Should You Even Train a Neural Network? A Forecasting-Based Early Rejection Strategy under Budget Constraints (draft)

Santiago Ramírez Castañeda

Universidad de La Sabana

Faculty of Engineering: Maestría en Analítica Aplicada

Félix Mohr

02 de septiembre de 2025

Should You Even Train a Neural Network? A Forecasting-Based Early Rejection Strategy under Budget Constraints (draft)

Abstract

Neural networks have demonstrated remarkable performance across diverse tasks, yet their design remains resource-intensive due to the vast search space of possible architectures and the high cost of full training. This thesis investigates whether early-training signals and lightweight evaluation strategies can be effectively used to predict whether neural network architectures will achieve competitive performance compared to strong non-neural models, thereby enabling a fast and resource-efficient neural architecture search (NAS) framework. To address this question, we propose Effort-Based Evolution for Neural Architecture Search (EBE-NAS), a method that introduces the concept of Effort Computation, where candidate architectures are allocated a constrained and uniform amount of training effort for evaluation. EBE-NAS integrates this principle within an evolutionary search strategy, balancing rapid exploration with selective refinement of promising architectures. The framework is evaluated against traditional machine learning baselines, fully trained oracle architectures, and strong non-neural models. Results demonstrate that EBE-NAS can reliably forecast and select architectures that approach oracle-level performance while remaining competitive with non-neural baselines, all at a fraction of the computational cost. These findings highlight the potential of early-training signals as a viable basis for efficient NAS.

Introduction

Background

Problem Definition

Research Question

Can early-training signals and lightweight evaluation strategies be effectively used to predict whether neural network architectures will achieve competitive performance compared to strong non-neural models, enabling a fast and resource-efficient NAS framework?

Research Objectives

General Objective

To develop a fast and resource-efficient neural architecture search (NAS) framework capable of determining whether neural networks are likely to yield competitive performance

compared to strong non-neural models, by leveraging early training information and lightweight evaluation strategies.

Specific Objectives

- Develop a predictive evaluation mechanism that leverages early-training signals to forecast which neural architectures are likely to achieve competitive performance.
- Implement an evolutionary search strategy within the NAS toolkit to efficiently explore and optimize candidate architectures based on the predictive evaluation and their gap with non neural models.
- Validate and benchmark the NAS toolkit by assessing its efficiency and accuracy in discovering top-performing architectures compared to conventional NAS methods and baseline models.

Methodology

Overview of the NAS Framework

The proposed NAS toolkit is designed to identify top-performing neural network architectures with minimal computational resources. The workflow consists of candidate architecture generation, predictive evaluation using early-training signals, evolutionary search for optimization, and benchmarking against strong non-neural models.

Candidate Architecture Generation

Candidate architectures form the foundation of the NAS process. In the proposed toolkit, each candidate is represented as a neural network configuration, including hyperparameters such as the number of layers, neurons per layer, activation functions, and other relevant design choices defined in the search space.

The generation process is implemented as follows:

1. **Population Initialization:** A predefined number of candidate architectures is created in each generation. To ensure reproducibility and diversity, a deterministic sampling strategy

is applied where the random seed for each candidate is derived from its index combined with a base seed. This approach balances randomness with controlled variation in the candidate population.

2. **Architecture Sampling:** The toolkit samples architectures from the defined search space. The search space encapsulates the allowed ranges for all neural network parameters, ensuring that generated candidates are valid for the target task. For each sampled architecture, a corresponding neural network model is instantiated. This method accounts for the task type (e.g., classification or regression) and configures the model accordingly.

3. **Candidate Tracking:** Each model and its architecture configuration are wrapped into a Candidate object, this will store its respective metrics after training. All candidate objects are stored in a dictionary keyed by their unique IDs. The total number of candidates generated over the NAS process is tracked as well. This allows the evolutionary algorithm to manage populations, apply selection pressures, and maintain lineage information across generations.

One-Effort (OE) Training and Effort Computation

After candidate architectures are generated, each is evaluated using **One-Effort (OE) training**, where the goal is to measure performance per unit of computational effort. This strategy forms the foundation of **Effort-Based Evolution (EBE)**, prioritizing candidates that achieve high performance for minimal effort.

One-Effort Training Procedure

1. **Active Candidate Selection:** All candidates in the generation are iterated over, with their model, architecture, batch size, and seed retrieved to ensure reproducible training outcomes.

2. **Dynamic Instance Budget:** Each candidate has a *dynamic instance budget*, defining the number of training samples used in the current OE unit. A corresponding fraction of the training data is sampled to create the training DataLoader.

3. **Training Execution:** Each candidate is trained for one OE unit, which may involve a partial epoch or a subset of the data, depending on its instance budget. After training, metrics such as training loss and accuracy are logged. This provides an *early, resource-aware signal* of a candidate's potential without committing to full training.

Effort Computation

1. **Definition:** Effort is quantified as:

$$\text{Effort} = \text{Dataset Fraction} \times \text{Number of OE Units Completed} \tag{1}$$

   where:

   - *Dataset Fraction:* Portion of the training data used in the current OE unit.
   - *Number of OE Units Completed:* Cumulative OE units the candidate has undergone.

2. **Logging and Tracking:** Each candidate's cumulative effort is updated after every OE unit. Candidates achieving high predictive performance with low cumulative effort are prioritized in subsequent generations.

3. **Role in EBE:** Effort serves as a core selection criterion. The evolutionary algorithm leverages these effort-weighted metrics to guide architecture propagation, promoting candidates that maximize performance per unit of computational effort while pruning inefficient or low-performing architectures.

Predictive Evaluation Mechanism

A central component of the NAS toolkit is the **predictive evaluation mechanism**, which estimates the potential final performance of each candidate architecture using data obtained from *One-Effort (OE) training*. This mechanism enables early selection of promising architectures without fully training all candidates, improving both speed and resource efficiency.

Forecasting Accuracy with the Rational Model

By default, the toolkit employs a *rational function model* to forecast validation accuracy based on the cumulative effort invested:

$$\hat{y}(E) = \frac{a}{E + b} \tag{2}$$

where E represents the cumulative effort (dataset fraction $\times$ number of OE units), and a and b are parameters fitted to the candidate's observed validation accuracies. Parameters are estimated using nonlinear least-squares curve fitting, and forecasts are clipped to the $[0, 1]$ range to represent valid accuracy values.

This approach allows the framework to project whether a candidate is likely to achieve competitive performance before fully committing computational resources.

Additional Lightweight Predictive Signals

To further refine candidate evaluation, the following supplementary metrics are computed from the OE training history:

- **Slope of Validation Accuracy (s):** Measures the rate of improvement over the observed OE units:

  $$s = \frac{\text{val\_acc}_{\text{last}} - \text{val\_acc}_{\text{first}}}{E_{\text{last}} - E_{\text{first}} + \varepsilon} \tag{3}$$

  where ε prevents division by zero. A positive slope indicates a candidate that is still improving rapidly.

- **Variance of Validation Accuracy (σ^2):** Captures stability of performance across OE units. Low variance suggests consistent learning, while high variance may indicate unstable training.

- **Gap of Last Accuracy (g):** The difference between the most recent validation accuracy

and the mean of previous accuracies:

$$g = \text{val\_acc}_{\text{last}} - \frac{1}{n-1}\sum_{i=1}^{n-1}\text{val\_acc}_i \tag{4}$$

This highlights sudden improvements or regressions in candidate performance.

Scoring Candidates and Baseline Comparison

Each candidate's forecasted accuracy is compared against a predefined *baseline metric* to guide selection. The toolkit logs whether a candidate is likely to surpass the baseline:

$$\text{fcst\_above\_baseline} = \begin{cases} 1 & \hat{y}(E) \geq \text{baseline} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Candidate scores are computed by combining forecasted accuracy, recent performance, and momentum indicators (slope and gap). For example:

- Candidates below the baseline but with positive momentum receive a moderate score:

$$\text{score} = 0.3 \cdot \text{val\_acc}_{\text{last}} + 0.5 \cdot \hat{y}(E) + 0.2 \cdot s \tag{6}$$

- Candidates exceeding the baseline are rewarded based on forecast gain and slope:

$$\text{score} = 0.6 \cdot \hat{y}(E) + 0.3 \cdot (\hat{y}(E) - \text{baseline}) + 0.1 \cdot s \tag{7}$$

This multi-signal scoring enables the NAS toolkit to balance predicted final performance, observed trends, and training efficiency, ensuring that only candidates with high potential for success and low computational cost are prioritized in the evolutionary search process.

Evolutionary Search Strategy

The evolutionary search strategy governs the iterative generation, selection, and refinement of candidate architectures within the NAS framework. It relies on the predictive

evaluation mechanism, which provides forecasted validation accuracy and additional lightweight signals, and integrates these with the *Effort-Based Evolution (EBE)* principle.

Candidate Pruning and Selection

After training the models and forecasting their future performance, the toolkit identifies underperforming candidates for removal. This process incorporates both predictive forecasts and observed validation metrics:

- Candidates below a *baseline performance metric* are flagged for potential removal.
- If forecasts are unavailable, raw validation accuracy is used as the fallback criterion.
- The top 10% of candidates by validation accuracy are preserved as *elite* individuals to maintain high-performing architectures.
- A configurable percentile (e.g., 15%) of the worst candidates is removed to free computational resources.

This selective pruning ensures that resources are concentrated on candidates with the highest potential for success, while preventing stagnation by continuously introducing new genetic material.

Generation of New Models

The toolkit replenishes the population by generating new candidates to replace dropped individuals. New candidates are created using a combination of:

- **Basic models:** Architectures sampled directly from the defined search space to maintain diversity.
- **Evolved models:** Offspring created through crossover and mutation of high-ranking parent architectures. Crossover combines architectural components from two parents, while mutation introduces stochastic alterations in some hyperparameters (i.e, hidden layers, learning rate, dropout, batch size). This process ensures exploration of promising regions in the search space while leveraging prior knowledge.

The proportion of basic versus evolved models is configurable (e.g., 50% each), allowing a balance between exploration and exploitation.

Weighted Random Parent Selection

Parents for crossover are chosen using a *weighted random selection* based on candidate scores derived from the predictive evaluation mechanism. Higher-scoring candidates have a greater probability of being selected, increasing the likelihood that successful traits propagate to the next generation. If insufficient high-scoring candidates exist, uniform random selection ensures diversity is preserved.

Crossover and Mutation

- **Crossover:** For each gene in the architecture (e.g., hidden layers, scheduler parameters, activation functions), a value is inherited from either parent using stochastic selection rules.
- **Mutation:** Each architectural parameter has a probability of mutation (configurable, e.g., 30%). Mutations can add or perturb hidden layers, adjust learning rate or batch size, and tweak dropout. This introduces diversity and prevents premature convergence.

Generation Update and Resource Management

Once new candidates are created:

- They are merged with the surviving population to form the next generation.
- GPU and CPU memory are managed to prevent leaks, with pruned candidates moved to CPU before deletion and caches cleared.
- The toolkit updates internal counters for the number of individuals and total OE units completed, maintaining accurate tracking of effort.

This iterative process continues until termination criteria are met, such as reaching a maximum number of generations, or exhausting the computational budget in time.

Benchmarking

The datasets on the which the tool was tested correspond to those found by (Zimmer et al., 2020) as part of LCBench. For each of the 35 datasets, the following experiments were conducted:

1. **Standard Non Neural Model performance** A broad set of classical machine learning models was trained on the classification datasets. The models included Logistic Regression, Support Vector Classifier (SVC), Decision Trees, Random Forests, Gradient Boosting, AdaBoost, k-Nearest Neighbors, Gaussian Naive Bayes, Quadratic Discriminant Analysis (QDA), and Linear Discriminant Analysis (LDA).

2. **Best Non Neural Model performance** This was performed using the AutoML software NaiveAutoML (Mohr & Wever, 2023) on its standard configuration with a set random state for replicability. The time taken by this tool to identify the best pipeline was recorded as well as their metrics.

3. **EBE-NAS** Using the time taken by NaiveAutoML, triple the budget was assigned to the tool to run the training loop up until a max number of epochs were reached or the time budget was exhausted.

4. **Oracle training** The Oracle consists of fully training all sampled architectures by EBE-NAS to convergence, independent of resource constraints. This experiment provides two insights: (i) the true potential of the architecture pool, and (ii) a reference point for evaluating the effectiveness of one-effort training forecasts.

Results

Non-neural methods outperform neural ones on this dataset. However, EBE efficiently reached the same best neural performance as an exhaustive search, cutting computation from 2.6 hours to 1 minute.

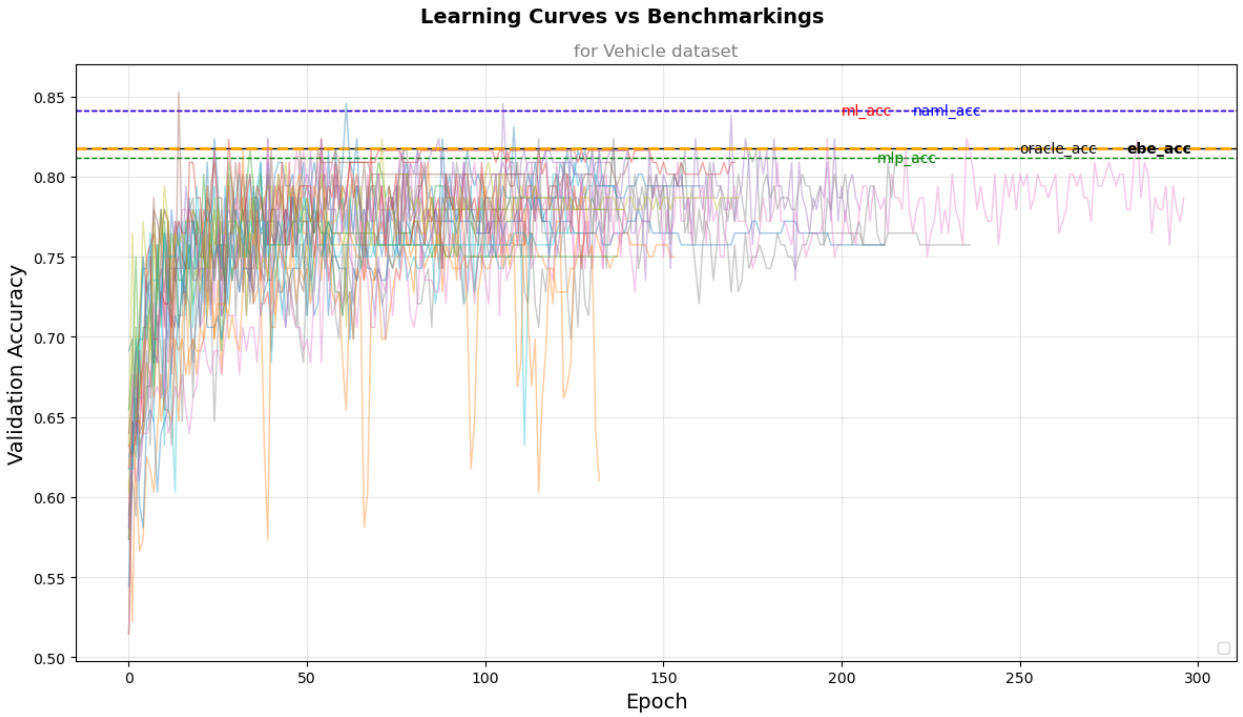

Figure 1

Horizontal lines correspond to test performance

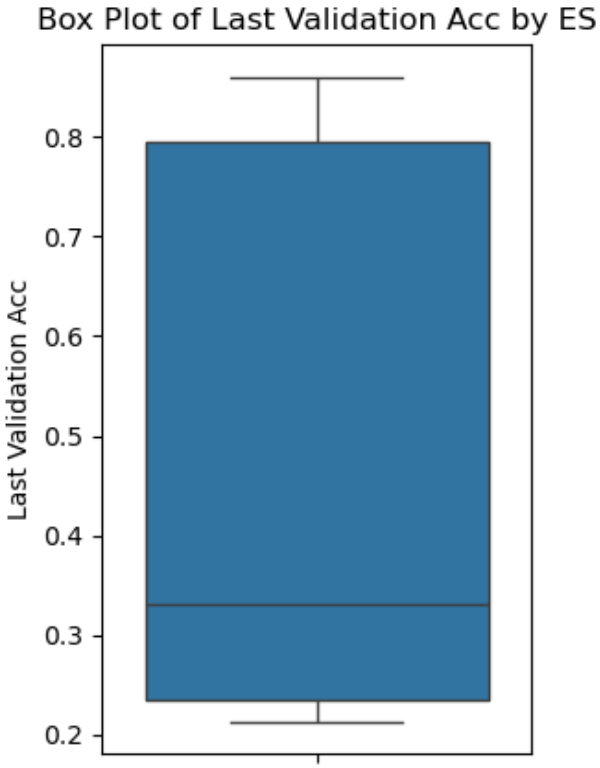

Figure 2
All 820 models were trained by ES

Referencias

Mohr, F., & Wever, M. (2023). Naive automated machine learning. *Machine Learning*, *112*(4), 1131–1170. https://doi.org/10.1007/s10994-022-06200-0

Zimmer, L., Lindauer, M., & Hutter, F. (2020). Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(9), 3079–3090. https://doi.org/10.1109/TPAMI.2021.3067763