad \sum_{t=1}^{T} \ell'_t(f_{t,i}) \tag{19}$$

Notice that the auxiliary losses are bounded by an input parameter in line 1 of algorithm 1,

$$|\ell'_t(f_{t,i})| \leq \mathcal{L} \tag{20}$$

We also define the set of remaining experts at time t,

$$I_t = \{i | i \in I_{t-1} \text{ and } v_{t,i} \geq \theta_t\} \quad ; \quad I_0 = \{1, \dots, N\} \tag{21}$$

Such that,

$$I_T \subset I_{T-1} \subset \cdots \subset I_0 \tag{22}$$

We Now bound the ratio of weights sums from both sides.
Let us derive a lower bound:

$$\begin{aligned}
\ln \frac{V_T}{V_0} &= \ln \frac{\sum_{i \in I_T} v_{T,i}}{N} \\
&= \ln \left(\sum_{i \in I_T} \exp\left\{-\eta L'_{T,i}\right\} \right) - \ln N \\
&\geq \ln \max_{i \in I_T} \exp\left\{-\eta L'_{T,i}\right\} - \ln N \\
&= \ln \max_{i=1,\dots,N} \exp\left\{-\eta L'_{T,i}\right\} - \ln N & (23) \\
&= -\eta \min_{i=1,\dots,N} L'_{T,i} - \ln N
\end{aligned}$$

Where we used lemma 1 in (23), assuring that loss minimizer is among the remaining experts.
Let us derive an upper bound:

$$\begin{aligned}
\ln \frac{V_t}{V_{t-1}} &= \ln \frac{\sum_{i \in I_t} v_{t,i}}{\sum_{i \in I_{t-1}} v_{t-1,i}} \\
&= \ln \frac{\sum_{i \in I_{t-1}} v_{t,i}}{\sum_{i \in I_{t-1}} v_{t-1,i}} + \ln \frac{\sum_{i \in I_t} v_{t,i}}{\sum_{i \in I_{t-1}} v_{t,i}} \\
&= \ln \frac{\sum_{i \in I_{t-1}} v_{t,i}}{\sum_{i \in I_{t-1}} v_{t-1,i}} + \ln \frac{\sum_{i \in I_t} v_{t,i}}{\sum_{i \in I_t} v_{t,i} + \sum_{i \in I_{t-1} \setminus I_t} v_{t,i}} & (24) \\
&= \ln \frac{\sum_{i \in I_{t-1}} v_{t,i}}{\sum_{i \in I_{t-1}} v_{t-1,i}} - \ln \Gamma_t & (25) \\
&= \ln \frac{\sum_{i \in I_{t-1}} v_{t-1,i} \cdot \exp\left\{-\eta \ell'_t(f_{t,i})\right\}}{\sum_{i \in I_{t-1}} v_{t-1,i}} - \ln \Gamma_t \\
&\leq -\eta \frac{\sum_{i \in I_t} v_{t-1,i} \cdot \ell'_t(f_{t,i})}{\sum_{i \in I_t} v_{t-1,i}} + \frac{\eta^2 \mathcal{L}^2}{2} - \ln \Gamma_t & (26) \\
&= -\eta \ell'_t(p_t) + \frac{\eta^2 \mathcal{L}^2}{2} - \ln \Gamma_t & (27)
\end{aligned}$$

Where (24) is due to (22) and (25) is by setting,

$$\Gamma_t := 1 + \frac{\sum_{i \in I_{t-1} \setminus I_t} v_{t,i}}{\sum_{i \in I_t} v_{t,i}} = \frac{\sum_{i \in I_t} v_{t,i} + \sum_{i \in I_{t-1} \setminus I_t} v_{t,i}}{\sum_{i \in I_t} v_{t,i}} \tag{28}$$

Inequality (26) results from lemma 3, with $[a, b] = [-\mathcal{L}, \mathcal{L}]$ and (27) is a result of the linearity of ℓ'_t. Summing the logs telescopic sum:

$$\ln \frac{V_T}{V_0} = \sum_{t=1}^{T} \ln \frac{V_t}{V_{t-1}} \leq -\eta \sum_{t=1}^{T} \ell'_t(p_t) + \frac{\eta^2 T \mathcal{L}^2}{2} - \sum_{t=1}^{T} \ln \Gamma_t \tag{29}$$

Setting $\gamma_T := \prod_{t=1}^{T} \Gamma_t$ with bounds specified in by lemma 2, we have,

$$\ln \frac{V_T}{V_0} \leq -\eta \sum_{t=1}^{T} \ell'_t(p_t) + \frac{\eta^2 T \mathcal{L}^2}{2} - \ln \gamma_T \tag{30}$$

Combining the lower and upper bounds and dividing by η,

$$\sum_{t=1}^{T} \ell'_t(p_t) - \min_{i=1,\ldots,N} L'_{T,i} \leq \frac{\eta T \mathcal{L}^2}{2} + \frac{1}{\eta} \ln N - \frac{1}{\eta} \ln \gamma_T \tag{31}$$

We now bound the accumulated regret, using the convexity of the loss,

$$\begin{aligned}
\text{Regret}_T &= \sum_{t=1}^{T} \ell_t(p_t) - \min_{i=1,\ldots,N} \sum_{t=1}^{T} \ell_t(f_{t,i}) \\
&= \max_{i=1,\ldots,N} \left\{ \sum_{t=1}^{T} \ell_t(p_t) - \ell_t(f_{t,i}) \right\} \\
&\leq \max_{i=1,\ldots,N} \left\{ \sum_{t=1}^{T} \nabla \ell_t(p_t) \cdot (p_t - f_{t,i}) \right\} \\
&= \max_{i=1,\ldots,N} \left\{ \sum_{t=1}^{T} \ell'_t(p_t) - \ell'_t(f_{t,i}) \right\} \\
&= \sum_{t=1}^{T} \ell'_t(p_t) - \min_{i=1,\ldots,N} L'_{T,i}
\end{aligned} \tag{32}$$

Combining 31 and 32 completes the proof,

$$\text{Regret}_T \leq \frac{\eta T \mathcal{L}^2}{2} + \frac{1}{\eta} \ln N - \frac{1}{\eta} \ln \gamma_T \tag{33}$$

□

8 Supporting Materials for the Key Properties Discussion

8.1 A Deterministic 3D Axes Toy Problem

In an attempt to demonstrate the possible recovery of *late bloomers*, we view an optimization problem in a three dimensional space as a prediction-with-experts problem, where each axis represents an expert with a constant prediction, i.e. $f_{t,x} \equiv (1,0,0), f_{t,y} \equiv (0,1,0), f_{t,z} \equiv (0,0,1)$ for the x, y, z axes respectively. The forecaster then makes a prediction according to the following,

$$p_t = \frac{v_{t,x} \cdot f_{t,x} + v_{t,y} \cdot f_{t,y} + v_{t,z} \cdot f_{t,z}}{v_{t,x} + v_{t,y} + v_{t,z}} = x_t \cdot f_{t,x} + y_t \cdot f_{t,y} + z_t \cdot f_{t,z} = (x_t, y_t, z_t)$$

with p_t in the simplex, i.e. $(x_t, y_t, z_t) \in \Delta = \{(x, y, z) \mid x, y, z \geq 0, x + y + z = 1\}$. Setting $v_{t,i} = \exp\{\alpha_{t,i}\}$ for $i \in \{x, y, z\}$, for a given loss function $\ell(p_t)$, the update terms for GD with softmax (section 3.2.1) and XNAS (section 2.2), associated with the z axis, are as follows,

$$(\mathbf{GD})\ \partial_{\alpha_z}\ell(p_t) = z((x+y)\partial_z\ell(p_t) - x\partial_x\ell(p_t) - y\partial_y\ell(p_t)) \quad (\mathbf{XNAS})\ \nabla_{p_t}\ell(p_t)f_{t,z} = \partial_z\ell(p_t)$$

as the corresponding terms for the x and y axes are similar by symmetry, see a full derivation in section 8.4.2.

Now let us present the following toy problem: A three dimensional linear loss function $\ell_1(x, y, z) = 2z$ is being optimized for 30 gradient steps over the simplex, i.e. $(x, y, z) \in \Delta$. Then for illustrating a region shift in terms of the best expert, the loss function shifts into another linear loss function $\ell_2(x, y, z) = -y - 2z$ which is then optimized for additional 90 steps. The trajectories are presented in Figure 3. At the first stage the z axis suffers many penalties as its weight shrinks. Then it starts receiving rewards. For GD with softmax, those rewards are attenuated, as explained in section 3.2.1 and can be seen in Figure 3 (right). Despite the linear loss, with constant gradients, the update term decays. Note that this effect is even more severe when dealing with more common losses of higher curvature, where the gradients decay near the local minimum and then further attenuated, as illustrated in section 8.2.4. Once the gradient shifts, it is already depressed due to past penalties, hence the z axis struggles to recover. XNAS, however, is agnostic to the order of the loss values in time. Once the rewards balance out the penalties, the path leads towards the z axis. In the meanwhile, the y axis takes the lead.

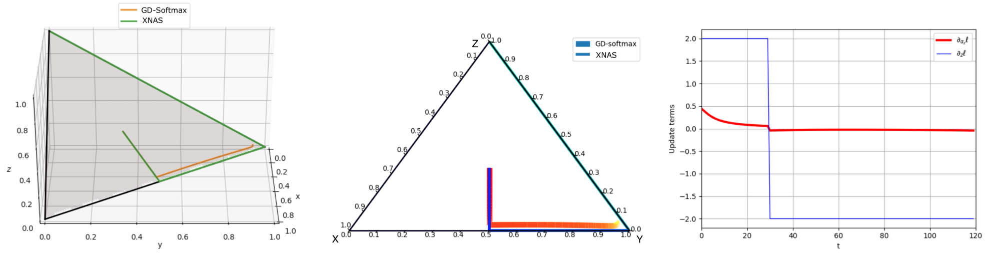


Figure 3: Trajectories of XNAS and GD with softmax in 3D (left), projected on the simplex (middle) and the update terms for the z axis (right). At the middle, color indicates progression through time, starting darker and ending lighter

8.2 A Deterministic 2D Axes Toy Problem

In section 8.1 we show the built-in attenuation of weak operations by GD with softmax. This is illustrated by a three dimensional toy example where the axes represent experts of constant predictions. Here we elaborate on this effect using a similar two dimensional toy problem, where the gradients with respect to the axes are the negative values of one another. See section 8.4.3 for the setup and full derivations. All the experiments in this section are conducted using a learning rate of 0.1 for both optimizers for 50 steps.

8.2.1 Balanced Initialization

Figure 4 illustrates the attenuation of gradients for GD with softmax, as although the gradients of the loss are constant, the gradients' magnitude decreases as we move away from the initialization $\alpha_x = \alpha_y$, i.e. $(x, y) = (0.5, 0.5)$. XNAS indeed receives constant gradients thus reaches the minima faster.

8.2.2 Imbalanced Initialization

The attenuated gradients also make GD with softmax more sensitive to the initialization, as demonstrated in Figure 5, where $\alpha_x = 0 < 5 = \alpha_y$ and GD with softmax, whose gradients are attenuated, makes no progress while XNAS reaches the minima.

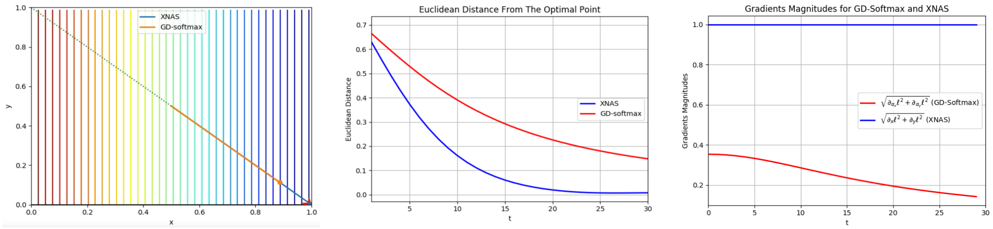


Figure 4: Trajectories of GD with softmax and XNAS for a $\ell(x, y) = -x$ (left) the distance from the optimal point (middle) and the corresponding magnitude of the gradients (right) for a balanced initialization $\alpha_y = \alpha_x$. The vertical lines are the contour lines of the loss function. The dashed line is the simplex $x + y = 1$, and the trajectories are the solid lines with circles at their ends.

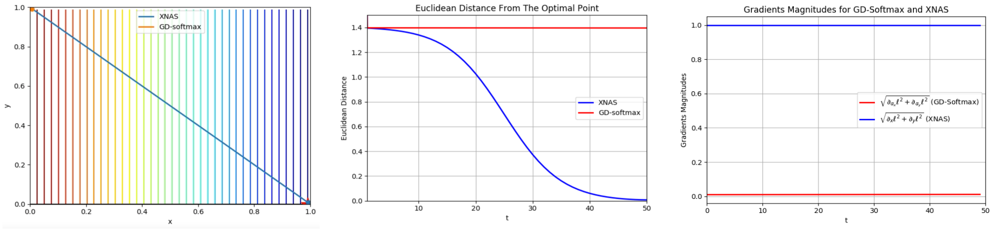


Figure 5: Trajectories of GD with softmax and XNAS for a $\ell(x, y) = -x$ (left) the distance from the optimal point (middle) and the corresponding magnitude of the gradients (right) for an imbalanced initialization $\alpha_x < \alpha_y$. The vertical lines are the contour lines of the loss function. The dashed line is the simplex $x + y = 1$, and the trajectories are the solid lines with circles at their ends.

8.2.3 The Preference of Dense Solutions

Presenting the attenuation factor $x \cdot y$ on the simplex, i.e. $x \cdot (1 - x)$, in Figure 6, demonstrates how gradients are harder attenuated as far away as the variables move from a dense solution, e.g. $(x, y) = (0.5, 0.5)$ at $\alpha_x = \alpha_y$. Hence, it is harder for GD with softmax to strengthen a single expert over the other. This effect encourages dense solutions over sparse solutions, i.e. a choice of a single expert. Due to the *descretization stage*, described in section 2.1, the denser the solution is, the more degradation in performance is incurred. Hence dense solutions should be discouraged rather than encouraged.

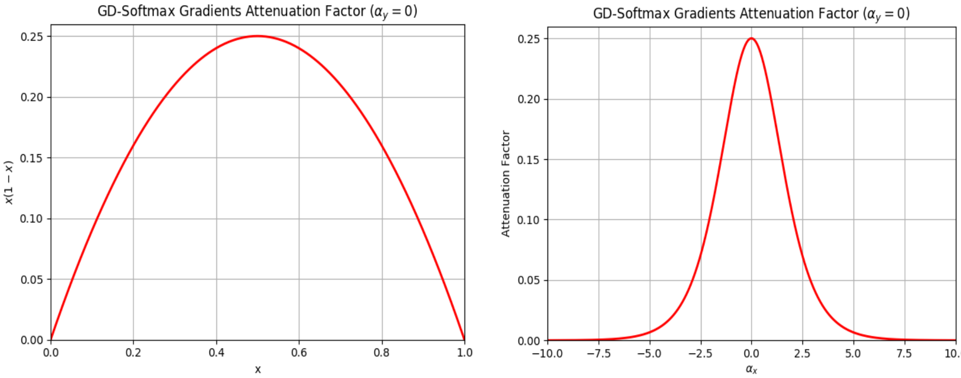


Figure 6: The attenuation factor of GD with softmax along the simplex (left) and versus α_x as $\alpha_y = 0$ (right)

8.2.4 Loss Functions of a Higher Curvature

Sections 8.2.1 and 8.2.2 deal with a linear loss function of no curvature and constant gradients. Once convex loss functions are considered, the gradients decrease towards the local minimum. Figure 7 illustrates the further attenuation of GD with softmax for a quadratic loss function $\ell(x,y) = 0.5(x-1)^2 + 0.5y^2$, which makes it even harder for GD with softmax to reach the local minimum.

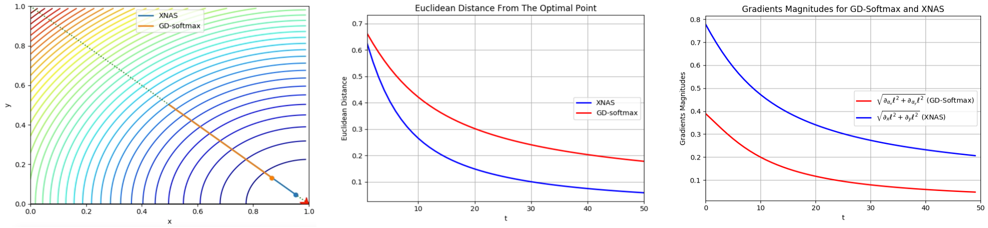


Figure 7: Trajectories of GD with softmax and XNAS for a $\ell(x,y) = 0.5(x-1)^2 + 0.5y^2$ (left) the distance from the optimal point (middle) and the corresponding magnitude of the gradients (right). The circular lines are the contour lines of the loss function. The dashed line is the simplex $x + y = 1$, and the trajectories are the solid lines with circles at their ends.

8.3 Regret and Correct Selection in Statistical Setting

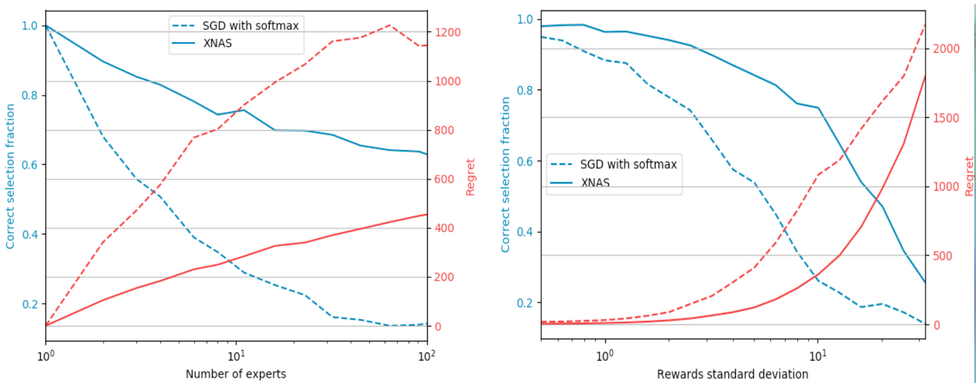


Figure 8: Correct selection fraction and regret of optimizers.

We consider a statistical setup for comparing XNAS with the common Gradient Descent (GD) with softmax, described in section 3.2.1. This setup simulates the iterative architecture optimization and final selection of the top expert for a single forecaster.

Two forecasters are compared, XNAS and GD with softmax. Both receive noisy independent and identically distributed (i.i.d) rewards of N experts. Each expert has an initial i.i.d bias $\{b_i\}_{i=1}^N \sim \mathrm{N}(0,1)$ simulating its inherent value, so its rewards satisfy $R_{t,i} \sim \mathrm{N}(b_i, \sigma_R^2)$ for $i = 1, \ldots, N$ and $t = 1, \ldots, T$, where $\mathrm{N}(\mu, \sigma^2)$ is a Gaussian distribution with a mean μ and a standard deviation σ.

The first forecaster updates its weights using GD with softmax update rule from Equation 6 (full derivation in section 8.4.1), common to previous NAS methods, while the second is using Algorithm 1.

The forecasters use their update rules to update weights along the run. At the end of the run each selects the expert with the largest weight. A correct classification satisfies $\max_{i=1,\ldots,N} \alpha_i = \max_{i=1,\ldots,N} b_i$. The average regret of those runs is also calculated based on equation 2.

Figure 8 shows a mean of 1000 Monte-Carlo runs, each of 1000 time-steps, plotting the regret and the fraction of correct selection (classification). In Figure 8 (left), both terms are plotted versus a varying number of experts. It can be seen that the regret of XNAS is significantly smaller, scaling with the number of experts like $O(\sqrt{\ln N})$, as implied by its regret upper bound in equation 4, while GD regret scales like $O(\sqrt{N})$ [14].

In Figure 8 (right), the noise standard deviation σ_R is varying, making it harder to correctly classify the expert with the highest bias. Again, XNAS dominates GD with softmax, which is more sensitive to the noisy rewards due to the 'late bloomers' described in 3.2.1, e.g. the best experts might suffer some large penalties right at the beginning due to the noise, thus might not recover for GD with softmax.
In both graphs it can be seen that the correct selection fraction is monotonically decreasing as the regret is increasing. This gives an additional motivation for the use of the regret minimization approach as a criterion for neural architecture search.

8.4 Gradients Derivations

For the comparison to previous work [25], we consider the decision variables $\alpha_{t,i} = \ln v_{t,i}$, as the right hand side is defined at (1).

8.4.1 The Derivation of Derivatives in the General Case

$$\begin{aligned}
\frac{\partial u_{t,j}}{\partial \alpha_{t,i}} &= \frac{\partial}{\partial \alpha_{t,i}} \frac{e^{\alpha_{t,j}}}{\sum_{k=1}^{N} e^{\alpha_{t,k}}} \\
&= \frac{\delta_{i,j} e^{\alpha_{t,i}} \cdot \sum_{k=1}^{N} e^{\alpha_{t,k}} - e^{\alpha_{t,j}} \cdot e^{\alpha_{t,i}}}{\left(\sum_{k=1}^{N} e^{\alpha_{t,k}}\right)^2} \\
&= \frac{e^{\alpha_{t,i}}}{\sum_{k=1}^{N} e^{\alpha_{t,k}}} \left(\delta_{i,j} - \frac{e^{\alpha_{t,j}}}{\sum_{k=1}^{N} e^{\alpha_{t,k}}}\right) \\
&= u_{t,i} \left(\delta_{i,j} - u_{t,j}\right) \qquad (34)
\end{aligned}$$

where $\delta_{i,j} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$ is the Kronecker delta.

Observe that:

$$\begin{aligned}
\frac{\partial \ell_t(p_t)}{\partial u_{t,i}} &= \nabla_{p_t} \ell_t(p_t)^T \cdot \frac{\partial p_t}{\partial u_{t,i}} \\
&= \nabla_{p_t} \ell_t(p_t)^T \cdot \frac{\partial}{\partial u_{t,i}} \sum_{j=1}^{N} u_{t,j} f_{t,j} \\
&= \nabla_{p_t} \ell_t(p_t)^T \cdot f_{t,i} \qquad (35)
\end{aligned}$$

Finally,

$$\begin{aligned}
\frac{\partial \ell_t(p_t)}{\partial \alpha_{t,i}} &= \sum_{j=1}^{N} \frac{\partial \ell_t(p_t)}{\partial u_{t,j}} \cdot \frac{\partial u_{t,j}}{\partial \alpha_{t,i}} \\
&= \sum_{j=1}^{N} \nabla_{p_t} \ell_t(p_t)^T \cdot f_{t,j} \cdot u_{t,i} \left(\delta_{i,j} - u_{t,j}\right) \qquad (36) \\
&= \nabla_{p_t} \ell_t(p_t)^T \cdot u_{t,i} \left(f_{t,i} - \sum_{j=1}^{N} u_{t,j} f_{t,j}\right) \\
&= \nabla_{p_t} \ell_t(p_t)^T \cdot u_{t,i} \left(f_{t,i} - p_t\right) \qquad (37)
\end{aligned}$$

where (36) is due to (34) and (35).

8.4.2 The Derivation of Derivatives for the 3D Axes Problem

In this section we derive the derivatives with respect to the x, y, z axes for the toy problem introduced at section 8.1. In this case,

$$\begin{aligned} f_{t,x} &\equiv (1,0,0)^T & u_{t,x} &= x_t \\ f_{t,y} &\equiv (0,1,0)^T & u_{t,y} &= y_t \\ f_{t,z} &\equiv (0,0,1)^T & u_{t,z} &= z_t \end{aligned} \tag{38}$$

Hence,

$$p_t = \sum_{i \in \{x,y,z\}} u_{t,i} f_{t,i} = x_t(1,0,0)^T + y_t(0,1,0)^T + z_t(0,0,1)^T = (x_t, y_t, z_t)^T \tag{39}$$

such that for a loss function ℓ,

$$\nabla_{p_t} \ell(p_t) = \left(\frac{\partial \ell(p_t)(p_t)}{\partial x_t}, \frac{\partial \ell(p_t)(p_t)}{\partial y_t}, \frac{\partial \ell(p_t)(p_t)}{\partial z_t} \right)^T = (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t), \partial_{z_t}\ell(p_t))^T \tag{40}$$

Then, setting (38) and (40) in (37), we have,

$$\begin{aligned} \partial_{\alpha_x}\ell(p_t) = \frac{\partial \ell(p_t)}{\partial \alpha_x} &= (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t), \partial_{z_t}\ell(p_t))^T \cdot x_t \left((1,0,0)^T - (x_t, y_t, z_t)^T \right) \\ &= (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t), \partial_{z_t}\ell(p_t))^T \cdot x_t \left(y_t + z_t, -y_t, -z_t \right)^T & (41) \\ &= x_t \left((y_t + z_t)\partial_{\alpha_x}\ell(p_t) - y_t \partial_{\alpha_y}\ell(p_t) - z_t \partial_{\alpha_z}\ell(p_t) \right) & (42) \end{aligned}$$

where (41) is since $(x, y, z) \in \Delta$, defined in section 8.1. By symmetry we have,

$$\partial_{\alpha_y}\ell(p_t) = y_t \left((x_t + z_t)\partial_{\alpha_y}\ell(p_t) - x_t \partial_{\alpha_x}\ell(p_t) - z_t \partial_{\alpha_z}\ell(p_t) \right) \tag{43}$$

$$\partial_{\alpha_z}\ell(p_t) = z_t \left((x_t + y_t)\partial_{\alpha_z}\ell(p_t) - x_t \partial_{\alpha_x}\ell(p_t) - y_t \partial_{\alpha_y}\ell(p_t) \right) \tag{44}$$

The update terms for XNAS are according to (35),

$$\nabla_{p_t}\ell(p_t)^T \cdot f_{t,x} = (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t), \partial_{z_t}\ell(p_t)) \cdot (1,0,0)^T = \partial_{x_t}\ell(p_t) \tag{45}$$

$$\nabla_{p_t}\ell(p_t)^T \cdot f_{t,y} = (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t), \partial_{z_t}\ell(p_t)) \cdot (0,1,0)^T = \partial_{y_t}\ell(p_t) \tag{46}$$

$$\nabla_{p_t}\ell(p_t)^T \cdot f_{t,z} = (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t), \partial_{z_t}\ell(p_t)) \cdot (0,0,1)^T = \partial_{z_t}\ell(p_t) \tag{47}$$

8.4.3 The Derivation of Derivatives for a 2D Axes Problem

In this section we derive the derivatives with respect to the x, y axes for a two dimensional toy problem. Similar to section 8.1 where a three dimensional problem was considered, now we consider only two axes. Each axis represents an expert of a constant prediction,

$$\begin{aligned} f_{t,x} &\equiv (1,0)^T & u_{t,x} &= x_t \\ f_{t,y} &\equiv (0,1)^T & u_{t,y} &= y_t \end{aligned} \tag{48}$$

Where $x_t + y_t \equiv 1$. Hence,

$$p_t = \sum_{i \in \{x,y\}} u_{t,i} f_{t,i} = x_t(1,0)^T + y_t(0,1)^T = (x_t, y_t)^T \tag{49}$$

such that for a loss function ℓ,

$$\nabla_{p_t}\ell(p_t) = \left(\frac{\partial \ell(p_t)}{\partial x_t}, \frac{\partial \ell(p_t)}{\partial y_t} \right)^T = (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t))^T \tag{50}$$

Then, setting (48) and (50) in (37), we have,

$$\begin{aligned} \partial_{\alpha_x}\ell(p_t) = \frac{\partial \ell(p_t)}{\partial \alpha_x} &= (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t))^T \cdot x_t \left((1,0)^T - (x_t, y_t)^T \right) \\ &= (\partial_{x_t}\ell(p_t), \partial_{y_t}\ell(p_t))^T \cdot x_t \left(y_t, -y_t \right)^T & (51) \\ &= x_t y_t \left(\partial_{x_t}\ell(p_t) - \partial_{y_t}\ell(p_t) \right) & (52) \end{aligned}$$

where (51) is since $x_t + y_t \equiv 1$. By symmetry we have,

$$\partial_{\alpha_y}\ell(p_t) = x_t y_t \left(\partial_{y_t}\ell(p_t) - \partial_{x_t}\ell(p_t) \right) = -\partial_{\alpha_x}\ell(p_t) \tag{53}$$

8.5 The Mean Normalized Entropy

In this section we provide the technical calculation details of the mean normalized entropy, referred to in section 3.2.2. The normalized entropy of forcaster (i, j) is calculated at the end of the search as following,

$$\bar{H}_T^{(i,j)} = -\frac{1}{\ln(N)} \sum_{i=1}^{N} u_{T,i}^{(i,j)} \ln\left(u_{T,i}^{(i,j)}\right) \tag{54}$$

The mean is taken over all the forecasters in a normal cell, i.e.

$$\bar{H}_T = \frac{1}{|\mathcal{I}| \cdot |\mathcal{J}|} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \bar{H}_T^{(i,j)} \tag{55}$$

where $\mathcal{I}$ and $\mathcal{J}$ are the sets of indices i and j respectively.

9 Detailed Experiments Setting

9.1 Classification Datasets Details

In this section we will describe the additional datasets that were used for transferability tests in section 4.3

CINIC-10: [8] is an extension of CIFAR-10 by ImageNet images, down-sampled to match the image size of CIFAR-10. It has $270,000$ images of 10 classes, i.e. it has larger train and test sets than those of CIFAR-10.

CIFAR-100: [42] A natural image classification dataset, containing 100 classes with 600 images per class. The image size is 32x32 and the train-test split is $50,000$:$10,000$ images respectively.

FREIBURG: [19] A groceries classification dataset consisting of 5000 images of size 256x256, divided into 25 categories. It has imbalanced class sizes ranging from 97 to 370 images per class. Images were taken in various aspect ratios and padded to squares.

SVHN: [28] A dataset containing real-world images of digits and numbers in natural scenes. It consists of $600,000$ images of size 32x32, divided into 10 classes. The dataset can be thought of as a real-world alternative to MNIST, with an order of magnitude more images and significantly harder real-world scenarios.

FMNIST: [47] A clothes classification dataset with a $60,000$:$10,000$ train-test split. Each example is a grayscale image of size 28x28, associated with a label from 10 classes of clothes. It is intended to serve as a direct drop-in replacement for the original MNIST dataset as a benchmark for machine learning algorithms.

9.2 CIFAR-10 XNAS Search details

Data pre-processing. We apply the following:

- Centrally padding the training images to a size of 40x40.
- Randomly cropping back to the size of 32x32.
- Randomly flipping the training images horizontally.
- Auto augment.
- Standardizing the train and validation sets to be of a zero-mean and a unit variance.

Operations and cells. We select from the operations mentioned in 4.1, used with stride 1 on normal cells, and with stride 2 on reduction cells in edges connected to the two previous cells. Other edges in reduction cells are used with stride 1. Convolutional layers are padded so that the spatial resolution is kept. The operations are applied in the order of ReLU-Conv-BN. Following [29],[33], depthwise separable convolutions are always applied twice. The cell's output is a 1x1 convolutional layer applied on all of the cells' four intermediate nodes' outputs concatenated, such that the number of channels is preserved. In CIFAR-10, the search lasts up to 0.3 days on NVIDIA GTX 1080Ti GPU.

9.3 Train Details

CIFAR-10. The training architecture consists of stacking up 20 cells: 18 normal cells and 2 reduction cells, located at the $1/3$ and $2/3$ of the total network depth respectively. For the three architectures XNAS-Small, XNAS-Medium and XNAS-Large, the normal cells start with 36, 44 and 50 channels respectively, where we double the number of channels after each reduction cell. We trained the network for 1500 epochs using a batch size of 96 and SGD optimizer with nesterov-momentum of 0.9. Our learning rate regime was composed of 5 cycles of power cosine annealing learning rate [17], with amplitude decay factor of 0.5 per cycle and initial value of 0.025. For regularization we used cutout [9] with a length of 16, scheduled drop-path [22] of 0.2, auxiliary towers [39] after the last reduction cell with a weight of 0.4, label smoothing [40] of 0.1, AutoAugment [7] and weight decay of $3 \cdot 10^{-4}$.

ImageNet. Our training architecture starts with stem cells that reduce the input image resolution from 224 to 56 (3 reductions), similar to [25]. We then stack 14 cells: 12 normal cells and 2 reduction cells. The reduction cells are placed after the fourth and eighth normal cells. The normal cells start with 46 channels, as the number of channels is doubled after each reduction cell. We trained the network, with a batch size of 1280 for 250 epochs, with one cycle of power cosine learning rate, weight decay of 10^{-4} and nesterov-momentum of 0.9. We add an auxiliary loss after the last reduction cell with a weight of 0.4. During training, we normalize the input image and crop it with a random cropping factor in the range of 0.08 to 1. In addition we use auto-augment and horizontal flipping. During testing, we resize the input image to the size of 256x256 and applying a fixed central crop to the size of 224x224.

Additional datasets. Our additional classification datasets consist of CINIC-10 [8], CIFAR-100 [42], FREIBURG [19], SVHN [28] and FashionMNIST [47]. Their training scheme was similar to the one used for CIFAR-10, described at 9.3, with some minor adjustments and modifications. For the FREIBURG dataset, we resized the original images from 256x256 to 96x96 and used a batch size of 16. For CINIC-10, we trained the network for 800 epochs instead of 1500, since this dataset is much larger then CIFAR-10. For Fashion-MNIST we edited the learned augmentations regime to fit a dataset of grayscale images.

10 Cells Learned by XNAS

Figure 9 presents the cells learned by XNAS on CIFAR-10.

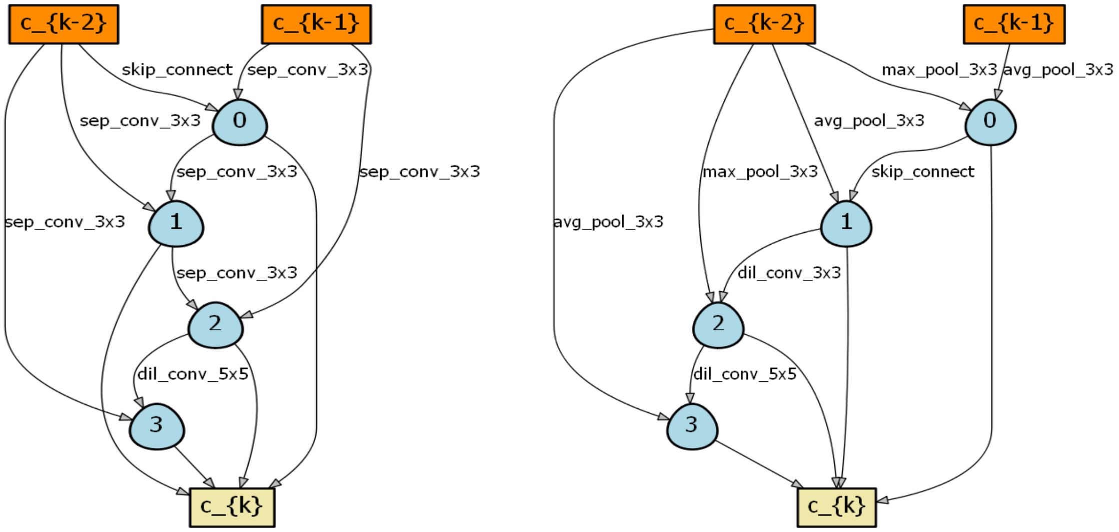


Figure 9: XNAS learned normal and reduction cells on CIFAR-10.

Cell depth: When comparing the cell in Figure 9 to other reported NAS cells [29, 25, 33, 56, 50, 24], it is clear visually that XNAS cell is "deeper" in some sense.

We wish to define a metrics for a cell "depth". A cell C_k contains four nodes. Each node can be connected to the previous cell's nodes or to the two previous cells' outputs C_{k-2}, C_{k-1}. Let us index the previous cells C_{k-2}, C_{k-1} and the cell's four nodes as $0, \ldots, 5$ respectively.

Define the depth of each connection in a cell as the index of the node (or previous cell) it came from. A simple metric for a cell depth can be the average depth of its inner connections. Table 4 presents the depth of XNAS and other NAS methods normal cells.

Cell	Depth
SNAS [50]	0.625
PNAS [24]	0.5
Amoeba-A [33]	0.9
NASNet [56]	0.4
DARTS [25]	0.625
ASAP [29]	0.875
XNAS	1.375

Table 4: XNAS and other NAS methods normal cell Depth.

We can see from Table 4 that the XNAS cell is much deeper then a typical NAS cell. This observation could provide a hint about the superior performance of the XNAS cell. Unlike most NAS methods, that usually produce shallow cells, XNAS cell utilizes better the possible degree of freedom of a NAS cell design, yielding a deeper and more complex architecture.